В. П. Дьяконов

Энциклопедия компьютерной алгебры

Москва

УДК 32.973.26-018.2
ББК 004.438
Д93

Д93 **Дьяконов В. П.**
Энциклопедия компьютерной алгебры. – М.: ДМК Пресс. – 1264 с.: ил.

ISBN 978-5-94074-490-0

Первая в России энциклопедия по компьютерной алгебре, ориентированная на пользователей систем компьютерной математики, нуждающихся в выполнении аналитических вычислений и их численной и графической визуализации. Содержит описание возможностей систем компьютерной алгебры Maple 9.5/10/11, Mathematica 5.1/5.2/6, Mathcad 11/12/13/14, Derive 5/6 и MuPAD 2.5/3/4. Особое внимание уделено их новейшим реализациям. Дает основополагающие понятия о системах компьютерной математики, подкрепленные более чем тысячью конкретных и наглядных примеров. Рассмотрены средства компьютерной математики, реализованные аппаратно.

Книга предназначена для студентов и преподавателей университетов и вузов, инженеров и научно-технических работников.

УДК 519.6
ББК В162я73

ISBN 978-5-94074-490-0

Краткое содержание

Содержание

Глава 14. Программирование в системах компьютерной алгебры ... 1033

Введение

В конце XX в. возникло и получило бурное развитие новое фундаментальное научное направление – *компьютерная математика* [1–8], которое зародилось на стыке математики и информатики. Это направление ныне достойно представлено на аппаратном уровне новейшими научными микрокалькуляторами [15] с встроенными системами компьютерной математики, сигнальными, звуковыми и видеопроцессорами, математическими сопроцессорами (ныне интегрированными с микропроцессорами компьютеров), цифровыми измерительными приборами и многими другими устройствами. Но основным продуктом компьютерной математики стали программные *системы компьютерной математики* – СКМ, в частности *системы компьютерной алгебры* – СКА.

Следует отметить, что в современной мировой литературе [3–12] имеются разночтения в отношении термина «компьютерная математика». Иногда этот термин отождествляется с математикой, которая преподается студентам специальности «информатика». Естественно, что в книгах, посвященных этому направлению компьютерной математики, излагаются в сущности основы обычной фундаментальной математики, приправленные отдельными материалами, представляющими интерес для разработчиков вычислительных устройств и программных средств [9–12].

Однако примерно с четверть века назад стало ясно, что компьютеры, при соответствующем *особом* алгоритмическом, математическом и программном обеспечении [3–8], способны решать не только численные, но и аналитические задачи, представляя результаты в виде математических формул. Первыми были задачи, относящиеся к *алгебре* как к наиболее формализованной и продвинутой области математики.

Данное направление возникло заметно раньше, чем появились коммерческие СКМ (в начале 80-х гг. XX в.). Первые работы в области компьютерной алгебры (КоАл) были выполнены еще в 60–70-х гг. прошлого века в ряде ведущих математических школ бывшего СССР. Из них особо следует отметить научную школу советского академика В. А. Глушкова, создавшую первые в мире малые ЭВМ серии «Мир» и язык для таких вычислений «Аналитик» [3]. На этих ЭВМ выполнялись достаточно сложные аналитические вычисления. К сожалению, в СССР данные разработки в должной мере не были поддержаны, и последующее развитие этого направления компьютерной математики получило развитие уже за рубежом в виде СКМ.

Побудительным мотивом в подготовке данной книги стали наметившиеся мотивы интеграции различных СКМ друг с другом и решения задач в ряде СКА и СКМ. Можно привести массу таких примеров. Например, система Derive вошла в массовые микрокалькуляторы фирмы Texas Instruments, превратив их из электронных арифмометров для подсчета стоимости покупок в миниатюрные устрой-

ства для сложнейших научно-технических расчетов, в том числе аналитических. Ядро мощной СКА Maple вошло в состав популярной СКМ Mathcad и мощной матричной системы MATLAB, превратив их в универсальные СКМ. Новейшая Mathcad 14 использует уже ядро от системы MuPAD. Можно привести ряд и других таких примеров, например интеграции систем Maple с системами Mathematica и MATLAB. Они показывают, что пользователи систем компьютерной математики уже не могут замыкаться на использовании какой-либо одной системы и должны владеть навыками применения нескольких систем. Происходит явная интеграция различных систем СКА. В то же время литературы, проводящей этот тезис в жизнь, пока нет – исключением была давно выпущенная и распроданная книга [1].

Наиболее известными средствами компьютерной математики стали ее программные средства – СКМ [21–121]. Первое время такие системы решали различные задачи только в численном виде. Это были табличные процессоры, такие как нынешний Excel, и всевозможные программные калькуляторы, например такие, как Eureka и Mercury [25, 26]. Отдельным направлением были программы для статистических вычислений, например Statistica [119, 120]. Но постепенно были созданы неизмеримо более мощные системы, способные решать математические задачи в символьном (аналитическом) виде, представляя результаты в виде математических формул. Важными в СКМ всегда были графическая и численная визуализации таких вычислений.

Известно, что аналитические решения математических задач, если они возможны, имеют более общий характер, чем численные решения. Поэтому для большинства задач науки и техники, как правило, весьма желательно решение в аналитическом виде, а лишь затем интерпретация и визуализация решений в более частном, численном или графическом виде. Нередко удается оптимизировать численное решение сложных задач, получив его вначале в аналитическом виде, а затем уже использовать возможности численного решения и его графической визуализации. На этом построена, например, оптимизация задач в популярной системе Mathcad [36–62]. Но, как правило, аналитические, численные и графические средства применяются совместно. В результате разница между СКА и СКМ стала условной, и все современные СКА стали универсальными СКМ. Это наглядно подтверждает материал данной книги.

Достойным представителем систем компьютерной алгебры стали системы Maple [63–77, 198–200], изначально созданные в университетских кругах. С возможностями новейшей системы Maple 12 можно познакомиться на сайте производителя этой системы, корпорации Waterloo Maple, Inc. (Канада) – www.maplesoft.com. По данным крупной поисковой системы Google, число ссылок на Maple в Интернете самое большое и достигает более 60 миллионов.

По системе Maple выпущены многие сотни книг, и среди них есть русскоязычные книги автора – рис. 0.1. Книга [68], завершившая серию книг по различным версиям системы Maple [63–68], сделала автора победителем всероссийского конкурса «Лучшая научная книга 2006», проводимого Фондом развития отечественного образования (www.fondro.sochi.ru), в номинации «Информационные техно-

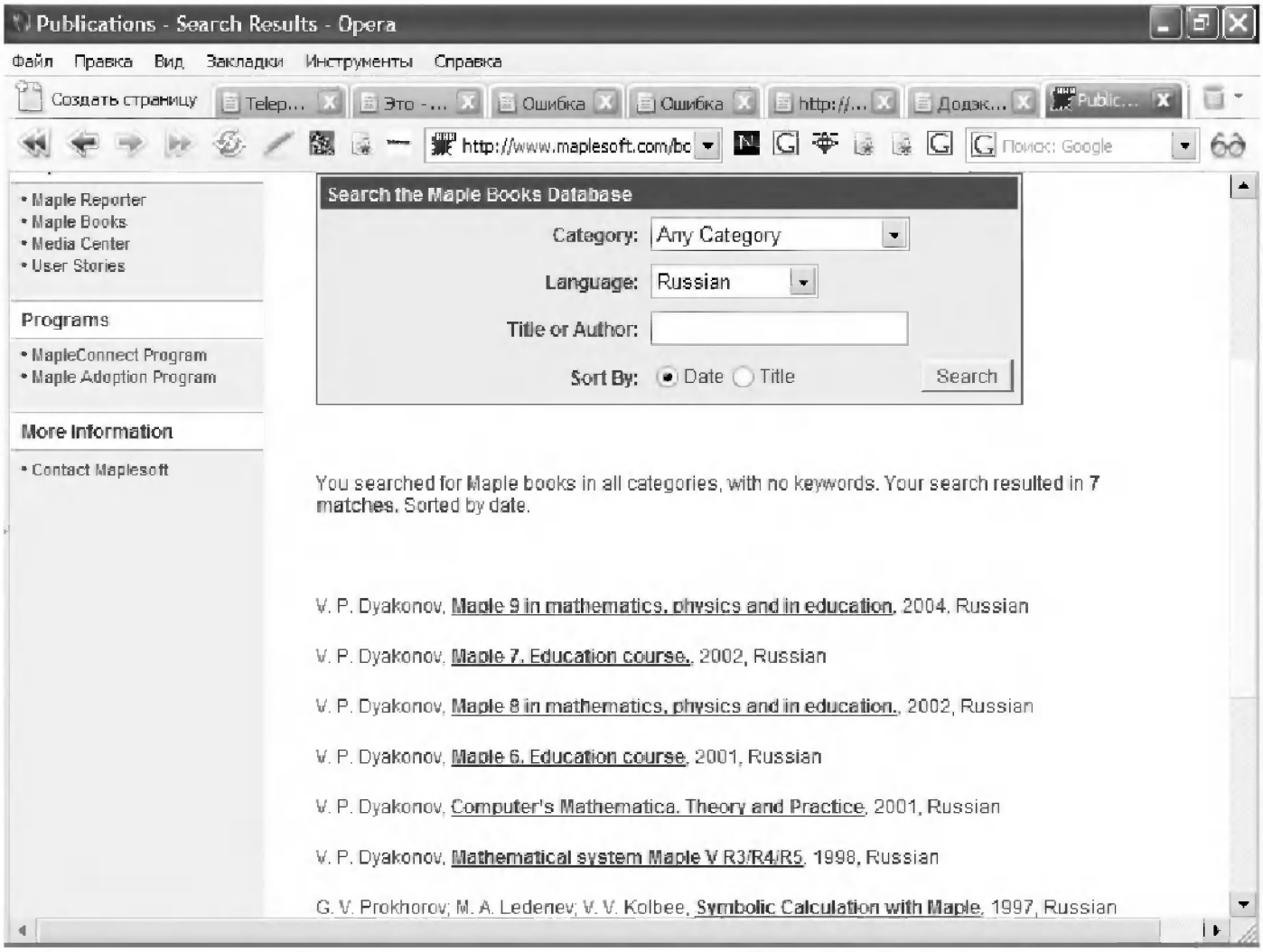


Рис. 0.1. Русскоязычные книги автора по системе Maple на сайте ее разработчика

логии». Многие, заметно обновленные и дополненные материалы книги [68] вошли в данную энциклопедию.

Системы Maple уже много лет лидируют в области символьных вычислений и имеют единственного очень серьезного конкурента в лице СКМ Mathematica [78–85], изначально созданной также как СКА. Широкое распространение у нас и во всем мире получили версии Mathematica 4/5 этой системы. Новейшая версия данной системы Mathematica 6 [170] представлена на сайте разработчика системы– корпорации Wolfram Research, Inc. (www.wolfram.com). Она вышла в мае 2007 г. Ныне это явный лидер среди универсальных СКМ.

Корпорация Wolfram Research, Inc. была создана ее создателем и нынешним руководителем Степфаном Вольфрамом. Им написана огромная книга [85], вошедшая в справочную базу данных по системе Mathematica. В 2000 г. автор данной книги прошел стажировку в корпорации Wolfram Research, Inc. (рис. 0.2) и детально познакомился с разработками системы Mathematica. Ее версиям 5.1/5.2 и особенно новейшей и поистине революционной версии Mathematica 6 в данной энциклопедии уделено должное место.

Корпорация Wolfram Research, Inc. уделяет огромное внимание развитию средств компьютерной математики, размещая документацию по ним в Интернете. Так,

Рис. 0.2. Автор данной книги Владимир Дьяконов (в центре) с главой фирмы Wolfram Research, Inc. Степфаном Вольфрамом (справа) и одним из ведущих менеджеров Рогером Гермундсоном (слева) на международной конференции по MathML (США)

под руководством нашего соотечественника, уехавшего в США, Олега Маричева создана интернет-страница по специальным математическим функциям, правилам их преобразований и визуализации. На этой странице размещено более ста тысяч формул и примеров их визуализации, связанных со специальными математическими функциями, и более десяти тысяч разделов по ним с рисунками. Эти данные весьма показательно свидетельствуют о мощи ядра символьных вычислений системы Mathematica. И их объем постоянно возрастает из месяца в месяц.

Другой масштабный проект World's of Science (Мир науки) создан одним из ведущих математиков компании Эриком Вэйштейном. В него вошли материалы его крупной энциклопедии Mathworld, существующей в виде огромной книги и в электронном виде. Размещение описаний алгоритмов математических вычислений для СКМ в Интернете снимает ограничения на объем документации по таким вычислениям и делает их общедоступными. Фундаментальные справочники по системе Mathematica с общим объемом свыше 5000 стр. написаны другим крупным математиком фирмы – Михаилом Троттом [182–185].

Широкую популярность, особенно у нас в России, получили СКМ класса Mathcad [36–62]. Вплоть до версии Mathcad 13 они создавались корпорацией MathSoft Engineering&Education и содержали ядро символьных операций системы Maple. Но недавно MathSoft Educations влилась в крупную корпорацию Parametric Technology Corporation (PTC), занятую разработкой и внедрением систем

автоматического проектирования. Новейшая Mathcad 14 выпускается уже от ее имени и использует ядро символьных операций от системы MuPAD (см. www.tpc.com).

Одной из первых СКА стала система Derive небольшой фирмы SoftWarehous, Inc. (США) [86–95]. Система отличалась поразительной компактностью благодаря ее реализации на языке искусственного интеллекта muLISP – версии известного языка LISP [96]. Благодаря этому на Derive «положила глаз» могущественная и огромная корпорация Texas Instruments (США), которая наряду с огромным числом выпускаемых измерительных приборов имеет свое микроэлектронное производство специализированных микросхем, обычных и сигнальных микропроцессоров и микрокалькуляторов. В итоге фирма SoftWarehous, Inc. влилась в Texas Instruments, и последние версии Derive анонсировались от имени последней. Результатом этого слияния стали первые в мире уникальные научные графические микрокалькуляторы фирмы Texas Instruments с встроенной СКА на основе системы Derive [16]. Так впервые СКА попали в hardware миниатюрных вычислительных устройств – микрокалькуляторов [1, 15].

Нынешние версии Derive 5/6 для персональных компьютеров – это довольно мощные СКА со своим ядром символьных операций и довольно эффектными средствами графической визуализации вычислений. На интернет-портале Google по Derive имеется 22,7 миллиона ссылок. Но Derive по-прежнему ориентирована на сферу образования и по вычислительной мощности не сопоставима с такими системами, как Maple, Mathematica и Mathcad.

Достойно с Derive конкурируют системы MuPAD, созданные в Германии (ныне на рынке их представляет компания Sci Face Software) [1, 97]. Для этих «легкоатлетов» характерны сравнительно невысокие требования к аппаратным ресурсам ПК и направленность на применение в образовании. Как уже отмечалось, ядро системы MuPAD используется в последней версии системы Mathcad 14, и никаких принципиальных ухудшений в возможностях последней не замечено. Последняя версия MuPAD 4.0.6 имеет установочный файл более 100 Мбайт и может использоваться на ряде компьютерных платформ. Ее можно скачать с интернет-сайта www.sciface.com разработчика системы.

В реализации численных методов вычислений и математического моделирования бесспорным лидером стали матричные системы класса MATLAB+Simulink [98–118], созданные крупной корпорацией The MathWorks. (www.mathworks.com). Это подлинные мастодонты среди СКМ – последняя версия MATLAB R2008a со всеми пакетами расширения занимает на жестком диске ПК объем памяти около 4 Гбайт. С MATLAB поставляется до сотни пакетов расширения по наиболее важным областям науки и техники. Многие пакеты созданы независимыми фирмами. Документация этой системы насчитывает многие десятки тысяч страниц.

По СКМ MATLAB подготовлено множество книг, например [98–118]. На интернет-сайте этой системы сообщается о выпуске по ней более 1000 книг на разных языках, в том числе и на русском. Среди них свыше десятка книг автора и его коллег. Объем и направленность данной книги не позволяют детально описать СКМ MATLAB. Отмечены лишь ее возможности в области символьных вычисле-

ний, реализованные пакетом Symbolic Math Toolbox, построенным на основе ядра системы Maple. По той же причине не описаны и другие программы, относящиеся к статистическим программам и электронным таблицам [119–121].

Выбор Maple в данной книге в качестве основы для описания возможностей компьютерной алгебры диктуется прежде всего тем, что именно ядро символьной (аналитической) математики системы Maple включено в целый ряд других массовых СКМ, например Mathcad 11/12/13 и MATLAB (с пакетом расширения Symbolic Math Toolbox). Недавно корпорация MapleSoft в союзе с MathWorks создала новый инструмент Maple Toolbox for MATLAB (рис. 0.3), позволяющий объединить мощь компьютерной алгебры системы Maple с уникальными возможностями мощной матричной системы MATLAB в реализации численных методов и имитационного математического моделирования с помощью пакета расширения Simulink.

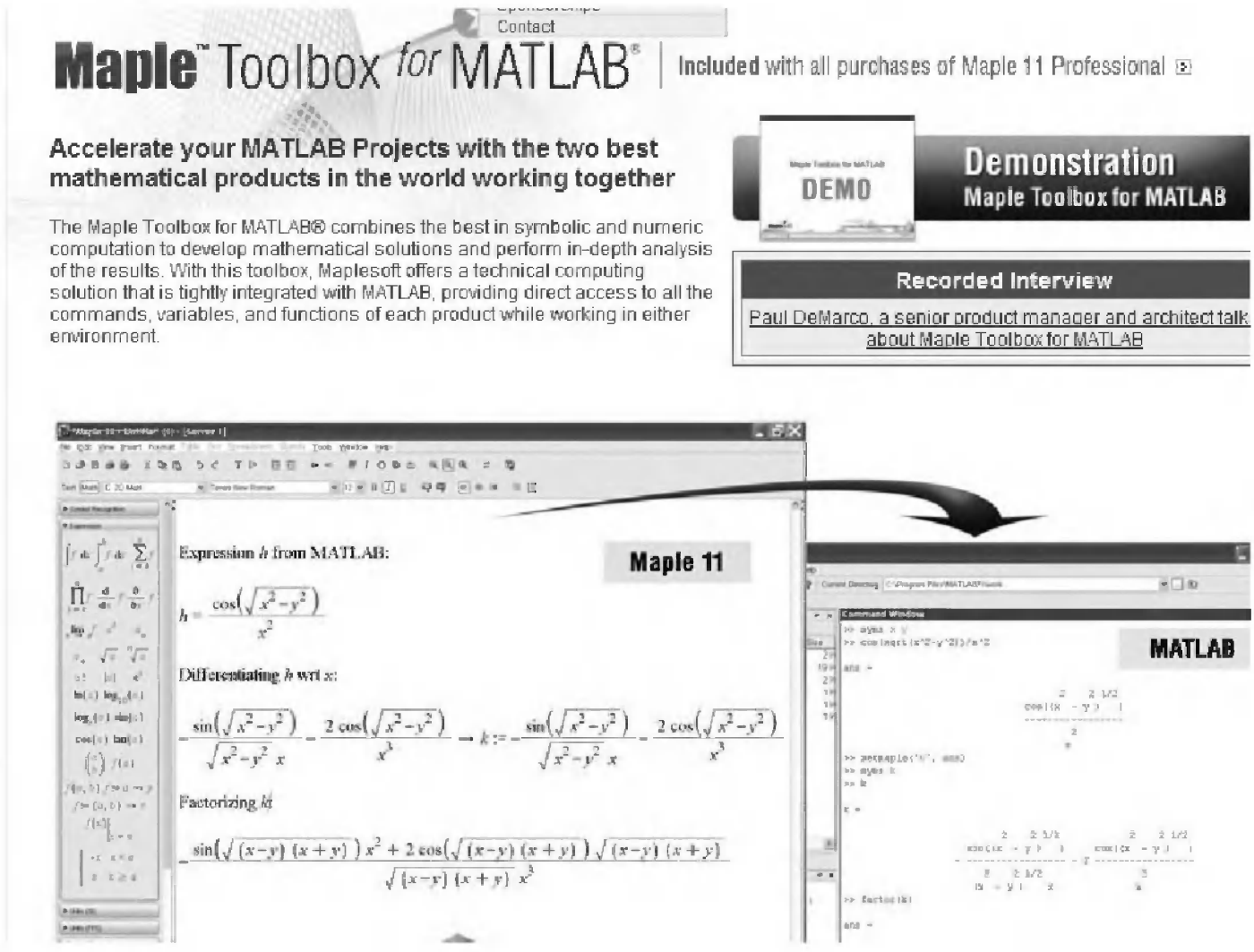


Рис.0.3. Интернет-страница, посвященная пакету Maple Toolbox for MATLAB

Итак, цели этой книги следующие:

- показать возможности нового научного направления – компьютерной математики с ориентацией на аналитические вычисления и их численную и графическую визуализацию;

- дать читателю книгу со сравнительным описанием сразу всех популярных СКМ (Maple, Mathematica Mathcad, Derive и MuPAD) с упором на описание их новейших версий;
- отразить интеграцию различных СКМ и СКА друг с другом;
- детально отразить области применения систем компьютерной математики в области символьных (аналитических) вычислений и их численной и графической визуализации;
- обеспечить читателю оптимальный выбор СКМ для выполнения его вычислений и расчетов.

Эта книга ориентирована на пользователей СКА и СКМ, а не на их разработчиков, которых у нас в России, увы, просто нет. Хотя и им она будет полезна. Основными читателями этой книги будут, конечно, студенты и преподаватели вузов, аспиранты и научные работники и инженеры. Исходя из этого, автор постарался дать описание на достаточно понятном и ясном языке с использованием простых и наиболее полезных примеров, число которых в этой книге превышает тысячу.

Энциклопедии обычно пишутся по известным материалам, прошедшим должную апробацию. Поэтому и эта книга обобщает материал многих ранее изданных книг автора и его коллег по системам компьютерной математики и содержит многие описанные в них примеры. В ней есть материалы по 8 научным конференциям по системам компьютерной математики и их приложениям, проведенным по инициативе автора в Смоленском государственном университете [179].

Автор намеренно не фокусирует внимания читателей на ряде недостатков, присущих СКМ, как и любым другим крупным программным продуктам. Отчасти это связано с тем, что такие недостатки устраняются в новых версиях СКМ. Хотя надо отметить, что иногда в новых версиях появляются новые недостатки и порою всплывают старые, когда-то уже устраненные. Однако эта книга посвящена тому, что могут делать современные СКМ м СКА, а не тому, на чем они спотыкаются. Пользоваться СКМ можно, только владея математикой, а достаточно подготовленный в ней пользователь сможет разобраться в редких ошибочных результатах и найти путь к получению верных решений. Полезно помнить нашу поговорку: не ошибается только тот, кто ничего не делает! Она относится не только к людям, но и к созданным ими СКМ. Другая поговорка «И на Солнце есть пятна!» тоже относится к СКМ.

Приобретая эту книгу, читатель получает возможность ознакомиться не только с достижениями в разработке систем компьютерной алгебры, но и с рядом ее новейших конкретных систем. Проще говоря, одна эта книга заменяет сразу несколько серьезных книг по системам компьютерной математики. При этом данная книга вовсе не является раздельным описанием СКМ, просто собранным в одном труде. Применение СКМ в ней описано по основным разделам математики, а не по классам СКМ, что позволяет оценивать возможности той или иной СКМ в том или ином конкретном разделе математики. Описание примеров в различных СКМ и их средств является взаимодополняющим. Детальное описание интерфейса СКМ в основные задачи книги не входит – предполагается, что в наше время читатель уже знает особенности интерфейса СКМ и может разобраться с ними

самостоятельно. Тем более что у всех СКМ интерфейс сделан интуитивно понятным и ориентированным на начинающего пользователя.

Особое внимание в книге уделено наглядности материала. Почти перед каждым разделом имеется теоретическая преамбула, поясняющая смысл основных математических понятий раздела. Особое внимание в книге уделено графической визуализации математических понятий. Впервые подробно описаны Maplet-средства такой визуализации, появившиеся в новейших версий системы Maple 9.5/10/11, и новейшие динамические объекты системы Mathematica 6.

По возможности энциклопедии придан практический характер. В книге реализована концепция обучения через примеры. Энциклопедия содержит свыше тысячи конкретных практических примеров по решению математических и научно-технических задач в наиболее популярных СКМ и СКА. Читатель может легко воспользоваться представленными примерами для решения своих задач.

Предостережения

Автор хотел бы предостеречь отдельных читателей от непонимания сути данной книги. Эта книга является энциклопедическим изданием по весьма обширному кругу *применений* СКА и СКМ, интересующих широкий круг читателей – пользователей таких систем. В соответствии с таким характером книга содержит большое число разделов, в совокупности дающих довольно полное (но не исчерпывающее) представление о возможностях компьютерной алгебры и ее наиболее известных программных продуктах. Большинство разделов книги носят вполне самостоятельный характер, и их можно применять для решения относящихся к ним задач.

В то же время книга не является учебником ни по обычной, ни по компьютерной математике. Средства последней описаны без приведения теорем и доказательств и порою даже без описания алгоритмов вычислений некоторых средств. Многие из них настолько сложны, что для их описания объема данной книги просто недостаточно. Кроме того, многие алгоритмы являются фирменными секретами разработчиков СКМ и широко не афишируются. Поэтому нередко даются лишь отдельные, достаточно краткие сведения о них. На роль полного справочника по каждой из описанных здесь СКА и СКМ книга также не претендует – достаточно полный справочник по каждой из таких систем имеет объем такой же, как у самой книги (а то и больший). Тем не менее в этой книге помещены достаточно полные сведения о каждой из описанных СКА и СКМ, вполне приемлемые для освоения работы с ними. В списке литературы указаны лишь отдельные книги по математике [122–148].

Мощным средством пропаганды СКА и СКМ становится Интернет. Но, к сожалению, некоторые российские интернет-форумы по системам компьютерной математики (например, форум по системе Maple на сайте Exponenta.Ru) ведутся на низком уровне и часто отражают мнение узкой (нередко малоквалифицированной, но с большими претензиями) категории анонимных участников таких форумов. Анонимность, безответственность и даже грубость отдельных участников таких форумов (нередко даже в адрес разработчиков СКМ) противоречат общепринятым нормам научной дискуссии и этики. При проверке (в том же Интернете и в письмах в адрес таких лиц) некоторые «специалисты» на таких форумах оказываются виртуальными «личностями», чья научная квалификация и состоятельность ничем не подкрепляются. Автор предостерегает серьезных читателей от участия в таких «форумах», дающих предвзятую и неверную информацию о СКМ.

При подборе примеров автор руководствовался основным принципом – примеры должны быть простыми, наглядными и поучительными. Ввиду огромного числа примеров отследить за авторством каждого из них практически невозможно, да это и не принято в энциклопедических и справочных книгах. Многие примеры в данной книге являются оригинальными, но это особо не выделяется. Список литературы содержит лишь наиболее важные источники. К тому же в этот список включены только те книги, которые, по мнению автора, можно рекомендовать научной общественности и студентам вузов и университетов.

Благодарности и адреса

Данная книга, как и ряд последних книг автора, подготовлена в инициативном порядке в Смоленском государственном университете. Частично работа автора была поддержана (скорее морально, чем материально) грантом Минобразования РФ в области фундаментальных наук, грантами Соросовского профессора, полученными автором книги в 1999 и 2001 гг., и грантом Фонда развития отечественного образования.

Особую благодарность автор выражает представителю корпорации Waterloo Maple г-ну Ph. D. Jason Schattman за любезно представленные программные продукты и документацию по СКМ Maple. Автор благодарит корпорацию Wolfram Research, Inc., ее создателя и главного разработчика систем класса Mathematica Степфана Вольфрама (S. Wolfram) и одного из ведущих сотрудников фирмы Wolfram Олега Маричева за интерес, проявленный к его работам, и прекрасные условия, созданные автору во время его стажировки в этой компании.

Автор благодарит Президента Фонда развития отечественного образования, академика РАО М. Н. Берулава за высокую оценку книги [68] на конкурсе «Лучшая научная книга 2006», а также своих коллег – декана физико-математического факультета Смоленского государственного университета профссора К. М. Расулова, профессоров В. В. Круглова, И. В. Абраменкову, В. Ф. Очкова, доцентов Р. Е. Кристалинского, Е. В. Петрову, А. А. Пенькова и др., а также А. Г. Лучаинова за интерес к тематике книги и обсуждение отдельных ее материалов.

Автор положительно относится к критическим и позитивным откликам и пожеланиям по данной книге, но не приемлет анонимного обсуждения. Отзывы по книге можно отправлять по месту работы автора: 214000, Смоленск, ул. Пржевальского 14, СмолГУ, – или по адресу его электронной почты: vpdyak@keytown.com. Вы можете также направлять свои отзывы издательству, выпустившему данную книгу.

Глава 1

Введение в компьютерную математику

Эта глава является введением в компьютерную математику. В ней описаны основы работы с программными и аппаратными средствами компьютерной математики. Основной материал ее относится к новейшим системам компьютерной алгебры (СКА) Maple 10/11. Приведены также данные о последних версиях СКА Derive, Mu-PAD и Mathematica, о системах компьютерной математики (СКМ) Mathcad и MATLAB. Описаны современные микрокалькуляторы с встроенными СКМ. В конце главы рассматривается применение средств компьютерной математики на аппаратном и программном уровнях в современных микрокалькуляторах и измерительных приборах и системах.

1.1. Краткая характеристика систем компьютерной алгебры

1.1.1. Понятие о символьных (аналитических) вычислениях

Системы компьютерной математики (СКМ) условно делятся на две категории – системы компьютерной алгебры (или системы символьной математики) СКА и системы для численных вычислений. Сразу отметим, что все современные системы являются универсальными, и их отношение к СКА или СКМ является довольно условным и определяется просто тем, к какой из систем они относились изначально. Разумеется, назначение систем зависит от истории их зарождения.

Символьные операции – это то, что кардинально отличает системы компьютерной алгебры (СКА) от систем для выполнения численных расчетов. При символьных операциях, называемых также *аналитическими*, задания на вычисление задаются в виде символьных (формульных) выражений, и результаты вычислений также получаются в символьном виде. Численные результаты при этом являются частными случаями результатов символьных вычислений.

К примеру, попытка вычислить в общем виде выражение $\sin(x)^2 + \cos(x)^2 = 1$ с неопределенной (не имеющей конкретного численного значения) переменной x с помощью численных математических систем или программ на обычных языках программирования к успеху не приведет. Вместо ожидаемого результата появится сообщение об ошибке вида: «Переменная x не определена!»

СКА не только не боятся применения *неопределенных переменных*, но и предпочитают работать с ними. Зададим, к примеру, в системе Maple (любой версии) квадратное уравнение, присвоив его выражение переменной *eq*:

```
> eq:=a*x^2+b*x+c=0;
```

$$eq := ax^2 + bx + c = 0$$

Проверим статус переменной x:

```
> x;
```

$$x$$

Переменная просто повторена в выводе, что и указывает на то, что она неопределенная, то есть ей не присвоено какое-либо конкретное значение. Теперь решим уравнение, используя функцию `solve`:

```
> solve(eq,x);
```

$$-\frac{b-\sqrt{b^2-4ac}}{2a},\ -\frac{b+\sqrt{b^2-4ac}}{2a}$$

Получено хорошо известное решение для квадратного уравнения. А теперь попробуем найти аналитическое решение относительно других переменных a, b и c:

```
> solve(eq,a);
```

$$-\frac{bx+c}{x^2}$$

```
> solve(eq,b);
```

$$-\frac{ax^2+c}{x}$$

```
> solve(eq,c);
```

$$-ax^2-bx$$

Решение прошло успешно – во всех случаях получены аналитические выражения для решения. Они более тривиальные, чем решение *eq* относительно x.

Не следует считать решения в аналитическом виде ограничением СКА. Все рассмотренные в данной книге СКА легко решают подавляющее большинство задач и в численном виде. Например, определив переменные, присвоив им некоторые значения

```
> a:=2:b:=3:c:=4:
```

получим решение в численном виде:

```
> solve(eq,x);
```

$$-\frac{3}{4}+\frac{1}{4}I\sqrt{23},\ -\frac{3}{4}-\frac{1}{4}I\sqrt{23}$$

Решение получено в виде комплексно-сопряженных чисел, в них I – это мнимая единица, то есть $\sqrt{-1}$.

1.1.2. Задачи, решаемые компьютерной алгеброй

Круг задач, решаемых компьютерной алгеброй и СКА, исключительно широк и постоянно расширяется. Тем не менее можно отметить некоторые наиболее характерные задачи, перечисленные ниже по разделам вычислительной математики.

Символьные и численные вычисления

- выполнение операций точной арифметики;
- вычисление фундаментальных констант с произвольной точностью;
- упрощение математических выражений;

- выполнение подстановок одних выражений в другие;
- расширение математических выражений;
- выделение общих множителей и делителей;
- комплектование выражений по степеням;
- операции со степенными многочленами;
- вычисление сумм и произведений рядов;
- дифференцирование выражений;
- интегрирование выражений;
- вычисление пределов функций;
- разложение функций в ряды Тейлора и Маклорена;
- поиск экстремумов функций и их асимптот;
- анализ функций на непрерывность (не у всех систем);
- интегральные преобразования Лапласа, Фурье и др.;
- дискретные Z-преобразования;
- прямое и обратное быстрое преобразование Фурье;
- работа с кусочно-нелинейными функциями;
- интерполяция, экстраполяция и аппроксимация различных функциональных зависимостей;
- решение специальных задач в теории графов, нейронных сетей и т. д.

Вычисление элементарных и специальных математических функций

- вычисление всех элементарных функций и осуществление их символьных преобразований;
- вычисление большинства специальных математических функций;
- пересчет координат точек для множества координатных систем.

Численное и символьное решение уравнений

- решение линейных и нелинейных систем алгебраических и тригонометрических уравнений;
- символьное вычисление рядов;
- работа с рекуррентными и рекурсивными функциями;
- решение трансцендентных уравнений;
- решение систем с неравенствами;
- решение систем обыкновенных дифференциальных уравнений;
- решение систем дифференциальных уравнений с заданными граничными условиями;
- решение систем дифференциальных уравнений с частными производными;
- решение задач нелинейного программирования.

Линейная алгебра

- задание векторов и матриц символьного и числового вида;
- множество операций с векторами и матрицами;
- решение систем линейных уравнений с действительными и комплексными коэффициентами;
- формирование специальных матриц и различные их преобразования;

- вычисление собственных значений и собственных векторов матриц;
- решение задач линейного программирования и оптимизации.

Графическая визуализация вычислений

- построение графиков многих функций;
- задание различных типов осей (с линейным и логарифмическим масштабом);
- графики функций в декартовой и полярной системах координат;
- специальные виды графиков (точки массивов, векторные, диаграммы уровней и др.);
- системы координат, определяемые пользователем;
- специальные графики, представляющие решения дифференциальных уравнений;
- графики фазовых траекторий и портретов;
- графики трехмерных 3D-поверхностей с функциональной закраской;
- контурные, векторные и прочие специальные графики;
- построение пересекающихся в пространстве объектов;
- задание пользователем окраски графиков;
- импорт графиков из других пакетов и программных систем;
- построение графиков, иллюстрирующих основные положения математики;
- задание анимационных графиков;
- проигрывание анимационных файлов с помощью специального проигрывателя;
- реализация всех средств современной машинной графики [155].

Возможности расширения

- задание функций пользователя со списком локальных переменных;
- задание операторов пользователя (не у всех систем);
- задание новых математических соотношений (не у всех систем);
- задание внешних функций и процедур с хранением их на диске;
- применение подключаемых библиотек операторов и функций;
- применение пакетов расширения, специализированных на определенные классы вычислений;
- интеграция с другими математическими и иными системами.

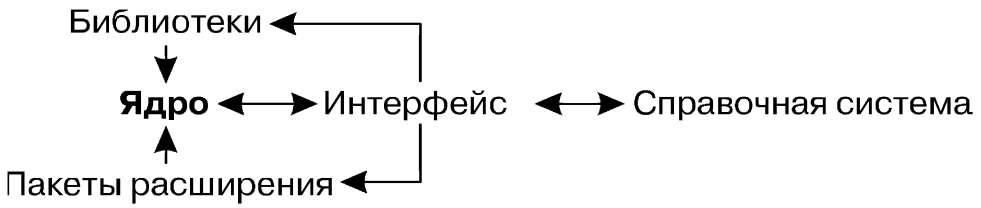
Программирование:

- встроенный язык программирования;
- простой синтаксис языка программирования;
- обширный набор математических типов данных;
- типы данных, задаваемых пользователем;
- средства работы с файлами и потоками информации;
- средства создания пользовательского интерфейса программ (не у всех систем);
- средства отладки программ;
- программный интерфейс с другими языками программирования C, Fortran и LaTeX (не у всех систем).

Некоторые системы могут иметь дополнительные возможности, например решение специальных систем дифференциальных уравнений с частными производными (Mathematica, Maple), решение задач линейного и нелинейного программирования, расширенные возможности аппроксимации данных и функций многими методами и т. д.

1.1.3. Структура и назначение систем компьютерной алгебры

Каждая система компьютерной математики может иметь нюансы в своей архитектуре или структуре. Тем не менее можно прийти к выводу, что современные универсальные системы компьютерной математики имеют следующую типовую структуру:



Ядро системы состоит из множества *заранее откомпилированных* функций и процедур, представленных в машинных кодах и обеспечивающих достаточно представительный набор встроенных функций и операторов системы. Оно содержит также множество правил преобразований математических выражений, команд, констант и их определений в символьном виде.

Ядро математических систем тщательно оптимизируется для ускорения вычислений. Этому способствует и компиляция функций и команд ядра. Доступ в ядро пользователя для его модификации, как правило, исключен. Объем ядра достигает нескольких мегабайт. Пишется ядро на языке реализации системы – чаще всего это C или C++ (лишь в системе Derive использован язык искусственного интеллекта MuLISP) и компилируется на фирме – разработчике системы. Поставка ядра в исходных кодах (на языке реализации) не практикуется ввиду огромного объема исходников. Нередко улучшенные алгоритмы вычислений ядра являются ноу-хау разработчиков и относятся к разряду тщательно скрываемых данных. Пожалуй, это один из главных недостатков современных коммерческих СКМ.

Интерфейс – это совокупность аппаратных и программных средств для работы ПК с внешним оборудованием и пользователем. Далее речь идет об интерфейсе пользователя (User Interface). Он дает пользователю возможность обращаться к ядру со своими запросами и получать результат решения на экране дисплея. Интерфейс современных систем символьной математики базируется на общеизвестных средствах операционных систем класса Windows (Windows XP или Windows Vista в данной книге) и имеет все их «прелести»: перемещаемые и масштаби-

руемые окна документов, диалоговые и информационные окна, кнопки управления, общение с периферийными устройствами и т. д. Интерфейс пользователя СКА в настоящее время почти исключительно является графическим и сокращенно именуется *GUI* (Graphics User Interface).

Функции и процедуры (в понятии языков программирования), включенные в откомпилированное ядро, выполняются предельно быстро. С этой точки зрения в ядро было бы выгодно включать как можно больше вычислительных средств. Однако это невольно приводит к замедлению поиска нужных средств из-за возрастания их числа. Поэтому объем ядра ограничивают, но к нему добавляют *библиотеки* более редких процедур и функций, к которым обращается пользователь, если в ядре не обнаружена нужная процедура или функция.

Кардинальное расширение возможностей систем и их адаптация к решаемым конкретным пользователем задачам достигается за счет *пакетов расширения* систем (packages). Эти пакеты, как правило, пишутся на собственном языке программирования той или иной системы, что делает возможным их подготовку как разработчиками СКМ, так и обычными пользователями.

Справочная система, или *справка*, обеспечивает получение оперативных справок по любым вопросам работы с системами компьютерной математики с примерами такой работы. Она содержит и многочисленный справочный материал – математические и физические таблицы, формулы для нахождения производных и интегралов, алгебраические преобразования и т. д. В справку включены многие тысячи примеров применения той или иной СКМ.

Ядро, библиотеки, пакеты расширения и справочная система современных СКА и СКМ аккумулируют знания в области математики, накопленные за тысячелетия ее развития. К сожалению, за редкими исключениями, справки написаны на английском языке, что затрудняет их применение русскоязычными пользователями. Объем справки очень велик – эквивалентен многим тысячам страниц книг. Система поиска с помощью гиперссылок удобна при работе на компьютере, но не всегда.

1.2. Системы компьютерной математики Maple

1.2.1. Назначение и место систем Maple

СКМ класса Maple были созданы корпорацией Waterloo Maple, Inc. (Канада) как СКА с расширенными возможностями в области символьных (аналитических) вычислений. Ядро и пакеты расширения Maple V R6 содержали до 3000 встроенных функций, операторов и правил символьных преобразований. Лишь система Mathematica способна, как и Maple, претендовать на роль лидера среди СКА.

Версия Maple 9.5 появилась на рынке в 2004 г. Уже весной 2005 г. появилась версия системы Maple 10, а в начале 2007 г. на рынок поступила версия Maple 11.

Эти версии позиционируются на рынке как универсальные СКМ, рассчитанные на широкий круг пользователей. Они, наряду с обширными средствами символьных вычислений, содержат средства для выполнения быстрых численных расчетов, лежащих в основе математического моделирования различных явлений окружающего нас мира, систем и устройств самого различного назначения. Основное развитие систем пошло по пути предельного упрощения применения средств системы наиболее массовыми категориями пользователей – математиками, физиками и инженерами, студентами и преподавателями вузов и университетов.

Maple – типичная интегрированная программная система. Она объединяет в себе:

- мощный язык программирования (он же язык для интерактивного общения с системой);
- редактор для подготовки и редактирования документов и программ;
- современный многооконный пользовательский интерфейс с возможностью работы в диалоговом режиме;
- мощную справочную систему со многими тысячами примеров;
- ядро алгоритмов и правил преобразования математических выражений;
- численный и символьный процессоры;
- систему диагностики;
- библиотеки встроенных и дополнительных функций;
- пакеты функций сторонних производителей и поддержку некоторых других языков программирования и программ.

Ко всем этим средствам имеется полный доступ прямо из окна программы. Система Maple прошла долгий путь развития и апробации. Она реализована на больших ЭВМ, рабочих станциях Sun, ПК, работающих с операционной системой Unix, ПК класса IBM PC, Macintosh и др. Все это самым положительным образом повлияло на ее отработку и надежность (в смысле высокой вероятности правильности решений и отсутствия сбоев в работе).

1.2.2. Ядро и пакеты применения Maple

Ядро системы Maple улучшается от версии к версии [63–67]. В версиях Maple 10/11 в ядре исправлены многие недостатки, выявленные в ходе обширного и поистине всемирного тестирования предшествующих версий.

В Maple имеется также основная библиотека операторов, команд и функций. Многие встроенные в нее функции, как и функции ядра, могут использоваться без какого-либо объявления, другие нуждаются в объявлении. Кроме того, имеется ряд подключаемых проблемно-ориентированных пакетов (packages), тематика которых охватывает множество разделов классической и современной математики.

Дополнительные функции из пакетов могут применяться после объявления подключения пакета с помощью команды `with(name)`, где `name` – имя применяемого пакета. Общее число функций в системе Maple, с учетом встроенных в ядро и размещенных в пакетах, уже превысило 3000.

1.2.3. Языки систем компьютерной алгебры

Большинство СКА, включая Maple, интегрирует в себя три языка: входной, реализации и программирования.

Входной язык является *интерпретирующим* языком сверхвысокого уровня, ориентированным на решение математических задач практически любой сложности в интерактивном (диалоговом) режиме. Он служит для задания системе вопросов, или, говоря иначе, задания входных данных для последующей их обработки. Язык имеет большое число заранее определенных математических и графических внутренних (или встроенных) функций, а также обширную библиотеку дополнительных функций, подключаемую по мере необходимости.

В состав СКА входит также *язык программирования*. Так, Maple имеет свой язык процедурного программирования – Maple-язык. Этот язык содержит вполне традиционные средства структурирования программ. Он описан в главе 14 и включает в себя все команды и функции входного языка. Многие из них являются весьма серьезными программными модулями: например, символьного дифференцирования, интегрирования, разложения в ряд Тейлора, построения сложных трехмерных графиков и т. д.

Начиная с Maple 9/9.5 в систему добавлены средства Maplets (*маплеты*) для визуально-ориентированного диалога с системой, включающие в себя задание множества диалоговых окон и иных типовых средств интерфейса, привычного пользователям Windows-приложений. Однако даже обычные средства диалога у систем класса Maple обеспечивают высокую наглядность и комфортность работы с системой при решении математических задач.

Языком реализации системы Maple является один из самых лучших и мощных универсальных языков программирования – Си (или C). На нем написано ядро системы, содержащее тщательно оптимизированные процедуры. Большинство же функций, которые содержатся в библиотеках расширения системы Maple, написаны на Maple-языке, благодаря чему их можно модифицировать и даже писать свои собственные библиотеки. По разным оценкам, лишь от 5 до 10 % средств Maple создано на языке реализации – все остальное написано на Maple-языке.

Для подготовки программ на языке Maple могут использоваться внешние редакторы, но система имеет и свой встроенный редактор, вполне удовлетворяющий требованиям большинства пользователей. Он открывается командами **New** и **Open** в меню **File**. Этот редактор можно использовать для редактирования файлов программ или математических выражений.

Maple-язык программирования считается одним из самых лучших и мощных языков программирования математических задач. Это наряду с упомянутыми новыми средствами пакета Maplets позволяет создавать высококачественные электронные уроки, статьи и даже целые книги.

1.2.4. Новые возможности Maple 11

Предшествующие версии системы Maple, вплоть до Maple 9.5/10, детально описаны в книгах [63–68]. Поэтому остановимся только на новых возможностях новей-

шей реализации Maple 11. Это существенно переработанная версия системы Maple. Она реализует концепцию «умных» документов и интуитивно понятного интерфейса пользователя. Это значительно облегчает и ускоряет освоение системы. Десятки интерактивных учебников и справочных инструментов, а также встроенный обучающий материал по математике, инженерным дисциплинам и физике дают возможность продуктивно выполнять работу с минимальными усилиями.

В области пользовательского интерфейса Maple 11 имеет следующие новинки:

- самодокументируемые контекстные меню;
- настраиваемая панель Favorites;
- более 35 новых шаблонов задач для курсов Calculus (Численные методы) и Algebra (Алгебраические методы), работающих по принципу «наведи и щелкни»;
- вспомогательный модуль Backsolver позволяет быстро найти значение для любой переменной в формуле, исходя из значений других параметров;
- вспомогательный модуль Special Functions – быстрый доступ к более чем 200 специальным функциям;
- вспомогательный модуль Scientific Constant – доступ к БД более 20 000 физических констант и свойств химических элементов;
- руководство по сообщениям об ошибках (Error Message Guide).

Новинки в области создания документов:

- режим слайд-шоу превращает документ в наглядную презентацию;
- к тексту и математическим выкладкам можно добавлять аннотации;
- расширенные возможности форматирования чисел;
- междокументные ссылки – доступ к математическим выражениям из другого документа;
- расширенная панель рисования и панель избранных инструментов;
- все инструменты рисования теперь можно напрямую использовать на графиках и изображениях;
- улучшенный механизм работы с блоками документа;
- специальные маркеры для дополнительных компонентов.

Существенно улучшены возможности графической визуализации:

- рендеринг 2D-графиков выполняется быстрее и с меньшим потреблением памяти;
- расширены возможности аннотации 2D-графиков, улучшены их вид и сопровождение;
- улучшены контекстные меню для параметров построения графиков;
- существенно расширен числовой диапазон для двумерных графиков;
- улучшена поддержка формата WMF;
- введены новые команды для построения трехмерного пересечения поверхностей, а также анимация с трассировкой;
- усовершенствованы команды для построения двумерных неявных кривых и диаграмм плотности;
- модернизирован механизм цветового выделения для анимированных элементов и выбранных областей;

- отображение массивов графиков в виде таблицы обеспечивает более глубокий контроль над каждым графиком в отдельности;
- улучшены инструменты Pan и Scale, теперь при отображении текста в графиках учитывается коэффициент масштабирования.

У Maple 11 улучшены возможности интеграции с другими СКМ, в частности с табличными процессорами Microsoft Excel и языками программирования C и Fortran. Обеспечена поддержка компилятора Intel Fortran на компьютерах с операционной системой Windows. При работе с матричной системой MATLAB обеспечен более полный набор преобразований данных, в том числе строк и структур, обеспечена работа с растровыми изображениями (пакет Image Tools), введено более 20 новых команд, расширена поддержка предварительного просмотра изображений. Maple с MATLAB стал работать быстрее, используется меньше памяти. Разработан пакет расширения Maple Toolbox for MATLAB для совместной работы Maple и MATLAB с числовыми и символьными данными, а также пакет расширения BlockBuilder for Simulink для осуществления блочного моделирования в среде Simulink. Но в стандартную поставку Maple он не входит.

Заметно улучшены средства программирования:

- введен пакет Threads (многопоточное исполнение);
- введено альтернативное написание параметров ключевых слов и явно объявляемые зависимые типы;
- для сопоставления цепочки из нуля или более аргументов одного типа можно задать всего один параметр;
- появились дополнительные необязательные параметры, которые порождают исключение, если следующий элемент в последовательности аргументов не отвечает требованиям объявленного типа;
- введен более удобный механизм извлечения и установки значений;
- введена дополнительная опция команды GetProperty, позволяющая извлекать содержимое компонента Math Expression и возвращать вычисленное выражение Maple;
- введен новый пакет ListTools для работы со списками.

Система Maple 11 предлагает широчайший набор новых и улучшенных математических инструментов как для серьезных ученых, так и для студентов технических вузов:

- пакет Graph Theory содержит всеобъемлющий набор инструментов для работы с графами;
- пакет Physics (Физика) для вычислений по теоретической физике предлагает все необходимые инструменты, от простых приложений классической механики до вычислений по теории квантовых полей;
- новый пакет Differential Geometry – набор тесно интегрированных инструментов для самых разных расчетов в области дифференциальной геометрии;
- полный учебный курс по дифференциальной геометрии (Differential Geometry) и ее приложениям;
- улучшения в алгоритмах решения ДУ, в частности в части эллиптических решений, новый набор алгоритмов для поиска точных решений для ДУ

в частных производных, новый интерактивный построитель графиков решений ДУ;
- улучшенный алгоритм решения задач с дифференциально-алгебраическими уравнениями высших индексов с поддержкой крупных систем, которые обычно возникают при инженерном моделировании;
- лучший в мире сертифицированный механизм поиска действительных корней многочлена гарантирует нахождение действительных корней в системах многочленов;
- благодаря включению знаменитой библиотеки FGb, где реализован самый эффективный механизм вычислений в базисе Грёбнера (Groebner bases), этот решатель стал мировым лидером по скорости работы;
- улучшения в быстродействии всех компонентов системы, которые автоматически будут пользоваться всеми преимуществами новых аппаратных компонентов для вычислений с плавающей запятой;
- множество улучшений и дополнений в различных алгоритмах, включая численное интегрирование и суммирование, векторное исчисление, линейную алгебру, теорию чисел, идеалы на алгебре многочленов, уравнения с линейной рекурсией и др.

Стоит особо отметить новые средства решения дифференциальных уравнений:
- эллиптические решения для нелинейных ОДУ первого и второго порядков;
- новый набор алгоритмов для поиска точных решений для ЧДУ;
- 20 новых команд, основанных на свойствах симметрии, в том числе несколько оригинальных алгоритмов;
- интерактивный модуль построения графика решения ДУ;
- улучшения в команде DEPlot: возможность анимации графика как по кривым решений, так и по направлениям поля; новые типы стрелок для обозначения направлений поля;
- цветовое оформление графика в соответствии с магнитудой поля.

В символьный процессор Maple 11 введены:
- новые команды для линейных интегральных уравнений;
- расширенные возможности интеграции: неопределенные интегралы в области существования специальных функций; обработка подынтегральных выражений, содержащих двоякопериодические функции;
- средства вычисления производных четырех функций Вейерштрасса при нулевом дискриминанте;
- возможности получения полных решений линейных и нелинейных неравенств с параметрами в форме интервальных выражений;
- обработка неизвестных (искомых) функций и преобразование выражений в формальные степенные ряды вблизи некоторой точки.

Дополнительная поддержка численных методов обеспечена за счет:
- новых функций в процедурах, позволяющих задействовать расширенные аппаратные средства вычислений с плавающей запятой;
- улучшения алгоритмов численного интегрирования и суммирования;

- проведения итераций по нулям функции в положительном вещественном направлении и для изоляции вещественных корней вещественных одномерных многочленов и систем многочленов.

Отметим и средства повышения скорости вычислений:

- в состав системы входит лучший в мире сертифицированный механизм поиска действительных корней многочлена, который гарантирует нахождение действительных корней в системах многочленов;
- стандартный численный решатель для жестких ОДУ и стандартные численные решатели для жестких/нежестких ДАУ эффективнее работают на больших задачах;
- команда Multiply теперь использует алгоритм Карацубы для многочленов достаточно высокой степени;
- конструкции Matrix и Vector и сокращенные записи конкатенации стали работать намного эффективнее;
- изменения в механизме обработки аргументов увеличили производительность и снизили потребность в свободной памяти почти для всех вызовов процедур Maple;
- более эффективное представление данных и алгоритмы вычислений на многочленах с коэффициентами – алгебраическими числами позволяет вычислять НОД для таких многочленов намного быстрее;
- есть версия для исполнения на 64-битных платформах Windows.

Такое обилие новшеств говорит о том, что Maple 11 – действительно новая и серьезно переработанная версия этого популярного программного продукта в области компьютерной математики. Впрочем, уже появилась очередная версия Maple 12.

1.2.5. Запуск Maple 11 и интерфейс пользователя

Установка Maple на ПК пользователям этой системы хорошо известна и в особых комментариях не нуждается. После инсталляции Maple в папке Program Files образуется новая папка с именем установленной версии Maple. В ней можно обнаружить несколько папок и ярлыков, которые используются для запуска компонентов системы.

В версиях Maple 9.5/10 наряду с обновленным стандартным интерфейсом было возможно применение классического интерфейса, характерного для первых версий системы. Полная идентичность в средствах стандартного и классического интерфейсов, к сожалению, отсутствовала. Иногда отдельные задачи, созданные средствами одного интерфейса, не работали при использовании другого интерфейса. Это оговорено в документации по системам Maple 9.5 и 10, и делать из этого трагедию, как в некоторых репликах на интернет-форумах по системе Maple, вряд ли стоит. Тем более что в Maple 11 классический интерфейс как отдельный программный модуль уже не используется.

Для запуска Maple 11 достаточно активировать ярлык системы на рабочем столе операционной системы Windows (для определенности примеры даны для Windows XP). Откроются основное окно Maple 11 и окно начальной подсказки **Startup**. Эти окна показаны на рис. 1.1. Сверху основного окна расположены титульная строка, меню и панель инструментов. Снизу имеется строка сообщений. В левой части окна расположены раскрывающиеся списки палитр математических символов, операторов, функций и команд.

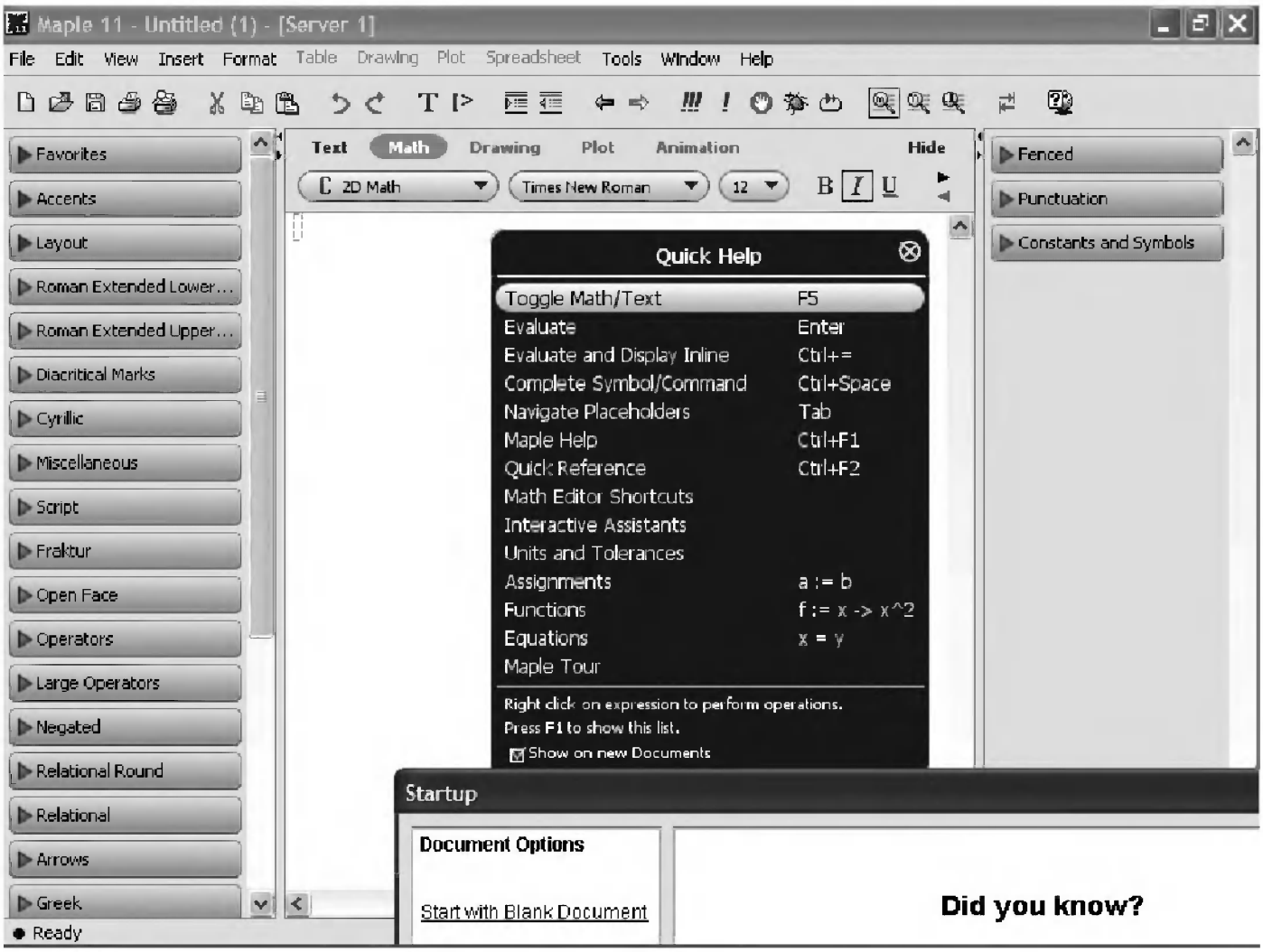


Рис. 1.1. Основное окно Maple 11 и окно начальной подсказки ***Startup***

Если в левой части окна не хватает места для палитр, то часть их переносится в правую часть окна. В результате область окна для документа заметно уменьшается. Над окном Maple 11 появилась новая панель **Toolbar Dock**, которая дополнительно уменьшает размеры окна документов. Первоначально в окне документов присутствуют окно быстрой справки **Quick Help** и большое окно **Startup** с многочисленными гиперссылками (опциями), позволяющими ознакомиться с системой.

Обилие деталей интерфейса пользователя свидетельствует о том, что разработчик системы продолжает развивать средства интуитивно понятного ввода документов с помощью готовых шаблонов. Однако стоит отметить, что интерфейс

Maple 11 имеет средства удаления практически любых палитр и панелей. В качестве примера на рис. 1.2 показан пример применения Maple 11 для проектирования цифрового фильтра с удалением правого набора палитр.

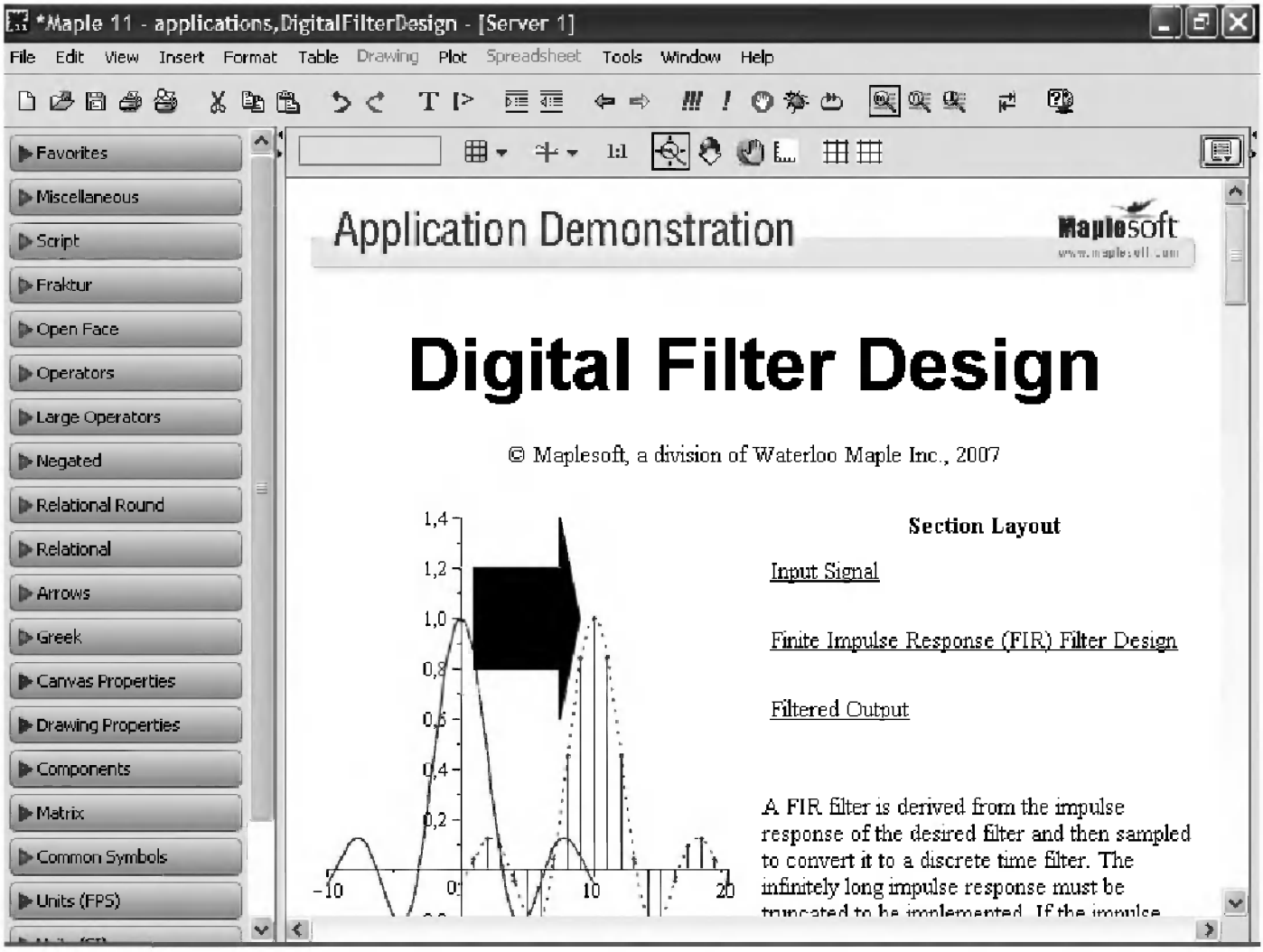


Рис. 1.2. Вид окна системы Maple 11 с примером проектирования цифрового фильтра и скрытым правым набором палитр

Поскольку интерфейс новых версий Maple интуитивно понятен, то с учетом характера данной книги его подробное описание лишено смысла. Его можно найти в книгах [63–68]. Отметим лишь, что средства интерфейса позволяют готовить документы, максимально приближенные по виду к страницам научных статей и книг.

Уже в Maple 10 был сделан очередной шаг к приближению документов к форме, принятой в литературе по математике. В режиме **Document Mode** этой системы не выводится знак **>**, отмечающий ячейки ввода и позволявший отличать их от текстовых ячеек и ячеек вывода. Курсор ввода в ячейках ввода имеет вид прямой черты |, если ячейка является текстовой и используется для задания комментария, и вид наклонной черты /, если это Math-ячейка для ввода математических выражений и функций. Разумеется, сохранен и режим **Worksheet Mode**, привычный пользователям Maple предшествующих версий. Выбор режи-

ма для новых документов возможен при задании нового документа командой **New** в позиции **File** меню.

Переключение между режимами ввода текстов **Text** и математических выражений **Math** осуществляется клавишей **F5**. При этом статус ячейки можно менять оперативно. Например, если ввести текст (везде с курсором |) , например 2+3, то при нажатии клавиши ввода ничего не произойдет – просто курсор ввода переместится на строку вниз. Но если набрать 2+3 при установке курсора ввода в виде / (режим **Math**), то получим результат 5.

Курсор ввода имеется только в последней ячейке ввода. Результаты вычислений в ячейках вывода автоматически нумеруются с простановкой номеров в круглых скобках (чуть ниже это описано более подробно).

Режим **Document Mode** обеспечивает высококачественную подготовку документов в стиле математических статей и книг. На его основе строятся электронные книги, например руководство по применению Maple. Однако по сравнению с привычным режимом **Worksheet Mode** в режиме **Document Mode** вычисления выполняются явно медленнее. В последующих главах книги почти исключительно используется режим **Worksheet Mode** как более быстрый и привычный пользователям всех версий системы Maple.

В Maple 10/11 широко используется разделение документов на блоки, например с помощью задания таблиц (**Tables**), в ячейках которых размещаются части документов. В сочетании с весьма эффектной графической визуализацией результатов вычислений и расширенной техникой использования в документах рисунков и графических набросков это позволяет создавать весьма наглядные и просто красивые документы, которые отличаются от страниц математических книг разве только возможностью представления в цвете и возможностями применения средств мультимедиа, например анимации графиков и использованием звуковых эффектов.

1.2.6. Помощники (ассистенты) и пакеты расширения Maple

Помощники (ассистенты) – еще одно новое средство, облегчающее работу с системой Maple 9.5 начинающих пользователей, особенно студентов вузов и учащихся других образовательных учреждений. В Maple 9.5 включено 7 ассистентов, список которых открывает команда **Assistents**.

- **Curve Fitting...** – помощник по подгонке кривых;
- **Library Browser...** – браузер для просмотра библиотек;
- **Matrix Builder...** – помощник по заданию матриц заданного размера;
- **ODE Analyser...** – анализатор систем обыкновенных дифференциальных уравнений;
- **Optimization...** – помощник по решению задач оптимизации;
- **Plot Builder...** – создатель (построитель) графиков по заданным выражениям;
- **Unit Convertor ...** – преобразователь единиц измерений научных величин.

В ядро систем Maple включена только часть реализованных в ней функций. Это те функции, которые используются достаточно часто и должны выполняться в минимально возможное время. Множество функций, в том числе применяемых довольно редко и в вычислениях специальных видов, реализовано в проблемно-ориентированных *пакетах расширения*. Напоминаем, что информацию о пакетах расширения Maple можно получить, используя команду

```
> ?packages
```

Приведем назначение нескольких наиболее важных пакетов расширения Maple:

- algcurves – работа с алгебраическими кривыми;
- combinat – функции комбинаторики;
- CurveFutting – приближение кривых;
- DEtools – решение дифференциальных уравнений;
- diffalg – дифференциальная алгебра;
- DiscreteTransform – пакет по дискретным преобразованиям;
- FileTools – пакет для работы с файлами;
- LinearAlgebra – линейная алгебра;
- Matlab – интеграция с MATLAB;
- MathematicalFunction – математические функции;
- Maplets – пакет визуально-ориентированного задания элементов интерфейса (окна, кнопки, линейки прокрутки и т. д.);
- networks – графы;
- numapprox – численная аппроксимация;
- plots – расширения графики;
- simplex – линейная оптимизация (симплекс-метод);
- stats – статистика;
- student – функции в помощь студентам;
- Student[Calculus1] – пакет анализа кривых первого порядка со специальными средствами визуализации;
- VectorCalculus – пакет работы с векторами, содержащий средства векторного анализа и решения задач теории поля.

Как следует из просмотра этого далеко не полного списка, пакеты расширения охватывают многие крупные разделы математики и существенно дополняют возможности системы, предоставляемые средствами ее ядра. Пакеты расширения пишутся на Maple-языке программирования, поэтому они могут легко модернизироваться и пополняться в достаточно короткие сроки.

Уже в системе Maple 9.5 состав пакетов расширения существенно обновлен. Из новых математически ориентированных новых пакетов расширения следует отметить:

- Optimization – пакет реализации методов оптимизации;
- RootFinding – пакет поиска корней уравнений;
- Student(MultivariateCalculus) – пакет Student для многовариантных вычислений.

С помощью команды

```
> ?name_package;
```

можно получить информацию о любом пакете расширения и найти список входящих в него функций. Названия пакетов расширения можно найти в справке.

Для обращения к функциям того или иного пакета используется его полная загрузка командой

```
> with(package):[;]
```

Знак : блокирует вывод списка функций пакета, а знак ; задает вывод этого списка.

Если вам необходима какая-то одна функция пакета или небольшая их часть, то не стоит загружать пакет целиком. Это может привести к избыточным затратам памяти компьютера и даже нарушить нормальную работу некоторых функций – следует помнить, что нередко пакеты переопределяют некоторые функции ядра. Для загрузки избранных функций используется команда `with` в форме

```
> with(package, f1, f2, …):
```

или

```
> with(package, [f1, f2, …]):
```

При этом загружаются функции `f1`, `f2`, ... из пакета с именем `packages`.

1.3. Основы работы с Maple в диалоговом режиме

1.3.1. Начальные навыки работы

После загрузки и запуска системы можно начать диалог с ней, используя ее операторы и функции (с параметрами) для создания и вычисления математических выражений. Во избежание грубых ошибок при исполнении того или иного примера рекомендуется перед этим исполнить команду `restart`, которая снимает определения со всех использованных ранее переменных и позволяет начать вычисления «с чистого листа».

Диалог идет в стиле: «задал вопрос – получил ответ». Вопросы и ответы занимают отдельные блоки, выделяемые в левой части квадратными скобками. Длина квадратных скобок зависит от размера выражений – исходных (вопроса) и результатов вычислений (ответов на вопросы). Знак **>** является знаком приглашения к заданию вопроса. Мигающая вертикальная черта | – маркер ввода (курсор).

Ввод выражений (вопросов) задается по правилам, давно принятым для строчных редакторов. Они хорошо известны, и мы не будем останавливаться на них подробно. Отметим лишь, что клавиша **Ins** позволяет задавать два основных режима ввода – замены и вставки. В режиме замены вводимый символ заменяет существующий символ, который отмечен маркером ввода. А в режиме вставки новый символ вставляется в текст, не уничтожая имеющихся символов.

В Maple возможны две формы задания ввода. Ниже они приведены для случая ввода двойного интеграла с помощью палитры выражений:

```
> int(int(%?, %?=%?..%?), %?=%?..%?);
```

$$> \int_{?}^{?}\int_{?}^{?} ?d?d?$$

Верхняя строка соответствует Maple-нотации, а нижняя – стандартной математической нотации. В обоих случаях на места вопросительных знаков (мест ввода) вводятся нужные символьные или численные значения.

Перемещение маркера ввода осуществляется клавишами перемещения курсора ← и →. Клавиша **Backspace** стирает символ слева от маркера ввода, а клавиша **Del** – справа от маркера ввода. Для ввода любого символа надо нажать соответствующую клавишу. Клавиша **Shift** включает верхний регистр для ввода заглавных (прописных) букв, а клавиша **Caps Lock** переключает верхний и нижний регистры клавиш с буквами (они меняются местами).

Знак фиксации конца выражения ; (точка с запятой) указывает, что результат его вычисления должен быть выведен на экран, а знак : (двоеточие) отменяет вывод и может использоваться как знак разделителя при записи нескольких выражений в одной строке. Клавиши перемещения курсора позволяют передвигаться по ранее введенным строкам на экране.

1.3.2. Вставка электронных таблиц и работа с ними

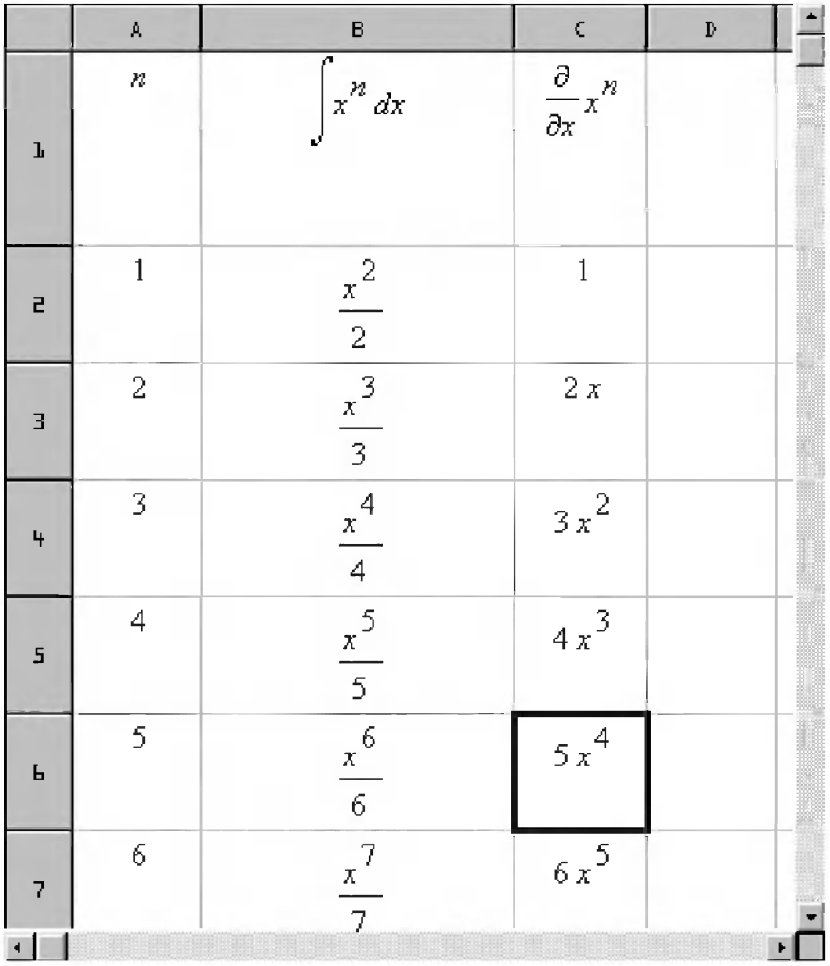
Электронные таблицы, давно известные пользователям приложения Excel из пакета Microsoft Office, долгое время в системах Maple не применялись. Впервые они были введены в реализацию Maple V R5. В последующих версиях для вставки электронных таблиц используется команда **Insert** ⇒ **Spreadsheet**. Она выводит шаблон пустой таблицы. Возможны ручное и автоматическое заполнения таблиц.

Электронная таблица представляет собой двумерный массив ячеек, имеющих адресацию по строкам и столбцам. Номера строк задаются цифрами, а номера столбцов – латинскими буквами. Верхняя левая ячейка имеет адрес **A1**, где **A** – номер столбца и **1** – номер строки. Если одиночные буквы в номерах столбцов заканчиваются, происходит переход на двухбуквенные адреса (**AA**, **AB**, **AC** и т. д.). Такая адресация используется в функциях обработки табличных данных, в том числе известного офисного приложения Excel из пакета Microsoft Office.

Возможности Maple в обработке табличных данных намного превосходят возможности обычных табличных процессоров, например Excel. В частности, наряду с текстовыми и численными данными электронные таблицы Maple могут работать с символьными данными – формулами.

При этом можно сослаться на любую другую ячейку. Такая ссылка указывается значком тильда (**~**) перед адресом ячейки. Так, обозначение **~A1** означает, что будут подставлены данные из ячейки **A1**.

В качестве примера на рис. 1.3 показана таблица значений `n`, интеграла `int(x^n,x)` и производной `diff(x^n,x)` для `n=1..9`.



	A	B	C	D
1	n	$\int x^n dx$	$\frac{\partial}{\partial x} x^n$	
2	1	$\frac{x^2}{2}$	1	
3	2	$\frac{x^3}{3}$	$2x$	
4	3	$\frac{x^4}{4}$	$3x^2$	
5	4	$\frac{x^5}{5}$	$4x^3$	
6	5	$\frac{x^6}{6}$	$5x^4$	
7	6	$\frac{x^7}{7}$	$6x^5$	

Рис. 1.3. Электронная таблица с символьными данными

Подготовка такой таблицы проходит в три этапа. Вначале формируется первый столбец вводом в ячейку **A1** имени переменной `n`, а в ячейку **A2** – значения 1. После этого выделяются ячейки от **A2** до **A10**, и с применением автоматического заполнения они заполняются числами от 1 до 9.

Затем во втором столбце в ячейку **B1** вводится инертная формула `Int(x^~A1,x)`, а в ячейку **B2** – исполняемая формула `int(x^~A2,x)`. После этого выделяются ячейки от **B2** до **B10** и исполняется команда **Spreadsheet Fill Down**. В результате формируется столбец с символьными значениями интегралов.

Аналогично (третий этап) задается формирование столбца с символьными значениями производной от `x^n`.

1.3.3. Понятие о функциях и операторах

Важным понятием системы Maple (да и математики вообще) является понятие *функции*, то есть зависимости одной величины от другой или нескольких величин. Последние именуют параметрами, или аргументами функции. Функция возвращает результат некоторого преобразования исходных данных – параметров функции по определенному правилу, обычно представленному в виде формулы или программного модуля. Maple имеет множество встроенных функций, включенных в его ядро и в пакеты.

Функция в выражениях задается вводом ее имени и списка параметров функции (одного или нескольких), заключенного в круглые скобки: например, `sqrt(2)` задает функцию вычисления квадратного корня с параметром 2 (численной константой). Основным признаком функции является возврат значения в ответ на обращение к ней по имени (идентификатору) с указанием списка параметров функции. Например:

```
> 2*sin(1.);
```

$$1.682941970$$

```
> 2*sin(1);
```

$$2\sin(1)$$

Обратите внимание на особую роль десятичной точки – здесь она служит указанием к выполнению вычисления значения sin(1.0) (или, что то же самое, sin(1.)). А вот синус целочисленного аргумента 1 не вычисляется – считается, что вычисленное значение менее ценно, чем точное значение sin(1).

Ради единства терминологии мы будем пользоваться расширительным понятием функции, относя к нему и те объекты, которые в некоторых языках программирования именуют процедурами, или командами. Например, команды `plot` и `plot3d` построения графиков мы также будем называть функциями, которые возвращают графики аргументов. Под командами же мы будем подразумевать прежде всего команды, содержащиеся в пунктах меню.

Помимо функций, в математических системах для записи математических выражений используются специальные знаки – *операторы*. К примеру, вычисление квадратного корня часто записывается с помощью его специального знака – √. Достаточно хорошо известны операторы сложения +, вычитания -, умножения *, деления / и некоторые другие. Операторы обычно используются с операндами в виде констант или переменных, например в записи `2*(3+4)` числа 2, 3 и 4 – это операнды, а знаки * и + – операторы. Скобки используются для изменения порядка выполнения операций. Так, без них `2*3+4=10`, тогда как `2*(3+4)=14`, поскольку вначале вычисляется выражение в скобках.

Пожалуй, самым распространенным оператором является оператор присваивания `:=`. Он используется для задания переменным конкретных значений, например:

```
> x:=y;
```

$$x := y$$

```
> y:=z;
```

$$y := z$$

```
> z:=2;
```

$$z := 2$$

```
> x;
```

$$2$$

```
> y;
```

$$2$$

Этот простой пример наглядно иллюстрирует эволюцию переменных и особую роль оператора присваивания в системе Maple. В частности, здесь перемен-

ные x, y и z взаимосвязаны с помощью операций присваивания. Поэтому задание значения 2 переменной z приводит к тому, что и переменные y и x принимают то же значение.

Другой распространенный оператор – оператор равенства = – используется для задания равенств и логических условий (например, a=b), указания областей изменения переменных (например, i=1..5 означает формирование диапазона изменения i от 1 до 5) и определения значений параметров в функциях и командах (например, color=black для задания черного цвета у линий графиков).

Операторы сами по себе результат не возвращают. Но они, наряду с функциями и своими параметрами (операндами), позволяют конструировать математические выражения, которые при их вычислении также возвращают результат.

С позиции канонов символьной математики квадратный корень из двух уже является основным результатом вычислений. Поэтому такая функция обычно не вычисляется в численном виде, а выводится в естественном виде, с применением знака квадратного корня √. Для вычисления в привычном виде (в виде десятичного числа с мантиссой и порядком) надо воспользоваться функцией evalf(sqrt(2)) – эта функция обеспечивает вычисление символьного выражения, заданного ее параметром (числом 2). Результат точных целочисленных операций Maple стремится представить в виде рационального числа – отношения двух целых чисел:

```
> (125-2)/(3980+58);
```

$$\frac{41}{1346}$$

1.3.4. Операторы и средства вывода выражений

Для вывода выражений чаще всего используется оператор-символ «точка с запятой», который ставится после соответствующего выражения. Однако есть и *оператор вывода* print:

```
> print(2*sin(1));
> print(2*sin(1.));
```

$$2\sin(1)$$
$$1.682941970$$

Обратите внимание на несколько необычный вывод в этом примере (до сих пор вывод каждого выражения шел после его завершения). Такой вывод обеспечивается, если строки ввода ряда выражений заключены в общую квадратную скобку слева от приведенных выражений. Для блокирования вывода используется оператор «двоеточия», а оператор «%» применяется для исполнения предшествующего выражения:

```
> print(2*sin(1.)):
```

$$1.682941970$$

```
> 2*sin(1.):
> %;
```

1.682941970

Обратите внимание на то, что знак «двоеточия» в первом случае не сработал. Это связано с тем, что сам оператор print выполнил свою функцию – вывода.

Некоторые операторы могут записываться в виде инертных функций, которые выводят записываемое выражение, но без их исполнения. Такие функции обычно записываются с большой буквы. Следующие примеры иллюстрируют применение функции интегрирования – обычной int и инертной Int:

```
> int(x^2,x=0..1);
```

$$\frac{1}{3}$$

```
> Int(x^2,x=0..1);
```

$$\int_0^1 x^2 dx$$

```
> evalf(%);
```

0.3333333333

В первом примере Maple вычисляет интеграл предельно точно и дает ответ в виде рационального числа. Во втором примере просто выводится запись интеграла в математической нотации. В третьем случае функция `evalf` вычисляет этот интеграл и возвращает результат уже в форме числа с плавающей точкой. Мы еще вернемся в дальнейшем к более подробному описанию этих и иных средств вывода.

1.3.5. Обработка и индикация ошибок

При работе с системой Maple надо строго придерживаться правил корректного ввода выражений и иных объектов Maple-языка, называемых *синтаксисом* языка. Однако, как гласит русская пословица, не ошибается только тот, кто ничего не делает. Даже у опытного пользователя возможны ошибки в ходе ввода выражений и задания алгоритмов вычислений.

Алгоритмические, но синтаксически корректные ошибки часто могут не распознаваться системой. Например, если в выражении `a*sin(x)` вы вместо аргумента `x` взяли аргумент `b`, то есть записали `a*sin(b)`, то такую ошибку Maple распознать не может, ибо синтаксически как `a*sin(x)`, так и `a*sin(b)` абсолютно корректны. Если вы перепутаете синус с косинусом и запишете `a*cos(x)`, то такая ошибка также не будет распознана.

Ошибки в записи выражений, описывающих те или иные алгоритмы вычислений, не нарушающие синтаксической корректности, системой Maple не распознаются. Контроль за такими ошибками целиком лежит на пользователе. Приведем еще один характерный пример ошибки, которую Maple не может распознать. Вводя выражение `X/Y*Z`, мы можем предположить, что это означает `X/(Y*Z)`. Однако в Maple приоритет операций деления и умножения одинаков. Поэтому Maple вначале делит `X` на `Y`, а затем полученный результат умножает на `Z`:

```
> X/Y*Z:
> %;
```

$$\frac{XZ}{Y}$$

Ошибки такого рода называют *семантическими*. Если бы мы не проверили вывод с помощью оператора вычисления предыдущего выражения (%), то эта ошибка осталась бы нераспознанной. Выявление и устранение семантических ошибок выполняется на этапе отладки вычислений и программ.

Используйте блокировку вычислений с помощью знака двоеточия только в том случае, когда вы абсолютно уверены в правильности записи выражения – например, когда она очевидна и без повторения в строке вывода. Иначе вы можете не заметить вкравшейся в вычисления серьезной ошибки. В нашем примере мы получили бы верный результат, заключив выражение `Y*X` в круглые скобки. Как обычно, они предназначены для задания явного приоритета выполнения операций – в нашем случае вначале будет вычислено выражение в скобках `(Y*Z)`, а затем уже `X` будет поделено на это выражение:

```
> X/(Y*Z);
```

$$\frac{X}{YZ}$$

Но вернемся к ситуации с *синтаксическими ошибками*, которые система Maple распознает с помощью встроенного в нее *синтаксического анализатора*. Например, если вы задали неправильное имя функции, то это будет опознано синтаксическим анализатором и вычисления не будут выполняться. Maple просто повторит выражение в строке вывода:

```
> son(1.0);
```

$$\mathrm{son}(1.0)$$

В этом примере вместо имени функции `sin` введено ошибочное имя `son`. Maple воспринимает его как некую введенную пользователем функциональную зависимость и потому просто повторяет запись и не выводит сообщения об ошибке. А вот другая ситуация – имя функции `sin` введено верно, но вместо десятичной точки при задании вещественного числа 1.0 использована запятая:

```
> sin(1,0);
Error, (in sin) expecting 1 argument, got 2
```

В данном случае Maple «знает», что работа идет с его встроенной функцией синуса, которая должна иметь единственный аргумент. Задание `(1,0)` означает, что пользователь ввел вместо вещественного числа два целочисленных числа, разделенных запятой. На это Maple отреагировал выдачей сообщения об ошибке (на экране дисплея оно имеет малиновый цвет). Исправьте ошибку, и синус единицы будет благополучно вычислен:

```
> sin(1.0);
```

$$.8414709848$$

А вот еще одна типичная ситуация – в последовательности выражений опущен знак-разделитель (двоеточие или точка с запятой):

```
> X:=2: Y:=3| Z:=4:
```
Error, missing operator or ';'

Тут Maple не только реагирует на ошибку, но и пытается подсказать, что именно пропущено. Более того, маркер ввода в виде мигающей вертикальной черточки будет помещен на место ошибки, и вы сможете тут же устранить ошибку. Правда, подсказки не всегда точны – в нашем случае явно пропущен разделитель в виде двоеточия, а Maple сообщает о пропуске точки с запятой. Впрочем, откуда системе знать, хотим мы вывести результат операции `Y:=4` сразу (для этого нужен разделитель в виде точки с запятой) или откладываем на потом (с помощью символа двоеточия).

Вот еще один пример характерной ошибки – три знака * подряд:

```
> 2**|*3*sin(1.);
```
Error, '*' unexpected

Здесь Maple подсказывает, что один оператор * надо убрать – два знака * подряд означают вполне законный вариант оператора – возведение в степень. При этом маркер ввода вновь указывает место ошибки. Проанализируйте следующие простые примеры:

```
> 2**3*sin(1.);
```
6.731767878

```
> 2^3*sin(1.0);
```
6.731767878

```
> 2^(3*sin(1.0));
```
5.753392735

В двух первых примерах Maple вначале вычисляет функцию синуса, затем производит возведение в степень и лишь потом операцию умножения. Впрочем, такой *приоритет операций* принят практически во всех системах компьютерной математики и в языках программирования. Третий пример показывает изменение приоритета с помощью круглых скобок.

Позже, при описании программирования в Maple, мы опишем более развитые средства контроля над допускаемыми пользователем ошибками. Пока же ограничимся приведенными выше сведениями, полезными уже в начале диалога с системой.

1.3.6. Примеры задания функции пользователя и построения ее графика

На рис. 1.4 показан ряд простых вычислений в среде системы Maple 10. Среди них задание *функции пользователя* `f(x)` с одним параметром `x`. Нетрудно заметить, что параметр указывается в скобках после имени функции, а для записи выражения функции используется знак присваивания `:=` (двоеточие со знаком равенства). Это старый способ задания функции пользователя, который (что видно из приведенного примера) еще работает, но уже не рекомендуется к применению.

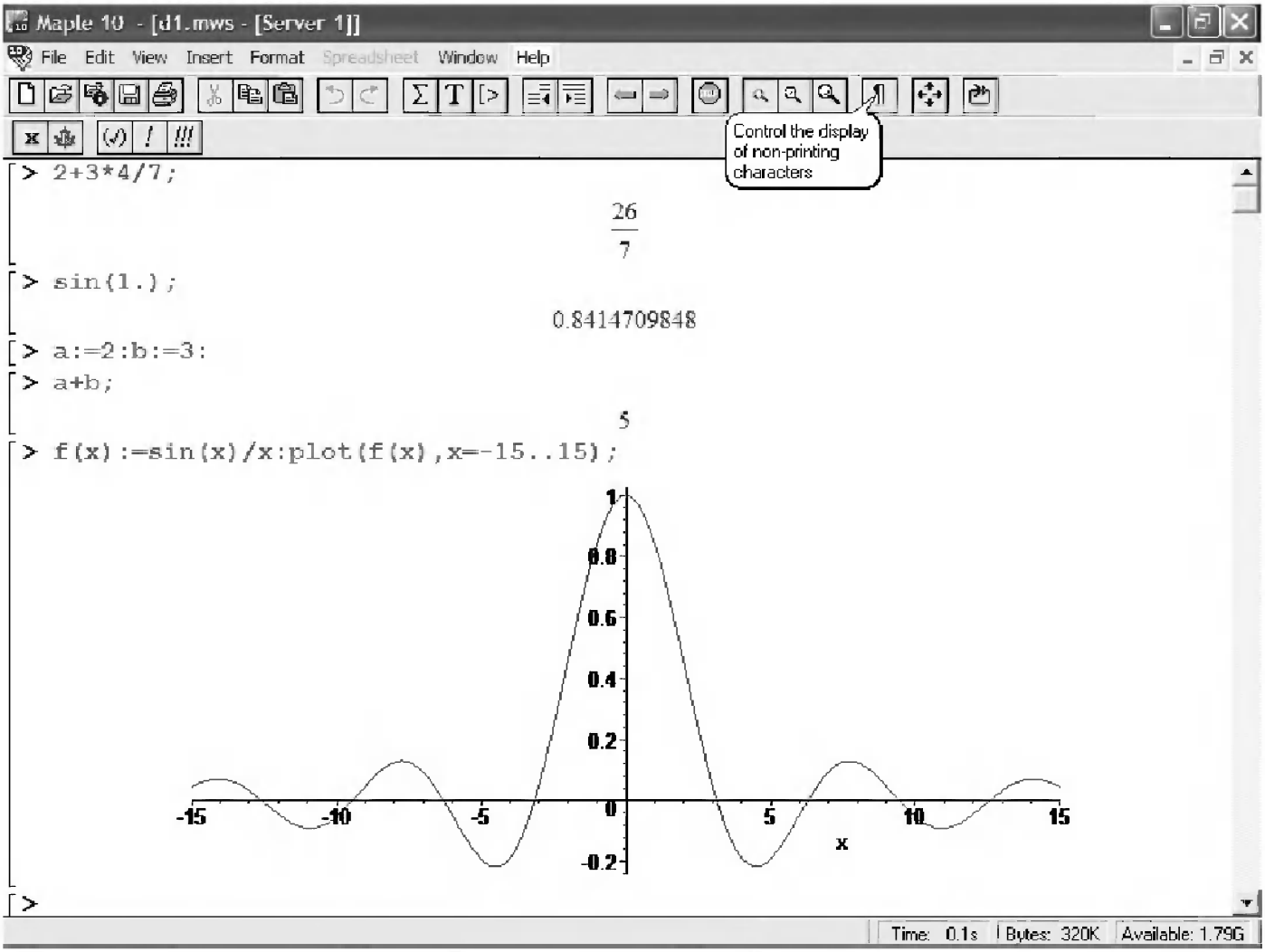


Рис. 1.4. Задание и графическая визуализация функции пользователя в среде Maple 10

Для построения графика функции `f(x)` одной переменной используется функция `plot` в форме

```
plot(f(x), x = -15..15);
```

Нетрудно заметить, что при наличии нескольких параметров функции (в нашем случае их два) они разделяются запятыми. Выражение `x=-15..15` задает, во-первых, указание, относительно какой переменной строится график, а во-вторых, говорит, в какой области значений меняются значения этой переменной – в нашем случае от –15 до +15. Шаг изменения переменной выбирается автоматически, в зависимости от размеров и вида графика.

Столь же просто, как и график обычной функции в декартовой системе координат, можно построить график трехмерной поверхности. Это показано на примере рис. 1.5. В данном случае задана функция двух переменных z(x,y):=sin(x*y), и ее график строится с использованием графической функции `plot3d`. Правила задания пределов изменения переменных `x` и `y` соответствуют описанным выше.

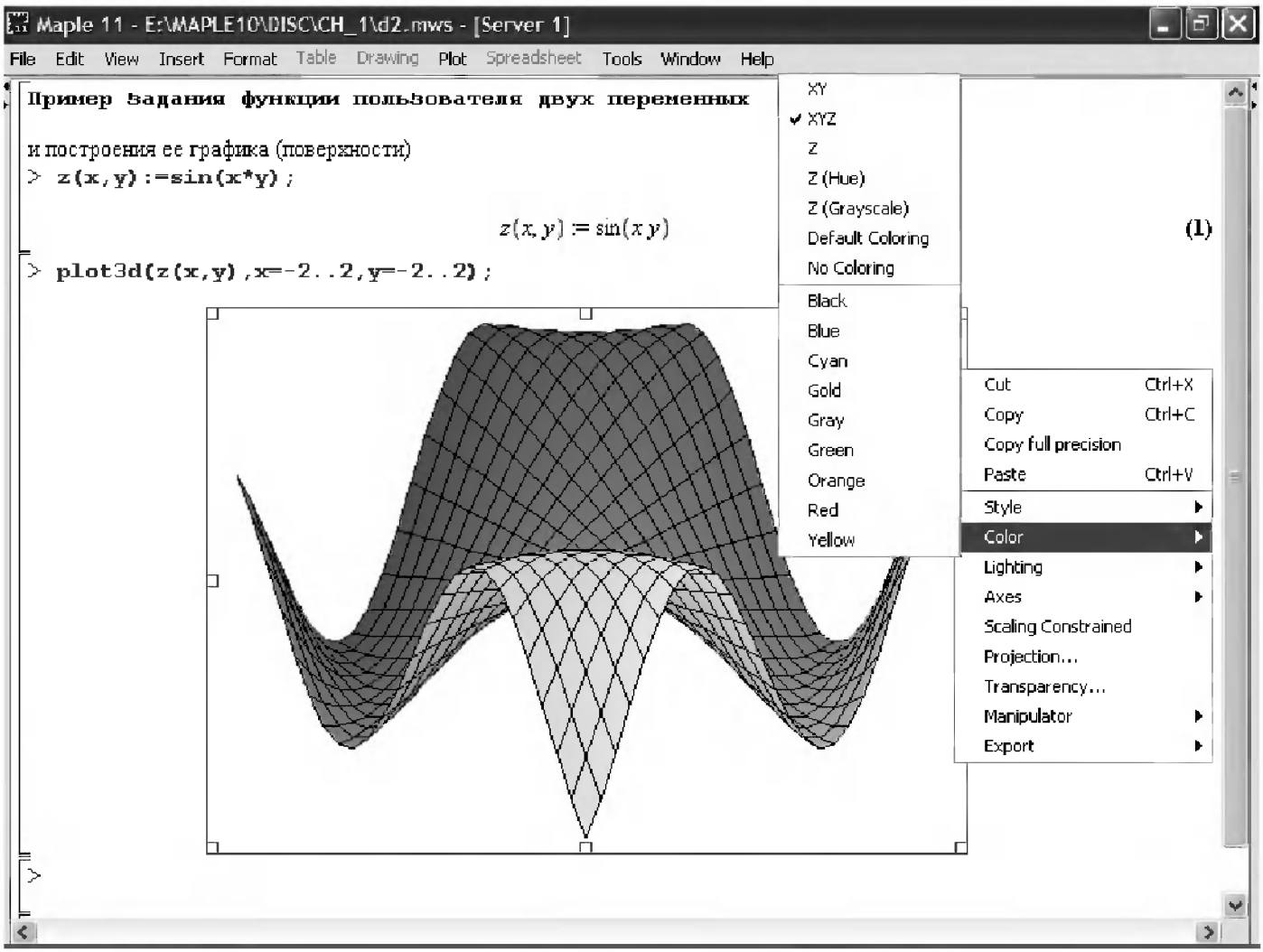


Рис. 1.5. Построение графика трехмерной поверхности, заданной функцией пользователя

В данном случае можно было бы задать функцию пользователя и по старинке в виде `z(x,y):= sin(x*y)`.

При выделении графика щелчком левой клавиши мыши на нем график обрамляется рамкой с местами ввода, за которые можно цепляться курсором мыши и растягивать график в ту или иную сторону. Кроме того, мышью при нажатой левой клавише можно вращать график в ту или иную сторону. Ряд возможностей форматирования графика дает контекстное меню правой клавиши мыши, показанное на рис. 1.5. С ними нетрудно разобраться самостоятельно.

В последних версиях Maple рекомендуется задавать функцию пользователя в виде

`Имя_функции:=(Список_параметров)->Выражение`

Например, в нашем случае: `z:=(x,y)->sin(x*y)`. При этом переменные `x` и `y` становятся локальными, а вызов функции задается как `z(x,y)`.

1.4. Символьные вычисления в Maple

1.4.1. Простой пример символьных вычислений

Maple, как и другие СКА, открывает обширные возможности выполнения *символьных* (*аналитических*) вычислений. Мы уже описывали примеры решения квадратного уравнения. Возьмем еще один простой пример из электротехники – требуется найти сопротивление трех параллельно включенных резисторов $R1$, $R2$ и $R3$ произвольной величины. Из курса электротехники известно, что можно задать следующее равенство, определяющее суммарное сопротивление $R0$:

```
> eq:=1/R0=1/R1+1/R2+1/R3;
```

$$eq := \frac{1}{R0} = \frac{1}{R1} + \frac{1}{R2} + \frac{1}{R3}$$

Теперь достаточно использовать функцию решения уравнений `solve`, чтобы найти значение $R0$ в общей аналитической форме:

```
> R0:=solve(eq,R0);
```

$$R0 := \frac{R1\,R2\,R3}{R2\,R3 + R1\,R3 + R1\,R2}$$

С таким же успехом мы можем найти аналитическое выражение для $R1$, определяющее $R1$ через $R0$, $R2$ и $R3$:

```
> R1:=solve(eq,R1);
```

$$R1 := -\frac{R0\,R2\,R3}{-R2\,R3 + R0\,R3 + R0\,R2}$$

Нетрудно проверить, что результат может быть получен и в численном виде для конкретных значений $R1$, $R2$ и $R3$:

```
> R1:=1:R2:=2:R3:=3:R0;
```

$$\frac{6}{11}$$

```
> evalf(%);
```

$$.5454545455$$

Позже мы рассмотрим не одну сотню примеров на решение в среде Maple задач в символьном виде с их визуализацией – как графической, так и численной.

1.4.2. Представление входных выражений в математической форме

В Maple ввод исходных данных производится привычными для языков программирования средствами – с помощью функций и операторов, задаваемых в командной строке. Зато результаты вычислений получаются по умолчанию в виде

обычных формул (хотя есть возможность их представления в другом виде, например принятом в редакторе LaTeX или языках программирования Fortran и C).

Для представления входных данных в математической форме Maple предлагает ряд средств. Во-первых, это *текстовые комментарии*, в которые можно вводить формулы. Во-вторых, это *инертные функции*, которые не вычисляются, но дают вывод на экран выражений в естественной математической форме (рис. 1.6). И в-третьих, это возможность быстрого преобразования строковых выражений ввода в естественные математические формулы.

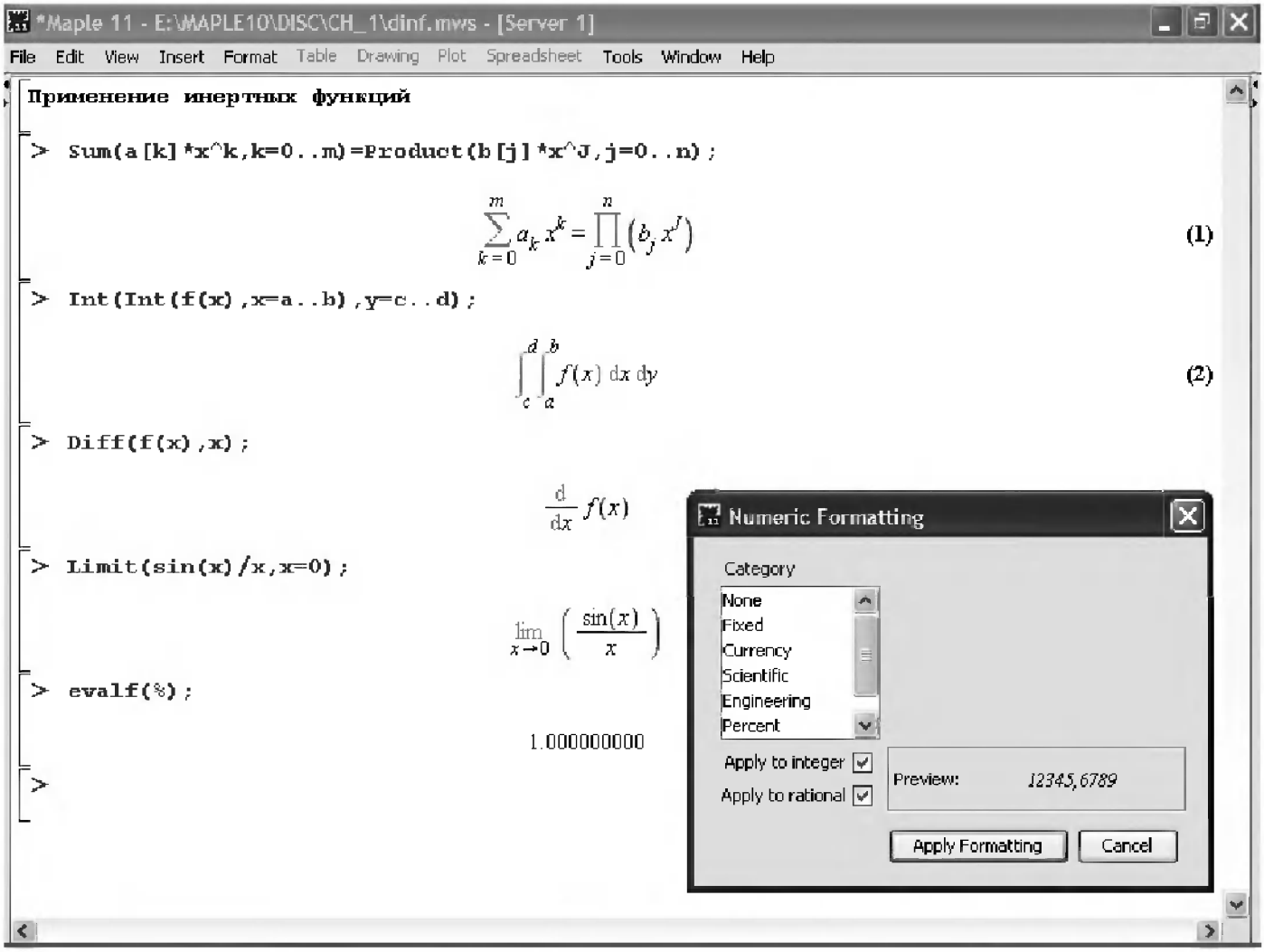


Рис. 1.6. Примеры применения инертных функций

Имена инертных функций начинаются с большой буквы, и функции выводят математическое выражение в естественной математической нотации. С помощью ряда функций, например `evalf`, можно вычислить математическое выражение, полученное инертной функцией. На рис. 1.6 внизу дан пример такого вычисления для предела функции $\sin(x)/x$. Обратите внимание на вывод из контекстной панели форматирования чисел. С ее помощью можно быстро изменить формат вывода числа, в данном случае задающего предел.

Для преобразования исполняемых выражений ввода на Maple-языке в обычные математические формулы достаточно, выделив входное выражение, нажать

первую кнопку контекстной панели (со знаком «x») – соответствующее выражение тут же приобретет вид обычной математической формулы.

1.4.3. Типовые символьные вычисления

На рис. 1.7 показаны несколько примеров выполнения символьных вычислений математического характера: преобразование тригонометрического выражения с помощью функции упрощения `simplify`, вычисление суммы ряда функцией `sum` и вычисление производной функцией `diff` и неопределенного интеграла функцией `int`.

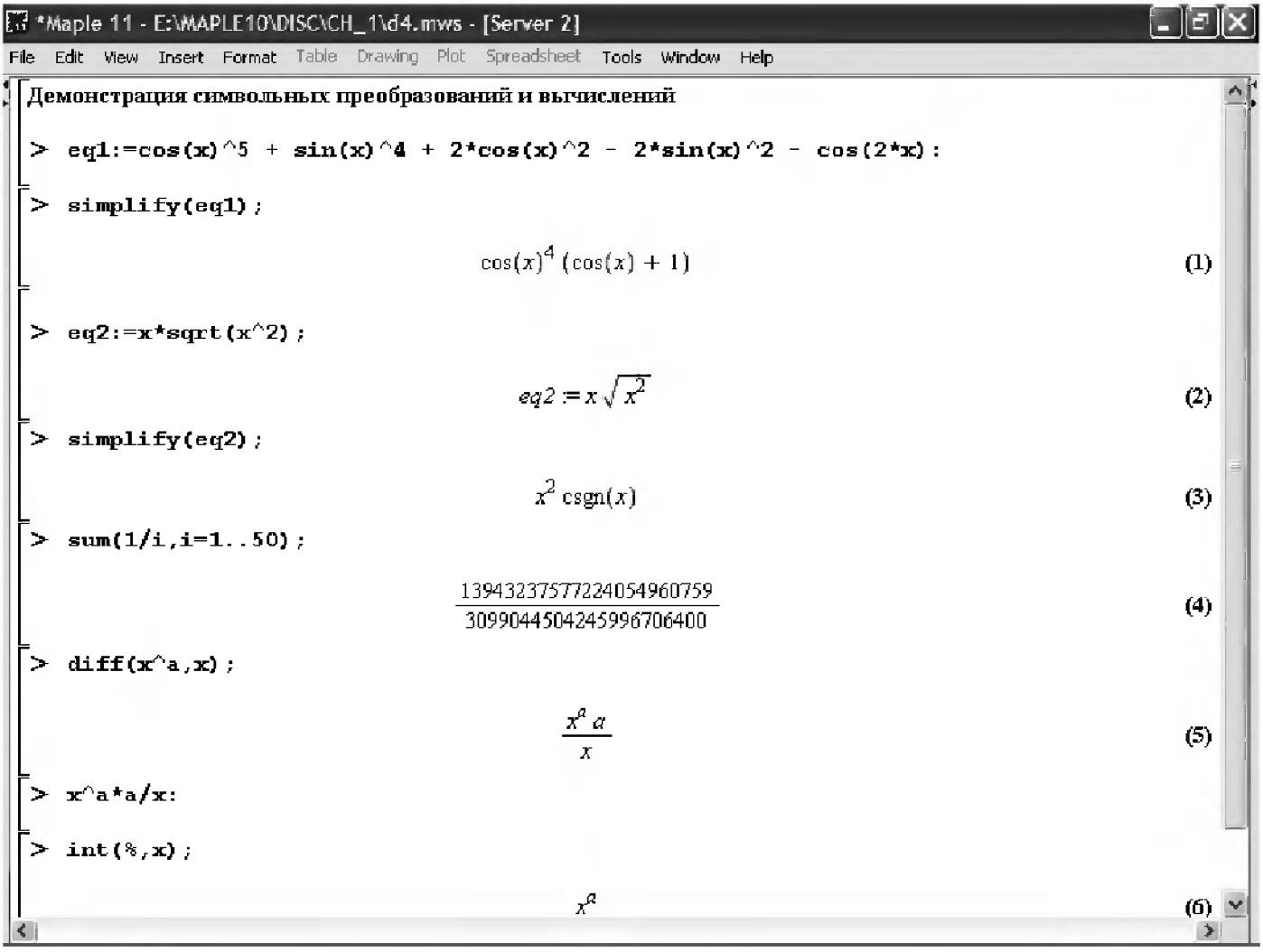


Рис. 1.7. Примеры символьных вычислений в среде Maple 11

Обратите внимание на последний пример. В нем используется знак % для ввода на его место предшествующего выражения. Два знака % вводят предшествующее предшествующему выражение и т. д.

1.4.4. Разбухание результатов символьных вычислений

Одной из проблем в применении систем компьютерной алгебры является «*разбухание*» результатов – как оконечных, так и промежуточных. К примеру, численное решение кубического уравнения не вызовет трудностей даже на калькуляторе [13, 14, 16], тогда как системы символьной математики выдают его в виде громоздких, хотя и точных формул – см. примеры на рис. 1.8. Заметьте, что для кубического уравнения в окно поместилась только небольшая часть решения. Просмотреть оставшуюся часть можно с помощью линейки прокрутки в правой части окна документа.

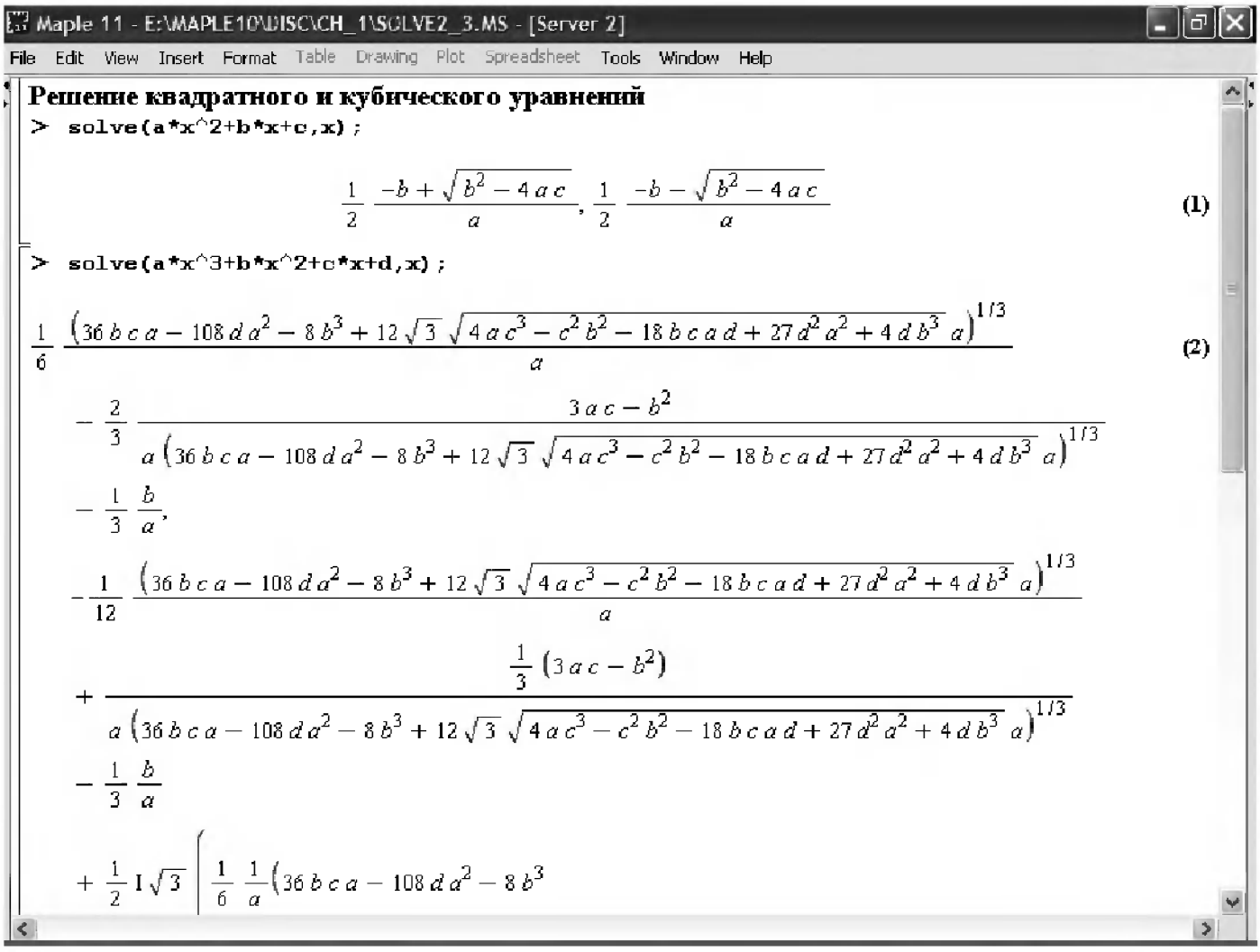


Рис. 1.8. Решение квадратного и кубического уравнений в символьной форме

Стремление системы выдать полный и математически предельно точный результат, безусловно, очень важно для математиков. Но для многих прикладных задач, с которыми имеют дело инженеры и техники, оно оборачивается большими неудобствами. Инженеры часто прекрасно знают, какие из членов математиче-

ских формул можно отбросить, тогда как для математика-теоретика или аналитика такое действо – типичное кощунство.

1.4.5. Решения системы линейных уравнений

Приведем еще один характерный пример – решение системы линейных уравнений с помощью функции `solve` (рис. 1.9). Обратите внимание на форму задания уравнений и выдачи результатов и поразительную естественность решения задачи. Значение переменной *z* на рис. 1.9 выделено, при этом видно, что Maple отображает его поле под панелью инструментов.

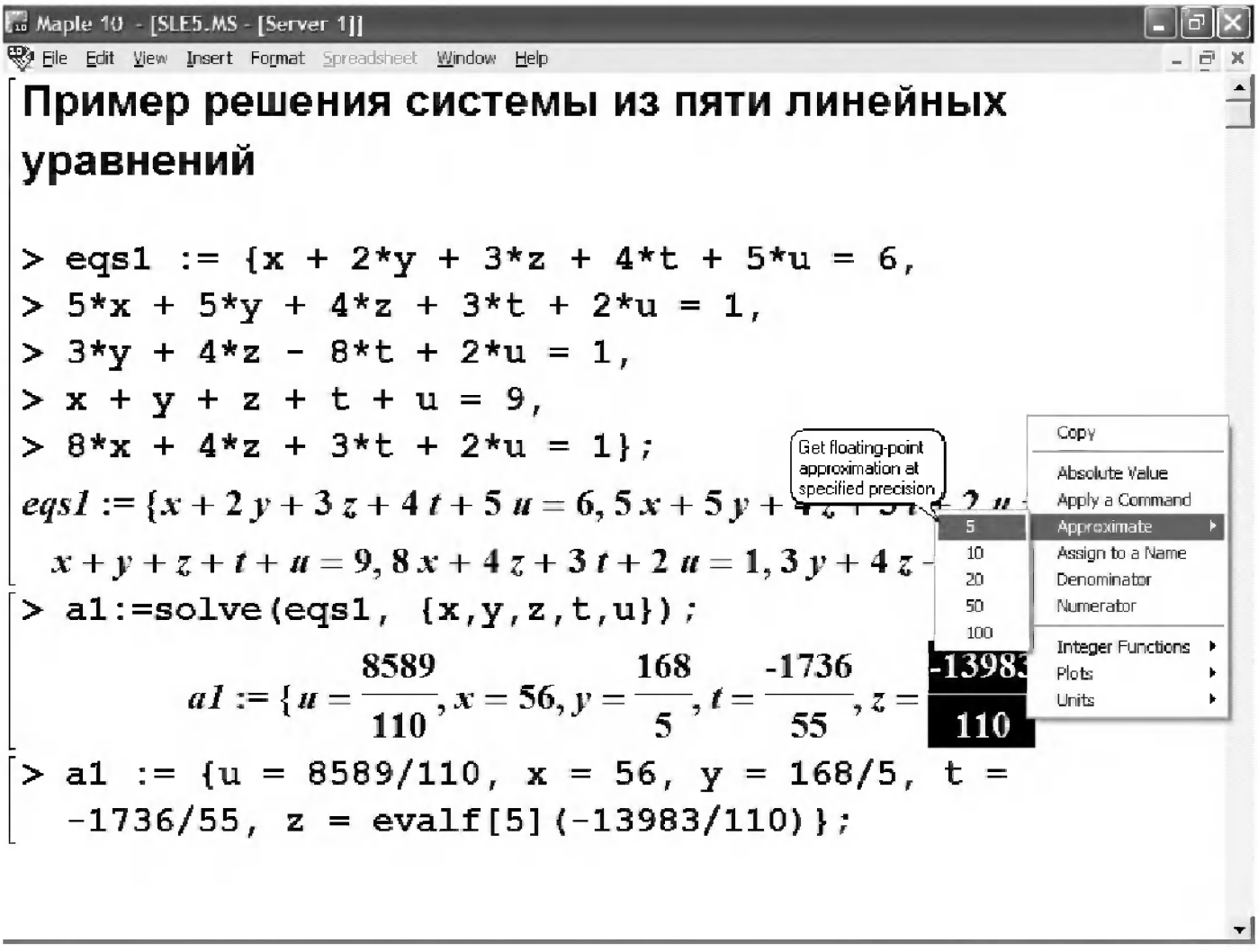


Рис. 1.9. Решение системы из пяти линейных уравнений (в среде Maple 10)

Maple стремится выдать результат с максимальной точностью – обычно в форме рациональных чисел. Но с помощью функции `evalf` можно получить результат в форме вещественных чисел в формате с плавающей точкой. Это можно сделать, и выделив нужное число, а затем используя команду `Approximate` из контекстного меню правой клавиши мыши – это показано справа на рис. 1.9.

Слова `solve`, `diff`, `int` и `evalf` с их аргументами являются именами встроенных в систему функций, возвращающих символьные значения результатов. В общих чертах назначение их пользователю Maple вполне понятно. Но в последующих главах книги мы познакомимся с этими и многими функциями гораздо более подробно и обстоятельно.

1.5. Системы компьютерной математики Mathematica

1.5.1. Особенности систем класса Mathematica

Версия Mathematica 1.0, появившаяся в 1988 г., и была «пробным камнем», и самой известной разработкой фирмы Wolfram Research, Inc. стала версия 2.0 СКА Mathematica 2 под Windows, появившаяся в 1991 г. У нас она впервые стала известна благодаря обзорам [21–23, 31–33].

Цели нового по тем временам проекта были достаточно амбициозными:

- разработка мощного и универсального ядра системы (названного Kernel), способного работать на различных компьютерных платформах;
- создание многофункционального языка программирования, ориентированного на математические приложения;
- разработка современного пользовательского интерфейса;
- создание обширных пакетов применений и расширений системы;
- обеспечение свойства адаптации – обучения новым математическим законам и закономерностям.

Несмотря на определенные недоработки, система быстро заняла ведущие позиции на рынке математических систем. Особенно привлекательны были обширные графические возможности системы и реализация интерфейса типа Notebooks – блокнотов, сочетающих в себе программы и команды с данными, представленными в формульном, текстовом, табличном и графическом виде. При этом система обеспечивала динамическую связь между ячейками документов в стиле электронных таблиц даже при решении символьных задач, что принципиально и выгодно отличало ее от других систем.

В середине 1997 г. появилась Mathematica 3. Был кардинально переработан пользовательский интерфейс системы, он вобрал в себе массу новинок – от раздельного вывода на экран деталей и панелей интерфейса до мощной и прекрасно реализованной справочной системы (подобной примененной в Maple с классическим вариантом интерфейса). Устранен недостаток предшествующих версий – небольшое число примеров в справочной системе. Все примеры стали «живыми» – их в любой момент можно переиначить на свой лад и перенести в свои документы. Продолжая линию развития универсального ядра системы, фирма Wolfram Research обеспечила применение этой системы на целом ряде операционных систем –

Windows 95, Windows NT, Macintosh, Power Macintosh, SunOS, Solaris, HP-UX, SGI, Linux и др.

Средства Mathematica 4 позволяют готовить документы в стиле Notebook на самом высоком полиграфическом уровне воспроизведения текстов, математических формул и графиков. Размеры блокнота практически не ограничены, и он может быть распечатан во всей красе с помощью струйного или цветного лазерного принтера.

Пожалуй, главной отличительной особенностью системы Mathematica 4 стало кардинальное ускорение численных расчетов. Кардинально уменьшено и время обращения к памяти при записи и считывании массивов, а заодно существенно повышена плотность упаковки массивов для данных различного типа (за счет применения особой технологии упаковки массивов).

Указанные достоинства системы Mathematica 4 достигнуты за счет выбора и тщательной оптимизации алгоритмов численных вычислений. Mathematica 4 способна при вычислениях давать миллион верных цифр. Число π, к примеру, с таким числом верных знаков Mathematica 4 выдает за 138 секунд!

1.5.2. Система Mathematica 5

После более трех лет разработок в середине 2003 г. на рынок была выпущена новейшая и давно ожидаемая версия системы Mathematica 5. Mathematica 5 включает важные расширения системы для широкого круга численных и символьных операций Mathematica на основе алгоритмов нового поколения. Новые возможности Mathematica 5 и последующих версий Mathematica 5.1/5.2 достаточно подробно описаны в книгах [78–81].

Отметим, что в ходе разработок этих версий существенно оптимизированы операции численной линейной алгебры плотных и разрешенных матриц, резко (порою в сотни раз) увеличена скорость вычислений, присутствует новая команда LinearSolveFunction для решения линейных систем уравнений, введена новая функция FindFit для нелинейной аппроксимации функций и кривых, введена новая команда глобальной оптимизации NMinimize, обеспечена поддержка решения n-мерных уравнений с частными производными в команде NDSolve, обеспечена поддержка решения алгебраических дифференциальных уравнений в команде NDSolve и др.

В Mathematica 5, помимо ранее имевшихся пакетов расширения, дополнительно были включены пакеты: Statistical plots and graphics по статистической графике и Algebraic number fields по алгебраическим числовым полям. Mathematica 5 поддерживает работу в среде операционных систем Windows 98/ME/NT4.0/2000/XP и требует для установки 345 Мбайт памяти на жестком диске (версии 4.2 в среде Windows 95 было достаточно 235 Мбайт).

1.5.3. Идеология систем Mathematica

Идеология систем Mathematica, кстати, как и систем Maple, базируется на двух, казалось бы, взаимно исключающих друг друга, положениях:

- решение большинства математических задач в системе может производиться в диалоговом режиме без традиционного программирования;
- входной язык общения системы является одним из самых мощных языков функционального программирования, ориентированных на решение различных задач (в том числе математических).

Противоречивость этих положений чисто кажущаяся. На самом деле Mathematica – типичная система программирования с проблемно-ориентированным языком программирования сверхвысокого уровня. Его, как и язык Maple, можно отнести к классу интерпретаторов. Благодаря этому работа с системой происходит в диалоговом режиме – пользователь задает системе задание, а она тут же выполняет его. Разумеется, Mathematica содержит достаточный набор управляющих структур для создания условных выражений, ветвления в программах, циклов и т. д., для полной автоматизации вычислений.

В новых реализациях Mathematica к указанным двум принципам добавлен ряд новых принципов:

- за счет устранения ограничений по скорости реализации численных методов системы Mathematica 4/5 стали поистине универсальными СКМ;
- системы стали существенно расширяемыми за счет применения встроенных (Add Ons) и внешних пакетов расширения;
- системы реализуют высочайшее качество подготовки ноутбуков – документов, содержащих одновременно текстовые записи, аналитические выражения, таблицы, графики, рисунки и другие компоненты;
- системы позволяют создавать полностью завершенные высококачественные электронные уроки, статьи и книги с высоким уровнем визуализации всех видов вычислений;
- системы стали интеллектуальными системами предоставления знаний в области фундаментальной и прикладной математики.

Теперь рассмотрим простейший пример работы системы Mathematica, показанный на рис. 1.10.

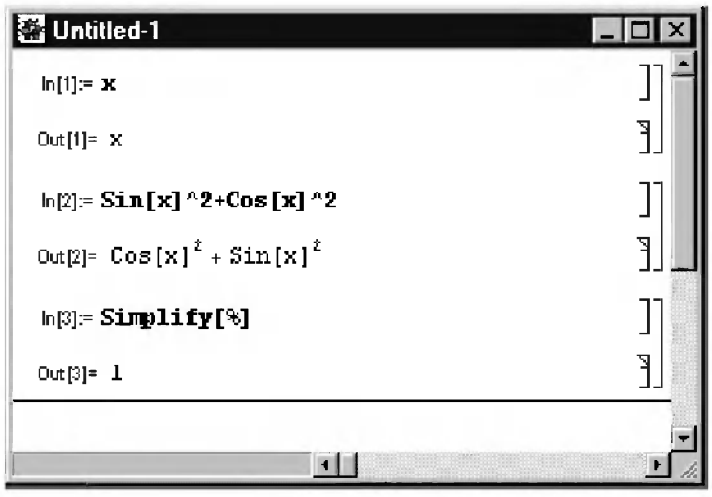


Рис. 1.10. Простейший пример символьных вычислений в окне ноутбука системы Mathematica 5

Для простых примеров удобно приводить их не в виде копий экрана или окон системы, а прямо в тексте книги. Воспроизведем примеры рис. 1.10 в тексте этой книги:

```
x
 x
Sin[x]^2+Cos[x]^2
 Cos[x]^2 + Sin[x]^2
Simplify[%]
 1
```

Здесь содержимое входных и выходных ячеек задано разным шрифтом. Кроме того, содержимое выходной ячейки дается с отступом. Это исключает путаницу в интерпретации содержимого ячеек.

При рассмотрении приведенного выше простейшего примера можно сделать несколько характерных выводов. Прежде всего видно, что вывод неопределенной переменной x дает саму переменную. Функции sin(x) и cos(x) в системе Mathematica обозначаются как Sin[x] и Cos[x], то есть имена функций начинаются с заглавной буквы. Само по себе выражение sin(x)2 + cos(x)2 просто повторяется, а для его вычисления используется функция Simplify (упростить), аргументом которой является знак %, означающий подставку предшествующего выражения. Два знака % можно использовать для подстановки предшествующего предшествующему выражения и т. д. Для вычисления строки ввода надо нажимать клавиши **Shift+Enter**, нажатие только клавиши **Enter** переводит строку в области ввода, именуемой также ячейкой ввода.

Обратите внимание на то, что система выделяет области ввода определителем In[N], а области вывода – определителем Out[N], где N – автоматически проставляемый номер строки. Кроме того, в левой части окна (рис. 1.46) отображаются квадратные скобки с особыми признаками, которые будут описаны позже. Далее мы, как правило, будем опускать определители и представлять документы в упрощенной и более компактной форме.

Ячейки нумеруются по мере их использования. При этом можно с конца документа вернуться к его началу или середине и, изменив содержимое ранее использованных ячеек, снова выполнить вычисления. При этом ячейки меняют номера. При загрузке файла ячейки перенумеруются в строго последовательном порядке. Таким образом, номера ячеек не являются жестко фиксированными и представляют собой сугубо техническое средство. Это говорит в пользу отказа от вывода определителей ячеек.

Разумеется, роль СКА Mathematica далеко не исчерпывается подтверждением указанного общеизвестного тождества. Эти системы способны преобразовывать сложнейшие алгебраические выражения, находить аналитические решения сложных систем линейных, нелинейных и дифференциальных уравнений, манипулировать со степенными многочленами, вычислять производные и интегралы, анализировать функции и находить их пределы и т. д. Это видно из следующих примеров для системы Mathematica, тоже достаточно простых и показательных:

```
D[x^n,x]
```

$$n x^{-1+n}$$

```
Integrate[x^n,x]
```

$$\frac{x^{1+n}}{1+n}$$

```
D[%,x]
```

$$x^n$$

```
Solve[a*x^2+b*x+c=0,x]
```

$$\left\{\left\{x \to \frac{-b-\sqrt{b^2-4ac}}{2a}\right\}, \left\{x \to \frac{-b+\sqrt{b^2-4ac}}{2a}\right\}\right\}$$

```
Series[Sin[x],{x,0,7}]
```

$$x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040} + O[x]^8$$

В этих примерах функция D (как исключение из правил обозначенная одной буквой) вычисляет производную, функция Integrate – интеграл, функция Solve решает нелинейное уравнение (в данном случае квадратное), а функция Series разлагает выражение в ряд относительно заданной переменной и при заданных начальном значении переменной и максимальной степени ряда. В фигурных скобках задаются списки некоторых входных и выходных параметров (аргументов).

Системы компьютерной алгебры по существу являются справочниками по многим специальным функциям. При этом они способны давать результаты вычислений в виде специальных функций, что демонстрируют следующие примеры:

```
Sum[1/k^9,{k,1,n}]
```

$$\text{HarmonicNumber}[n, 9] + \frac{\text{PolyGamma}[8, 1]}{40320} + \text{Zeta}[9]$$

```
Integrate[Log[x]*Exp[-x^4],{x,0,Infinity}]
```

$$-\frac{1}{32}\,\text{Gamma}\left[\frac{1}{4}\right]\,(2\,\text{EulerGamma} + \pi + \text{Log}[64])$$

```
DSolve[y ''[t]+y'[t]+y[t]/t==0,y[t],t]
```

$$\{\{y[t] \to e^{-t}\,t\,C[1] - e^{-t}\,C[2]\,(e^{t} - t\,\text{ExpIntegralEi}[t])\}\}$$

Здесь специальные функции получаются в результате вычисления суммы, символьного интегрирования и решения в аналитическом виде дифференциального уравнения. Соответствующие функции будут более подробно описаны в дальнейшем.

1.5.4. Интерфейс системы Mathematica 5

Окна системы Mathematica 5 при ее запуске показаны на рис. 1.11. В целом окна Mathematica 5 выполнены в светлых золотистых тонах, тогда как окна Mathematica 4/4.1/4.2 более темные. Вид окна ноутбука (документа) у всех версий одинаков (см. также рис. 1.10). В дальнейшем мы будем останавливаться на отличиях описываемых версий системы Mathematica только в тех случаях, когда они заметны и принципиальны.

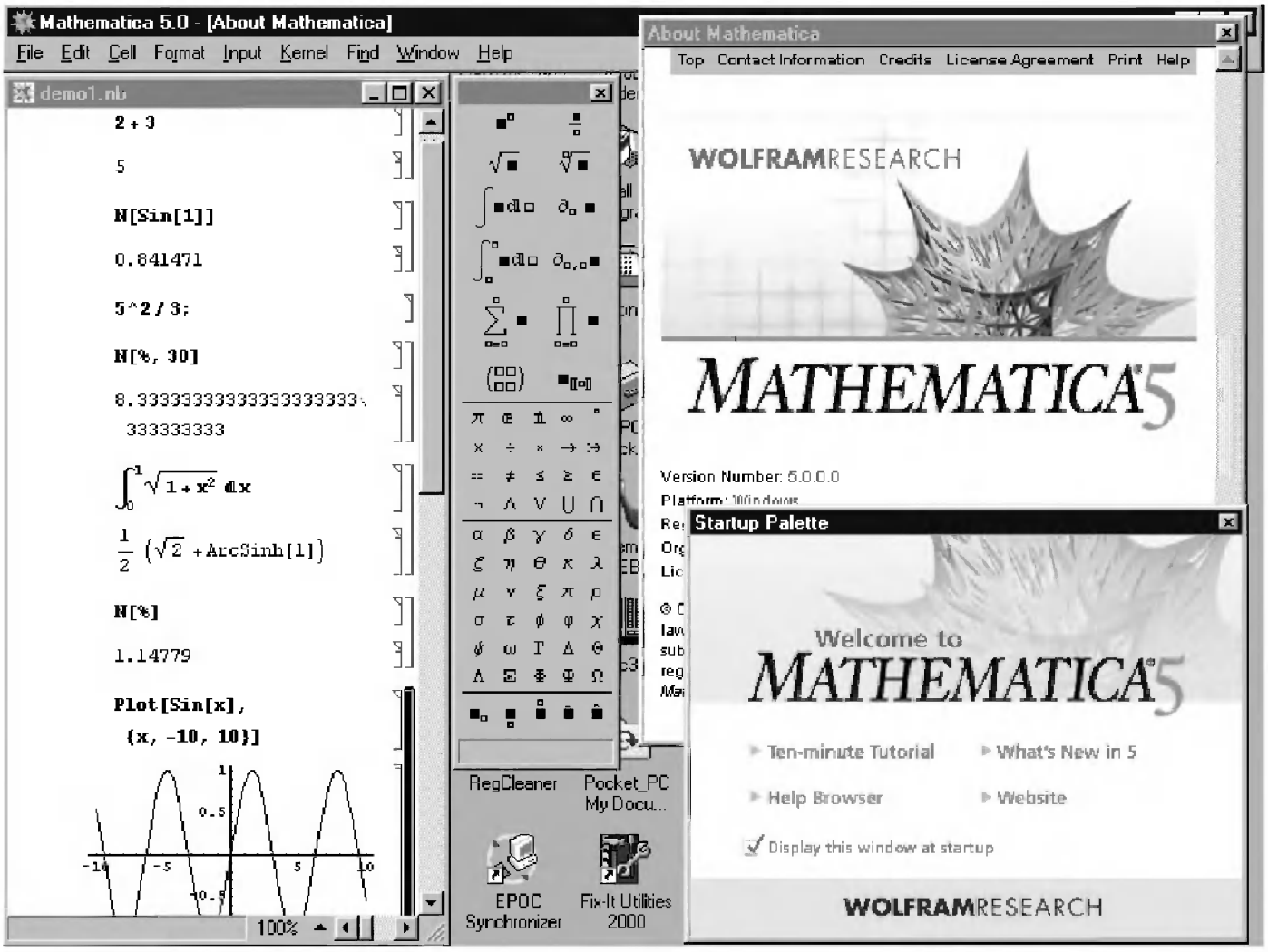


Рис. 1.11. Окна системы Mathematica 5

Панель главного меню имеет всего две строки:

- с названиями системы и загруженного файла;
- позициями главного меню.

Висящее главное меню системы (рис. 1.11 сверху) содержит следующие позиции:

- **File** – работа с файлами: задание нового файла, выбор файла из каталога, закрытие файла, запись текущего файла, запись файла с изменением имени, печать документа и выход в Windows;
- **Edit** – основные операции редактирования (отмена операции, копирование выделенных участков документа в буфер с их удалением и без удаления, перенос выделенных участков, их стирание);
- **Cell** – работа с ячейками (объединение и разъединение ячеек, установка статуса ячейки, открытие и закрытие);
- **Format** – установка множества форматов документов;
- **Input** – задание элементов ввода (графиков, матриц, гиперссылок и др.);
- **Kernel** – управление ядром системы (ручное и автоматическое исполнение ноутбука и т. д.);
- **Find** – поиск заданных данных;

- **Window** – операции с окнами и их расположением;
- **Help** – управление справочной системой.

Элементы интерфейса, в частности окно редактирования, можно перетаскивать мышью (зацепившись курсором мыши за титульную строку и держа нажатой левую клавишу) или растягивать в разные стороны.

В последние версии системы фирма – разработчик этих систем ввела выводимые по желанию пользователя и перемещаемые по экрану инструментальные панели с множеством пиктограмм ввода математических символов, функций и команд управления системой. Они выводятся с помощью подменю **Palettes** (Палитры) в позиции **File** главного меню системы. Если вывести все десять инструментальных панелей, то они едва умещаются во всем главном окне системы – рис. 1.12.

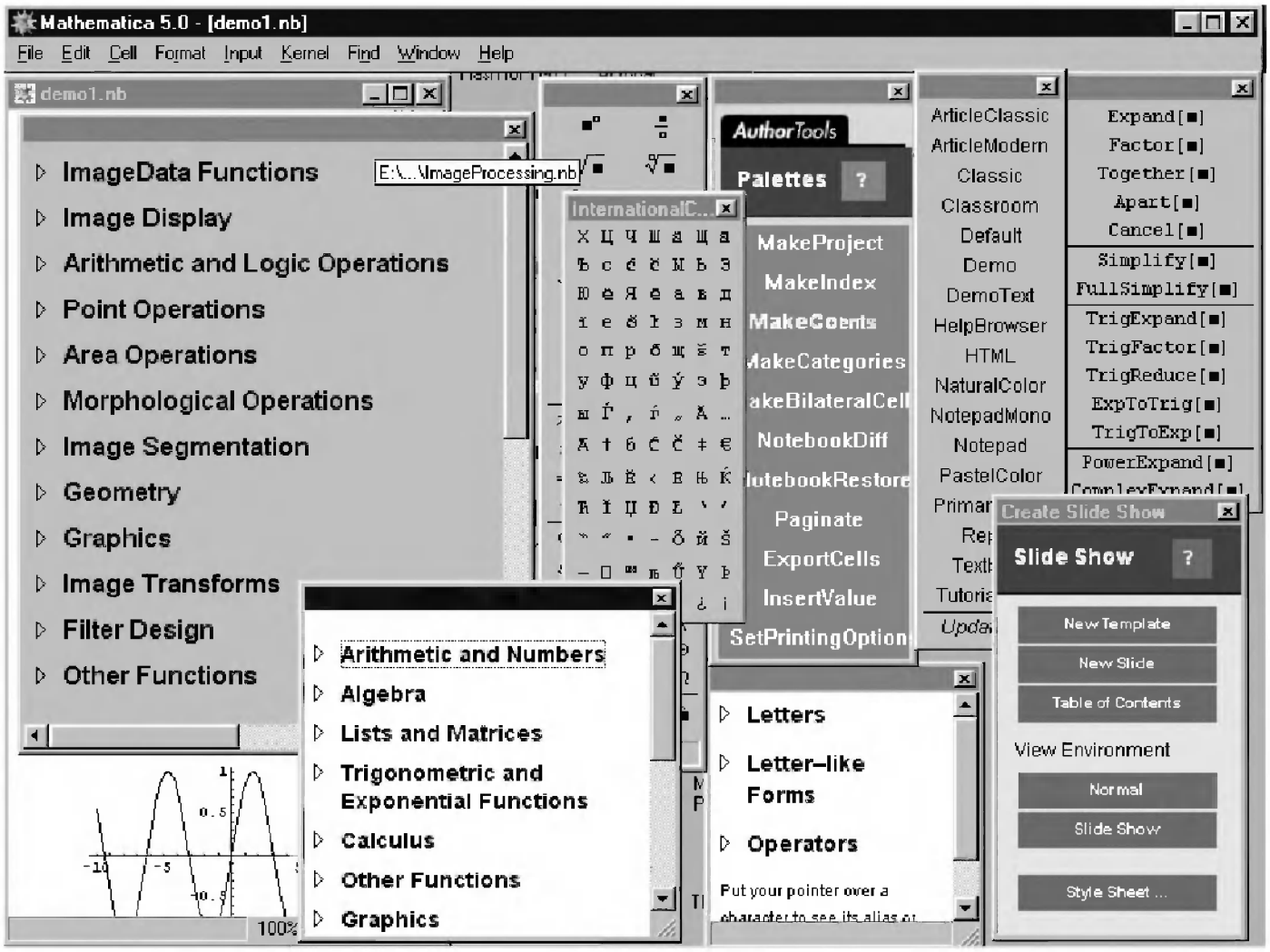


Рис. 1.12. Инструментальные панели системы Mathematica 5

Инструментальные панели, предназначенные для вывода математических спецзнаков, намного облегчают их ввод и упрощают работу по подготовке документов. Общее число специальных математических знаков (греческих и латинских букв, операторов, функций и команд), вводимых с помощью палитр, около 700. Многие знаки имеют альтернативные варианты ввода с применением комбинаций клавиш – их можно найти в справочной базе данных системы.

1.5.5. Первые навыки работы и понятие о ноутбуках

Как уже отмечалось, для выполнения простых арифметических операций достаточно набрать необходимое математическое выражение и нажать клавиши **Shift** и **Enter** одновременно (сама по себе клавиша **Enter** используется только для задания перевода строки внутри текущей строки ввода).

Нетрудно заметить, что вычисления в оболочке системы проходят так же, при вычислениях на обычном калькуляторе. Однако, прежде чем получить результат первого вычисления, даже столь простого, как вычисление 2 + 3, вам придется дождаться, когда система загрузит свое ядро. Обычно это занимает несколько секунд. Впрочем, последующие вычисления (если они не слишком сложны) проходят уже почти мгновенно.

Отдельные ячейки с математическими выражениями и результатами их вычислений отмечаются в правой части главного окна редактирования характерными тонкими квадратными скобками синего цвета. Это позволяет следить за тем, к чему относятся математические выражения, – к исходным данным или результатам. Кроме того, ячейки могут иметь различный статус, который отмечается соответствующими значками над квадратными скобками, – речь об этом более подробно пойдет ниже.

Изображение в окне редактирования очень напоминает записи аккуратного и педантичного инженера или ученого в записной книжке или на листке бумаги. В общем случае в окне видны поясняющие текстовые комментарии, числа, таблицы, математические выражения и формулы и графики различных типов. Мы будем называть эти записи документом – в оригинале они именуются «блокнотами», «записными книжками» и даже «*ноутбуками*» (от слова *notebook*, которое с английского языка на русский переводится как записная книжка или блокнот). Представление протокола работы с системой в такой форме считается наиболее целесообразным для математических систем. Файлы ноутбуков имеют расширение .nb.

Для того чтобы документ имел форму ноутбука, надо предпринять определенные операции по форматированию документа и приданию ему нужного вида. Прежде всего каждый шаг вычислений следует снабжать поясняющими надписями. Их можно прямо вводить в строки ввода, но затем отформатировать под текстовый формат подходящего стиля. Для этого выделяется строка ввода (установкой маркера ввода на ее скобку и щелчком левой клавиши мыши) с текстовой надписью. Пространство внутри скобки при этом затеняется (делается черным). Затем выполняется команда **Format** ⇒ **Style** ⇒ **Text** (**Alt+7**). Она задает текстовый формат надписи, который является неисполняемым.

С помощью других команд позиции **Format** главного меню, которые мы рассмотрим в дальнейшем, можно задать надпись разным шрифтом, разным цветом с выделением фона и т. д. Как уже отмечалось, для ввода математического выражения по шаблону и представления его в естественной математической форме исполь-

зуется стандартный формат StandardForm ячеек ввода. Рисунок 1.13 показывает созданный таким образом простой блокнот, в котором наряду с поясняющими текстовыми надписями задана операция вычисления суммы квадратов чисел от 1 до 10 и построение графиков двух функций с помощью графической функции **Plot**.

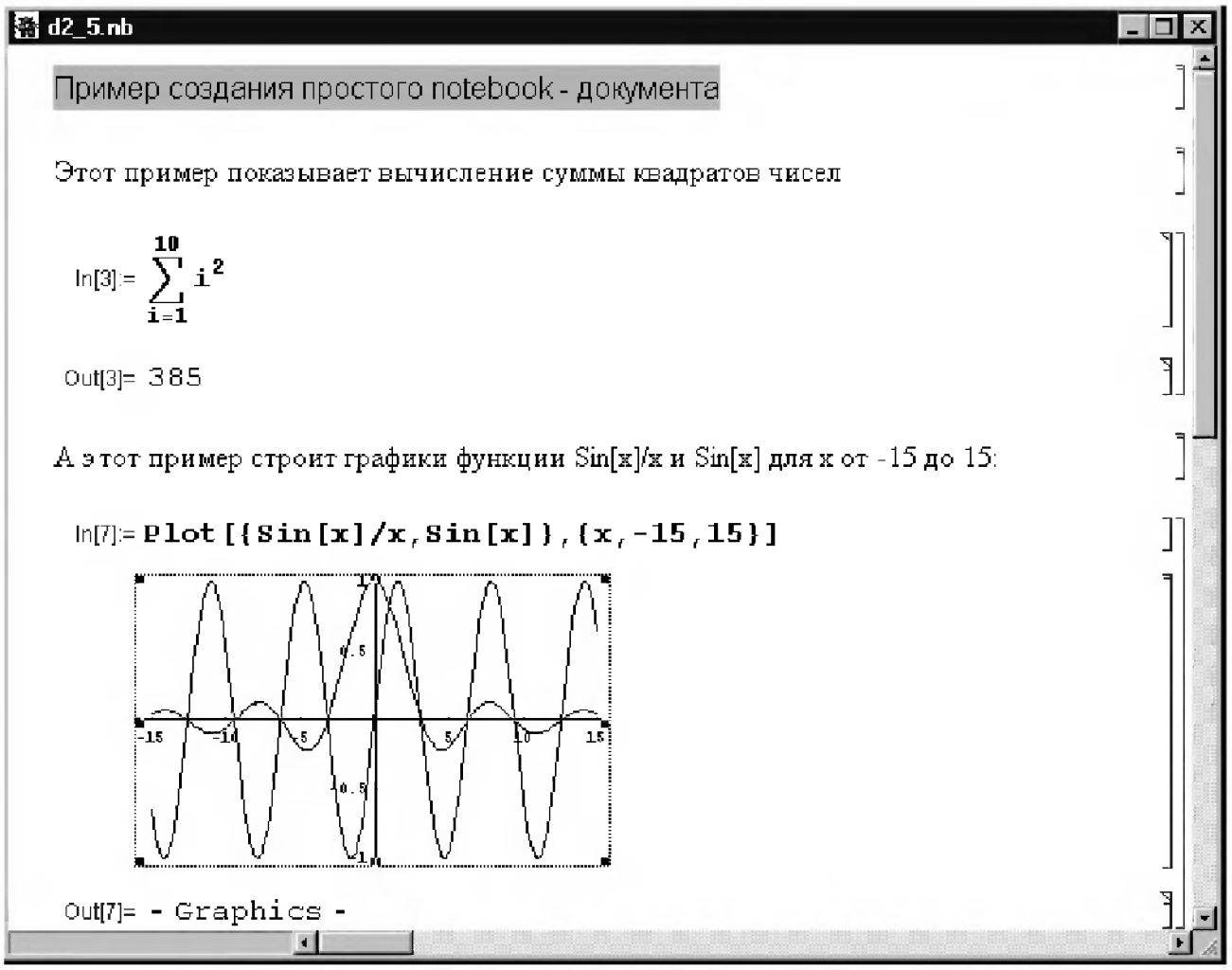


Рис. 1.13. Простой блокнот (ноутбук) системы Mathematica 5

Каждая надпись, математическое выражение или график занимают отдельную ячейку – Cell. Ячейка может занимать одну или несколько строк и всегда выделена своей квадратной скобкой. Важным свойством ячеек систем Mathematica является возможность их эволюции (изменения) по всему документу. Этим осуществляется динамический обмен данными в ходе символьных преобразований.

1.5.6. Пакеты расширения системы Mathematica 5

Часть функций система Mathematica 5 хранит во встроенных пакетах расширения (в оригинале дополнения – AddOns), расположенных в каталоге ADDONS:

- **Algebra** – работа с полиномами, алгебраическими неравенствами, Гамильтоновой алгеброй и др.

- **Calculus** – символьные вычисления производных, интегралов и пределов функций, прямое и обратное преобразования Фурье и Лапласа, решение систем нелинейных уравнений, реализация инвариантных методов, решение дифференциальных уравнений в частных производных, нахождение полных интегралов и дифференциальных инвариантов нелинейных уравнений, аппроксимация Паде, вычисление эллиптических интегралов и работа с векторами.
- **DiscreteMath** – вычисления из области дискретной математики, комбинаторики, вычислительной геометрии и теории графов, решение рекуррентных и разностных уравнений, операции с целыми числами и т. д.
- **Geometry** – функции для выполнения геометрических расчетов, задания правильных прямоугольников и многогранников, вращения геометрических фигур в плоскости и в пространстве.
- **Graphics** – построение графиков специального вида, геометрических фигур и поверхностей, графиков параметрически и неявно заданных функций, представления функций комплексного переменного, отображение ортогональных проекций трехмерных фигур, имитация теней, функции оформления графиков.
- **LinearAlgebra** – решение задач линейной алгебры, дополнительные векторные и матричные операции, задание ортогональных векторных базисов и др.
- **Miscellaneus** – задание единиц измерения физических величин, данные о химических элементах, физические константы, географические данные и все «прочее».
- **NumberTheory** – функции теории чисел.
- **NumericalMath** – реализация важнейших численных методов, аппроксимация данных и аналитических функций полиномами, сплайнами и тригонометрическими рядами, численное интегрирование и дифференцирование, решение дифференциальных уравнений, вычисление корней нелинейных уравнений, нахождение вычетов и разложений в комплексной плоскости и т. д.
- **Statistics** – статистические функции для непрерывных и дискретных распределений, реализация линейной и нелинейной регрессий, вычисление параметров ряда распределений (особенно нормального), функции сглаживания и подгонки данных и т. д.
- **Utilities** – дополнительные утилиты для работы с бинарными файлами, с памятью ПК, поддержки языков, работы с системами класса AutoCAD и т. д.

Они содержат множество библиотечных файлов с расширениями .m., в каждом из которых определен ряд новых функций системы. Число функций в одном пакете расширений лежит в пределах от нескольких функций до нескольких десятков, а общее число дополнительных функций и их вариантов достигает примерно тысячи. С их помощью можно реализовать новые алгоритмы решения математических задач и постоянно расширять возможности системы. Все библиотечные файлы подробно прокомментированы, что облегчает их использование пользователями, владеющими английским языком.

1.5.7. Возможности версии Mathematica 5.1/5.2

Версия Mathematica 5 вплоть до выпуска Mathematica 6 в 2007 г. постоянно модернизировалась. Достаточную известность получили промежуточные версии Mathematica 5.1/5.2.

СКМ Mathematica 5.1 имеет следующие отличительные особенности:

- оптимизированные операции бинарного ввода/вывода;
- встроенный словарь на 90 000 английских слов для проверки орфографии;
- встроенная база данных на основе графического интерфейса пользователя GUI;
- существенно улучшенные алгоритмы линейной алгебры;
- улучшенная поддержка кусочных функций для многих символьных операций;
- новые алгоритмы символьного и численного решений дифференциальных уравнений;
- расширенная поддержка визуализации массивов;
- новые средства выполнения кластерного анализа;
- средства интерактивного решения дифференциальных уравнений;
- поддержка построения графиков большого размера;
- расширенные возможности форматирования графиков;
- поддержка импорта/экспорта XLS- и AVI-файлов;
- новые инструменты на основе GUI;
- интегрированный с системой Web-сервер.

Все эти возможности, безусловно, украшают новую реализацию системы и могут способствовать расширению круга ее пользователей.

В Mathematica 5.2 существенно повышена скорость вычислений многих функций, особенно целочисленных. Обеспечена работа с новыми 64-разрядными и многоядерными (multicore) микропроцессорами. Впрочем, в области символьных вычислений различия в версиях 5.* не существенны и далее не обсуждаются.

1.5.8. Возможности версии Mathematica 6

Вышедшая в 2007 г. новейшая версия Mathematica 6 представляет собой не просто кардинально переработанную систему, а поистине революционный программный продукт. Он справедливо приравнивнивается разработчиками (по влиянию на развитие компьютерной математики) к первым версиям системы Mathematica 1/2. Последние в свое время (конец 80-х гг.) открыли новое научное направление – системы компьютерной математики для персональных компьютеров.

Отметим основные наиболее крупные и значимые нововведения в системе Mathematica 6:

- включение в ядро системы более 1000 новых функций различного рода, операторов и команд интерфейса;

- дальнейшее увеличение скорости вычислений в несколько раз, иногда и больше:
- новая концепция интерактивного динамического интерфейса, в корне меняющая и резко упрощающая создание ноутбуков с новейшими деталями интерфейса (кнопками, переключателями, слайдерами и т. д.), подобными маплетам в системе Maple и окнам GUI в MATLAB;
- новая позиция Graph в меню, позволяющая вводить в документы графические окна и с помощью встроенного графического редактора создавать в них рисунки из графических объектов;
- введение ряда пакетов расширений с сохранением пакетов Add-On, имеющихся в прежних версиях системы;
- поддержка документов, созданных в прежних версиях;
- многочисленные новые функции построения графиков самого различного вида со средствами управления мышью и интерактивными динамическими средствами;
- существенное расширение типов файлов, которые поддерживает система;
- полностью переработанная и легкая в применении справочная система, удобная как начинающим пользователям, так и профессионалам;
- огромное число самых разнообразных примеров буквально на каждую функцию;
- уменьшенное время загрузки системы;
- заметно расширенные средства обращения к обширным интернет-ресурсам.

Меню системы Mathematica 6 (рис. 1.14) серьезно переработано. Можно сразу заметить, что в меню Mathematica 6 исчезла позиция **Find**. Операции поиска перенесены в позицию **Edit** меню, как это сделано в большинстве приложений под операционные системы класса Windows. Исчезла также позиция **Kernel** с командами выбора ядра системы и управления им. Эти команды перекочевали в новую позицию **Evaluation** (Оценка) меню. Появились также новые позиции **Graphics** (вывод окна для создания рисунков с помощью простого графического редактора) и **Palletes** (для вывода палитр с различными математическими символами).

В позиции **Palletes** имеются команды для вызова шести палитр. Хотя число палитр в Mathematica 6 уменьшено, число вводимых ими знаков, операторов и функций даже увеличено за счет улучшения организации палитр. Названия команд в позиции **Palletes** соответствуют названию палитр в их титульных строках.

В окне ноутбука на рис. 1.14 виден новый элемент графического интерфейса пользователя (Graphics User Interface или GUI) – слайдер. Передвигая мышью его движок, можно менять значение переменной, к которой относится слайдер. Подобных элементов GUI (двухкоординатных слайдеров, кнопок, меню и т. д.) Mathematica 6 имеет множество. Они позволяют создавать динамический GUI ноутбуков чрезвычайно простыми средствами. Масса примеров этого приведена в справке системы, кстати, существенно переработанной и ставшей более простой в использовании.

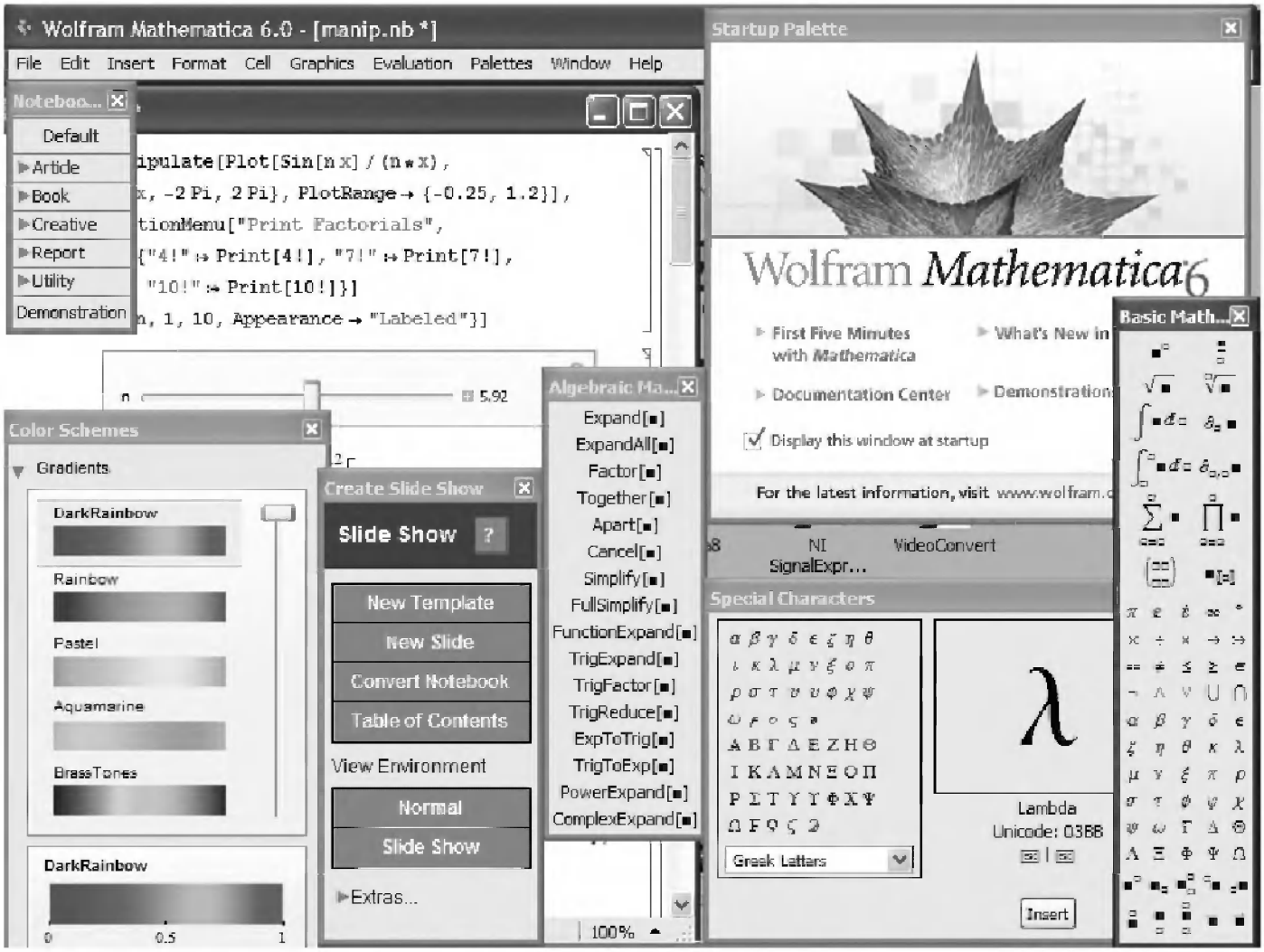


Рис. 1.14. Окна и палитры системы Mathematica 6

1.6. Другие системы компьютерной алгебры

Разумеется, системы компьютерной алгебры представлены не только лидирующими на рынке системами Maple и Mathematica. В системах образования ряда стран нашли применения СКА начального уровня Derive и MuPAD. Особую популярность заслужила система Mathcad с ее превосходным и математически ориентированным пользовательским интерфейсом и визуально-ориентированным программированием. Мощная матричная система MATLAB обзавелась пакетом расширения Symbolic Math Toolbox, использующим ядро символьной математики от системы Maple. Наконец произошло знаменательное событие – СКА стали встраиваться в постоянное запоминающее устройство современных научных микрокалькуляторов [15]. Этой чести удостоилась СКМ Derive.

1.6.1. Системы класса Derive 5/6

Derive (полное название Derive a Mathematical Assistant) – одна из первых систем компьютерной алгебры, реализованная на языке программирования muLISP, ориентированном на решение задач искусственного интеллекта. Благодаря этому первые версии Derive размещались даже на гибких дисках, и именно Derive стала основой операционных систем первых микрокалькуляторов с встроенными в их hardware СКА.

Новые версии Derive 5/6 под Windows лишены ущербности интерфейса пользователя первых версий. У них сняты ограничения в создании документов класса «ноутбуков» и графики, встроенной в документы. Существенно улучшены возможности трафики, в частности появилась возможность построения трехмерных фигур с функциональной окраской, пересекающихся в пространстве. Однако объем системы вырос до 30 Мбайт и выше.

При запуске Derive 6 (как обычно, из меню программ или активизацией ярлыка системы) появляется окно системы, которое представлено на рис. 1.15. Там же даны примеры работы в системе и построения 2D-графика (две функции в одном окне). Работа с Derive (вплоть до версии Derive 5) описана в [90–95].

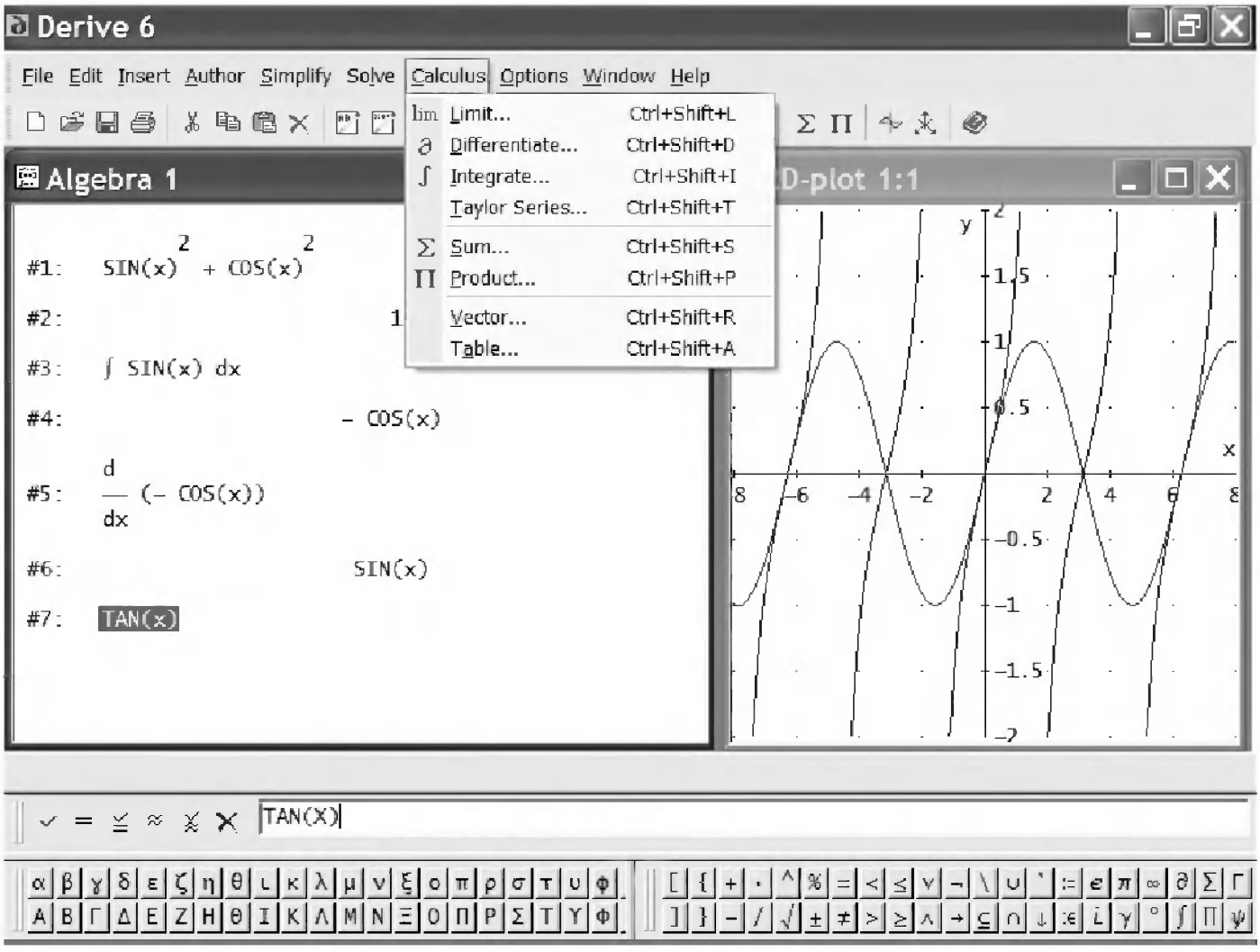


Рис. 1.15. Окно Derive 6 под Windows с элементами интерфейса и примерами работы

Derive имеет не вполне стандартное меню со следующими позициями:

- **File** – стандартные средства работы с файлами;
- **Edit** – стандартные средства редактирования документов и работы с буфером;
- **Insert** – вставка в документ объектов (графических, текстовых и OLE);
- **Author** – создание пользователем выражения, вектора или матрицы;
- **Simplify** – выполнение вычислительных операций и подстановок;
- **Solve** – решение отдельных уравнений и систем уравнений;
- **Calculus** – выполнение символьных операций (вычисление пределов, дифференцирование, интегрирование, разложение в ряд Тейлора, вычисление сумм и произведений последовательностей, создание векторов и таблиц);
- **Declare** – декларация переменных, массивов и функций, установки входа/выхода;
- **Help** – доступ к справке по системе.

Внизу окна расположена, как обычно, строка состояния (статуса) системы. Над ней имеются строка ввода выражений Author и панель ввода специальных символов и основных операторов. Это существенно облегчает ввод выражений и уменьшает необходимость в использовании панели инструментов и меню системы.

К новым возможностям Derive 5/6 относится возможность выделения частей выражений указанием на них курсора мыши и двойным щелчком левой клавиши, а также полноценные функции правой клавиши мыши. Теперь нажатие правой клавиши ведет к появлению контекстно-зависимого меню, в котором присутствуют те команды, которые допустимы для выделенного объекта. Строку ввода можно постоянно держать в окне документа, что упрощает и ускоряет ввод выражений. Доступны все типовые операции с буфером обмена (в том числе и через контекстно-зависимое меню правой клавиши мыши).

Важно отметить, что документы в стиле «ноутбуков» сохраняются при использовании команд **Save** и **Save As...** в позиции **File** меню. Таким образом, Derive 5/6 избавился от такого порока прежних систем, как невозможность сохранения одновременно математических выражений с текстовыми комментариями и графиков.

Следующий шаг в превращении документов в «ноутбуки» заключается в возможности ввода текстовых комментариев различного стиля. Derive 5/6 предусматривает возможность создания текстовых областей без номеров, характерных для областей ввода и вывода. Это и придает документам вид вполне полноценных «ноутбуков» – см. рис. 1.16 с такой надписью сверху.

Рисунок 1.16 заодно демонстрирует и еще одну новинку в новых версиях Derive 5/6 – возможность построения трехмерных графиков с цветной функциональной окраской поверхностей или фигур. При этом подобное построение обеспечивается по умолчанию и существенно повышает наглядность трехмерных графиков.

Говоря о графиках, важно отметить, что Derive 5 позволяет записывать их в файлы различного формата: DIB, JPEG, PCX, TARGA и TIF. Это делается с по-

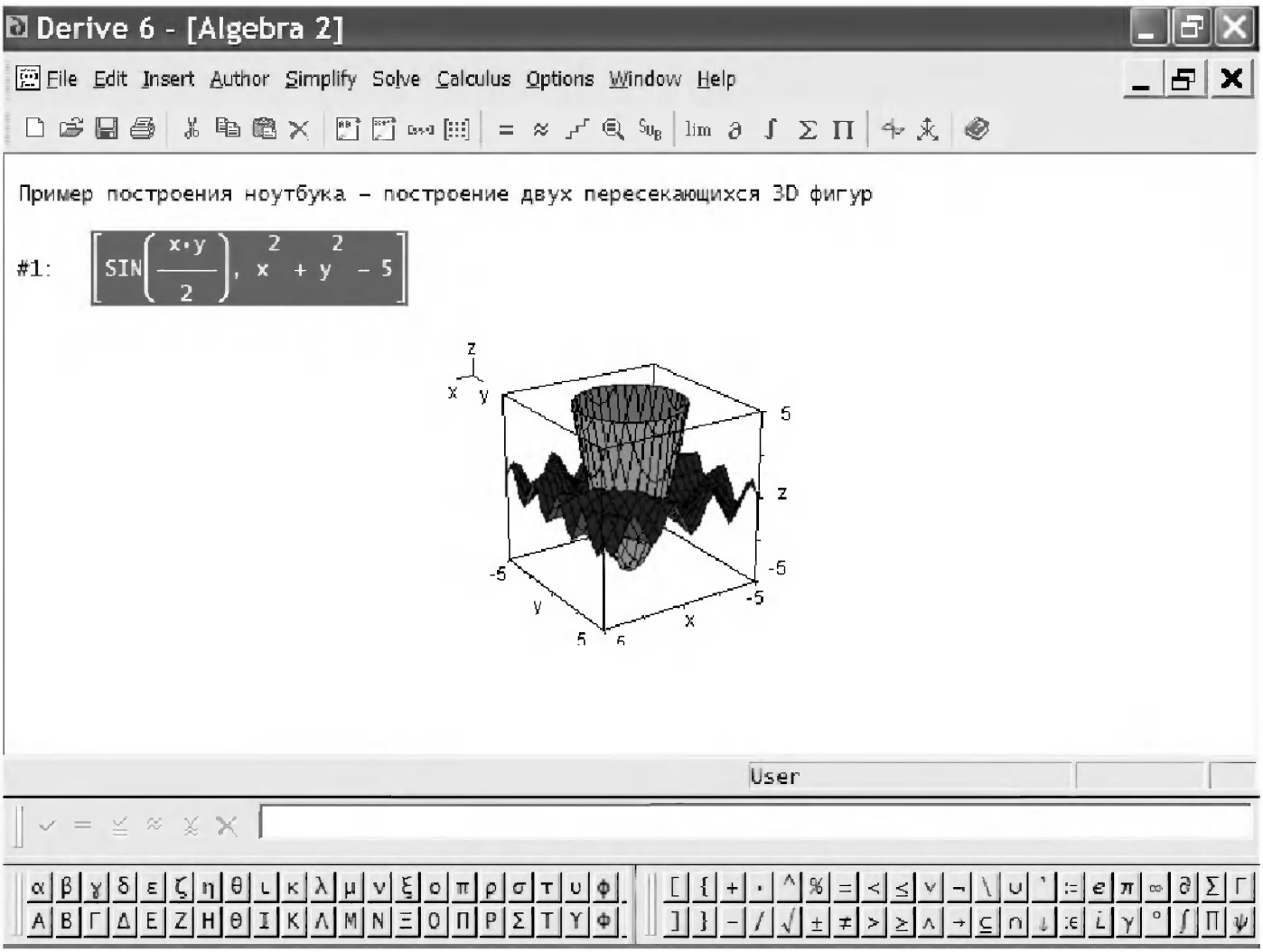


Рис. 1.16. Пример построения документа в стиле «ноутбука»

мощью команды **File Export** меню графики (кстати говоря, меню модифицируется в зависимости от того, какое окно – выражений или графики – активно). Весьма полезной является и опция **Simplify Before Ploting**, которая введена в подменю опций Options меню графического окна. Эта опция при ее введении обеспечивает вычисление выражения перед построением графика.

Еще одна интересная возможность графики Derive 5/6 – построение пересекающихся в пространстве поверхностей или трехмерных фигур. Для этого достаточно задать список выражений для таких фигур в виде функций двух переменных x и y, задающих координаты точек $z(x,y)$ каждой из фигур. Рисунок 1.16 показывает простой пример построения двух поверхностей в пространстве. При этом график можно получить в отдельном окне или поместить в окно алгебраических выражений. Вставка графиков в это окно осуществляется соответствующими командами в позиции **Insert** меню.

Еще один очень любопытный и полезный вид двумерной графики есть у системы Derive – это графики с окраской отдельных их сегментов, задаваемых с помощью неравенств. Пример такого графика приведен на рис. 1.17. Здесь показано построение графика функции $\sin(x)/x$ с закраской областей, ограниченных гра-

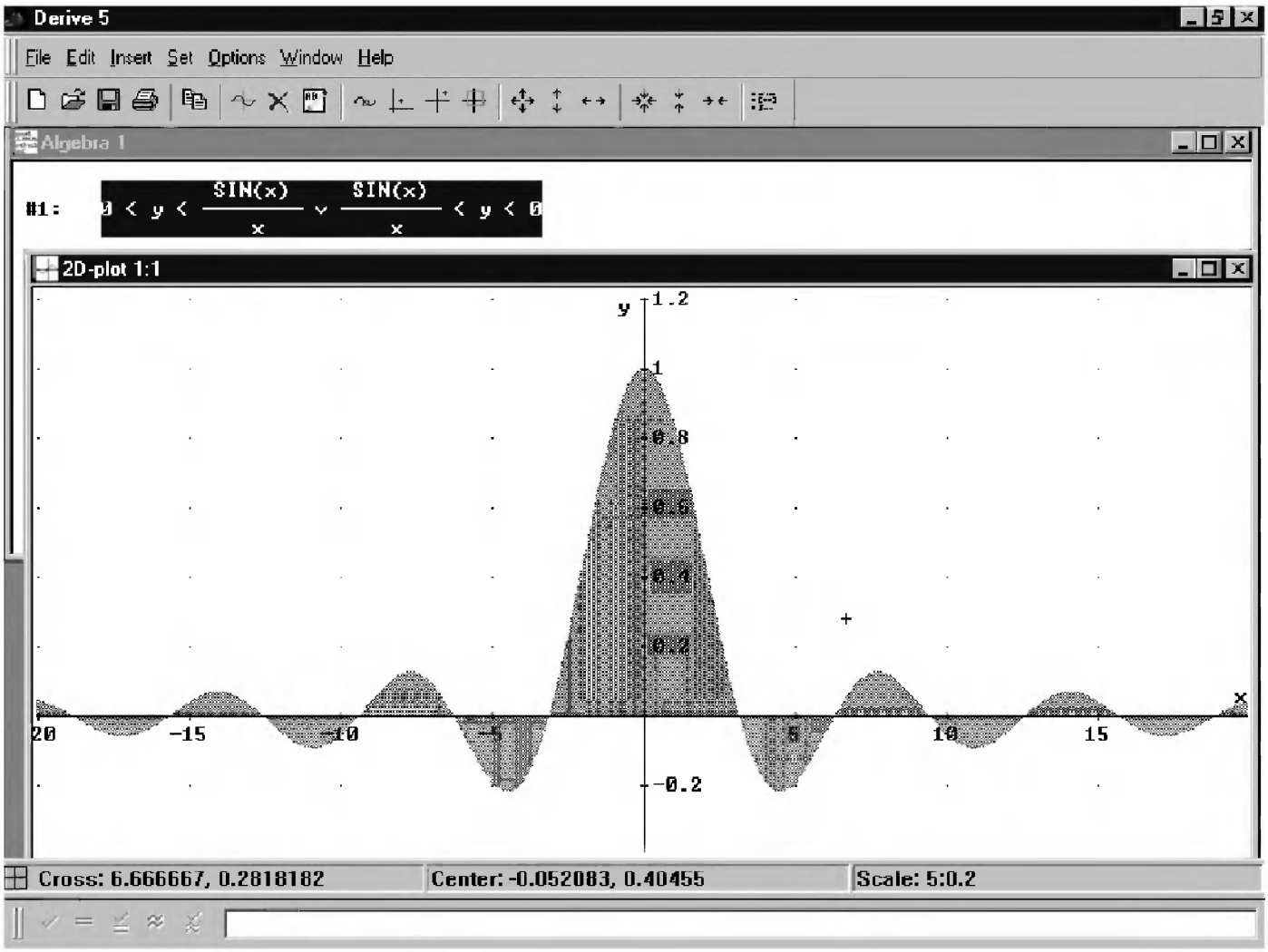


Рис. 1.17. График функции sin(x)/x с разной окраской положительных и отрицательных полуволн

фиком функции и осью абсцисс x. При этом окраска положительных и отрицательных полуволн выполняется разным цветом.

Derive, несмотря на свое учебное назначение, имеет достаточно мощное ядро символьных операций. Возможности системы будут описаны в последующих главах данной книги.

1.6.2. Системы класса MuPAD 2.5/3

MuPAD – относительно новая система компьютерной алгебры, разработанная в Германии под руководством профессора B. Fuchssteiner и распространяемая фирмой SciFice Software. У нас достаточно широкую известность заслужили версии MuPAD 1.41 PRO, MuPAD 2.5* PRO и MuPAD 3* PRO. Новейшая версия MuPAD 4.* установлена в Интернете на сайте этой фирмы с адресом www.sciface.com (Email: bugs@sciface.com). Срок действия версии без оплаты – 30 дней с момента инсталляции. Видимо, этим можно объяснить популярность данной системы в ряде наших школ, колледжей и учебных центров.

Но и не только этим! По сравнению с Derive, пользовательский интерфейс MuPAD выглядит куда более современно и привлекательно. Он позволяет создавать почти полноценные документы в стиле notebooks – см. рис. 1.18. Из этого рисунка можно составить представление о виде пользовательского интерфейса данной системы – он вполне обыденный для приложений Windows 95/98.

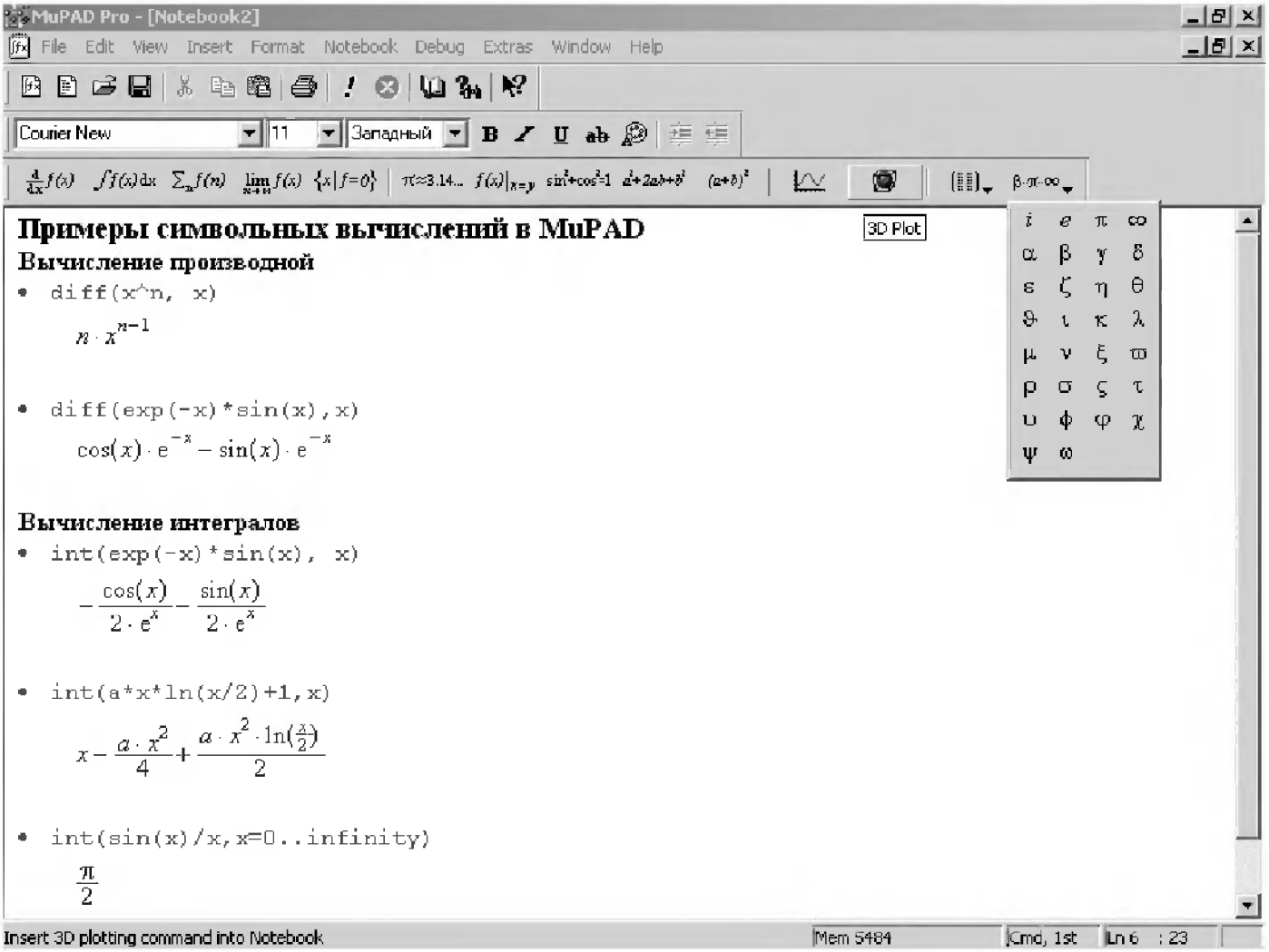


Рис. 1.18. Окно системы MuPAD с примерами вычислений

Для ввода выражений, в том числе с символами производных и интегралов, удобно использовать панель инструментов с математическими символами, расположенную под панелью форматирования. В ее конце есть открывающийся список греческих букв – он виден на рис. 1.18.

Возможности графики у системы MuPAD довольно специфические, поскольку требуют применения множества опций. Это, впрочем, не относится к простейшей двумерной графике. Пример построения графиков трех функций одной переменной представлен на рис. 1.19. График можно выделить одним щелчком левой клавиши мыши, в результате чего он попадает на рамку с метками (черными прямоугольниками), за которые график можно растягивать или сжимать в разных направлениях.

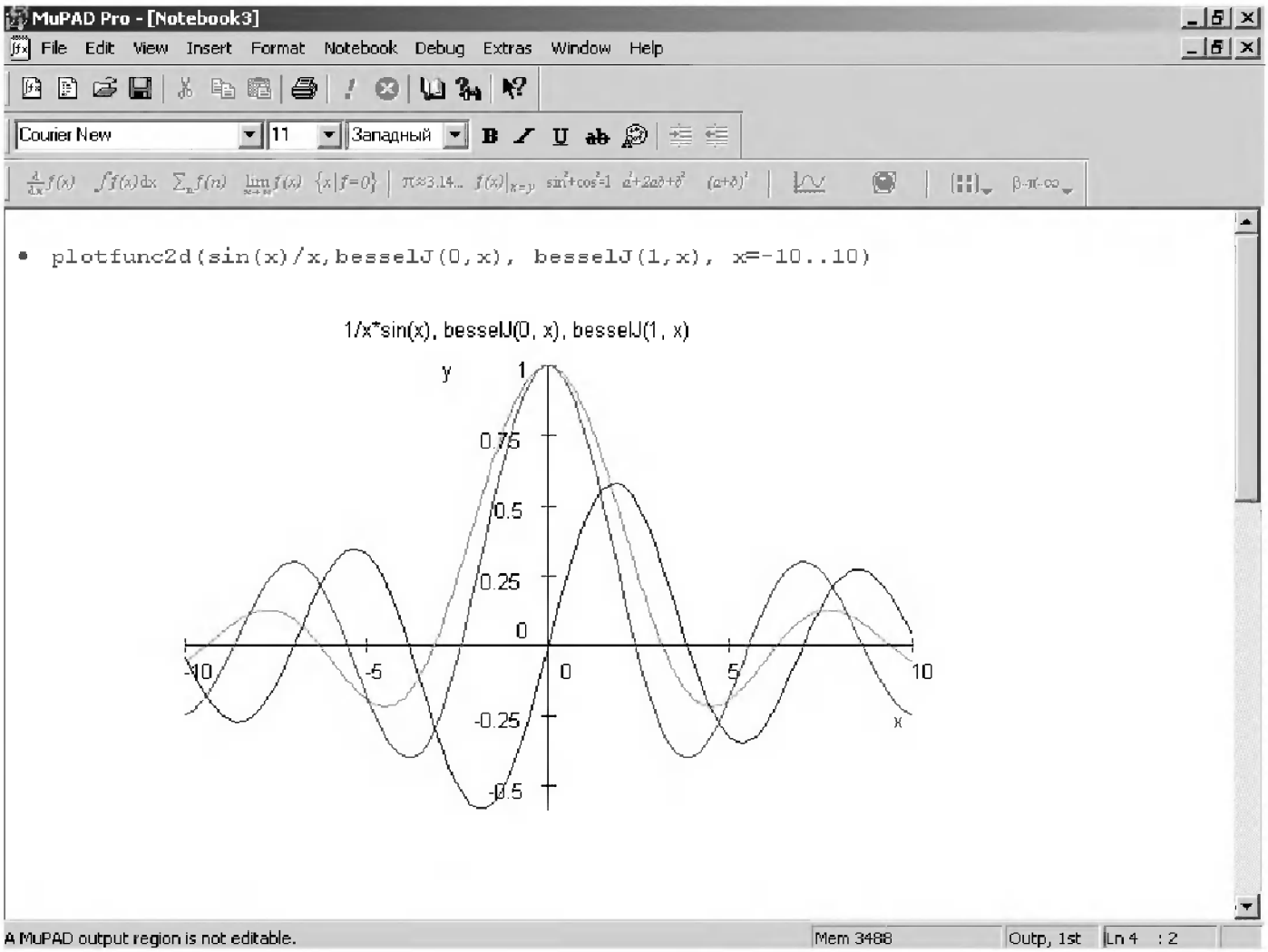


Рис. 1.19. Пример построения трех графиков функций одной переменной системой MuPAD

Трехмерные графики могут быть построены при использовании специальной функции plot3d – рис. 1.20. С помощью различных опций могут устанавливаться различные схемы цветового оформления графика, размеры его, углы поворота вокруг осей, задание перспективного изображения и др. Это сближает графику MuPAD с графикой более мощных систем, хотя до их возможностей графика MuPAD все же явно не дотягивает. Она удобна для обучения основам трехмерной графики, но не слишком порадует тех, кто хотел бы сразу построить трехмерный график, не задумываясь о программировании его деталей.

При активизации 3D-графика двойным нажатием левой клавиши мыши график выделяется характерной жирной рамкой из коротких отрезков прямых, а на экране появляется панель инструментов с рядом наглядных кнопок управления. С их помощью можно выполнить дополнительное форматирование графиков. К сожалению, вращение графика мышью не предусмотрено.

Система MuPAD 3.* по интерфейсу мало отличается от версий MuPAD 2.*. Это хорошо видно из рис. 1.21, где, кстати, показано решение квадратного уравнения с помощью функции solve.

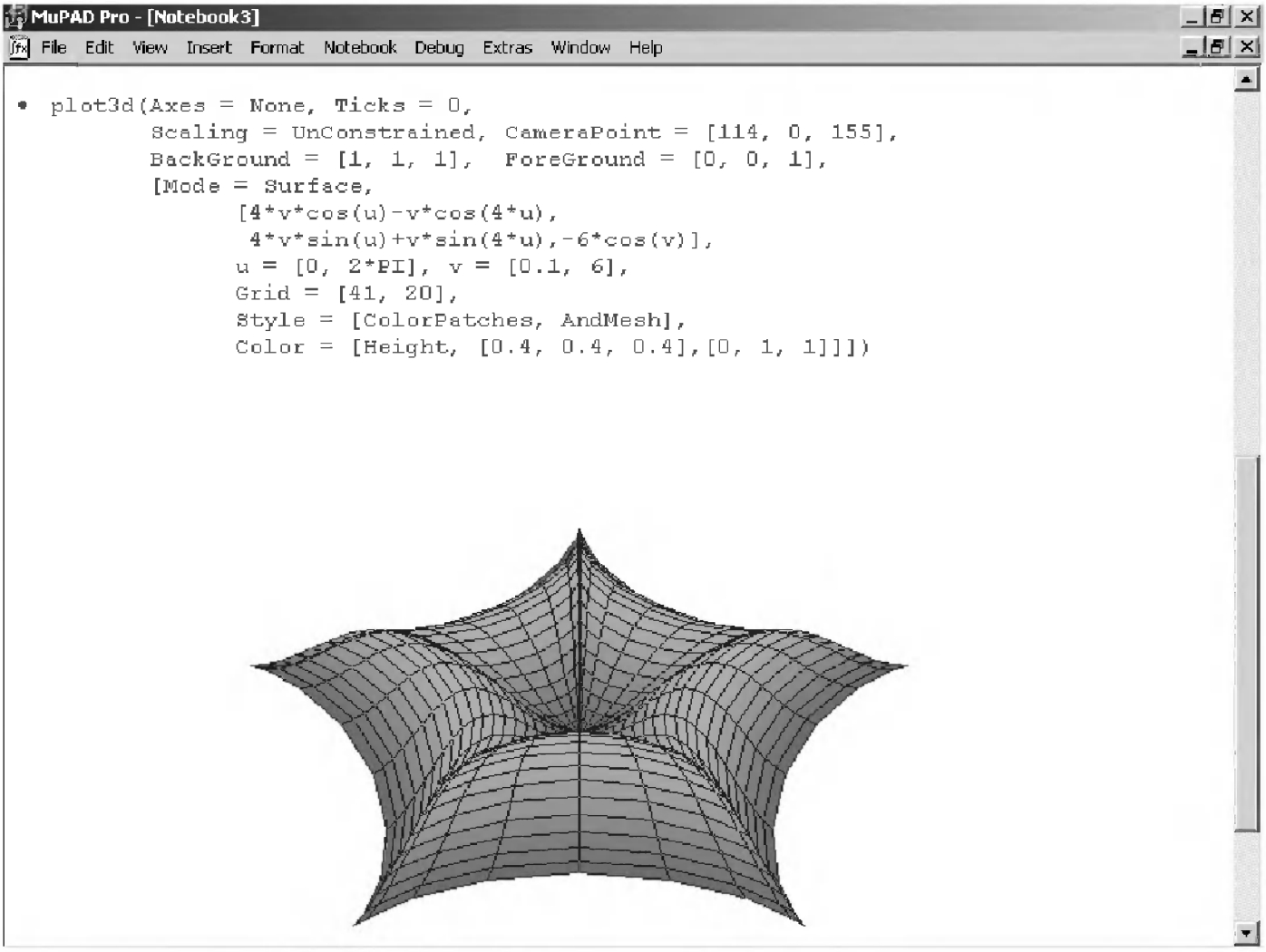


Рис. 1.20. Построение графика 3D-поверхности с рядом опций графика

Как система программирования MuPAD 3.* имеет довольно удобный редактор/отладчик исходных программных кодов. Его применение показано на рис. 1.21. Окно редактора/отладчика исходных программных кодов показано снизу. На рис. 1.22 показано также окно с данными о переменных, которые использованы при создании функции интегрирования `int`.

По математическим возможностям MuPAD, особенно последняя версия MuPAD 4, сопоставима с Derive, но далеко не доходит до таких монстров компьютерной алгебры, как Maple и Mathematica.

1.6.3. Системы класса Mathcad

Mathcad – это популярная система компьютерной математики, предназначенная для автоматизации решения массовых математических задач в самых различных областях науки, техники и образования. Название системы происходит от двух слов – MATHematica (математика) и CAD (Computer Aided Design – системы автоматического проектирования, или САПР). В мире сейчас более 2 млн только легальных пользователей системы Mathcad.

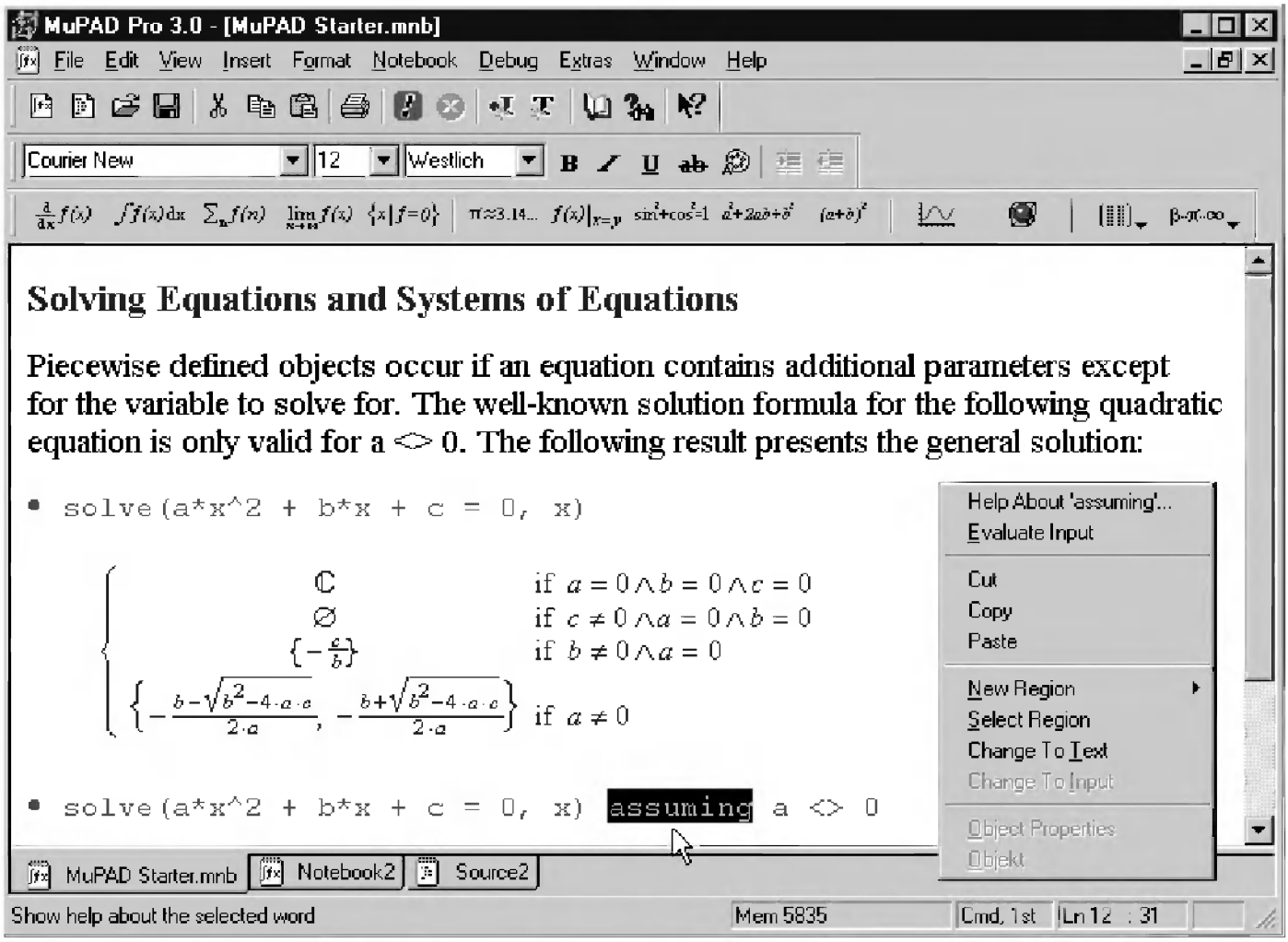


Рис. 1.21. Окно системы MuPAD 3. с примером решения квадратного уравнения*

Первые версии системы Mathcad 1.0/2.0 работали в среде MS-DOS и предназначались лишь для численных расчетов. Но сегодня различные версии Mathcad [37–46] являются математически ориентированными универсальными вычислительными системами. Помимо собственно вычислений, как численных, так и аналитических, они позволяют с блеском решать сложные оформительские задачи, которые с трудом даются популярным текстовым редакторам или электронным таблицам. Применение библиотек и пакетов расширения обеспечивает профессиональную ориентацию Mathcad на любую область науки, техники и образования.

Среди других систем компьютерной математики, таких как Maple, Mathematica и MATLAB, система Mathcad по-прежнему выделяется своим дружеским по отношению к пользователю интерфейсом (рис. 1.23), а также удобным и чрезвычайно простым в применении математически и визуально ориентированным языком общения с пользователем. Именно это объясняет популярность системы. Число ее только легальных пользователей в мире превысило 2 млн.

Как видно по рис. 1.23, Mathcad имеет множество палитр с математическими символами, операторами и функциями. Их вывод в окно документа (рис. 1.24, например), как и изменение интерфейса, осуществляется из позиции **View** меню. Палитр так много, что в окне не остается места для объектов документа. Поэтому

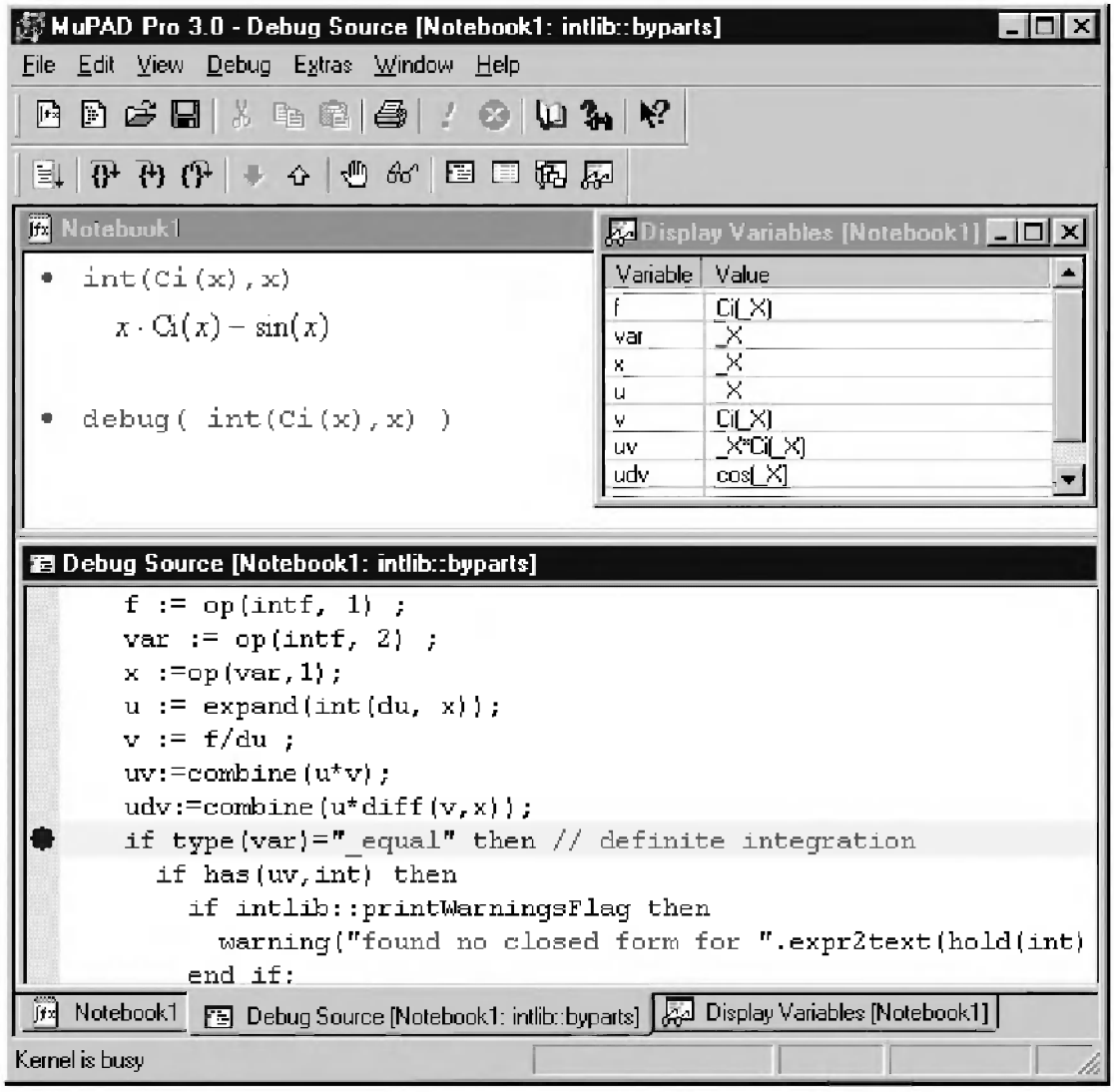


Рис. 1.22. Работа с MuPAD 3. в режиме отладки функции интегрирования*

обычно выводятся только нужные палитры, а ненужные убираются активизацией кнопки со знаком «×» в правом верхнем углу каждой палитры. Естественно, что любой объект (символ, число, оператор, функцию и т. д.) можно вводить с помощью клавиатуры, панели инструментов и позиции меню **Insert**. Для форматирования объектов служат позиция меню **Format** и панель форматирования.

Как нетрудно заметить, для вычислений достаточно набрать нужное выражение (с клавиатуры или с помощью палитр) и поставить за выражением оператор вывода – знак «=». Сразу появится результат операции. Mathcad имеет ряд расширенных операторов для вычисления сумм, произведений, производных, интегралов и т. д. Задание выражений осуществляется в математической нотации, привычной всем, кто привык выполнять математические вычисления. На рис. 1.24 показаны также задание простой функции пользователя и построение ее графика. Графики строятся в окне документа, их можно перемещать, растягивать и форматировать.

Построение графиков в версии Mathcad делается удивительно просто. Для построения графиков функций достаточно ввести выражение для функции и шаблон графика – график будет тут же построен. Основные типы графиков системы

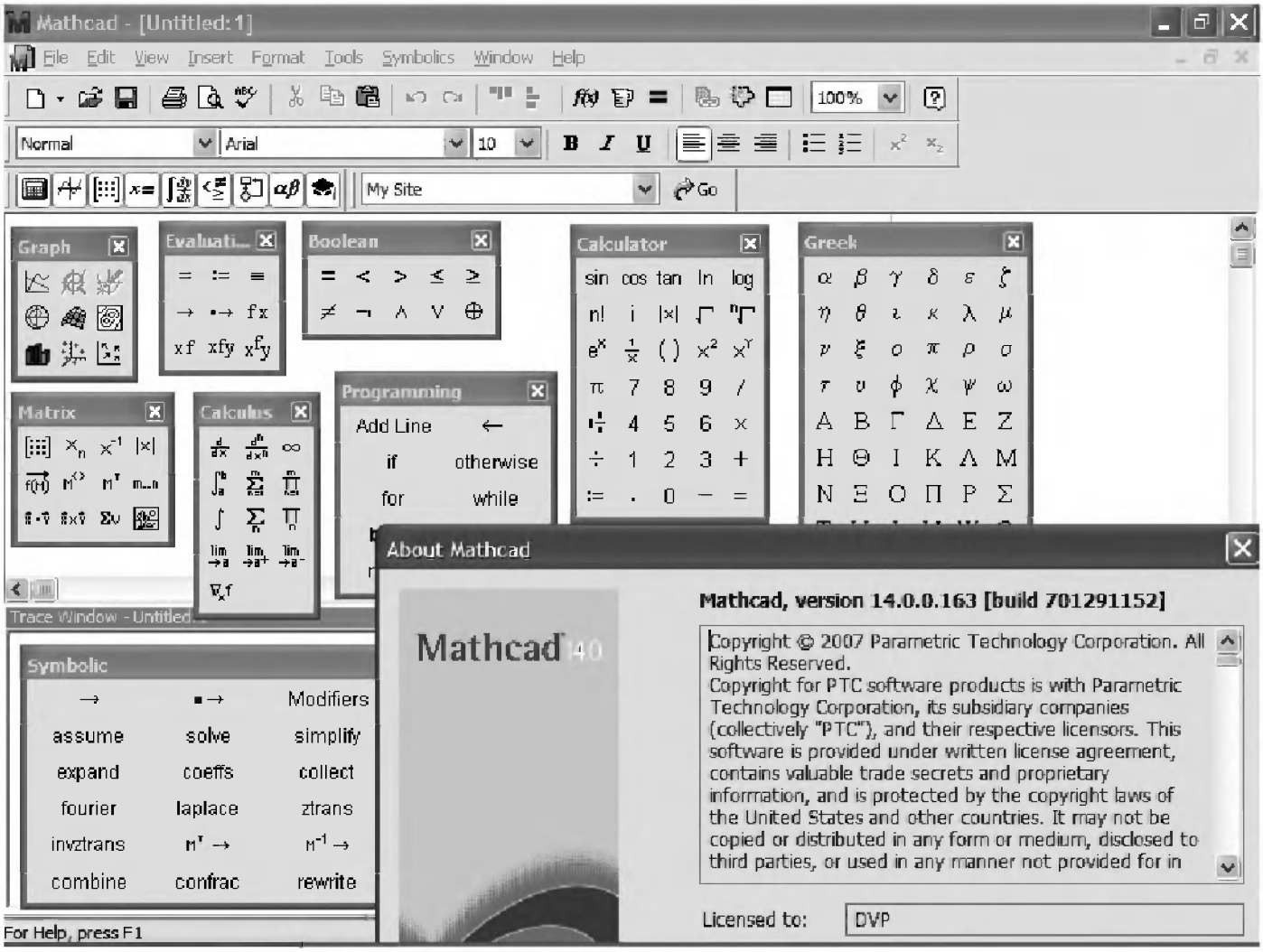


Рис. 1.23. Окно Mathcad 14 со всеми палитрами математических знаков и операций

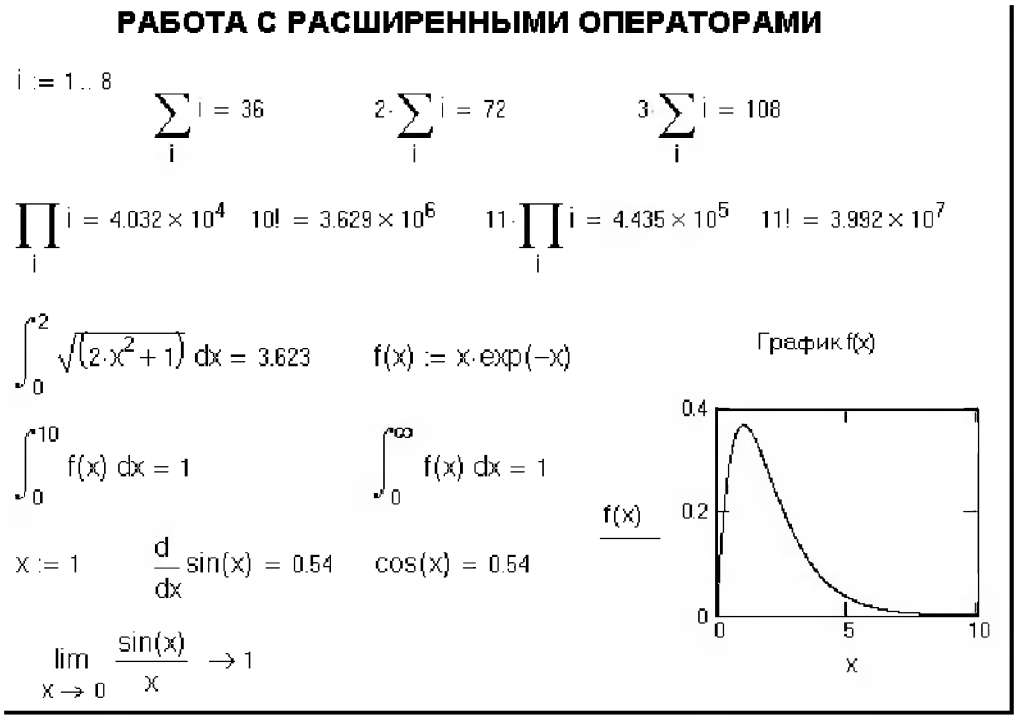


Рис. 1.24. Примеры вычислений в окне системы Mathcad (детали интерфейса обрезаны)

Mathcad представлены на рис. 1.25. Среди них показаны и трехмерные графики для функции пользователя $f(x,y)$, заданной в верхнем правом углу окна.

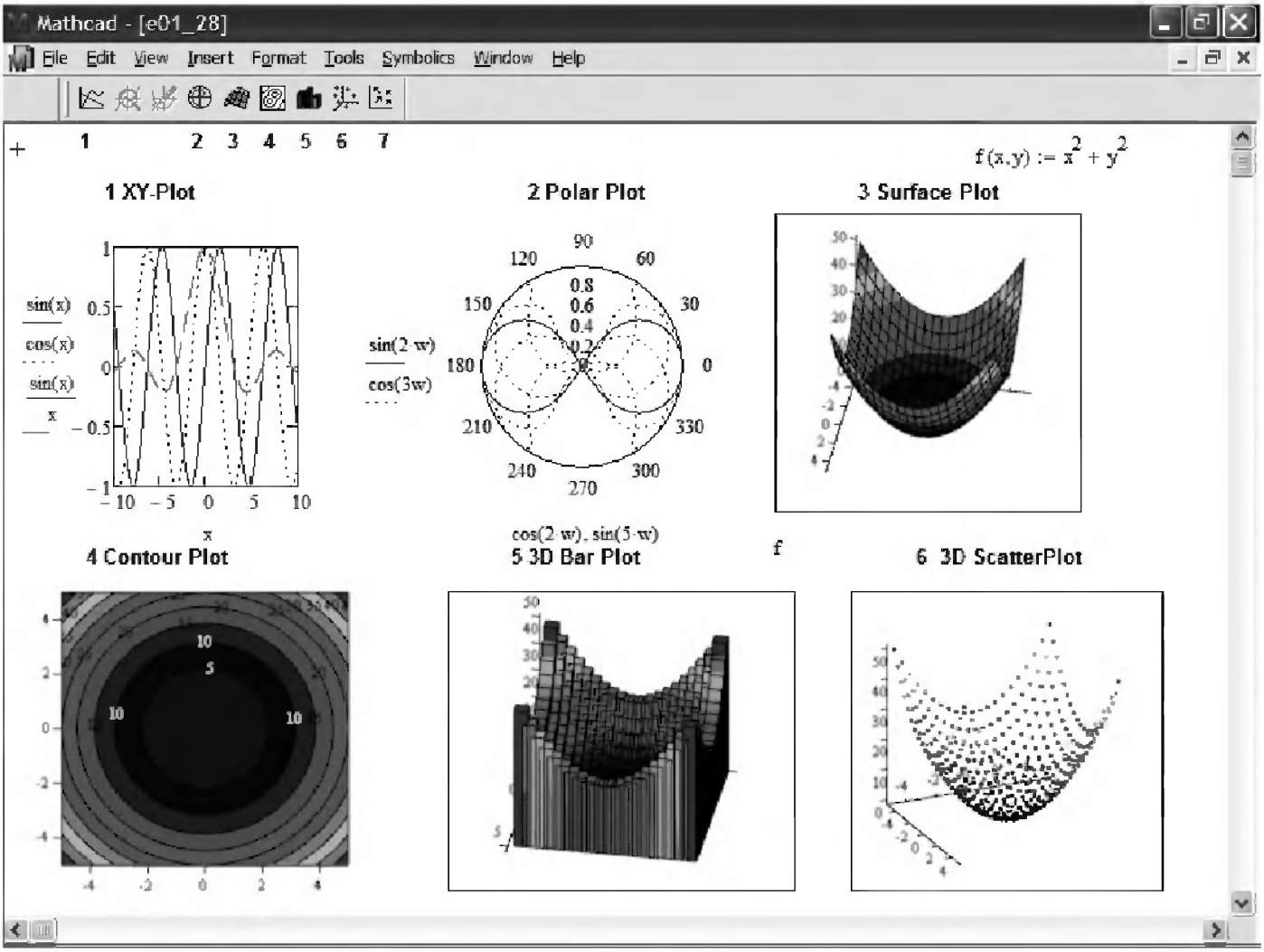


Рис. 1.25. Основные типы графиков системы Mathcad

Интересной возможностью новых версий Mathcad стало построение на одном рисунке нескольких трехмерных объектов, пересекающихся в пространстве и имеющих различные типы графиков, например в виде поверхности и контурного графика. Пример такого построения дан на рис. 1.26.

Для форматирования графиков Mathcad имеет окно форматирования. Для его вызова достаточно поместить курсор мыши в область графика и дважды щелкнуть левой клавишей мыши. На рис. 1.27 показано окно форматирования трехмерного графика. Оно имеет множество вкладок, с помощью которых можно задать вид графика, условия освещения, проявление тех или иных световых эффектов, задание координатных осей того или иного типа и т. д. Это эквивалентно заданию опций графических функций в других системах. Они сохраняются при записи документов с графиками на диск.

Mathcad 12/13/14 позволяет создавать двумерные графики с двумя вертикальными осями, причем масштаб их может быть разным. Это небольшое удоб-

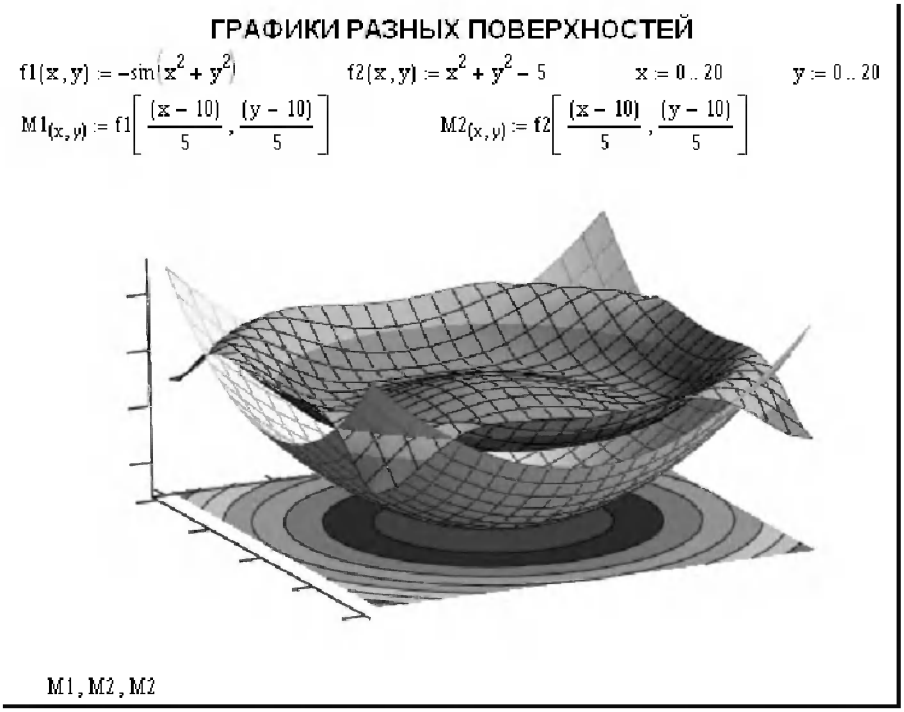


Рис. 1.26. Графики поверхностей разного типа, пересекающиеся в пространстве

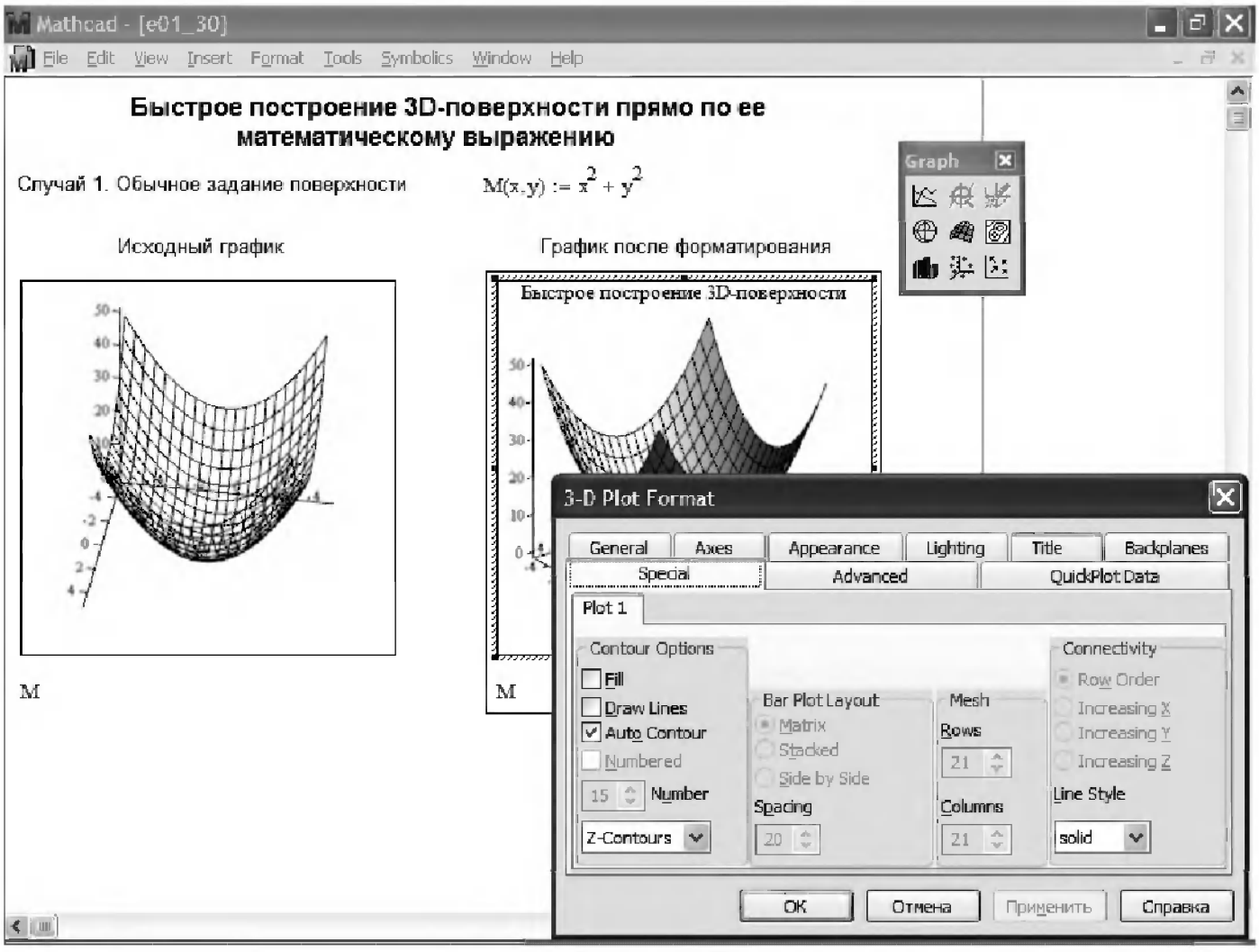


Рис. 1.27. Пример форматирования трехмерного графика

ство принципиального значения, разумеется, не имеет. Более важным отличием является поддержка работы одновременно нескольких потоков. Например, редактирование блоков документов теперь возможно, даже если одновременно происходит выполнение других блоков. Расширены возможности комментирования файлов и документов.

Для выполнения символьных операций в Mathcad (кроме последней версии Mathcad 14) использует несколько урезанное ядро системы Maple. Поэтому результаты символьных операций подобны таковым для системы Maple – см. примеры на рис. 1.28. В связи с этим детально описывать их не имеет смысла – в данной книге эти результаты даны для тех или иных версий системы Maple. Mathcad дает доступ примерно к трем десяткам символьных операций из более чем 4000 операций в последних системах Maple. Однако для промежуточных операций используются почти все операции ядра Maple. К сожалению, многие результаты в виде специальных математических функций в Mathcad выводятся просто как комментарии, и в дальнейшем использовать их для символьных вычислений нельзя.

Примеры символьных вычислений с оператором символьного вывода → представлены на рис. 1.29. Оператор символьного вывода позволяет работать не толь-

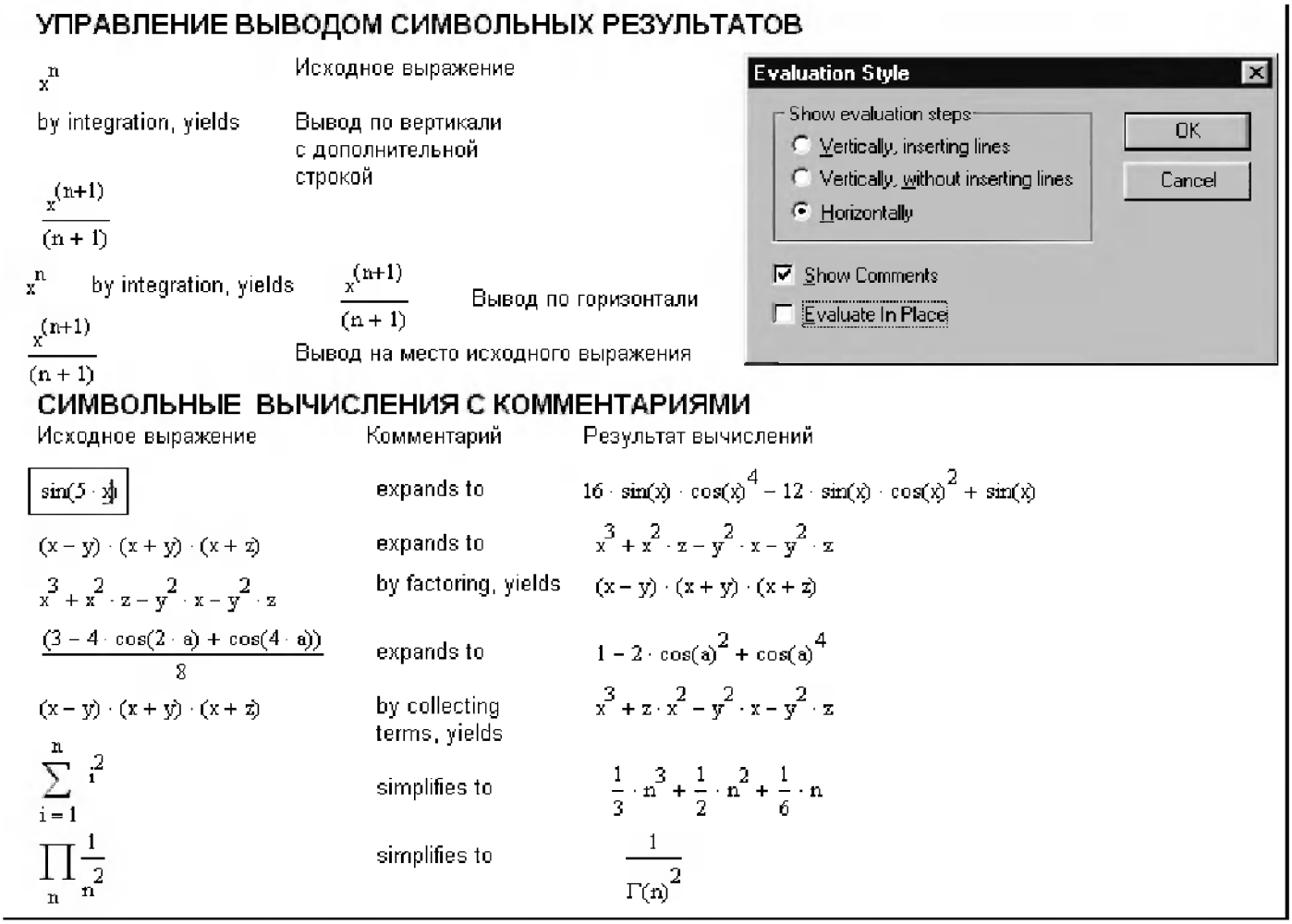


Рис. 1.28. Примеры выполнения символьных операций в командном режиме с выводом комментариев

ко с явно выраженными формулами, но и с выражениями, содержащими функции пользователя. Для выбора функций символьных операций могут использоваться палитры этих операций, показанные на рис. 1.29 справа.

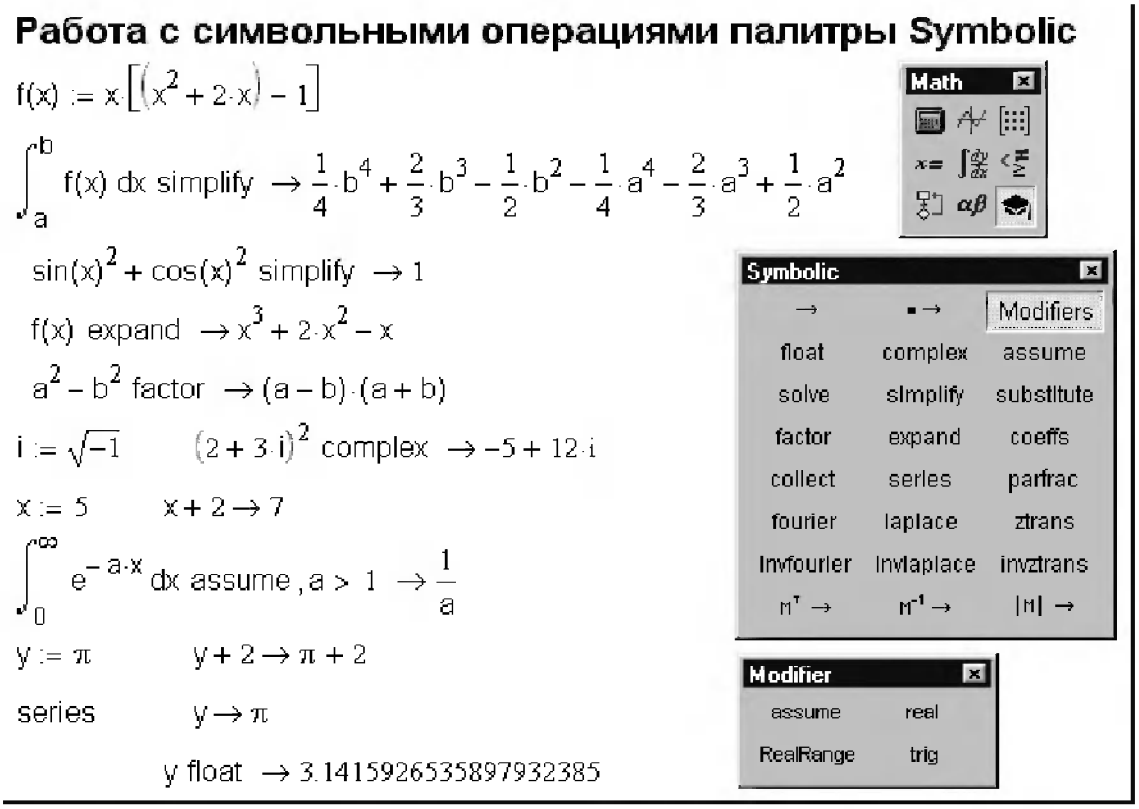


Рис. 1.29. Примеры выполнения символьных операций с помощью оператора символьного вывода

Среди части пользователей Mathcad распространено применение нерегламентированных возможностей системы Mathacd. Для этого лучшей оказалась версия Mathcad 11, в которой есть ряд недокументированных возможностей использования ядра Maple. Они заметно сократились в Mathcad 12/13, а в Mathcad 14 ядро Maple было заменено ядром системы MuPAD. Обнаружить изменения результатов символьных операций, вызванных столь радикальным решением, к счастью, не удалось. Но потенциально смена ядра грозит несовместимостью документов, содержащих символьные вычисления, выполненных в разных версиях Mathcad.

1.6.4. Система MATLAB с пакетом расширения Symbolic Math Toolbox

В последние годы на роль ведущей СКМ, ориентированной на реализацию численных методов и математического моделирования различных систем и устройств, уверенно выдвинулась матричная система MATLAB (ее название происходит от английских слов MATrix LABoratory) [98–118]. Данные в системе (даже одиночные числа) представлены как матрицы и векторы, и для их обработки ис-

пользуется весь арсенал матричных методов, накопленный за десятки лет развития этой системы и сотни лет развития математики.

Ныне MATLAB – это ориентированные на научно-технические расчеты мощный язык программирования. Это расширяемый язык – на нем можно строить различные определения, которые сохраняются в виде M-файлов (с расширением .m) и могут применяться без какого-либо объявления.

Но что особенно характерно для MATLAB (на момент написания книги система была представлена версией MATLAB R2008a), так это множество пакетов ее расширения (packages), входящих в состав инструментальных ящиков Toolbox и Blockset, относящихся к собственно системе MATLAB и ее главному пакету расширения Simulink. Этот пакет ориентирован на блочное имитационное моделирование различных систем и устройств.

В настоящее время в состав MATLAB входит до сотни пакетов расширения. Еще больше пакетов расширения создано внешними разработчиками. Документация по системе многократно превосходит по объему данную энциклопедию. В разработке пакетов расширения принимают участие многие научные школы, коллективы и отдельные научно-педагогические работники. Объем системы на жестком диске ПК при полной установке достигает 2 Гбайт и выше.

Несмотря на все это, для данной книги, посвященной в основном символьным вычислениям, система MATLAB представляет ограниченный интерес, поскольку в своей основе является численной системой. Однако разработчики не могли не учесть потребностей в символьных вычислениях даже при работе с этой системой. В связи с этим был создан пакет расширения для реализации символьных вычислений Symbolic Math Toolbox, который был основан на ядре символьных вычислений системы Maple. Пакет дает доступ к сотне функций этого ядра. Это заметно больше, чем у системы Mathcad, но по-прежнему составляет лишь малую часть от полного числа функций системы Maple, которое перевалило за 3000.

Очевидно, что при таком подходе от MATLAB (как и Mathcad) невозможно получить результаты символьных операций, хоть в чем-то отличные от получаемых системой Maple (или MuPAD в Mathcad 14). Более того, ввод математических выражений и вывод результатов в Symbolic Math Toolbox осуществляются в примитивной текстовой форме. В связи с этим описание системы MATLAB и ее пакетов расширения из данной энциклопедии исключено.

1.7. Компьютерная математика в аппаратных средствах

1.7.1. Научные микрокалькуляторы со встроенными СКА

В последнее десятилетие произошло важное событие, последствия которого большинство ученых еще не ощущает в полной мере, – появились миниатюрные микрокалькуляторы с встроенными СКА. В конце 80-х гг. крупная корпорация Texas

Instruments, Inc. (США) создала первые научные графические микрокалькуляторы [15]. Уже к 2000 г. выпуск их превысил 20 млн экземпляров!

В некоторые модели калькуляторов стали встраиваться возможности выполнения отдельных символьных вычислений – например, решения в аналитическом виде квадратных уравнений, простые символьные матричные операции и т. д. Их можно найти в научных графических калькуляторах фирмы Hewlett Packard HP48 и HP49. Однако возможности компьютерной алгебры у этих калькуляторов носят рудиментарный характер и далеки от возможностей даже простых СКА, таких как Derive и MuPAD.

Среди научных графических микрокалькуляторов фирмы Texas Instruments несколько моделей имели встроенную СКА класса Derive [92]. Выбор этой системы был обусловлен поразительной компактностью ее кодов, созданных на основе языка искусственного интеллекта muLISP. Texas Instrument сделала разработчика Derive – фирму SoftWarehouse, Inc. (США) – своим подразделением и поручила ему доработать Derive под применение в микрокалькуляторах. Что и было успешно выполнено. Таким образом, СКА впервые появились в виде hardware («твердого» программного обеспечения).

Первый микрокалькулятор такого типа TI-89 выполнен в стиле классического калькулятора – рис. 1.30. Экран его жидкокристаллического дисплея, с разрешением 160 × 100 точек (пикселей), расположен у верхней короткой стороны корпуса. Расположение клавиш характерно для микрокалькуляторов – оно удобно для ввода чисел и арифметических операторов, но неудобно для ввода текстов. В частности, клавиши с буквами расположены в алфавитном порядке.

Рис. 1.30. Научные графические микрокалькуляторы TI-89 с встроенной СКА

Калькулятор TI-92/92 Plus выполнен в стиле микрокомпьютера – рис. 1.31. У него QWERTY-клавиатура, дисплей повышенного разрешения (240 × 128 пикселей) и размера и даже вполне современный встроенный графический манипулятор с 8 направлениями перемещения графического маркера.

Texas Instruments не ограничилась выпуском этих калькуляторов. На базе TI-89 был создан современный калькулятор TI-89 Titanium, а на базе TI-92 Plus недавно был выпущен самый мощный калькулятор этого типа VOYGE 200 – рис. 1.32. Он имеет модерновый и эргономический дизайн. Дисплей калькулятора имеет разрешение 128 × 240 пикселей, что позволяет отображать не только алфавитно-цифровую информацию, но и графику и окна. Около 188 Кбайт памяти отведены пользователю. Кроме того, калькулятор имеет флэш-память объемом около 2,7 Мбайта, что втрое больше, чем у TI-92 Plus.

Операционная система калькуляторов может обновляться через Интернет. Калькуляторы могут объединяться в сеть, для них выпущены многочисленные дополнительные внешние устройства, например интерфейсы связи с ПК и даже

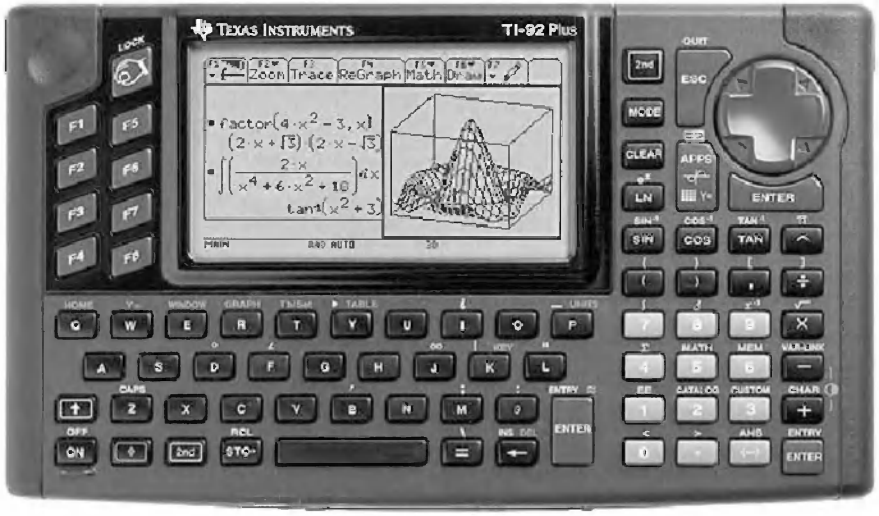


Рис. 1.31. Внешний вид калькулятора TI-92 Plus

Рис. 1.32. Внешний вид калькулятора VOYGE 200

простые видеопроекторы. Специальное математическое программное обеспечение калькуляторов обеспечивает выполнение как обычных, матричных и статистических вычислений, так и символьных вычислений с их графической визуализацией (более 100 графических функций). Обеспечивается интерактивный анализ функций (кстати, пока отсутствующий у СКА): вычисление значений функций в заданной точке, поиск корней, минимумов и максимумов, точек пересечений, длины дуг и т. д. Реализованы многочисленные функции статистических вычислений, интерактивный ввод и преобразования матриц, решение задач линейной алгебры и математического анализа.

В калькулятор VOYGE 200 встроено следующее дополнительное программное обеспечение (пакеты расширения):

- Cabri Geometry™ – пакет геометрических вычислений и построений;
- CellSheet™ – табличный процессор;
- Finance – пакет финансовых вычислений;
- The Geometer's Sketchpad® – дополнительный пакет по геометрии;

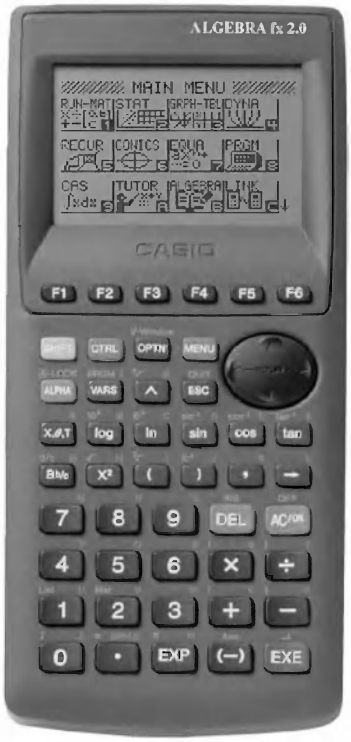


Рис. 1.33. Внешний вид калькуляторов ALGEBRA fx-20 фирмы Casio

- Polynomial Root Finder – пакет для работы с полиномами и поиска корней полиномов;
- Simultaneous Equation Solver – решатель уравнений;
- Statistics with List Editor – пакет статистических вычислений с табличным вводом;
- StudyCards – пакет для работы с картами;
- Language Localization (French, German and Spanish) – пакет локализации изначально англоязычного программного обеспечения.

Калькуляторы с СКА выпускает также фирма Casio. Так, она выпускает калькулятор GFX-9850GB Plus фирмы. Отличительная черта этого калькулятора – трехцветный дисплей с меню, содержащим пиктограммы. Возможности символьных вычислений довольно скромные, но они есть. Большими возможностями в решении задач символьной математики обладают калькуляторы ALGEBRA fx-20 фирмы Casio – рис. 1.33. В них встроена простая СКА, позволяющая выполнять многие виды символьных вычислений, но уступающая все же СКА калькуляторов фирмы Texas Instruments.

1.7.2. Примеры работы с научными микрокалькуляторами со встроенными СКА

Калькуляторы фирмы Texas Instruments способны в командном режиме выполнять практически все встречающиеся на практике простые и умеренно сложные вычисления. Для примера на рис. 1.34 представлено вычисление неопределенных интегралов в символьном виде. Решение в символьном виде дифференциальных уравнений показано на рис. 1.35.

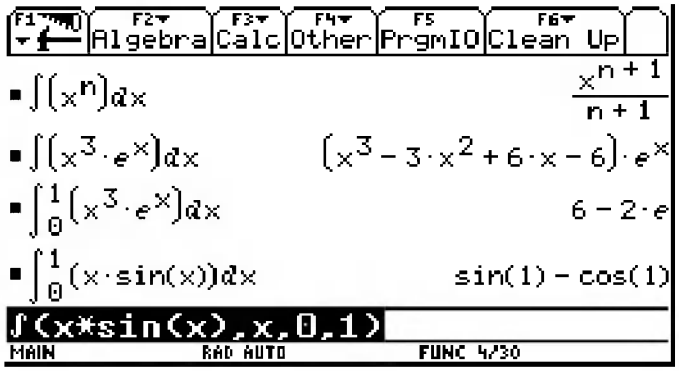


Рис. 1.34. Вычисление неопределенных интегралов в символьном виде (копия экрана)

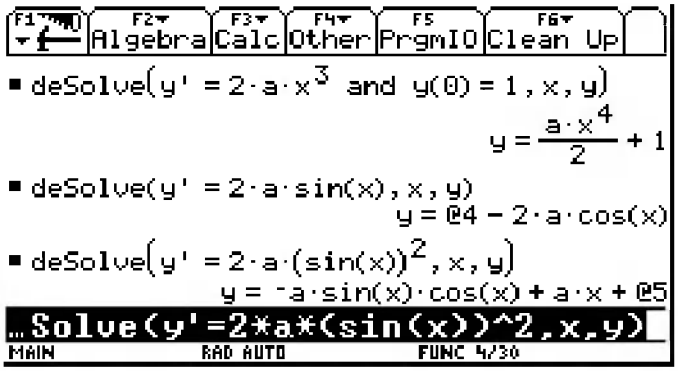


Рис. 1.35. Примеры решения дифференциальных уравнений второго порядка в аналитическом виде (копия экрана)

Калькуляторы имеют редакторы формул, текстовых комментариев, векторов, матриц и таблиц. Они могут выполнять самые распространенные статистические расчеты, проводить регрессию и вычислять множество статистических параметров для больших массивов данных, задаваемых в форме электронных таблиц. Большую часть таких вычислений они выполняют в «ручном» режиме, но имеют весьма мощные средства программирования на мощной и современной версии языка BASIC и большой объем памяти для хранения множества программ и даже библиотек программ, которые можно заполучить из Интернета.

Калькуляторы позволяют строить графики различных функций в декартовой и полярной системах координат, а также графики трехмерных поверхностей и фигур различного типа (рис. 1.36).

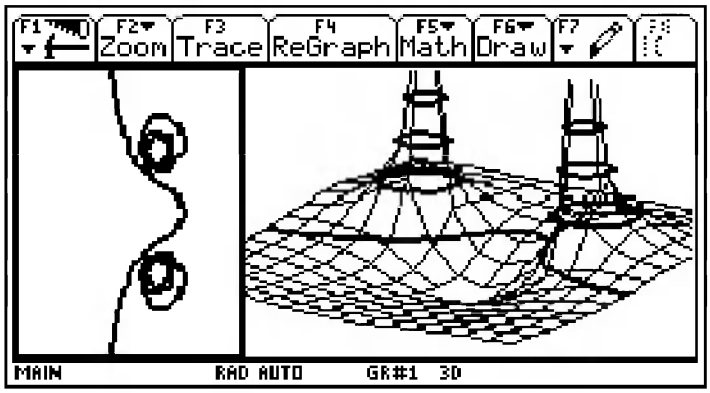


Рис. 1.36. Пример графической визуализации решения в виде контурного и трехмерного графиков (копия экрана)

Кажется почти невероятным, но они обеспечивают даже анимацию графиков и интерактивное управление углами обзора 3D-фигур путем их вращения в разные стороны с помощью графического манипулятора. Возможно разбиение экрана на два окна с выводом в каждое окно любой информации. Управление калькуляторами происходит с помощью многочисленных выпадающих меню и встроенного графического манипулятора. Наиболее часто используемые меню вызываются нажатием функциональных клавиш от F1 до F10. Широко используются и привычные для пользователей ПК окна (конечно, более скромные, чем в Windows, но очень похожие по сути).

1.7.3. Компьютерная математика в цифровых мультиметрах

В XXI в. происходит знаменательное событие – внедрение средств компьютерной математики в современные цифровые измерительные приборы [190–196]. Оно идет по двум направлениям:

- введение специализированных средств компьютерной математики непосредственно в приборы;
- использование приборов совместно с СКМ и СКА, имеющимися на рынке.

Одним из первых применений средств компьютерной математики стало измерение параметров переменного периодического напряжения произвольной формы u(t), например:

- среднего значения (постоянной составляющей) $U_0 = \frac{1}{T}\int_0^T u(t)dt$;
- средневыпрямленного значения $U_{СР} = \frac{1}{T}\int_0^T |u(t)|dt$;
- эффективного, или действующего, значения $U_Д = \sqrt{\frac{1}{T}\int_0^T u^2(t)dt}$.

Эти параметры, требующие вычисления интегралов, измеряются многими современными мультиметрами, имеющими лейб True RMS, указывающий на правильное измерение среднеквадратического значения напряжения (или тока) – см. рис. 1.37. Выпускаются специальные интегральные микросхемы для измерителей среднеквадратического значения напряжений или токов.

Рис. 1.37. Мультиметр M-3890DT с режимом True RMS и USB-портом для подключения к ПК (сверху)

Элементы математической обработки оцифрованных сигналов введены во многие классы измерительных приборов, например в измерители имитанса и импеданса, в цифровые осциллографы и анализаторы спектров.

1.7.4. СКМ в современных цифровых осциллографах

Массовое применение средства компьютерной математики нашли в современных цифровых осциллографах, выпускаемых многими фирмами (Tektronix, LeCroy, Agilent Trchnologies и др.). Мы рассмотрим это на примере осциллографов компании Tectronix, которые занимают более 50% мирового рынка этих приборов [190, 191].

Встроенные в осциллографы средства компьютерной математики в цифровых осциллографах обеспечивают:

- автоматическое выполнение от 11 до 53 автоматических измерений (включая True RMS сигналов, амплитудные, частотные и иные измерения);
- проведение спектрального анализа с помощью быстрого оконного преобразования Фурье (БПФ);
- курсорные измерения;

- статистические вычисления, включая построение гистограмм;
- математические вычисления с анализируемыми сигналами;
- создание с помощью математического редактора опорных зависимостей.

На рис. 1.38 показан внешний вид осциллографов DPO/MSO4000 фирмы Tektronix, построенных на основе закрытой архитектуры. Приборы DPO4000 – это двух- или четырехканальные осциллографы с полосой частот 350/500/1000 МГц, использующие технологию цифрового фосфора для получения эффекта послесвечения. Приборы имеют 25 автоматических измерений, курсорные измерения и реализацию оконного БПФ (см. описание спектрального анализа в главе 15). Осциллограммы смешанных сигналов MSO4000 позволяют, кроме 2/4 аналоговых сигналов, наблюдать до 16 логических сигналов.

Рис. 1.38. Осциллографы серии DPO4000 (справа) и MSO4000 (слева)

На рис. 1.39 показаны осциллограммы двух каналов осциллографа DPO4004 (4 канала с полосой 1000 МГц) в режиме проведения математических арифметических операций над ними. Выполняется операция умножения, и ее результат представлен средней осциллограммой.

Обратите внимание на позицию меню «Матем. расширенный» внизу экрана. Она вводит математический редактор, позволяющий задать куда более сложные математические операции над осциллограммами, например дифференцирование или вычисление многих функций. Другая позиция меню «БПФ» задает выполнение спектрального анализа методом оконного БПФ. На рис. 1.40 показан пример получения спектра прямоугольных импульсов (меандра) с применением окна Блекмана-Харриса.

Еще большими математическими возможностями обладают осциллографы с открытой архитектурой, которые строятся на основе встроенного в них полноценного ПК, имеющего типовую (по архитектуре) системную плату и встроенный жесткий диск достаточно большой емкости. К таким приборам относятся осциллографы серий DPO5000 (с полосой 350, 500 и 1000 МГц), DPO7000 (с полосой 500 МГц, 1, 2,5 и 3,5 ГГц) и DPO/DSA70000 (анализаторы сигналов с полосой от 4 до 20 ГГц). Внешний вид семейства осциллографов DPO7000 показан на рис. 1.41.

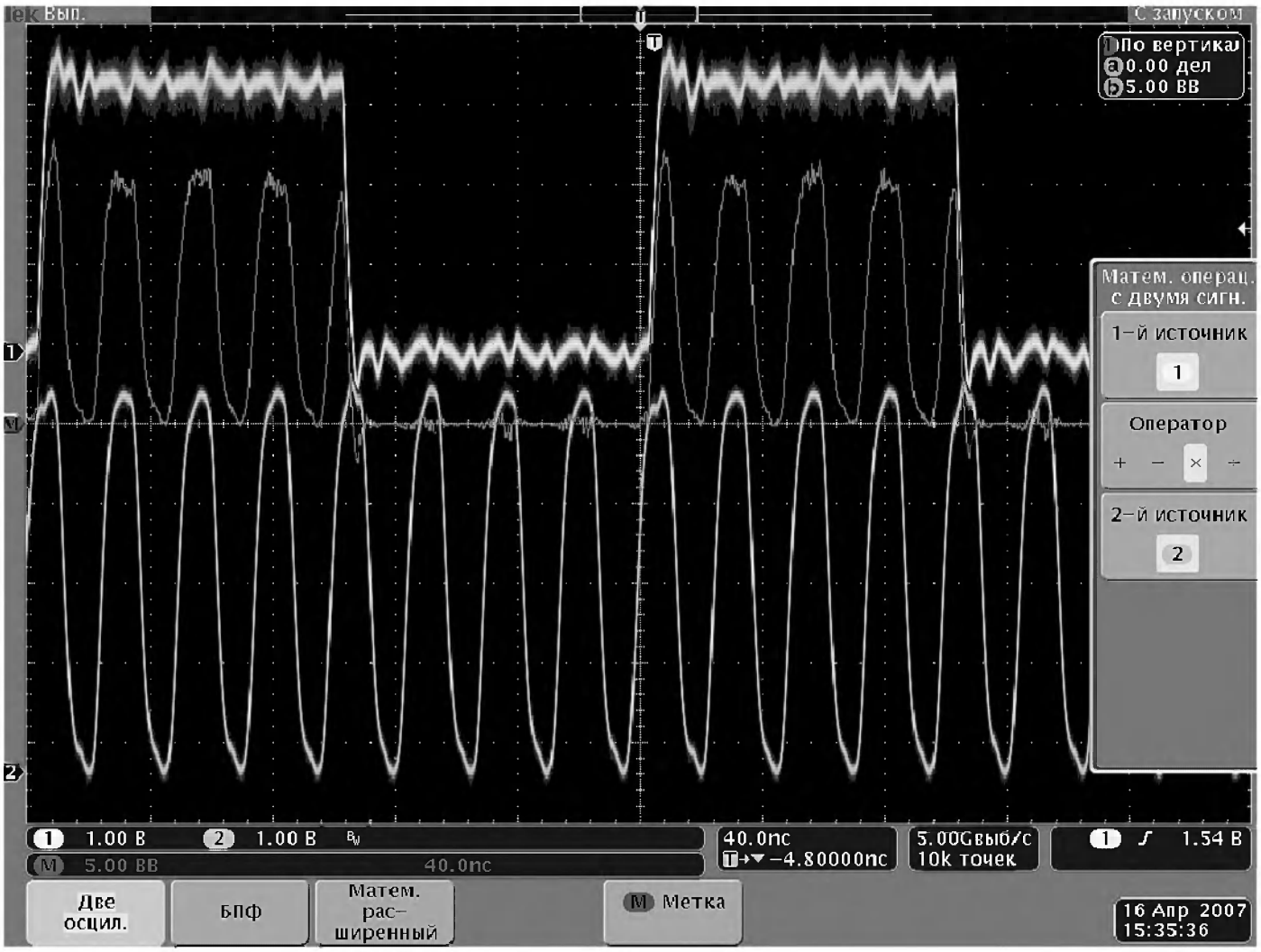


Рис. 1.39. Осциллограммы двух сигналов и результат их перемножения на экране осциллографа DPO4104

Число автоматически измеряемых параметров у осциллографов этой серии достигает 53. На рис. 1.42 представлен вид экрана осциллографа DPO7000 в режиме просмотра трех сигналов и измерения 8 параметров для сигналов первого и второго каналов. Обратите внимание, что выводится не только текущее значение каждого измеряемого параметра, но и статистические данные по каждому измерению (среднее значение, минимальное, максимальное и среднеквадратическое значения и др.). Вопросительный знак в конце некоторых строк с измеренными параметрами указывает на сомнительность измерения. Внизу экрана представлена панель настройки измерений, которую можно убрать для просмотра осциллограмм на полном экране.

На рис. 1.43 представлен вид экрана осциллографа DPO7000 в режиме просмотра фронтов импульса и автоматическом измерении уровней (верхнего и нижнего) перепадов сигнала и времени нарастания и спада. Времена нарастания и спада измеряются на уровнях 10% и 90%. Обратите внимание на показ этих уровней с помощью широких стрелок.

Рисунок 1.44 показывает пример построения спектра сигнала треугольной формы. Внизу экрана представлена панель установки параметров спектрального

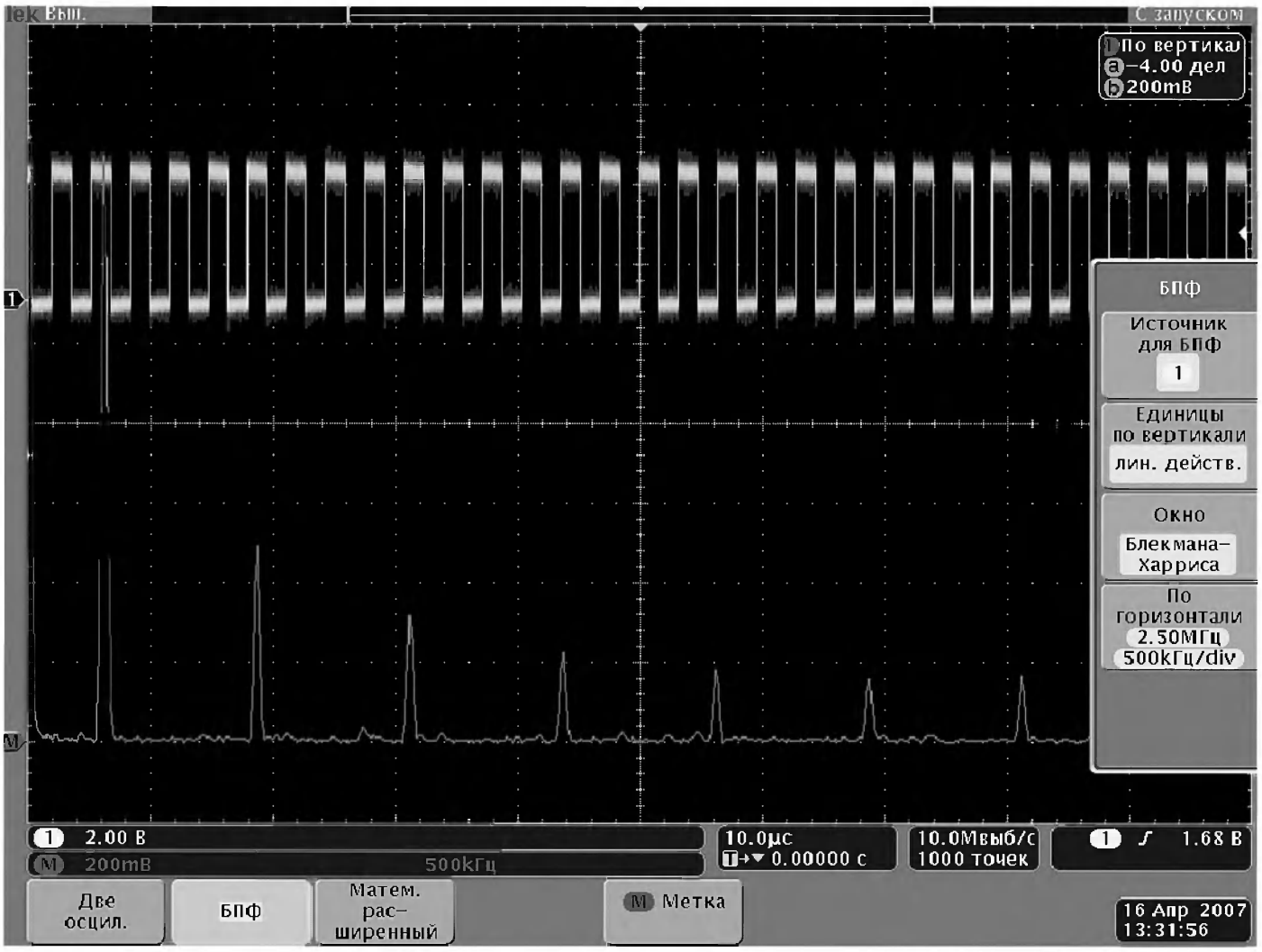


Рис. 1.40. Пример получения спектра меандра (осциллограф DPO4104)

Рис. 1.41. Внешний вид семейства осциллографов DPO7000

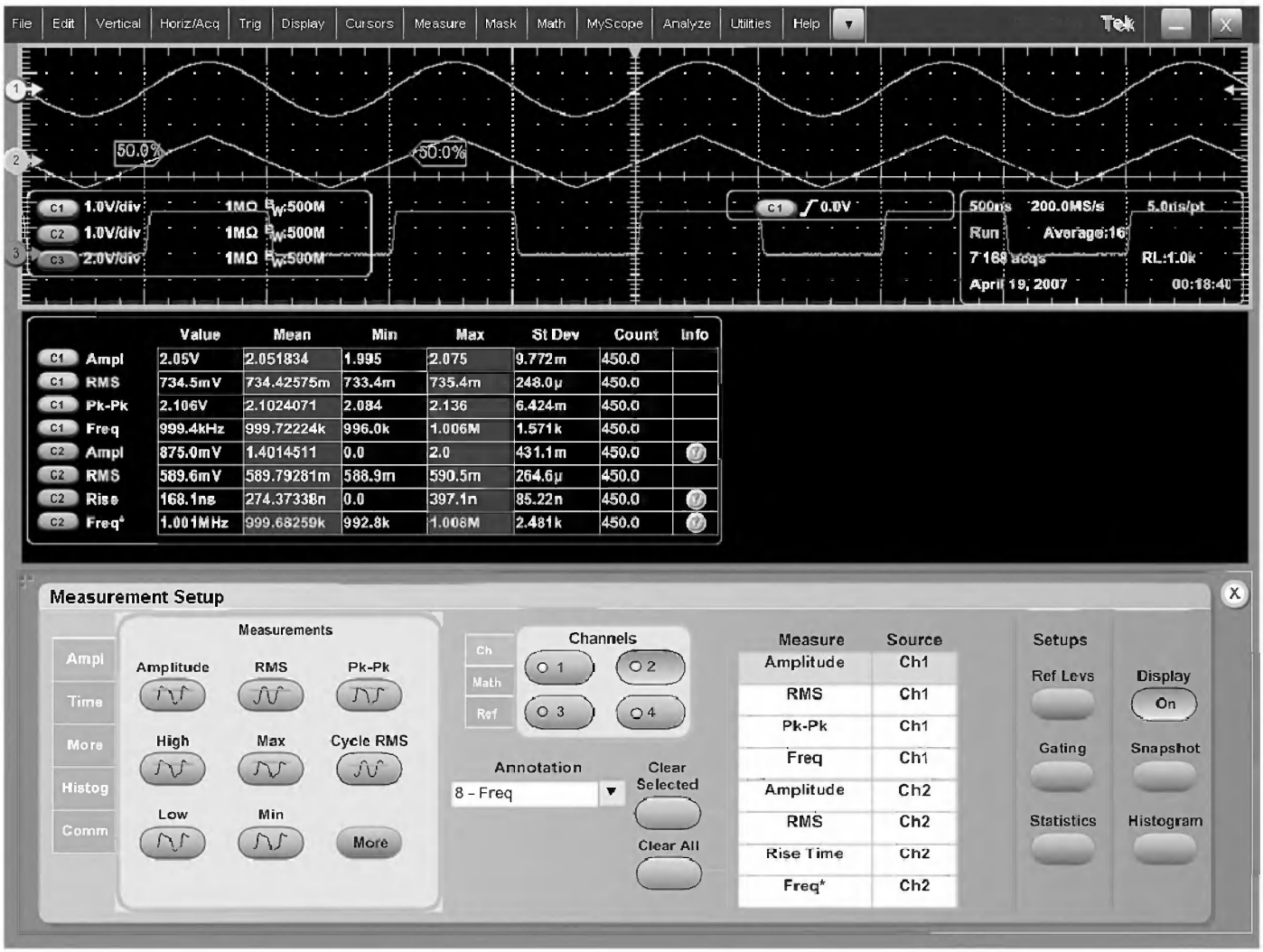


Рис. 1.42. Вид экрана осциллографа DPO7000 в режиме просмотра трех сигналов и автоматического измерения восьми параметров двух сигналов

анализа, в частности центральной частоты, полосы частот анализа и разрешения по частоте для заданного окна (у этого прибора также используется оконное БПФ).

Хотя возможности осциллографов с открытой архитектурой (как, увы, и их цены) впечатляют, нельзя не отметить, что они все же намного ниже возможностей современных СКМ и СКА, таких как Mathcad, MATLAB и др. В связи с этим огромный интерес представляет совместное применение современных цифровых осциллографов с обычными СКМ и СКА. Эти системы могут быть прямо установлены на жесткий диск осциллографов с открытой архитектурой.

Приборы с закрытой архитектурой, как и приборы с открытой архитектурой, могут подключаться к отдельному ПК (например, ноутбуку или настольному) через имеющиеся у них интерфейсы USB (универсальная последовательная шина), RS232 (стандартный коммуникационный последовательный интерфейс), LPT (параллельный интерфейс), GPIB (приборный интерфейс) и LAN (сетевой интерфейс). Современный уровень развития цифровых приборов обеспечивает их объединение в локальные сети, дистанционное управление от ПК и даже через Интернет. Это создает основы создания многофункциональных контрольно-из-

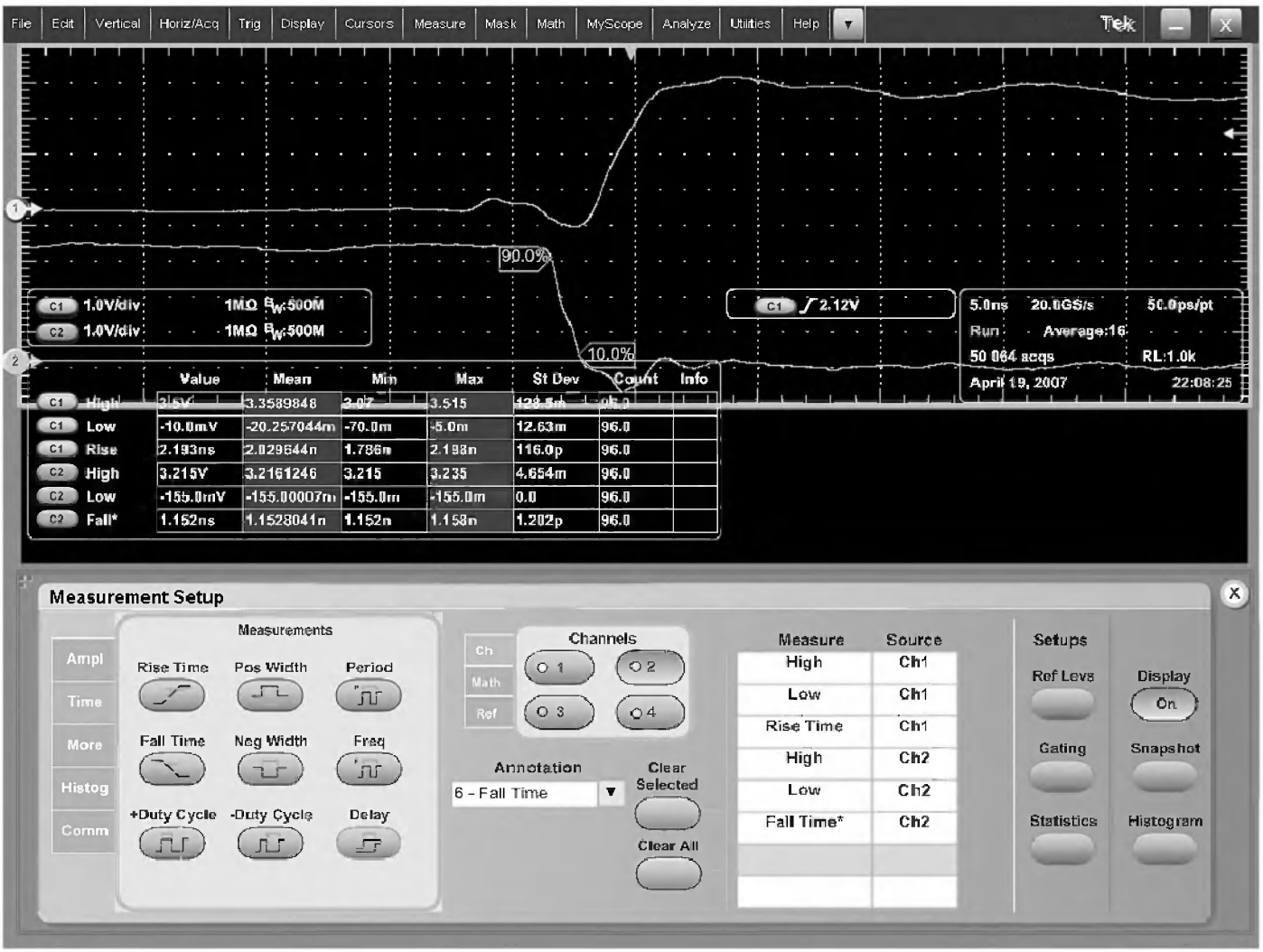


Рис. 1.43. Пример просмотра перепадов сигнала и автоматического измерения их уровней и длительностей нарастания и спада (осциллограф DPO7054)

мерительных и вычислительных систем. Описание программного обеспечения для связи приборов с ПК можно найти в [192, 193].

1.7.5. Анализаторы спектра со специализированными СКМ

Анализаторы спектра – еще один класс приборов, в которых в наше время включаются специализированные СКМ, предназначенные для выполнения наряду с аппаратным анализом спектра программного анализа [194–196]. Благодаря применению цифровых фильтров и оконного преобразования Фурье эти приборы обеспечивают высокое качество спектрального анализа при весьма высоком частотном разрешении. Математические аспекты анализа спектра изложены в главе 15.

Мировая промышленность выпускает десятки типов анализаторов спектра. Среди них наиболее насыщенными средствами компьютерной математики являются анализаторы спектра радиочастот реального времени. Основным их разработчиком и производителем является корпорация Tektronix.

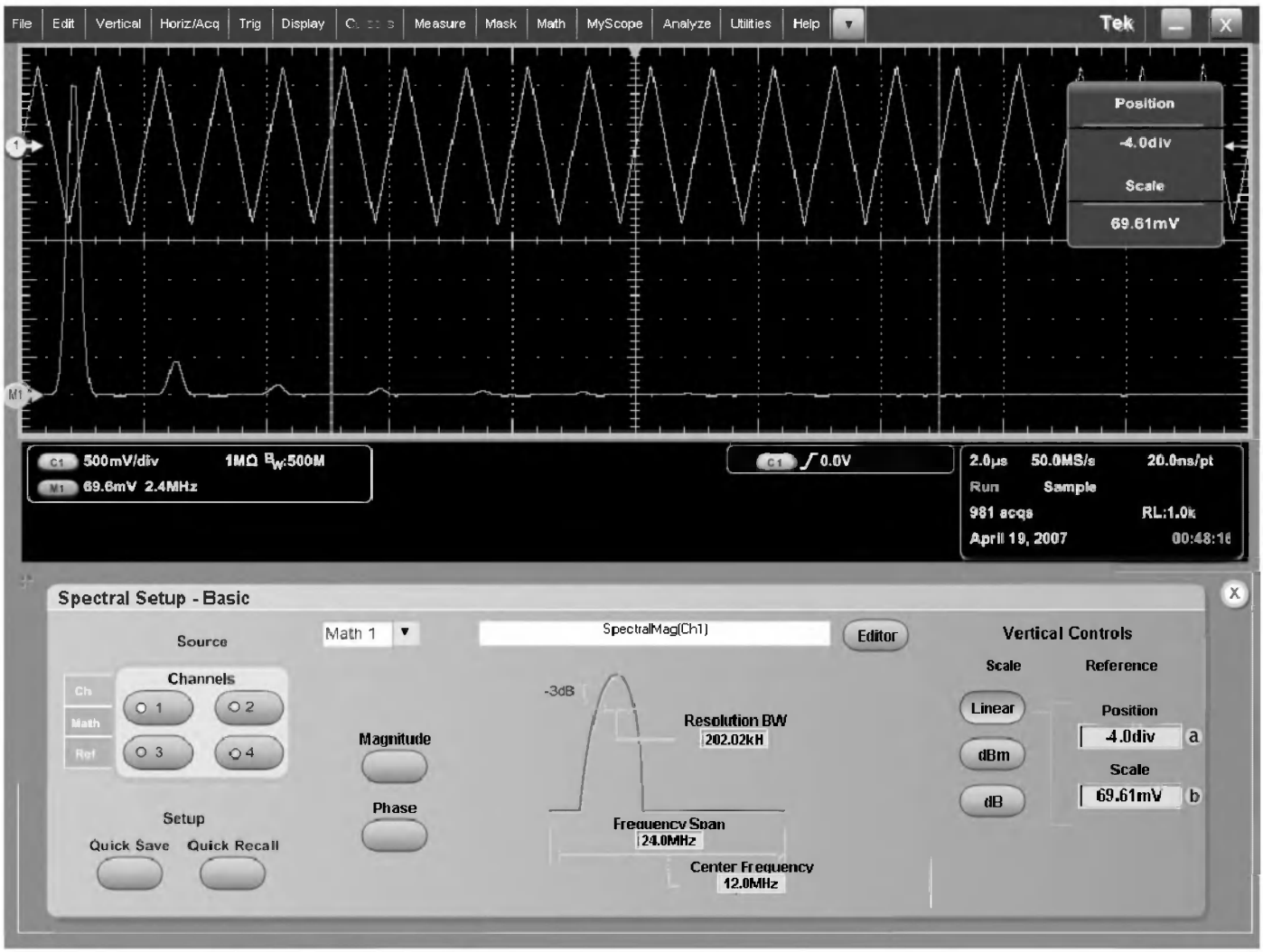


Рис. 1.44. Пример получения спектра треугольного сигнала (осциллограф DPO7054)

Анализатор спектра реального времени представляет собой супергетеродинный радиоприемник (рис. 1.45), переносящий спектр радиосигнала в область промежуточной частоты ПЧ [195, 196]. После фильтра ПЧ сигнал сразу подается на высокоскоростной аналого-цифровой преобразователь АЦП, и полученный цифровой сигнал подвергается цифровой обработке сигнала в блоке ЦОС. Именно тут осуществляются математические операции БПФ, сохранение отсчетов сигнала и синхронизация. В правой части функциональной схемы анализатора спектра представлены способы представления спектров, реализуемые программным путем с помощью встроенного в анализатор компьютера.

На рис. 1.46 показан внешний вид самого мощного анализатора спектра реального времени – RSA6100A фирмы Tektronix. Приборы рассчитаны на диапазон частот от 9 кГц до 6 ГГц (модель RSA6106A) и от 9 кГц до 14 ГГц (модель RSA6114A). Полоса частот обзора – до 40 МГц или до 110 МГц опционально (RSA6106A и RSA6114A). Длина непрерывного участка спектрограммы составляет до 1,7 секунды в полосе обзора 110 МГц и возрастает по мере уменьшения полосы обзора.

На рис. 1.47 показана копия экрана анализатора с данными анализа спектра синусоидального сигнала от генератора Tektronix AFG3252 в диапазоне частот

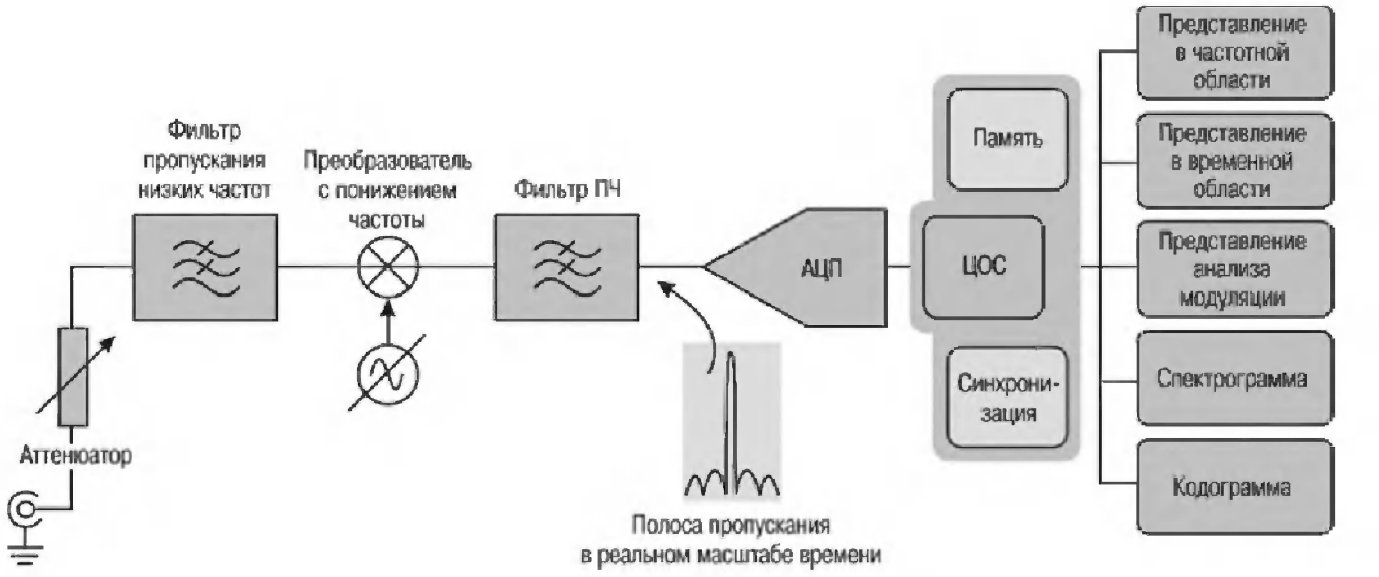


Рис. 1.45. Функциональная схема анализатора спектра реального времени

Рис. 1.46. Внешний вид анализатора спектра реального времени RSA6114A

просмотра Span в 40 МГц при средней частоте CF 20 МГц. Отчетливо видны первая, вторая и третья гармоники сигнала генератора и помеха вблизи второй гармоники.

Окно **Amplitude vs Time** показывает временную зависимость сигнала, которая в данном случае представляет собой слившиеся периоды синусоиды в виде дорожки. Сигнал воссоздается из его спектра очень приближенно из-за помех, шумов и ограничения полосы спектра. В окне **DPX Spectrum** видна фирменная новинка от Tektronix – спектр мощности реального сигнала, построенный с применением технологии цифрового фосфора (имитация запоминания множества спектрограмм). В режиме **DPX Spectrum** анализатор выводит до 47 000 спектров в секунду.

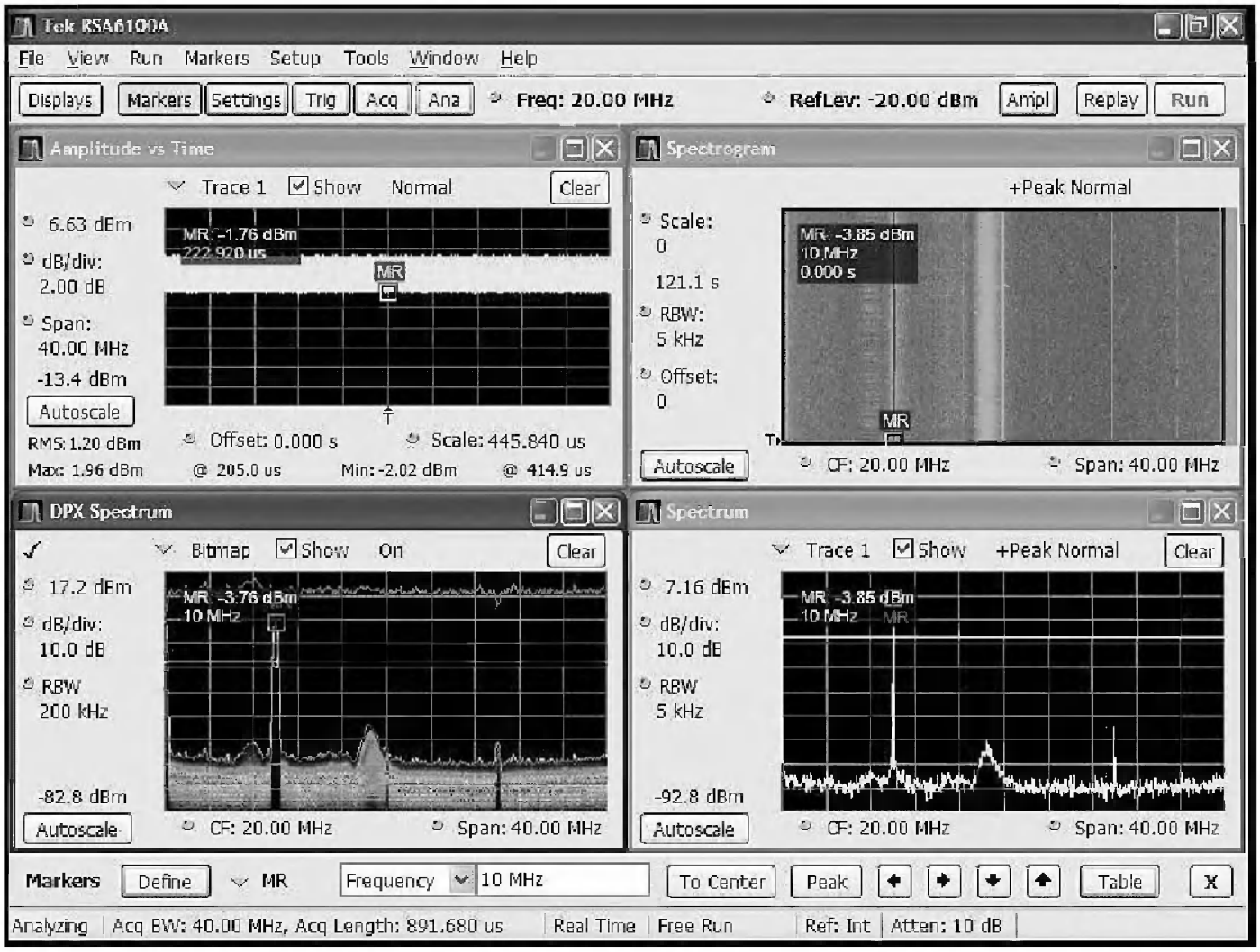


Рис. 1.47. Вид экрана дисплея анализатора спектра реального времени

В окне **Spectrum** представлено построение обычного спектра. В данном случае и он достаточно информативен. Но особенно информативна уникальная спектрограмма сигнала, представленная в верхнем правом углу экрана. Спектрограмма строится с помощью оконного БПФ в плоскости частота–время, причем интенсивность спектральной составляющей в каждой ее точке задается ее цветом и яркостью. В отличие от спектра на плоскости ампдитуда–частота, спектрограмма позволяет оценивать положение во времени любой особенности спектра. При этом возможно наблюдение спектров нестационарных сигналов, например с изменяющейся во времени частотой, с амплитудной, фазовой, частотной, кодоимпульсной и другими видами модуляции.

Глава 2

Типы данных и работа с ними

Системы компьютерной математики, как и любые другие программные средства, работают с *данными* и осуществляют их обработку. Поскольку СКМ ориентированы на подготовку документов самого различного характера (в том числе электронных документов и книг), то они обладают обширным набором возможных типов данных и средствами для работы с ними. Это и рассматривается в данной главе на примере системы Maple. Описаны и специфические особенности задания типов данных в других системах (Mathematica и Mathcad).

2.1. Простые типы данных

Описание типов данных разумно начать с простых типов данных. Обычно к ним относят такие типы данных, как отдельные числа и числовые константы, символы языков программирования и строки.

2.1.1. Системы счисления и основания чисел

Числа в виде исходных данных, констант и результатов вычислений являются, пожалуй, самыми распространенными объектами математических систем. Числа несут *количественную оценку* представляемых ими понятий и более сложных объектов. Математические системы могут работать с различными *типами чисел*. Числа часто используются в математических выражениях и формулах.

Существуют различные способы представления чисел – *системы счисления*: позиционные и непозиционные. При *позиционной системе* значение каждой цифры зависит от ее положения – *разряда*. Количество значений p каждого разряда задается *основанием* числа.

Любое число в позиционной системе исчисления можно представить в виде:

$$a_{m-1}p^{m-1} + a_{m-2}p^{m-2} + ... + a_1p^1 + a_0p^0 + a_{-1}p^{-1} + a_{-2}p^{-2} + ... + a_{-s}p^{-s}. \quad (2.1)$$

Здесь a_i задает *вес* каждого разряда и его положение: положительные значения i (не путать с мнимой единицей) относятся к целой числа с m разрядами, а отрицательные – к порядку с s разрядами.

Чаще всего используются десятичные числа с основанием 10, относящиеся к арабской позиционной системе счисления. При основании 10 веса a_i кратны 10 при $a_0 = 1$, $a_1 = 10$, $a_2 = 100$ и т. д. и $p = 0, 1, ... 9$. Для дробной части числа $a_{-1} = 0{,}1$, $a_{-2} = 0{,}01$ и т. д. Непозиционные системы счисления (например, римская система) в СКМ пока не применяются.

2.1.2. Натуральные и простые числа

Натуральными называют целые положительные числа 1, 2, 3, ... Эти числа возникли исходя из потребностей счета отдельных и неразделимых предметов. Они могут быть простыми и составными. Ряд натуральных чисел бесконечен, поскольк-

ку к каждому «последнему» числу всегда можно добавить единицу и получить очередное число.

Со временем появилось понятие *отрицательных чисел*, причем знак минус «–» обычно означает недостаток предметов при счете – например, число 8 можно представить как 10 за вычетом двух предметов, то есть 8 = 10 + (–2). Здесь –2 означает два недостающих предмета. Сами по себе натуральные числа в математических системах как отдельный класс данных не используются. Однако никаких ограничений к их применению как к подвиду целых чисел нет.

К натуральным числам относят и *простые числа* – это такие числа (за исключением 1), которые делятся только на себя. Все они нечетные, за исключением единственного простого четного числа 2. Многие СКМ имеют реализацию алгоритмов для поиска простых чисел и разложения натуральных чисел на простые множители.

2.1.3. Целые десятичные числа

Целые числа – это такие числа (тип integer), которые могут быть представлены в виде разности натуральных чисел. Обычно они задаются набором только цифр и, возможно, знака перед таким набором. Примеры целых чисел: 0, 1, 123, –456 и т. д. Таким образом, целые числа могут быть положительными и отрицательными. Пока речь идет о числах с основанием 10.

Знак «–» перед числом рассматривается как *унарный минус*, если перед ним нет другого числа – тогда знак «–» является оператором вычитания. Например, –4 означает минус четыре, а 6 – 2 даст результат 4. Можно использовать скобки для уточнения роли этого знака, например 6–(–2) даст 8. Подряд два знака, то есть символ «––», использовать нельзя.

Минимальное (не равное нулю) и максимальное значения целых чисел при m разрядах имеют значения:

$N_{min} = P^{-s}$ и $N_{max} = P^{m} - 1$.

При этом всего можно представить p^{m+s} чисел.

Для целых чисел определен ряд специальных функций, например разложение на простые множители, нахождение общего делителя, вычисление факториала и т. д. Для целых чисел характерна дискретность значений, минимальное отличие которых составляет 1 или –1. Такие числа широко применяются для задания индексов для данных сложных типов, например векторов и матриц.

Как известно, обычно целые числа в системах программирования делятся по своему формату на короткие, средние и длинные. Как правило, они занимают два, четыре и восемь байтов в памяти. Их часто обозначают int2, int4 и int8. Применение таких чисел оправдано уменьшением затрат памяти и ускорением вычислений.

В отличие от представления целых чисел в обычных языках программирования, у символьных систем таких жестких форматов нет. Для целочисленных операций используются не обычные команды микропроцессора, ориентированные на

фиксированные форматы, а специальные (реализованные программно) алгоритмы вычислений с целыми числами произвольной разрядности. Естественно, это ведет к замедлению вычислений, но зато они выполняются абсолютно точно. Данная возможность лежит в основе операций *точной арифметики* и вычислений *произвольной разрядности*.

Фактически целые числа произвольной разрядности в системах символьной математики представляются *списками* отдельных цифр каждого числа. Например, число 1 234 567 может быть представлено списком из семи элементов {1,2,3,4,5,6,7}, тогда как число 12 представляется списком всего из двух элементов {1,2}. Разумеется, это не самый компактный способ хранения чисел и манипулирования с ними, но он позволяет легко получить произвольную разрядность целых чисел. В действительности применяемые в системах символьной математики способы представления чисел более компактны, чем приведенный простейший, но это не меняет сути главного – чем больше количество цифр числа, тем больше памяти отводится на его хранение.

2.1.4. Числа двоичные, восьмеричные и шестнадцатеричные

Некоторые математические системы могут работать с числами, имеющими произвольное основание. По умолчанию используются *десятичные числа* (*decimal*) с основанием 10. Достаточно широкое применение имеют и числа с основаниями 2 (*binar* – бинарные, или *двоичные, числа*), 8 (*octal – восьмеричные числа*) и 16 (hexadecimal, или сокращенно *hex*, – *шестнадцатеричные числа*). Разряд чисел этого типа может иметь следующие значения:

b	**binary**	0	1													
o	**octal**	0	1	2	3	4	5	6	7							
d	**decimal**	0	1	2	3	4	5	6	7	8	9					
h	**hex**	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Здесь перед названием типа числа указан его отличительный символ (обычно может использоваться как строчная, так и заглавная буква, например h или H для шестнадцатеричного числа). Этот символ используется после числа, например 11001b, 1234o или A2CDH. Числа с разным основанием, за исключением десятичных чисел, в математических расчетах используются редко. Но их применяют при описании вычислительных систем.

Как известно, минимальной единицей информации в компьютерной технике является двоичная единица – *бит* (*bit*). Она имеет представление в виде 0 или 1, удобное для реализации простейшими электронными схемами с двумя состояниями электрического равновесия (например, триггерами или конденсаторными ячейками памяти). Восемь бит обычно называют *байтом*. Байт имеет $2^8 = 256$ значений от 0 до 255. Он положен в основу кодов ASCII (American Standard Code for Information Interchange – Американский стандартный код для обмена информацией). 1024 байта образуют килобайт (Кбайт), 1024 Кбайта дают 1 мегабайт

(Мбайт) и т. д. Шестнадцатеричные числа применяются для указания *адресов* ячеек памяти ОЗУ.

2.1.5. Рациональные числа

Рациональные числа (их определяют типом *rational*) задаются отношением отношения целых чисел, например 7/9, –123/127 и т. д. Системы компьютерной алгебры стремятся представить результаты вычислений (в том числе промежуточных в математических выражениях) в точном виде, то есть в форме целых или рациональных чисел.

При проектировании микропроцессорных устройств нередко приходится хранить многочисленные числовые константы в постоянном запоминающем устройстве – ПЗУ. Часто представление таких констант в форме рационального числа является наиболее экономичным способом хранения чисел при заданной погрешности их представления. Указанная выше возможность математических систем представлять любое число в рациональном виде в этом случае очень полезна.

2.1.6. Вещественные (действительные) числа

Вещественные числа возникли из-за ограничений рациональных чисел в представлении непрерывно изменяющихся (аналоговых) величин. Оказалось, например, что диагональ квадрата со стороной, выраженной рациональным числом, нельзя представить в виде рационального числа. Вещественные числа являются мерой непрерывных величин. Они задаются в виде мантиссы и порядка:

$$[-]iii.ddddd\cdot 10^{[-]p} \quad \text{или} \quad [-]iii.ddddd\cdot E^{[-]p},$$

где [–] – необязательный унарный минус, iii.ddddd – мантисса числа с разделительной десятичной точкой, iii – целая часть мантиссы, ddddd – дробная часть мантиссы, p – порядок числа (в виде целого числа, включая 0).

Примеры задания вещественных чисел: $1.23*10^{-5}$, 1.23E5, 123.4567E–10. Некоторый разнобой в их задании (порядок может задаваться как степень числа 10 или степень E и т. д.) не слишком затрудняет использование математических систем, поскольку быстро распознается – неверная форма задания числа ведет к появлению сообщения об ошибке.

Вещественные числа имеют фиксированный по числу байтов (или битов) формат. Они могут быть короткими, обычными или длинными. Работа с вещественными числами ограниченной разрядности предусмотрена на аппаратном уровне микропроцессора при поддержке ее и математическим сопроцессором.

2.1.7. Нотация вещественных чисел

В практике математических расчетов используются различные формы представления вещественных чисел – *нотации*. Рассмотрим основные из них.

Десятичные числа с фиксированной запятой (или точкой) обычно используются в финансово-экономических и некоторых иных видах расчетов. К примеру, любая сумма денег может быть представлена в виде суммы рублей Р и копеек К, будучи записанной в виде Р,К (например, 95,34 рубля, что означает 95 рублей и 34 копейки).

В математике при записи вещественных чисел принято использовать разделительную запятую, тогда как в информатике обычно разделителем выступает точка (дабы исключить восприятие запятой в ее общепринятом понимании). Поэтому понятия чисел с фиксированной точкой и с фиксированной запятой идентичны. Мы будет далее говорить о таких числах как о числах с *фиксированной точкой* (*floating point*). Недостаток подобных чисел – малый диапазон представляемых ими значений N: $p^{-s} \le N \le p^m - p^{-s}$. Поэтому числа в таком формате применяются редко.

Десятичные числа с плавающей точкой (их часто определяют опцией *float*) характерны возможностью изменения положения разделительной точки. Например, $1.23456*10^5$, $12.3456*10^4$, $123.456*10^3$ и $1234560*10^{-1}$ представляют одно и то же число 123 456 в форме чисел с плавающей точкой.

В общем случае такие числа содержат мантиссу M и порядок p и записываются в виде

$$N = \pm M \cdot p^{\pm r}.$$

Десятичные числа с нормализованной мантиссой, или числа с *научной нотацией*. У таких чисел мантисса нормализуется, так что до разделительной точки может быть только одна цифра (от 0 до 9). Так, число 123 в этом случае будет представлено в виде $1.23 \cdot 10^2$, а число 123 456 – как $1.23456 \cdot 10^5$. Еще чаще задается $|M| < 1$, так что число 123 456 будет представлено как $.123456 \cdot 10^6$.

Главное достоинство такой формы представления чисел – большой диапазон их значения:

$$p^{-m} p^{-(p^s-1)} \le N \le (1 - p^{-m}) p^{(p^s-1)}.$$

Благодаря этому именно данный формат чисел применяется наиболее часто в научно-технических расчетах.

Вещественные числа всегда имеют некоторую погрешность представления результатов из-за неизбежного округления их и существования так называемого *машинного нуля* – наименьшего числа, которое воспринимается как нуль. В терминах математических систем часто говорят о приближении числовых данных как об их аппроксимации, хотя в отечественной литературе под аппроксимацией чаще подразумевают приближенное описание некоторой зависимости между данными достаточно приближенной аналитической зависимостью.

СКМ обычно имеют специальные системные переменные или функции, возвращающие минимально возможное и максимально возможное (*машинная бесконечность*) значения вещественных чисел.

Числа в инженерной нотации характерны тем, что порядок их равен 0 или кратен 3. Они часто применяются в физических расчетах, где нередко размерные величины имеют кратность порядка, равную трем (например, 1 г – это тысячная доля килограмма, 1 мс – это тысячная доля секунды, 1 мл – это тысячная доля

одного литра и т. д.). Так, число 1234 в такой нотации будет иметь вид $1.234 \cdot 10^3$, а 12 345 – $12.345 \cdot 10^3$. Нетрудно заметить, что в данном случае положение десятичной точки в мантиссе не фиксировано.

2.1.8. Комплексные числа

Комплексные числа, пожалуй, являются наиболее общим типом числовых данных. Уже при решении квадратных уравнений появляются решения, в составе которых имеются квадратные корни из отрицательных чисел. Однако такие решения являются мнимыми. Но уже при вычислении корней кубических уравнений вычисление таких корней оказывается обязательной операцией, что привело к осознанию реальности комплексных чисел. Такие числа обычно задаются в одной из двух возможных форм – см. ниже.

2.1.9. Алгебраическая форма комплексных чисел

Алгебраическая форма задания комплексного числа следующая:

$Z = a + ib = \mathrm{Re}(Z) + I * \mathrm{Im}(Z)$.

В этой форме числа содержат действительную $\mathrm{Re}(Z)$ и мнимую $\mathrm{Im}(Z)$ части. Сами по себе значения $\mathrm{Re}(Z)$ и $\mathrm{Im}(Z)$ могут иметь целые или действительные значения, но в записи Z мнимая часть содержит множитель i (или j), который является квадратным корнем из -1.

2.1.10. Экспоненциальная форма комплексных чисел

Экспоненциальная форма комплексного числа соответствует следующему виду:

$Z = M \cdot e^{i\varphi}$,

где M – *модуль* комплексного числа Z, а φ – *фаза*.

В экспоненциальной форме комплексные числа особенно часто встречаются в электро- и радиотехнических расчетах, поскольку удачно описывают гармонические колебания. Как известно, они также характеризуются амплитудой (модулем) и фазой (а также частотой). С применением комплексных чисел в таких расчетах эти числа окончательно избавились от налета нереальности, или, как принято сейчас говорить, виртуальности.

2.1.11. Преобразование комплексных чисел

Преобразование комплексных чисел из алгебраической формы в экспоненциальную выполняется с применением соотношений

$M = \sqrt{a^2 + b^2}$

и

$\varphi = \arccos(a/|M|)$ при $b \geq 0$ и $\varphi = -\arccos(a/|M|)$ при $b < 0$.

Подобные преобразования легко выполняются всеми современными научными микрокалькуляторами и СКМ. Для угла φ возможны и иные представления, но данное дает непрерывность при изменении a в широких пределах.

2.1.12. Характерные правила ввода и вывода чисел

Ввод и *вывод* чисел в математических системах имеет следующие особенности:

- для отделения целой части мантиссы от дробной используется разделительная точка;
- нулевая мантисса не отображается (число начинается с разделительной точки);
- мантисса отделятся от порядка знаком умножения или пробелом, который рассматривается как знак умножения;
- мантисса отрицательного числа задается унитарным знаком минуса перед ней;
- признаком ввода порядка являются символы E или e либо число 10;
- порядок задается как целое число после знаков E или e либо как степень 10 и может иметь унарный знак минус;
- мнимая часть комплексных чисел задается умножением ее на символ мнимой единицы i (или I, j и J), который означает квадратный корень из –1;
- в выводе комплексного числа знак умножения на символ мнимой единицы может заменяться пробелом;
- представление чисел задается их установленным форматом.

У большинства СКМ десятичная точка в числах имеет особый статус – указание ее в любом месте числа, в том числе в начале или в конце, делает число вещественным и ведет к переводу вычислений в режим явных вычислений с вещественными числами. При этом количеством выводимых после десятичной точки цифр можно управлять, задавая их значение специальной системной переменной. Например, в системе MuPAD системная переменная DIGITS задает вывод числа знаков после десятичной точки, а преобразование в форму вещественного числа обеспечивает функция `float`.

Многие математические системы (Maple, Derive, Mathcad и др.) имеют специальные средства для задания формата вывода чисел. Чаще всего это специальные окна, в которых можно задавать число цифр после разделительной точки при представлении действительных чисел, погрешность представления нуля для действительных и комплексных чисел и т. д.

2.1.13. Символы и строковые данные

Символы – это элементарные объекты, из которых создаются слова, предложения и тексты и конструируются математические выражения. К символам относятся буквы обычных разговорных языков и языков программирования. Например,

слова английского языка состоят из малых (строчных) латинских букв a, b, c, ..., z и больших (заглавных) букв A, B, C, ..., Z. Отдельные цифры, например 0, 1, 2, ..., 9, и математические спецзнаки также являются символами. Символы входят в *алфавит* языков программирования СКМ. Символы обычно кодируются кодами, например упомянутыми однобайтовыми кодами ASCII. В последнее время получила распространение двухбайтовая система кодирования Unicode, при которой максимальное число символов достигает 65 536.

Из символов создаются *строки* – последовательности символов. Чтобы отличить их от числовых данных, строковые данные должны особым образом идентифицироваться. Чаще всего для этого их размещают в кавычках или, при задании имен переменных, одиночных апострофах. Например, «123» – это строка, а 123 – целое число. Ниже показана еще пара примеров строковых данных:

«Hello my friend!» или ?Hello my friend!? – строка символов;

«2+3» или ?2+3? – неисполняемая строка символов – цифр.

Обратите внимание на то, что строка «2+3» или ?2+3? – это просто цепочка символов 2, + и 3. В СКМ такие строки не оцениваются и не исполняются и просто повторяются в строках вывода. Но в каждой системе есть функции преобразования, которые могут превратить строку в исполняемое выражение 2+3, которое в строке вывода даст результат 5.

Иногда встречаются также *программные комментарии*. Это строки, которые можно прочесть при выводе листинга программы, но которые не исполняются при исполнении программы. Признаком таких строк может быть тот или иной символ, например % или # (стоит уточнить такой символ при написании программ в тех или иных СКМ).

2.1.14. Объекты

В общем случае математические системы оперируют с *объектами* (objects). Все перечисленные выше типы данных порождают объекты в виде чисел (numbers), символов (symbols), строк (string) и т. д. Объектами являются и математические выражения (expr), символы, строки из символов (strings), а также константы, переменные, графические и звуковые объекты и т. д.

2.2. Работа с простыми данными Maple-языка

2.2.1. Использование знаков алфавита

Простые типы данных и алфавит на уровне символов практически одинаковы у всех систем компьютерной математики. Поэтому их достаточно рассмотреть на примере одной системы – Maple.

Алфавит Maple-языка содержит 26 малых латинских букв (от **a** до **z**), 26 больших латинских букв (от **A** до **Z**), 10 арабских цифр (от **0** до **9**) и 32 специальных

символа (арифметические операторы +, –, *, /, знак возведения в степень ^ и др.). Кроме того, имеется множество особых математических символов. Все они будут в данной главе. Для ввода символов используются клавиатура и панели математических символов.

Имеются пять пар альтернативных символов (означающих одно и то же):

^ и ** [и (|] и |) { и (* } и *)

К специальным одиночным и составным знакам относятся элементы синтаксиса языка:

- **%** – системная переменная, хранящая результат предшествующей операции;
- **:** – фиксатор выражения, предотвращающий вывод результата вычисления в ячейку вывода;
- **;** – фиксатор выражения, дающий вывод результата вычисления в ячейку вывода;
- **#** – указатель программного комментария;
- **“** – ограничитель строки (например, `'string'`);
- **:=** – оператор присваивания (например, `x:=5`);
- **;;** – пустой оператор;
- **::** – указатель типа переменной (например, `n::integer` или `z::complex`);
- \ – знак обратного деления, который имеет множественные значения в зависимости от контекста (см. справку по этому знаку – backslash).

Комментарии в программе, не выводимые в ячейки вывода, в Maple 9.5 задаются после символа **#**. В них допустимо использовать все символы кодовых таблиц, что важно при вводе русскоязычных комментариев, использующих символы кириллицы. Применение последних для идентификаторов (имен) объектов недопустимо, хотя иногда и возможно.

2.2.2. Зарезервированные слова

Зарезервированные слова используются для создания условных выражений, циклов, процедур и управляющих команд. Список зарезервированных слов, имеющихся у всех версий Maple, дан ниже. Этими словами нельзя называть объекты пользователя. Список зарезервированных слов у каждой системы свой.

And	**Break**	**by**	**catch**	**description**
Do	**Done**	**elif**	**else**	**end**
Error	**Export**	**fi**	**finally**	**for**
From	**Global**	**if**	**in**	**intersect**
Local	**Minus**	**mod**	**module**	**next**
Not	**Od**	**option**	**options**	**or**
Proc	**Quit**	**read**	**return**	**save**
Stop	**Then**	**to**	**try**	**union**
Use	**While**			

Совокупность правил, по которым записываются определения всех объектов Maple-языка, называется его *синтаксисом*. Некоторые особенности синтаксиса полезно знать уже в начале освоения Maple. Например, то, что знак – (минус) имеет двойное значение. Соответственно, двойное назначение имеет и знак **+**, причем число без знака считается положительным, так что **+5=5**.

При вводе действительных чисел с порядком для указания порядка используется символ ^ (например, **2*10^100** или **2*10^-100**). Для возведения числа в степень наряду с оператором ^ можно использовать и составной оператор ** (две звездочки подряд). Для изменения общепринятого приоритета вычислений используются круглые скобки, в них же задаются параметры функций и процедур. Более подробно синтаксис Maple-языка рассматривается ниже.

Некоторые операторы представлены двумя символами – например, оператор присваивания переменным их значения **:=** содержит двоеточие и знак равенства. В таких операторах между символами недопустим знак пробела. Однако его можно использовать между отдельными частями выражений – так, **(a+b)/c** эквивалентно **(a + b) / c**.

2.2.3. Работа с числами и арифметические вычисления

Maple обеспечивает вполне естественную работу с целыми числами. В частности, обеспечиваются смена знака числа и выполнение основных арифметических операций с числами. Ввиду общеизвестности арифметических операций их определения не приводятся. Ограничимся примерами простых операций с числами, приведенными ниже:

```
> 12+34/47;
```

$$\frac{598}{47}$$

```
> -12+34*47;
```

$$1586$$

```
> 1/3;
```

$$\frac{1}{3}$$

```
> 12*10^(-15)*3;
```

$$\frac{9}{250000000000000}$$

Результаты операций с целыми числами в общем случае представляются рациональными числами.

Десятичная точка в числах имеет особый статус – указание ее в любом месте числа, в том числе в конце, делает число вещественным и ведет к переводу вычислений в режим работы с вещественными числами:

```
> 1./3;
```

.3333333333

```
> 12.*10^(-15)*3;
```

$$.3600000000\ 10^{-13}$$

Количеством выводимых после десятичной точки цифр можно управлять, задавая значение системной переменной окружения `Digits`:

```
> Digits:=3:1./3;
```

.333

```
> Digits:=10;exp(1.);
```

$$Digits := 10$$

2.718281828

```
> Digits:=40:evalf(Pi);
```

3.1415926535897932384626433832795028841971

Как видно из этих примеров, ввод и вывод чисел имеют следующие особенности:

- для отделения целой части мантиссы от дробной используется разделительная точка;
- нулевая мантисса не отображается (число начинается с разделительной точки);
- мантисса отделятся от порядка пробелом, который рассматривается как знак умножения;
- мнимая часть комплексных чисел задается умножением ее на символ мнимой единицы **I** (квадратный корень из –1);
- по возможности Maple представляет численный результат в виде точного рационального числа (отношения двух целых чисел).

Для работы с числами Maple имеет множество функций. Они будут рассмотрены в дальнейшем. С помощью многофункциональной функции `convert` Maple может преобразовывать числа с различным основанием (от 2 до 36, в том числе бинарные и шестнадцатеричные) в десятичные числа:

```
> convert("11001111", decimal, binary);
```

207

```
> convert("1AF.C", decimal, hex);
```

431.7500000

```
> convert("Maple", decimal, 36);
```

37451282

2.2.4. Точная арифметика

Благодаря возможности выполнения символьных вычислений Maple, как и другие СКА, реализует *точную арифметику*. Это значит, что результат может быть получен с любым числом точных цифр. Однако надо помнить, что идеально точные численные вычисления выполняются только в случае целочисленных операций, например таких, как приведены ниже:

```
> 101!;
```

942594775983835942085162312448293674956231279470254376832 \

```
     788935341697759931622147650308786159180834691162349000  \
     354959958336970630260326400000000000000000000000000
> (101!+1)-101!;
                                        1
> (10005!)/10000!;
                              100150085022502740120
> 2^101-2^100;
                        126765060022822940149670320537 6
> 2^101-2^100.0;
                                0.1267650600  10^31
```

Обратите внимание на то, что в последнем примере точность резко потеряна, так как показатель степени 100.0 был задан как число с плавающей точкой. Соответственно, и результат оказался в форме такого числа.

Приведем еще несколько примеров точных вычислений некоторых функций (с точностью до 150 знаков мантиссы):

```
> evalf(exp(1),150);
2.7182818284590452353602874713526624977572470936999595749 6  \
     6967627724076630353547594571382178525166427427466391 93  \
     2003059921817413596629043572900334295 26
> evalf(sin(1.),150);
0.8414709848078965066525023216302989996225630607983710656 7  \
     2751709991910404391239668948639743543052695854349037 90  \
     7920674293259118920991898881193410327729
> evalf(sqrt(2),150);
1.4142135623730950488016887242096980785696718753769480731 7  \
     6679737990732478462107038850387534327641572735013846 23  \
     0912297024924836055850737212644121497 10
> evalf(Si(1.0),150);
0.9460803070367183014941353313823179657812337954738111790 47  \
      1454773566687036540797918088702133081740711215023985 39  \
      8458909963018871921565883288920609191883
```

2.2.5. Вычисление числа π с произвольной точностью по алгоритму Рамануджана

У Maple и Mathematica в принципе возможны вычисления и с плавающей точкой с заданием до миллиона точных цифр мантиссы. Практически такая точность почти никогда не нужна, по крайней мере для физиков и инженеров. Например, всего 39 точных цифр числа π достаточно, чтобы вычислить длину окружности всей Вселенной с точностью до диаметра атома водорода. Однако истинные математики древности были увлечены вычислением числа π с большой точностью. Кое-кто потратил на это всю жизнь. Выдающийся вклад в такие расчеты внес Рамануджан, который еще в 1916 г. предложил алгоритмы и формулы для вычисления

числа π с произвольной точностью. Они часто используются для оценки эффективности супер-ЭВМ и программ точных вычислений.

Приведем и мы простой пример торжества Рамануджана. На рис. 2.1 представлено задание одной из самых известных формул Рамануджана. Уже первый член суммы этой формулы (k = 1) дает значение числа π с погрешностью вычисления менее $3 \cdot 10^{-8}$. Увеличение k на 1 каждый раз увеличивает число верных десятичных знаков на 8, то есть в сто миллионов раз! В принципе, эта формула может дать до миллиарда и более точных знаков числа π!

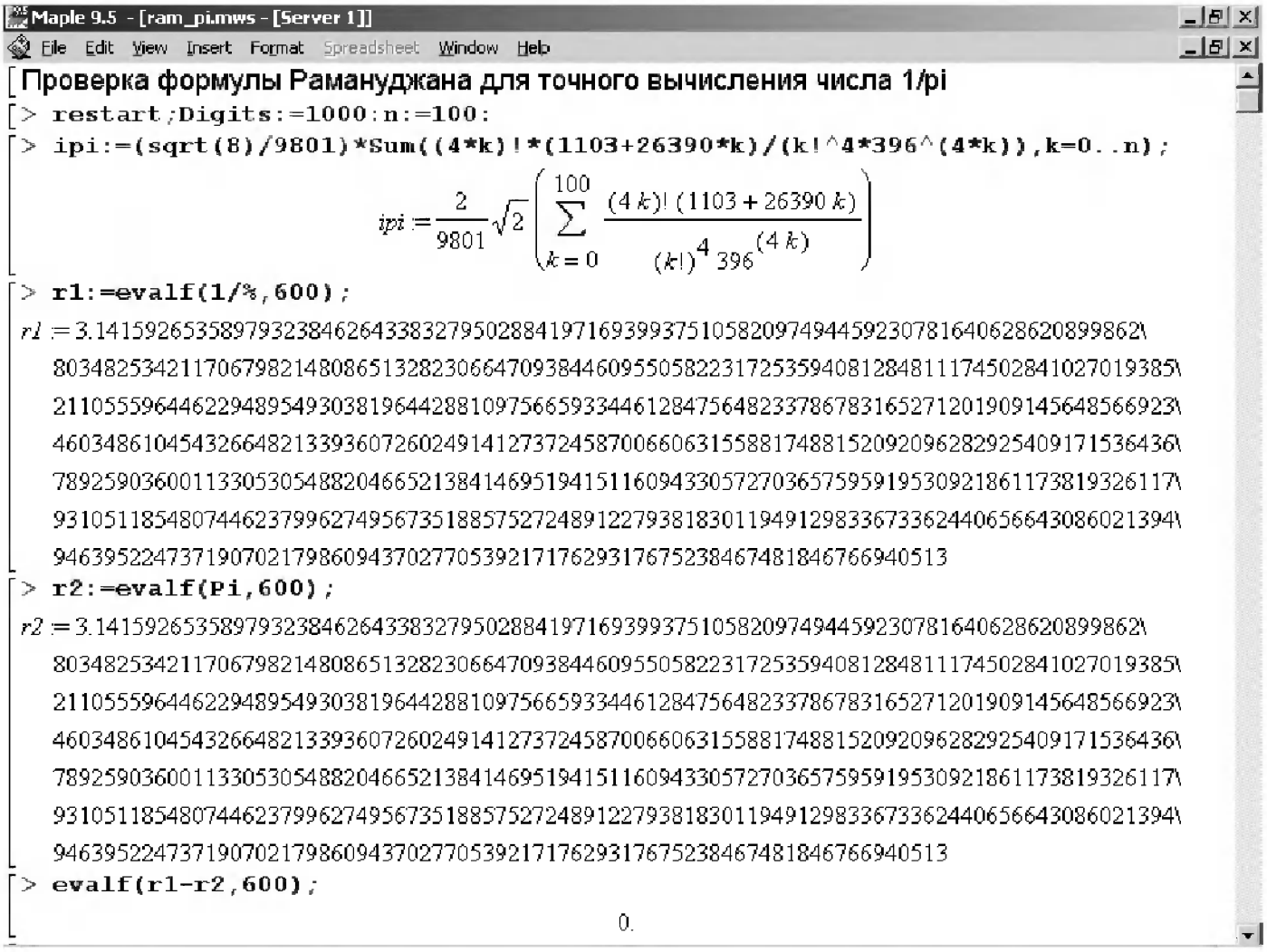


Рис. 2.1. Проверка вычислений по формуле Рамануджана

У инженеров формула Рамануджана может вызвать приступ головной или зубной боли. Уж больно несуразна она с первого взгляда. О какой точности можно говорить, если на подавляющем большинстве языков программирования корень квадратный из двух, факториал и степень вычисляются всего с 8–15 точными знаками?

Но Maple благодаря встроенному аппарату точной арифметики способен обеспечить эффективную проверку и реальное применение подобных формул. В нашем случае мы ограничились случаем n = 100 (максимальное значение k) и прове-

ли вычисления «всего» 600 цифр числа π – с тем чтобы результаты вместились в один рисунок. И он говорит сам за себя – все цифры при вычислении числа π по формуле Рамануджана и по встроенному в Maple алгоритму полностью совпали, а вычисленная ошибка равна нулю!

Тут уместно отметить, что многие неопытные пользователи часто допускают ошибки, пользуясь невысокой точностью вычислений Maple по умолчанию. Подчас из этого следуют довольно «неожиданные» и обычно просто неверные выводы. Приведенный на рис. 2.1 пример, как и другие примеры, в этом отношении весьма показателен – он говорит о возможности точных вычислений, но при непременном условии правильного применения аппарата точной арифметики.

2.2.6. Работа с комплексными числами

Maple, естественно, как и другие СКМ, может работать с *комплексными числами* вида **z=Re(z)+I·Im(z)**. Мнимая единица в комплексном числе (корень квадратный из –1) обозначается как **I**. Функции `Re(z)` и `Im(z)` возвращают действительную и мнимую части комплексных чисел. На комплексной плоскости числа задаются координатами точек (x, y) – рис. 2.2.

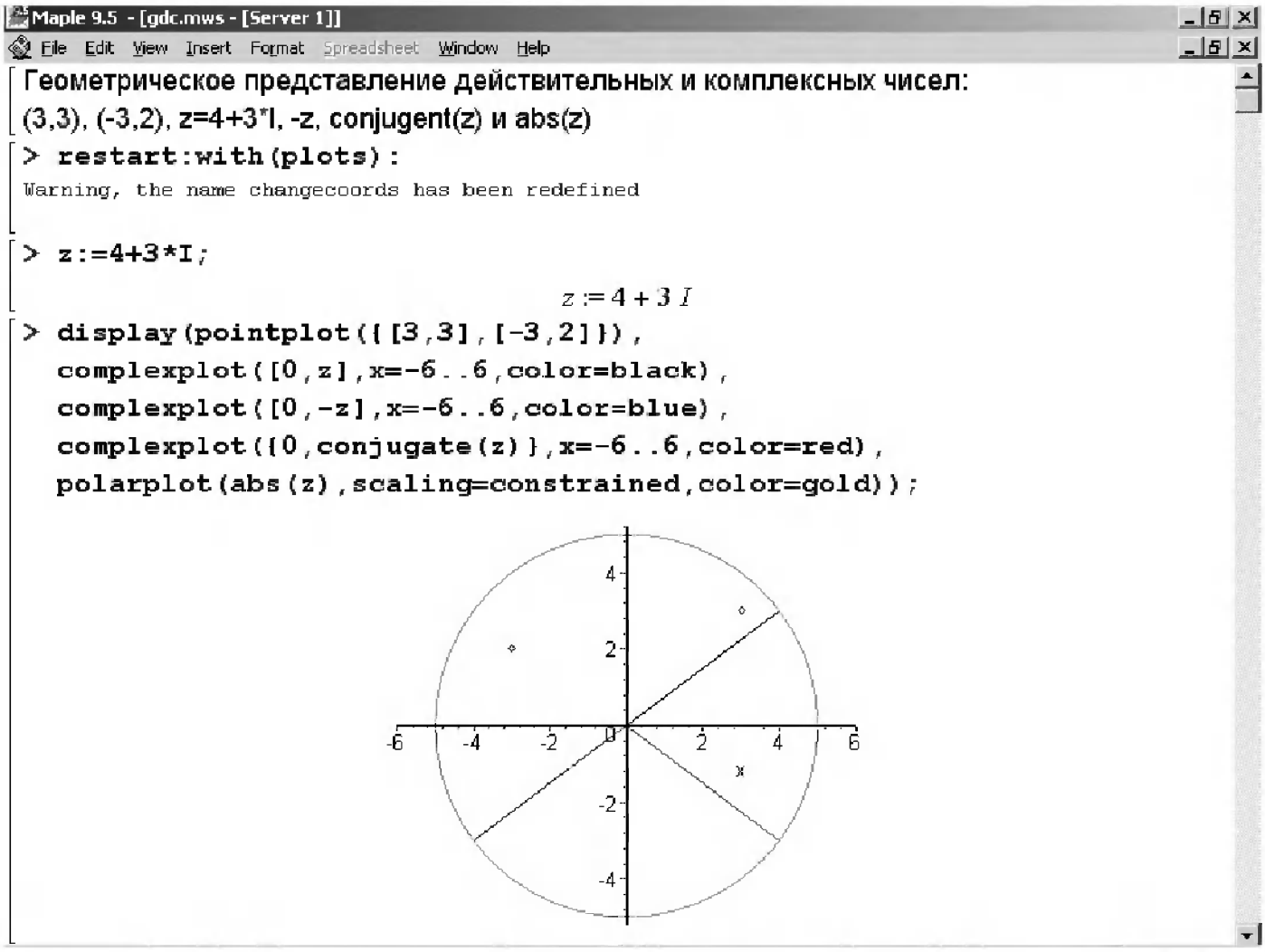


Рис. 2.2. Представление обычных и комплексных чисел на плоскости

Для представления чисел на рис. 2.2 используется функция `pointplot(list)`, где `list` – список координат точек. Эта функция становится доступной при подключении пакета `plots` командой `with(plots)`. Кроме того, использована функция вывода ряда графических объектов на один график – `display` (см. далее описание представления комплексных чисел).

Примеры задания комплексного числа и вывода его действительной и мнимой частей представлены ниже:

```
> a+b*I;
```

$$a + b\,I$$

```
> 1.25+Pi*I;
```

$$1.25 + I\pi$$

```
> Re(1.25+Pi*I);
```

$$1.25$$

```
> Im(1.25+Pi*I);
```

$$\pi$$

Комплексные числа обычно представляют на так называемой комплексной плоскости, у точек которой координата x задает действительную часть комплексного числа, а y (мнимая ось) показывает мнимую часть такого числа. На рис. 2.2 показано задание в виде радиус-векторов комплексного числа **z=4+3I**, **–z** и комплексно-сопряженного числа **4-3I**. А на рис. 2.3 показан пример вычисления корней уравнения **z^n=1** для случая **n=16** (другие случаи читатель может рассмотреть самостоятельно, просто изменив **n**). Нетрудно заметить, что корни уравнения – комплексные числа и что на комплексной плоскости они ложатся на окружность единичного радиуса.

Окружность радиуса abs(z)= $\sqrt{a^2 + b^2}$ представляет абсолютное значение комплексного числа **z=a+b*I**. Она является геометрическим множеством комплексных чисел, образованных концом вращающегося радиус-вектора числа **z** вокруг его начала в точке (0, 0) комплексной плоскости, иллюстрацией чего и является частный пример рис. 2.2. Позже мы рассмотрим ряд функций для работы с комплексными числами.

2.2.7. Контроль над типами чисел

Числа могут служить объектами ввода, вывода и константами, входящими в математические выражения. Функция `type(x, numeric)` позволяет выяснить, является ли `x` числом. Если является, то она возвращает логическое значение `true` (истина), а если нет, то `false` (ложь). Например:

```
> type(2,numeric);
```

true

```
> type(2.6,numeric);
```

true

```
> type(Pi,numeric);
```

false

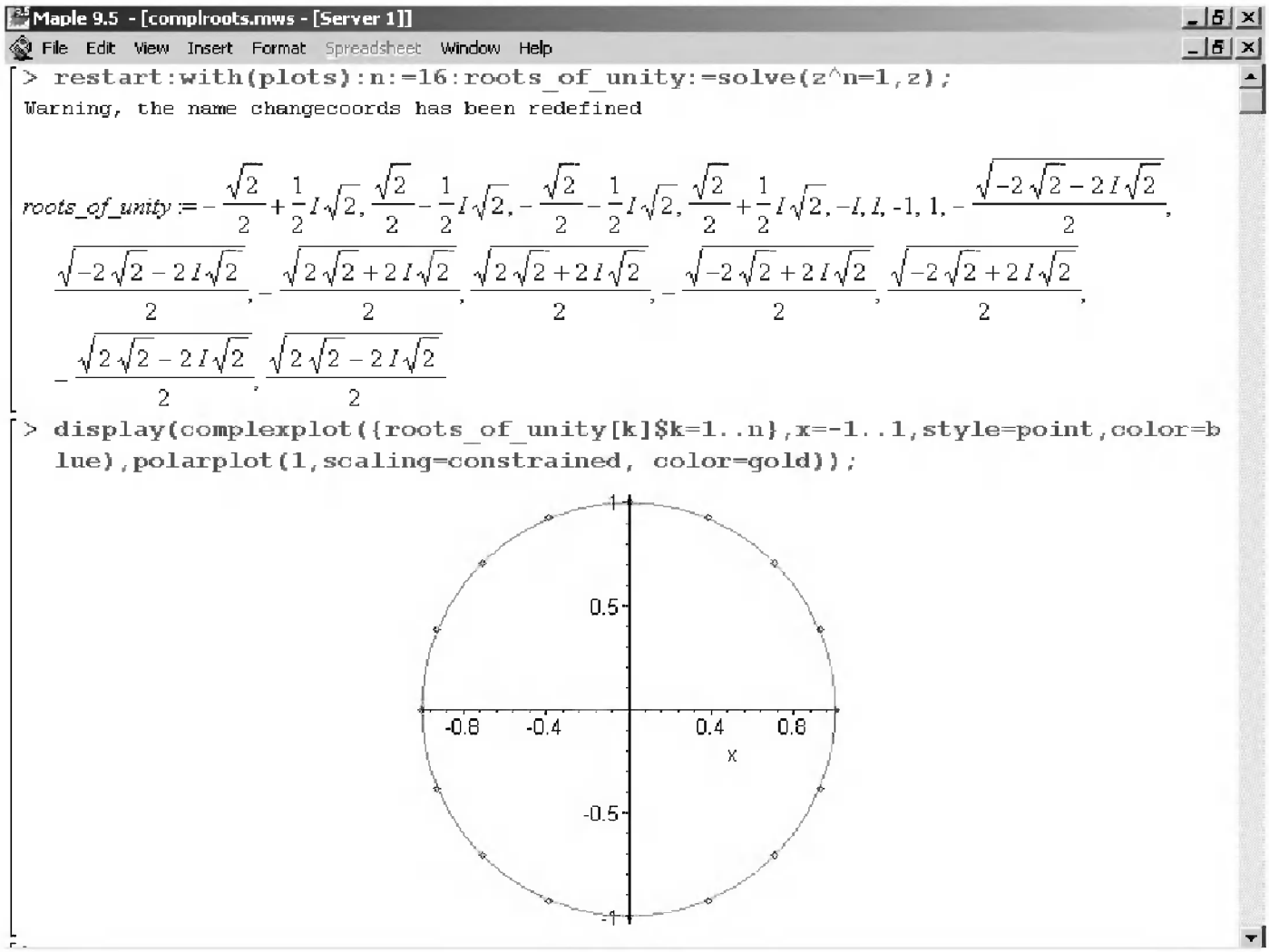


Рис. 2.3. Вычисление корней уравнения ***z^n=1*** *и расположение корней на комплексной плоскости*

```
> type(I,numeric);
```

false

```
> type(3^7,numeric);
```

true

```
> type(x^2,numeric);
```

false

Функции `type(x, integer)`, `type(x, rational)` и `type(x, fraction)` можно использовать для проверки того, имеет ли `x` значение, соответственно, целого числа, рационального числа или простой дроби:

```
> type(123,integer);
```

true

```
> type(123.,integer);
```

false

```
> type(123/456,rational);
```

true

```
> type(1./3,rational);
```

false

```
> type(1/2,fraction);
```

true

```
> type(0.5,fraction);
```

false

2.2.8. Преобразования чисел с разным основанием

В Maple возможна работа с числами, имеющими различное *основание* (base), в частности с двоичными числами (основание 2 – binary), восьмеричными (основание 8 – octal) и шестнадцатеричными (основание 16 – hex). Функция `convert` позволяет легко преобразовывать форматы чисел:

```
> convert(12345,binary);
```

11000000111001

```
> convert(%,decimal,binary);
```

12345

```
> convert(12345,octal);
```

30071

```
> convert(123456,hex);
```

1E240

```
> convert(%,decimal,hex);
```

123456

Помимо приведенных вариантов, функция `convert` имеет еще ряд других форм. С ними можно познакомиться с помощью справки по этой мощной функции. В дальнейшем будет приведен ряд других применений этой функции.

2.2.9. Пакет RealDomain для вычислений с действительными данными

В целом ряде случаев работа вычислителей Maple по умолчанию в области комплексных значений данных нежелательна, поскольку приводит к представлению результатов также в *комплексном виде*:

```
> restart:simplify( sqrt( x^2 ) );ln( -2 );solve(x^3-8=0,x);
```

$$\mathrm{csgn}(x)x$$

$$\ln(2)+\pi I$$

$$2,-1+\sqrt{3}\,I,-1-\sqrt{3}\,I$$

В связи с этим в Maple введен математический пакет расширения RealDomain, переводящий вычисления в область *реальных* значений данных. Вызов пакета обеспечивается следующим образом:

```
> restart:with(RealDomain);
```

[ℑ, ℜ, ^, arccos, arccosh, arccot, arccoth, arccsc, arccsch, arcsec, arcsech, arcsin, arcsinh, arctan, arctanh, cos, cosh, cot, coth, csc, csch, *eval*, exp,

expand, *limit*, ln, log, sec, sech, signum, *simplify*, sin, sinh, *solve*, sqrt, *surd*, tan, tanh]

Нетрудно заметить, что этот пакет переопределяет элементарные функции и некоторые другие вычислительные функции таким образом, что вычисления с ними ведутся только с реальными (вещественными, действительными) числами. Это видно из представленных ниже примеров:

```
> simplify( sqrt( x^2 ) );
```

$$|x|$$

```
> ln( -2 );
```

$$undefined$$

```
> solve(x^3-8=0,x);
```

$$2$$

2.2.10. Модификация графической функции plot

В прежних версиях Maple функция plot нередко отказывалась строить графики функций, значения которых были комплексными числами. Но уже в Maple 8 алгоритм построения графиков переработан. Теперь, если выражение, по которому строится график, в ходе оценивания дает мнимую часть, она отбрасывается, так что строится график только действительной части выражения. Малые по модулю мнимые части также нередко отбрасываются – впрочем, когда именно, не совсем ясно.

Рисунок 2.4 дает примеры этого. В верхней части документа строятся графики функции квадратного корня от x, логарифма и синуса. Нетрудно заметить, что для квадратного корня и логарифма строится и впрямь только та часть графиков, где значения функций действительны – при x положительном. Для $x < 0$ строится только график функции синуса, поскольку синус дает вещественные значения при любом x – как положительном, так и отрицательном.

Еще более интересен случай, представленный снизу рис. 2.4. Здесь функция задана как решение выражения f, которое дает корни в виде комплексных выражений. Несмотря на это, возможные части графика функции $f(x)$ строятся.

2.3. Сложные типы данных

Сложными являются такие типы данных, которые являются представлением множественных и подчас разнохарактерных объектов. Нередко такие данные включают, как часть себя, рассмотренные выше простые типы данных.

2.3.1. Массивы

Массивы относятся к числу сложных типов данных, поскольку представляют множественные числовые, строковые или комбинированные значения. Например, массив может представлять данные о числе выпущенных по месяцам автомо-

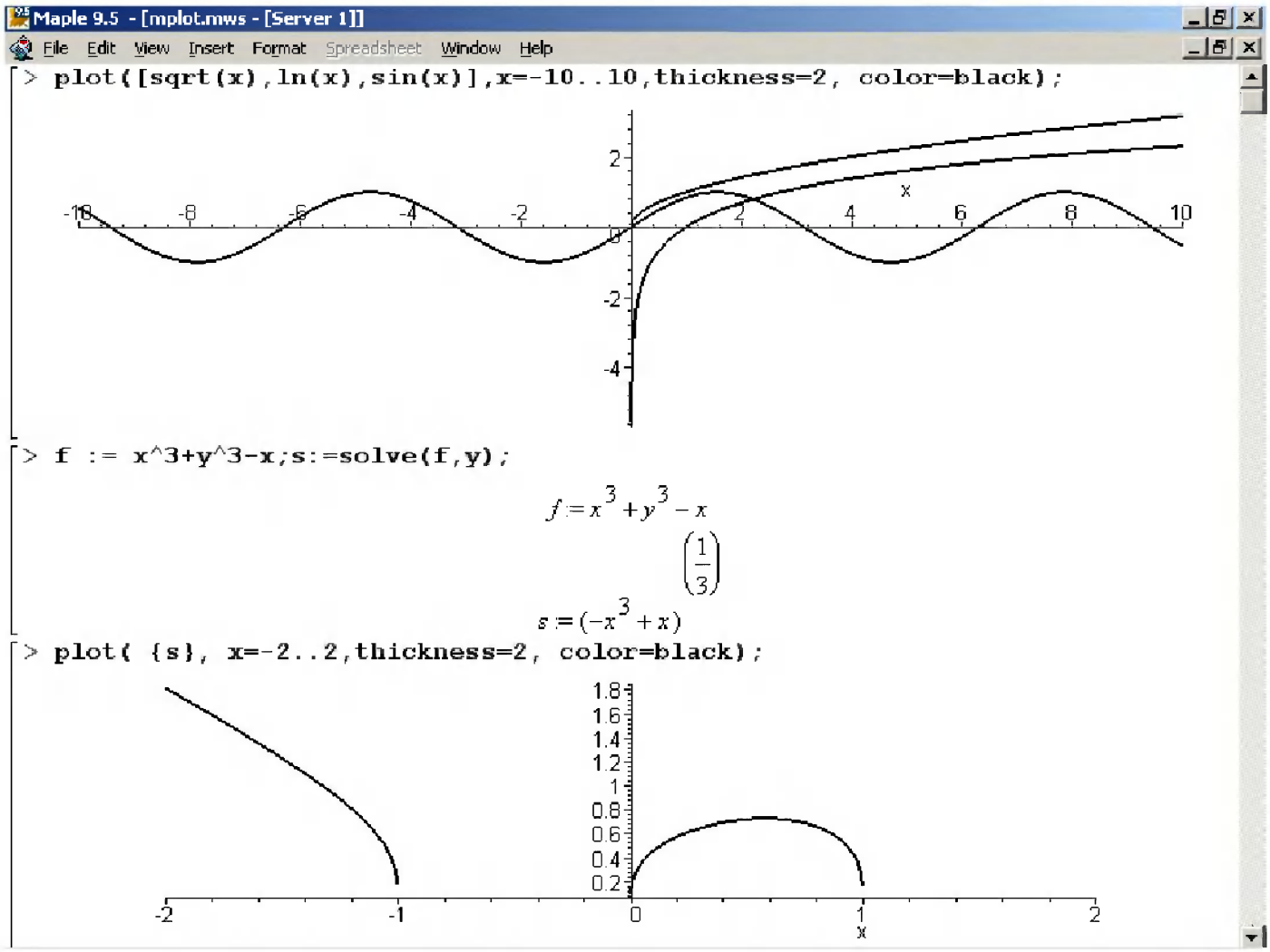


Рис. 2.4. Особые случаи применения функции plot

билей каким-то заводом. В этом случае говорят об одномерном массиве. А данные о числе книг в нескольких библиотеках (имеющих полки, шкафы с книгами и разные комнаты) можно представить, используя *многомерные массивы*, – число их размерностей больше двух.

Массивы можно характеризовать *размерностью* и *размером*. Так, вектор имеет размерность 1, матрица – размерность 2, куб с пронумерованными «кирпичами» – размерность 3 и т. д. Число элементов по каждой размерности характеризует ее размер, а произведение размеров по каждой размерности характеризует общий размер (число элементов) массива. Так, размер матрицы $m \times n$ определяется произведением числа элементов в строке m на число элементов в столбце n.

2.3.2. Векторы и матрицы

Самые распространенные одномерные и двухмерные массивы называются *векторами и матрицами* [66–68]. Массивы могут содержать как числовые, так и символьные данные. Например,

$\begin{bmatrix} 2 \\ 4 \\ 8 \end{bmatrix}$ – вектор-столбец с числовыми данными;

$(a\ b + c\ d)$ – вектор-строка с символьными данными.

$\begin{bmatrix} \sin(1) & 1 & "string" \\ 1 & a+b & 1 \\ 0 & 1 & \cos(1) \end{bmatrix}$ – матрица с элементами различного типа.

Как видно из этих примеров, элементами массивов могут быть числовые и строковые константы, переменные и математические выражения. Матрица с одинаковым числом строк и столбцов называется *квадратной матрицей*. Существует множество специальных типов матриц, которые будут рассмотрены позже.

2.3.3. Задание массивов

К сожалению, полного единообразия в средствах создания массивов, векторов и матриц в математических системах нет. В некоторых системах массивы можно задавать в квадратных скобках, в других – в фигурных, в третьих для этого используются специальные функции для задания массивов.

В последних версиях ряда математических систем (Mathcad, Maple, Mathematica и др.) предусмотрен удобный способ задания массивов с помощью их *шаблонов*, имеющихся в палитре математических спецзнаков (см. главу 4). В таких системах есть и специальные функции для задания векторов и матриц. В системах Mathematica 2/3 векторы и матрицы создаются с помощью списков в фигурных скобках – см. далее раздел о списках.

Многочисленные другие приемы создания и преобразования матриц будут описаны в дальнейшем по мере описания возможностей систем компьютерной математики по работе с массивами, векторами и матрицами.

Вообще говоря, массивы бывают статическими и динамическими. *Статические массивы* имеют жесткие (неменяемые) размеры и занимают в памяти ПК строго определенное место. *Динамические массивы* могут менять размер по ходу вычислений и изменять свое положение в памяти. Операции со статическими массивами обычно более быстрые, чем с динамическими массивами. Однако специфика всех систем символьной математики и универсальных математических систем такова, что размеры массивов и даже их ячеек могут меняться в широких пределах. Поэтому применяемые в них массивы, как правило, динамические.

2.3.4. Использование индексированных переменных массивов

Во всех математических системах массив задается именем – идентификатором, как и любая переменная. Однако массив имеет ряд элементов с определенным

порядком расположения. Порядковый номер элемента задается его *индексом* для вектора или двумя индексами для матрицы. Обычно существуют специальные системные переменные, задающие начальное значение индексов. Например, в системе Mathcad это переменная переменной ORIGIN, которая может принимать значение 0 или 1.

Таким образом, элементы массива являются *индексированными переменными*. Это значит, что помимо имени такие переменные имеют подстрочный индекс. Наиболее естественно он вводится в системах класса Mathcad, в которых i-й элемент вектора V задается как V_i. Таким образом, в этих системах индексы указываются как *подстрочные*, что принято и в обычной математической литературе.

Элементы матриц также являются индексированными переменными, имена которых совпадают с именами матриц. Например, можно задать матрицу

$$\mathbf{M} := \begin{bmatrix} a & 1 & 0 \\ 1 & a+b & 1 \\ 0 & 1 & b \end{bmatrix}.$$

Но в этом случае для каждой индексированной переменной указывают два индекса: один – для номера строки, другой – для номера столбца. Например, для указанной матрицы **M** средний элемент обозначается как $\mathbf{M}_{1,1}$, а последний – как $\mathbf{M}_{2,2}$. В системах Mathcad индекс вводится с помощью клавиши со знаком «[».

2.3.5. Списки и ранжированные переменные

Многие математические системы (Maple, Mathematica и др.) широко используют еще один сложный тип данных – *списки*, или *листы* (lists). Имеется множество функций для работы со списками, массивами и матрицами. Они будут рассмотрены в дальнейшем. В принципе, размерность массивов, создаваемых списками, не ограничена, и массивы могут быть многомерными. В системах Mathematica списки являются наиболее общим видом сложных типов данных.

В системе Mathcad последовательность чисел задается специальными *ранжированными переменными*. Например, если записать `n := 1..5`, то переменная `n` будет представлять список из целых чисел от 1 до 5 с шагом 1, то есть значения 1, 2, 3, 4 и 5. Возможность доступа отдельно к каждому значению отсутствует. Поэтому заменой вектора ранжированная переменная в Mathcad не является.

Для ввода знака `:=` надо нажать клавишу `:` (двоеточие), а для ввода знака `..` (две точки) – клавишу `;` (точка с запятой). Если требуется задать ряд чисел с шагом `d`, то ранжированная переменная записывается следующим образом:

```
X := Xstart, Xstart + d .. Xend
```

Здесь Xstart – начальное значение переменной X, Xend – конечное значение переменной X. Например, X := 1,1.25..2 дает ранжированную переменную X со значениями 1, 1.25, 1.5, 1.75 и 2.

Обратите внимание на то, что второй член в задании ранжированной переменной – это сумма ее начального значения с приращением d. Можно задать приращение отрицательным, но в этом случае значение Xstart должно быть больше, чем Xend. В явном виде приращение обычно не задается и фигурирует как разность чисел, разделенных запятыми, например в нашем примере X := 1,1.25..2 приращение есть 1.25 – 1 = 0.25.

2.3.6. Таблицы и их представление

Таблица – это двумерный набор тех или иных простых данных. Таблица имеет строки и столбцы. Некоторые математические системы, например Maple и Mathematica, позволяют задавать электронные таблицы как тип данных. Последние задают множественные данные с произвольной индексацией. Положение каждого элемента электронной таблицы характеризуется его адресом.

2.3.7. Создание наборов (множеств) в Maple

В Maple 9.5/10/11 любые выражения могут включаться в *наборы*, относящиеся к *множественным данным*. Такие наборы в виде *множеств* создаются с помощью фигурных скобок { }:

```
> {a,b,a,a,b,d,e,c,d};
```

$$\{a, b, c, e, d\}$$

```
> {10,2+3,4+4,8,5,1};
```

$$\{1, 5, 8, 10\}$$

```
> {"Hello","my","friend"};
```

$$\{friend, Hello, my\}$$

Отличительная черта множеств – автоматическое устранение из них повторяющихся по значению элементов. Кроме того, Maple расставляет элементы множеств в определенном порядке – числа в порядке увеличения значения, а символы и строки – в алфавитном порядке. Для множеств нет строгого математического определения, и мы будем считать их наборами, удовлетворяющими перечисленным выше признакам.

2.3.8. Создание и применение списков выражений

Для создания упорядоченных наборов – *списков* – служат квадратные скобки:

```
> [10,2+3,4+4,8,5,1];
```

$$[10, 5, 8, 8, 5, 1]$$

```
>[a,b,c,a,a,d,d,e];
```

$$[a, b, c, a, a, d, d, e]$$

Как нетрудно заметить, элементы списков преобразуются и выводятся строго в том порядке, в каком они были заданы. Списки широко применяются для задания векторов и матриц.

В ряде случаев, например при подготовке данных для двумерных графиков, возникает необходимость в подготовке парных списков – скажем, координат точек (x, y) графика. Для этого можно использовать функцию `zip(f, u, v)` или `zip(f, u, v, d)`. Здесь `f` – бинарная функция, `u`, `v` – списки или векторы, `d` – необязательное значение. Примеры применения функции `zip` даны ниже:

```
> X:=[1,2,3,4,5]:Y:=[3,2,1,1.5,2.5]:
> pare:=(X,Y)->[X,Y];
```

$$pare := (X, Y) \to [X, Y]$$

```
> CoordXY:=zip(pare,X,Y,2);
```

$$CoordXY := [[1, 3], [2, 2], [3, 1], [4, 1.5], [5, 2.5]]$$

```
> zip((X,Y)->X+Y,X,Y);
```

$$[4, 4, 4, 5.5, 7.5]$$

Рисунок 2.5 показывает применение этих средств для построения точек, представляющих множество действительных чисел на плоскости. Для этого использована функция `pointplot` из пакета `plots`.

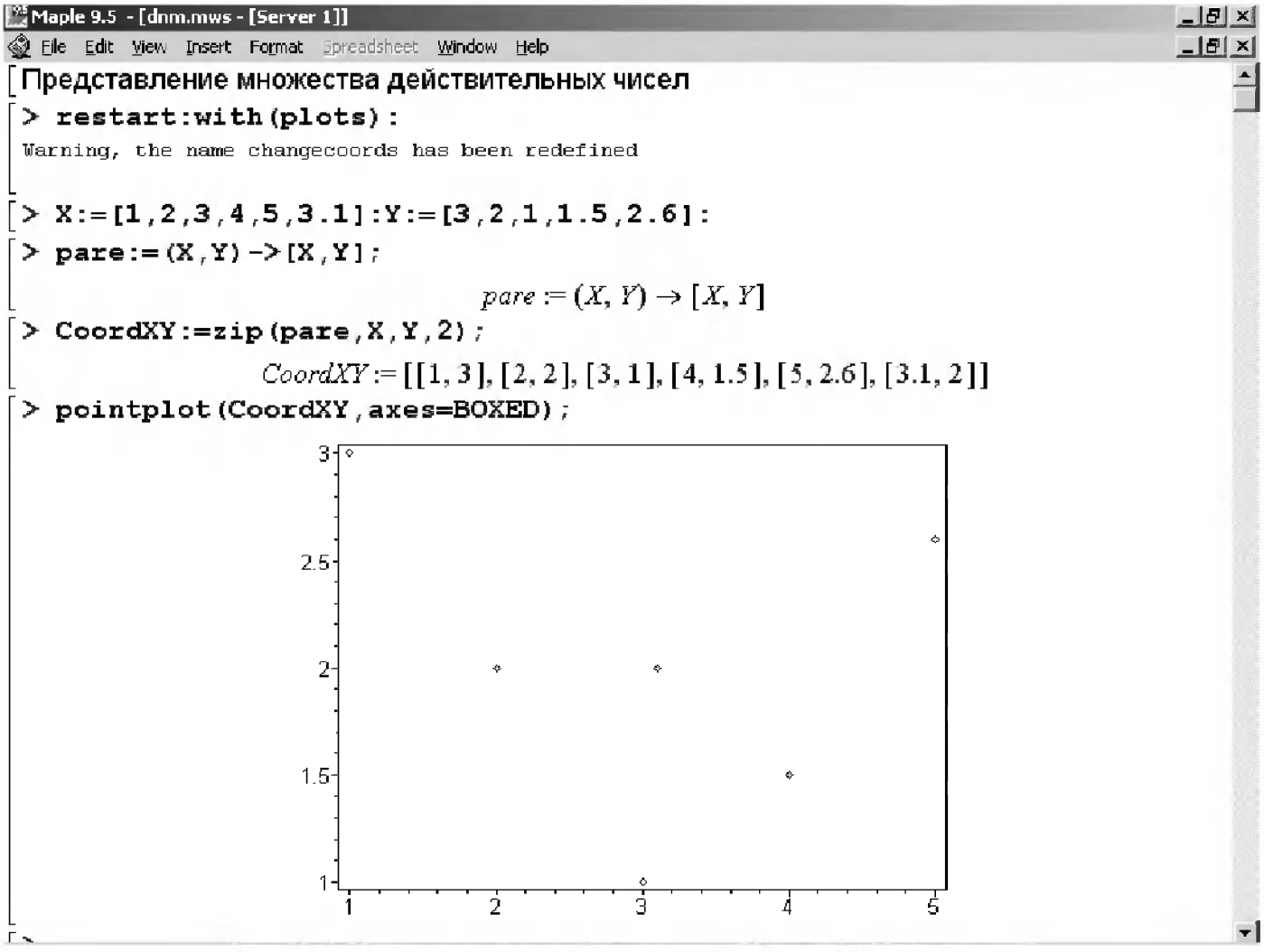


Рис. 2.5. Представление множества чисел на плоскости

2.3.9. Создание в Maple массивов, векторов и матриц

Как отмечалось, важным типом данных являются *списки* (lists). В Maple 9.5/10 они создаются с помощью квадратных скобок, например:

- `[1,2,3,4]` – список из четырех целых чисел;
- `[1.,2.34,5]` – список из двух вещественных и одного целого числа;
- `[a,b, ?Привет?]` – список из двух символов (переменных) и строковой константы;
- `[sin(x), 2*cos(x),a^2-b]` – список из трех математических выражений.

Для создания *векторов* (одномерных массивов) и *матриц* (двумерных массивов) служит функция `array`. Обычно она используется в следующих формах:

- `array[a..b,s1]` – возвращает вектор с индексами от `a` до `b` и значениями в одномерном списке `s1`;
- `array[a..b,c..d,s2]` – возвращает матрицу с номерами строк от `a` до `b`, номерами столбцов от `c` до `d` и значениями в двумерном списке `s2`.

Примеры задания вектора и матрицы представлены ниже:

- `array(1..3,[x,y,x+y])` – создает вектор с элементами x, y и $x + y$;
- `array(1..2,1..2,[[a,b],[c,d]])` – квадратная матрица $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$.

Для создания векторов может использоваться также конструктор векторов `Vector[o](d, init, ro, sh, st, dt, f, a, o)` с рядом опционально заданных параметров. В этой книге эта конструкция практически не используется. Векторы и матрицы можно также задавать с помощью угловых скобок:

```
> V:=<a,b,c>;
```

$$V := \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

```
> Vector[row]([a,b,c]);
```

$$[a, b, c]$$

```
> Vector[row](<a, b, c>);
```

$$[a, b, c]$$

```
> M:=<<a,b,c>|<d,e,f>>;
```

$$M := \begin{bmatrix} a & a \\ b & e \\ c & f \end{bmatrix}$$

Имеется множество функций для работы со списками, массивами и матрицами. Они будут рассмотрены в дальнейшем. В принципе, размерность массивов, создаваемых списками, не ограничена, и массивы могут быть многомерными.

2.3.10. Работа с построителем матриц Matrix Builder

Для интерактивного задания матриц и векторов уже в Maple 9.5 был введен ассистент Matrix Builder. При его применении открывается окно, показанное на рис. 2.6.

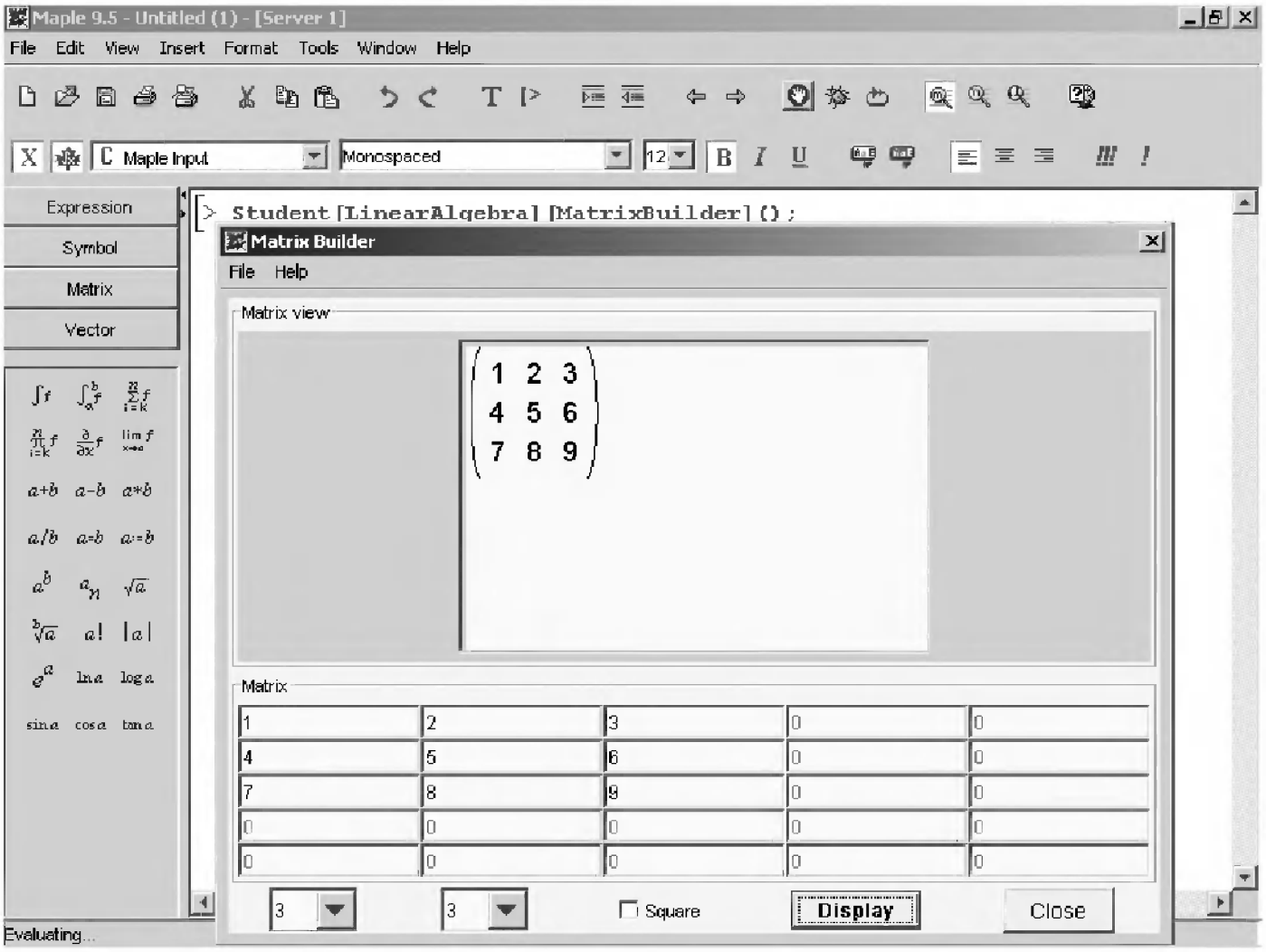


Рис. 2.6. Окно интерактивного ввода матриц и векторов ассистента Matrix Builder

В окне с помощью списков можно задать размер матрицы (или вектора, если один из размеров задан равным 1), определить рабочую область ввода с помощью кнопки Display и вводить значения элементов матрицы с помощью имеющегося шаблона. По завершении ввода всех элементов достаточно закрыть окно, и созданная матрица появится в документе.

2.3.11. Создание в Maple таблиц и их применение

Для создания в Maple таблиц служит функция `table`, которая при вызове в простейшем виде `table[]` создает шаблон пустой таблицы:

```
> table[];
```

$$\text{table}[\ ]$$

Пустая таблица резервирует память под данные. Когда параметром функции `table` является список выражений, он выводится в естественном порядке расположения элементов таблицы, но с произвольным порядком индексации:

```
> T:=table({1,2,Pi, "string"});
```

$$T := \text{table}([1 = 1, 2 = 2, 3 = \pi, 4 = string])$$

```
> T[3];
```

$$\pi$$

```
> S:=table([(one)=1,(two)=2,(three)=3]);
```

$$S := \text{table}([one = 1, three = 3, two = 2])$$

```
> S[1];
```

$$S1$$

```
> S[two];
```

$$2$$

```
> S[three];
```

$$3$$

```
> entries(S);
```

$$[1], [3], [2]$$

```
> indices(S);
```

$$[one], [three], [two]$$

В конце приведенных примеров показано, как можно выделить отдельные компоненты таблицы и вывести значения и индексы таблицы с помощью функций `entries` и `indices`. Следующие примеры показывают, что таблицу можно использовать для выполнения математических преобразований:

```
> F := table([sin=cos, cos=-sin]):
op(op(F));
```

$$[\cos = -\sin, \sin = \cos]$$

```
> F[cos](Pi/2);
```

$$-1$$

```
> F[sin](0);
```

$$1$$

```
> evalf(cos(Pi/2));
```

$$0.$$

```
> evalf(sin(0));
```

$$0.$$

Следует внимательно присмотреться к этим примерам – они демонстрируют замену функции косинуса на отрицательный синус и синуса на косинус.

2.3.12. Пакет ListTool для работы со списками

Для работы со списками в Maple имеется пакет расширения ListTool. Его вызов и состав новых определений – функций представлены ниже:

```
> with(ListTools);
Warning, the assigned name Group now has a global binding
```
[*BinaryPlace*, *BinarySearch*, *Categorize*, *DotProduct*, *Enumerate*, *FindRepetitions*, *Flatten*, *FlattenOnce*, *Group*, *Inerleave*, *Join*, *JoinSequence*, *MakeUnique*, *Pad*, *PartialSums*, *Reverse*, *Rotate*, *Sorted*, *Split*, *Transpose*]

Применительно к задачам данной книги применение этого пакета ограничено. Поэтому ограничимся несколькими примерами его применения:

```
> myList := [seq( ithprime(i), i=1..20 )];
```
myList :=
[2, 3, 5, 7, 11, 12, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71]
```
> BinarySearch( myList, 61, '<');
```
18
```
> Reverse(myList);
```
[71, 67, 61, 59, 53, 47, 43, 41, 37, 31, 29, 23, 19, 17, 13, 11, 7, 5, 3, 2]
```
> FindRepetitions(myList);
```
[ ]
```
> FindRepetitions([1,2,3,1,5,3]);
```
[1, 3]
```
> L := [0., .84, .91, .14, -.76, -.96, -.28, .66, .99, .41, -.54];
```
L := [0., .84, .91, .14, -.76, -.96, -.28, .66, .99, .41, -.54]
```
> M := [1., .54, -.42, -.99, -.65, .28, .96, .75, -.15, -.91, -
.84];
```
M := [1., .54, -.42, -.99, -.65, .28, .96, .75, -.15, -.91, -.84]
```
> DotProduct(L, L);
```
5.0063
```
> DotProduct(L, M);
```
.3162

Нетрудно заметить, что применение этих функций (как и ряда других) достаточно очевидно.

2.3.13. Структура разбросанных полиномов ряда переменных SDMPolynom

В Maple 9.5 введена новая структура данных – разбросанные полиномы ряда переменных – SDMPolynom (Sparse Distributed Multivariate Polynom). Примеры применения новой структуры:

```
A:= SDMPolynom(x^3 - 2 x^2 - 2 x + 4, [x])
B:=SDMPolynom(x*y^2+x^2*y^3 + (2 + I) y + 1, [x, y])
```

Из этих примеров форма задания таких полиномиальных структур очевидна. С ними можно проводить различные операции – вычислять степень по каждой переменной, выполнять арифметические операции и т. д.

2.4. Применение констант

2.4.1. Символьные и числовые константы

Константы – это простейшие именованные объекты, несущие заранее предопределенные значения. Их имена (идентификаторы) также заранее определены и не могут меняться. Подробную информацию о константах в Maple можно найти, исполнив команду `?constant`. Константы могут быть символьными, то есть представленными только своим символьным именем.

Обычные числовые константы не имеют имени и представлены просто числами, типы которых были указаны выше. Можно считать, что именем такой константы является само ее значение. Например, в выражении `2*sin(1.25)` числа `2` и `1.25` являются числовыми константами. При этом указание десятичной точки делает константу действительным числом – например, `2` – это целочисленная константа, а `2.`, `2.0` или `1.25` – это уже действительные константы.

2.4.2. Строковые константы

Строковыми константами являются произвольные цепочки символов, заключенные в двойные кавычки, например `"Hello"`, `"Привет"`, `"My number"` и т. д. Числа, заключенные в апострофы, например `"123456"`, также становятся строковыми константами, которые нельзя использовать в арифметических выражениях. Строковые константы представляют значения строковых переменных. В них можно использовать символы кириллицы, при условии что соответствующий шрифт имеется.

2.4.3. Встроенные в ядро Maple константы

Есть также ряд констант, которые правильнее считать заведомо определенными глобальными переменными:

```
> constants;
```

$$false, \gamma, \infty, true, Catalan, FAIL, \pi$$

Ниже указано их назначение:

- **false** – логическое значение «ложно»;
- γ или **gamma** – константа Эйлера, равная 0.5772156649...;
- ∞ или **infinity** – положительная бесконечность (отрицательная задается как `-infinity`);
- **true** – логическое значение «истинно»;
- **Catalan** – константа Каталана, равная 0.915965594...;
- **FAIL** – специальная константа (см. справку, выдаваемую по команде `?FAIL`);

- **I** – мнимая единица (квадратный корень из –1);
- π или **Pi** – представляет константу π = 3.141....

Любопытно, что в этот список не входит основание натурального логарифма – число *e*. В качестве этой константы рекомендуется использовать `exp(1)`. Она отображается как жирная прямая буква **E**. `A exp(1.0)` выводит 2.71828... (что и следовало ожидать).

2.4.4. Идентификация констант

Функции `type(x, constant)` и `type(x, realcons)` возвращают логическое значение `true`, если `x` представляет целочисленную или вещественную константу, и `false`, если `x` не является константой. Таким образом, эти функции можно использовать для идентификации констант, например:

```
> type(Pi,constant);
```

true

```
> type(1/2,constant);
```

true

```
> type(x/y,constant);
```

false

```
> type(x*y,realcons);
```

false

2.4.5. Защита идентификаторов констант

Имена встроенных констант (как и имена функций) защищены специальным атрибутом `protected`. Поэтому (без его снятия) константам нельзя присваивать какие-либо значения:

```
> Pi;
```

π

```
> Pi:=1;
Error, attempting to assign to 'Pi' which is protected
> gamma;
```

γ

```
> gamma:=10;
Error, attempting to assign to 'gamma' which is protected
```

Стоит упомянуть о такой экзотической возможности, как задание в Maple собственных констант путем описания алгоритма генерации входящих в константу цифр. Примеры этого творчества можно найти на сайте фирмы Wateloo Maple.

2.4.6. Задание новых констант

Следующий пример показывает, как можно определить новую константу g и ввести ее в список встроенных констант:

```
> type(g,constant);
```

$$false$$

```
> constants:=constants,g;
```

$$constants := false, \gamma, \infty, true, Catalan, FAIL, \pi, g$$

```
> type(g,constant);
```

$$true$$

2.5. Работа с размерными величинами

2.5.1. Пакет поддержки размерных величин Units

В некоторых областях науки и техники, например в физике, широко используются размерные величины, у которых, помимо их значения, указываются единицы измерения. Довольно развитую поддержку таких расчетов обеспечивает пакет расширения Units. Он содержит следующие функции, список которых выводит команда

```
> with(Units);
```

Большинство функций этого пакета достаточно просты и даже очевидны.

2.5.2. Примеры работы с размерными величинами

Ограничимся несколькими характерными примерами их применения:

```
> convert(4.532, units, N/m^2, (lb*ft/s^2)/ft^2);
```

$$3.045363395$$

```
> convert(W, dimensions), convert(W, dimensions, base);
```

$$power, \frac{length^2\ mass}{time^3}$$

```
> with(Units[Standard]):
> distance := 3.5*Unit(ft) + 2.4*Unit(m);
```

$$distance := 3.466800000\ [m]$$

```
> force := distance*Unit(lb)/Unit(s)^2;
```

$$force := 1.572514028\ [N]$$

```
> convert(force, units, lbf);
```

$$.3535152166\ [lbf]$$

```
> V := i*R;
```

$$V := iR$$

```
> eval(V, [i = 2.3*Unit(mA), R = 50.0*Unit(uOmega)]);
```

$$.1150000000\ 10^{-6}\ [V]$$

```
> convert(%, units, nV );
```

$$115.0000000\ [nV]$$

2.5.3. Применение ассистента преобразования размерных величин

В Maple 9.5/10/11 преобразования размерных величин упрощаются применением ассистента Unit Converter... (рис. 2.7), который преобразует значение Value размерной величины с размерностью Dimension с одной величины в другую. На рис. 2.7 дан пример преобразования 1 фута в метрическую величину (1 фут = = 0,3048 м).

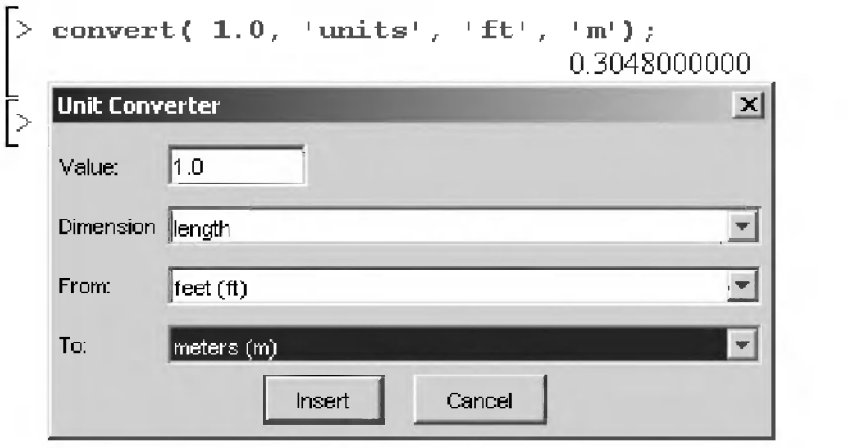


Рис. 2.7. Окно ассистента – конвертора размерных величин

Списки Dimension, From и To задают наименование размерной величины, начальную и конечную единицы ее измерения. Нетрудно заметить, что для преобразования используется функция **convert**.

2.5.4. Пакет научных констант ScientificConstants

В Maple 8 был добавлен пакет расширения для добавления и применения научных констант, химических элементов и свойств, что резко расширяет возможности применения системы в физических и химических расчетах. Вызов пакета ScientificConstants осуществляется командой:

```
> with(ScientificConstants);
Warning, the name Element has been rebound
```

[*AddConstant*, *AddElement*, *AddProperty*, *Constant*, *Element*, *GetConstant*, *GetConstants*, *GetElement*, *GetElements*, *GetError*, *GetIsotopes*, *GetProperties*, *GetProperty*, *GetUnit*, *GetVatue*, *HasConstant*, *HasElement*, *HasProperty*, *ModityConstant*, *ModifyElement*]

Нетрудно заметить, что функции пакета обеспечивают следующие возможности для констант (Constant), элементов (Eltment) и свойств (Property):

- **Add** – добавление;
- **Get** – вывод;
- **Has** – проверка на наличие объекта в пакете;
- **Modify** – модификация.

Функция

```
GetError(sc_obj)
```

возвращает значение ошибки, с которой задана константа, – объект sc_obj. Знание ошибки существенно при организации критичных к ошибкам научно-технических расчетов.

2.5.5. Работа с научными константами

Вызов всех научных констант осуществляется следующим образом:

```
> with(ScientificConstants):
> GetConstants();
```

E_h, F, G, G_0, K_J, M_{Earth}, M_{Sun}, N_A, Φ_0, R, R_{Earth}, R_K, R_∞, V_m, Z_0, a_0, a_e, a_μ, α,
b, c, $c_{1,L}$, c_1, c_2, e, ε_0, g, g_e, g_μ, g_n, g_p, γ_e, γ_n, γ_p, $gamma\_prime_h$,
$gamma\_prime_p$, h, k, l_p, $\lambda_{C,T}$, $\lambda_{C,\mu}$, $\lambda_{C,n}$, $\lambda_{C,p}$, λ_C, m_p, m_T, m_α, m_d, m_e,
m_h, m_μ, m_n, m_p, m_u, μ_0, μ_B, μ_N, μ_d, μ_e, μ_μ, μ_n, μ_p, mu_prime_h, mu_prime_p,
n_0, r_e, σ, σ_e, $sigma\_prime_p$, t_p

Теперь уточним данные по константе g – ускорению свободного падения:

```
> Constant( g );
```

$$\text{Constant}(g)$$

```
> GetValue( % ); GetUnit( %% );
```

$$9.80665$$

$$\left[\frac{m}{s^2}\right]$$

```
> Units:-UsingSystem();
```

$$\text{SI}$$

```
> Constant( g, units );
```

$$Constant_{SI}(g)\left[\frac{m}{s^2}\right]$$

А теперь проверим, есть ли в пакете константы g и edu:

```
> HasConstant(g);
```

$$true$$

```
> HasConstant(edu);
```

$$false$$

и вычислим погрешность, с которой задана константа G:

```
> GetError(Constant(G));
```

$$0.10\ \ 10^{-12}$$

2.5.6. Вызов списка и свойств химических элементов

Для вызова имен всех элементов периодической таблицы Менделеева можно использовать следующую команду:

```
> GetElements();
```

Она выводит список общепринятых обозначений химических элементов: H, He, Li, ..., Uuh. Выведем, к примеру, характеристики элемента A – алюминия:

```
> GetElement( Al, name, meltingpoint, boilingpoint );
```

13, *name* = *aluminum*,
meltingpoint = [*value* = 933.47, *uncertainty* = *undefined*, *units* = *K*],
boilingpoint = [*value* = 2792., *uncertainty* = *undefined*, *units* = *K*],

2.5.7. Применение пакета ScientificConstants

В справке системы Maple 9.5/10 можно найти примеры применения пакета ScientificConstants в химических и физических расчетах. Интересные примеры таких расчетов даны в документе «Applications of the ScientificConstants Package». На рис. 2.8 показано начало этого документа, в котором содержится пример на вычисление количества молекул, необходимого для получения 10 г вещества с химической формулой C_3H_60. Вначале математически синтезируется молекула этого вещества, ее вес конвертируется в систему единиц SI, и в конце находится число молекул вещества.

Приведенный документ является наглядным примером создания в среде Maple электронных документов, уроков и книг. Он построен с применением открывающихся и закрывающихся ячеек.

В другом примере (рис. 2.9) вычисляется энергия ионизации вещества и строится график ее зависимости от порядкового номера элемента вещества в таблице периодической системы элементов. График наглядно демонстрирует характерные почти периодические колебания энергии ионизации.

2.6. Функции для работы со строковыми данными

2.6.1. Неисполняемые программные комментарии в Maple

Часто возникает необходимость в задании программных комментариев. Любой текст после знака # в Maple рассматривается как невыводимый (неисполняемый)

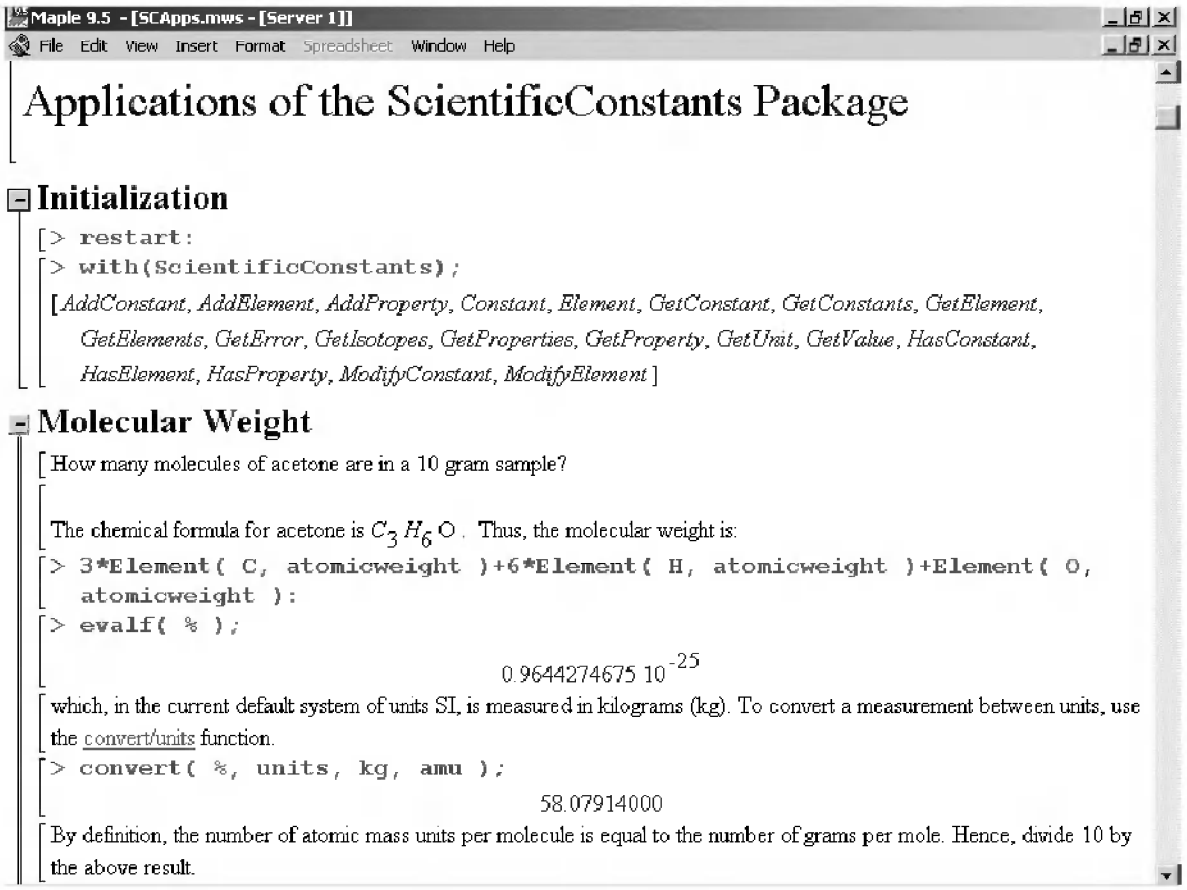


Рис. 2.8. Пример вычисления числа молекул для получения 10 г вещества C_3H_60

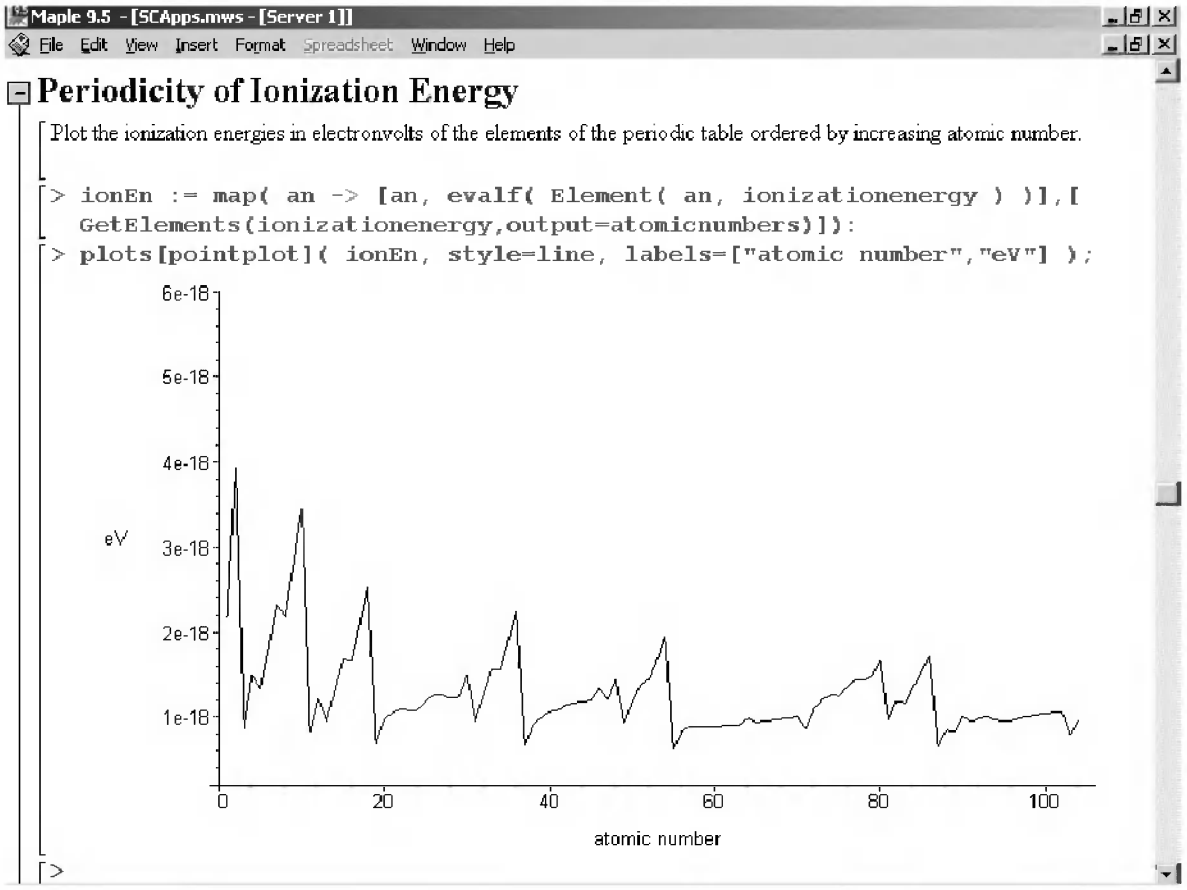


Рис. 2.9. Построение зависимости энергии ионизации вещества от его номера в таблице периодической системы элементов

программный комментарий – даже если это математическое выражение. При этом он не вычисляется. Например:

```
> 2+3;#Это пример. А это выражение не вычисляется: 4+5
```

$$5$$

Комментарии полезны в программах на Maple-языке и обычно используются для объяснения особенностей реализованных алгоритмов.

2.6.2. Контроль типа строковых данных

Строковые данные представляются совокупностью любых символов в двойных кавычках, например `"Привет"` или `"2+2"`. Для задания имен переменных могут использоваться обратные апострофы.

Для контроля объектов на принадлежность к строковым данным служит функция `type` с параметром `string`:

```
> str:= "Hello!";
```

$$str := Hello!$$

```
> type(Hello,string);
```

$$false$$

```
> type('2+3',string);
```

$$false$$

```
> char:=a;
```

$$char := a$$

```
> char:='a';
```

$$char := a$$

Из приведенных примеров видно, что контроль строкового типа осуществляется не очень строго – в частности, единичные символы рассматриваются как строковые и без заключения их в апострофы. В строках могут быть символы кириллицы, но гарантии в правильности обработки таких символов нет – надо мириться с тем, что Maple – англоязычная программа, и ее возможности в поддержке других языков ограничены.

2.6.3. Интерактивный ввод строк

Для интерактивного ввода строк можно использовать функцию `readline(filename)`, задав в качестве имени файла `terminal` или опустив имя файла. В этом случае ввод строки осуществляется с клавиатуры компьютера:

```
> s:=readline();
> Привет, мой друг!
```

s:= "Привет, мой друг!"

Полезно обратить внимание на то, что запрос в ходе интерактивного ввода может быть сделан на русском языке (если установленный для запросов шрифт имеет символы кириллицы). Нужно также, чтобы и шрифт строки вывода содержал кириллицу, иначе в строке вывода будет типичная «абракадабра» – смесь непонятных символов.

2.6.4. Обработка строк

Имеется ряд функций для работы со строками. Из них наиболее важны следующие:

- `lenght(str)` – возвращает число символов, содержащихся в строке `str`;
- `substring(str,a..b)` – возвращает подстроку строки `str` от `a`-го символа до `b`-го;
- `cat(str1,str2,...)` – возвращает строку, полученную объединением строк `str1`, `str2`, ... (альтернатива – оператор конкатенации в виде точки .);
- `SearchText(s,str)` – производит поиск подстроки `s` в строке `str` и при его успехе возвращает номер позиции `s` в строке `str` (при отсутствии `s` в `str` функция возвращает 0).

Примеры применения этих функций (в виде продолжения ранее приведенных примеров) представлены ниже:

```
> length(str);
```

6

```
> substring(str,1..3);
```

Hel

```
> substring(str,4..6);
```

lo!

```
> s:=cat("Hello"," my"," friend! ");
```

$s := Hello\ my\ friend!$

```
> SearchText(my,s);
```

7

```
> ss:= "Hello "||"my friend!";
```

$ss := Hello\ my\ friend!$

```
> seq(Name||i,i=1..4);
```

$Name1, Name2, Name3, Name4$

Эти функции дают достаточно средств для обработки данных строкового типа, которые можно применять не только для создания текстовых комментариев, но и для управления вычислительным процессом в программах.

2.6.5. Преобразование строки в математическое выражение

Часто возникает необходимость в интерактивном вводе математических выражений. Для ввода с запросом выражения используется функция `readstat(promt)`, где `promt` – строка с текстовым комментарием. Пример ее применения дан ниже:

```
> y:=readstat("Введите выражение ");
Введите выражение a*x^2+b;
```

$$y := ax^2 + b$$

Альтернативой может стать ввод строкового выражения с последующим преобразованием его в математическое с помощью функции `parse`:

```
> s:= "2+3*5";
```

$$s := = 2 + 3*5$$

```
> evaln(s);
```

$$s$$

```
> parse(%);
```

$$17$$

Обратите внимание на то, что функция `evaln` не смогла вычислить строковое выражение «`2+3`», поскольку оно не является числовым типом данных. Однако функция `parse` преобразовала это выражение в числовое, что и привело к его вычислению.

2.7. Переменные в Maple и их применение

2.7.1. Типы переменных

Как следует из самого названия, *переменные* – это объекты, значения которых могут меняться по ходу выполнения документа. Пока мы рассматриваем лишь *глобальные переменные*, доступные для модификации значений в любом месте документа. Тип переменной в системе Maple определяется присвоенной ей значением – это могут быть целочисленные (`integer`), рациональные (`rational`), вещественные (`real`), комплексные (`complex`) или строчные (`string`) переменные и т. д. Переменные могут также быть символьного типа (их значением является математическое выражение) или типа списка (см. далее). Для явного указания типа переменных используется конструкция

```
name::type
```

где `name` – имя (идентификатор) переменной, `type` – тип переменной, например целочисленный (`integer`), вещественный с плавающей точкой (`float`), с неотрицательным значением (`nonneg`), комплексный (`complex`) и т. д.

2.7.2. Назначение переменным имен

Переменные задаются своим именем – *идентификатором*, который должен начинаться с буквы и быть уникальным. Это значит, что ключевые слова языка Maple нельзя использовать в качестве имен переменных. Хотя имена ряда команд и функций можно использовать в качестве идентификаторов переменных, делать это крайне нежелательно. Ограничений на длину идентификатора практически нет – точнее, она не должна превышать 524 275 символов!

Имена переменных могут содержать одну букву (например, `x`, `Y` или `Z`) либо ряд букв (`Xmin` или `Xmax`). В любом случае имя переменной надо начинать с бук-

вы. Некоторые символы, например знак _, могут использоваться в именах (например, `Var_1`, `Var_2`). Нельзя, однако, вводить в имена переменных знаки, обозначающие операторы, например `a/b` или `a-b` будет истолковано как деление `a` на `b` или вычитание из переменной `a` переменной `b`.

Имена могут задаваться в обратных апострофах. При этом они просто тождественны именам без апострофов:

```
> var1:=123;var2:='Hello';
```

$$\text{var1} := 123$$
$$\text{var2} := \text{Hello}$$

```
> 'var1';'var2';
```

$$123$$
$$\text{Hello}$$

Строчные и прописные буквы в идентификаторах различаются, так что `Var1` и `var1` – это разные переменные.

Для проверки предполагаемого имени на уникальность достаточно выполнить команду `?name`, где `name` – выбранное имя. Если при этом откроется окно справки с этим именем, значит, оно уже использовано в Maple. Лучше воздержаться от его применения, так как связанная с этим именем команда или функция перестает работать, как только это имя закрепляется за какой-либо переменной.

2.7.3. Присваивание переменным значений

Для *присваивания* переменным конкретных значений используется комбинированный символ присваивания «`:=`», например:

- `n:=1` – переменной `n` присваивается целочисленное значение 1;
- `x:=123.456` – переменной `x` присваивается вещественное значение 123.456;
- `y:=17/19` – переменной `y` присваивается рациональное значение 17/18;
- `name:='Piter'` – переменной `name` присваивается строковое значение `'Piter'`;
- `expr:=2*Pi/3` – переменной `expr` присваивается значение выражения $2\pi/3$;
- `V:=[1,2,3]` – переменной `V` присваивается значение списка чисел [1,2,3];
- `M:=[[1,2,3],[4,5,6]]` – переменной `M` присваивается значение двумерного массива;
- `f:=x->x^2` – переменной `f` присваивается значение функции пользователя `f(x)=x^2`.

Правая часть выражения присваивания определяет тип переменной. Например, она может быть целочисленной, действительной, строковой, индексированной (элемент массива) и т. д.

2.7.4. Отмена операции присваивания и команда restart

При выполнении символьных операций часто нежелательно присваивание переменным значений перед такими операциями. Рассмотрим следующий пример:

```
> x:=10;
```

$$x := 10$$

```
> x;
```

$$10$$

```
> int(x^2,x);
Error, (in int) wrong number (or type) of arguments
```

Здесь не удалось вычислить интеграл с подынтегральной функцией `x^2` из-за того, что переменная `x` уже была определена ранее как целочисленная переменная со значением `10`, тогда как для вычисления интеграла она должна быть необъявленной или строковой (убедитесь в этом сами).

Для *отмены присваивания* надо использовать следующее выражение:

```
> x:='x';
```

$$x := x$$

Итак, заключение имени переменной в прямые апострофы ликвидирует присваивание. Так что запись `x:='x'` означает, что переменной `x` возвращается статус неопределенной переменной. Теперь можно вычислить интеграл:

```
> int(x^2,x);
```

$$\frac{1}{3}x^3$$

Можно сделать переменную `x` неопределенной и с помощью выражения вида `x:=evaln(x)`. Это поясняет следующий пример:

```
> x:=123;
```

$$x := 123$$

```
> x:=evaln(x);
```

$$x := x$$

```
> int(x^n,x);
```

$$\frac{x^{(n+1)}}{n+1}$$

Для отмены присваивания значений разом для всех переменных (и введенных функций пользователя) можно использовать команду `restart`:

```
> x:=5;
```

$$x := 5$$

```
> x^2;
```

$$25$$

```
> restart;
> x;
```

$$x$$

```
> x^2;
```

$$x^2$$

Важно отметить, что Maple сохраняет в памяти все определения и присваивания, которые были сделаны во всех загруженных в систему документах. Поэтому результаты вычислений в текущем документе могут зависеть от определений в других документах. Команда `restart` позволяет исключить эту зависимость.

2.7.5. Придание переменным статуса предполагаемых

Часто нужен определенный тип, или статус, переменных. Для придания переменным *статуса предполагаемых* используется функция `assume`:

```
assume(x,prop);
```

где `x` – переменная, имя или выражение, `prop` – свойство. Следующие примеры показывают применение функции `assume`:

```
> restart;
> assume(x,positive);
> x;
```

$$x\sim$$

```
> s:=x->sqrt(x);
```

$$s := \mathrm{sqrt}$$

```
> s(2);
```

$$\sqrt{2}$$

```
> s(2.);
```

$$1.414213562$$

```
> s(-2);
```

$$I\sqrt{2}$$

```
> is(x,positive);
```

$$true$$

```
> is(x,negative);
```

$$false$$

```
>> about(x);
Originally x, renamed x~:
  is assumed to be: RealRange(Open(0),infinity)
```

Обратите внимание, что в этом примере переменная `x` помечена как положительная и при выводе сопровождается знаком тильды `~`, как бы предупреждающем нас о ее особом статусе. Это не означает, что она не может принять отрицательное значение. Однако с помощью функции `is` можно убедиться в ее особом статусе и при необходимости программным путем исключить вычисления для $x < 0$. Кроме того, о свойствах переменной можно узнать с помощью функции `about(name)`.

Иногда к уже имеющимся признакам надо добавить новые. Для этого используется функция `additionally`:

```
> assume(a,nonnegative);
> additionally(a<=0);
> about(a);
Originally a, renamed a~:
  is assumed to be: 0
```

В этом примере переменной a вначале задан признак положительности, а затем a<=0. Оба признака удовлетворяются только при $a = 0$, что и подтверждает вывод информации о статусе этой переменной функцией about(a).

Предполагаемую переменную можно также изменить путем присваивания ей нового значения, противоречащего ее статусу:

```
> a:=123;
```

$$a := 123$$

```
> about(a);
123:
  All numeric values are properties as well as objects.
  Their location in the property lattice is obvious,
  in this case integer.
```

Для отмены переменным статуса предполагаемых используются те же приемы, что и при отмене присвоенного значения. Например, запись x:='x' отменяет статус предполагаемой для переменной x.

2.8. Работа с файлами и документами

2.8.1. Типы файлов

К числу широко распространенных данных относятся *файловые данные*, которые представлены *файлами*. Файлом называют имеющую имя упорядоченную совокупность данных, размещенную на том или ином носителе – обычно на жестком, гибком или компакт-диске.

В Maple используются файлы различных форматов, который указывается расширением файла (знак * означает произвольное имя файла):

- *.ms – файлы документов для систем с графическим интерфейсом (Windows/Macintosh);
- *.msw – файлы документов (Worksheets);
- *.txt – текстовые файлы (включая формат Maple-текст);
- *.tex – файлы в формате LaTeX;
- *.ind и *.lib – файлы библиотек;
- *.m – файлы внутреннего Maple-языка.

Работа с файлами документов удобна с применением команд меню и панели инструментов (см. главу 1). Предусмотрена возможность записи документов и в особом формате LaTeX, предназначенном для создания книг и статей по математике. Текстовые файлы (с расширением .txt) можно просматривать и редактировать текстовыми редакторами, работающими с ASCII-кодировкой.

Важно отметить, что даже при записи документов со сложными рисунками используется не прямая запись их растровой или векторной копии, а сохранение данных для построения графиков. Поэтому размеры файлов Maple невелики, и их легко передавать по современным средствам телекоммуникаций, например по сети Интернет. Они требуют небольшого свободного пространства на дисках для записи. Тем не менее чем сложнее график, содержащийся в документе, тем больше объем памяти, необходимой для хранения файла. Помимо обычных операций по работе с файлами (запись на диск и загрузка с диска), предусмотрены возможности распечатки документов принтерами различного типа.

2.8.2. Запись данных в файлы

Обмен информацией между Maple и внешней средой (к ней, кстати, относятся и другие программы) чаще всего осуществляется через файлы текстового формата, поскольку именно с такими файлами работают практически все программы. Для *записи* данных в файл служит оператор `writedata`:

```
writedata[APPEND](fileID, data)
writedata[APPEND](fileID, data, format)
writedata[APPEND](fileID, data, format, default)
```

Здесь `fileID` – имя или дескриптор файла данных, `data` – список, вектор или матрица данных, `format` – спецификация формата данных (`integer`, `float` или `string`), `default` – процедура, задающая запись нечисловых данных, например:

```
writedata(F,A,float,proc(f,x)fprintf(f,'CMPLX(%g,%g)',Re(x),Im(x)) end);
```

Необязательный указатель `APPEND` используется, если данные должны дописываться в уже созданный файл.

2.8.3. Считывание данных из файлов

Считывание данных из файла `filename` обеспечивает функция `readdata`:

```
readdata(fileID, n)
readdata(fileID, format, n)
readdata(fileID, format)
```

Здесь `n` – целое положительное число, задающее число считываемых столбцов.

Ниже представлены примеры этих операций:

```
> data:=array([[1,2,3],[4,5,6],[7,8,9]]);
```

$$data := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
> writedata("C:\\mydata.txt",data);
> restart;
> data;
```

$$data$$

```
> data:=readdata("C:\\mydata.txt",3);
```

$$data := [[1., 2., 3.], [4., 5., 6.], [7., 8., 9.]]$$

Maple имеет также типичные файловые операции:

- `writeto` – запись в файл;
- `appendto` – добавление к файлу;
- `open` – открытие файла;
- `close` – закрытие файла;
- `write` – запись в открытый файл;
- `save` – запись выражений в файл;
- `read` – считывание из файла.

Их реализация, однако, зависит от платформы, на которой установлена система, и от ее настройки.

2.8.4. Запись и считывание m-файлов

Основным способом записи различных объектов в файлы и считывания их из них является применение команд **save** и **read** применительно к файлам формата .m. Это наглядно иллюстрирует рис. 2.10.

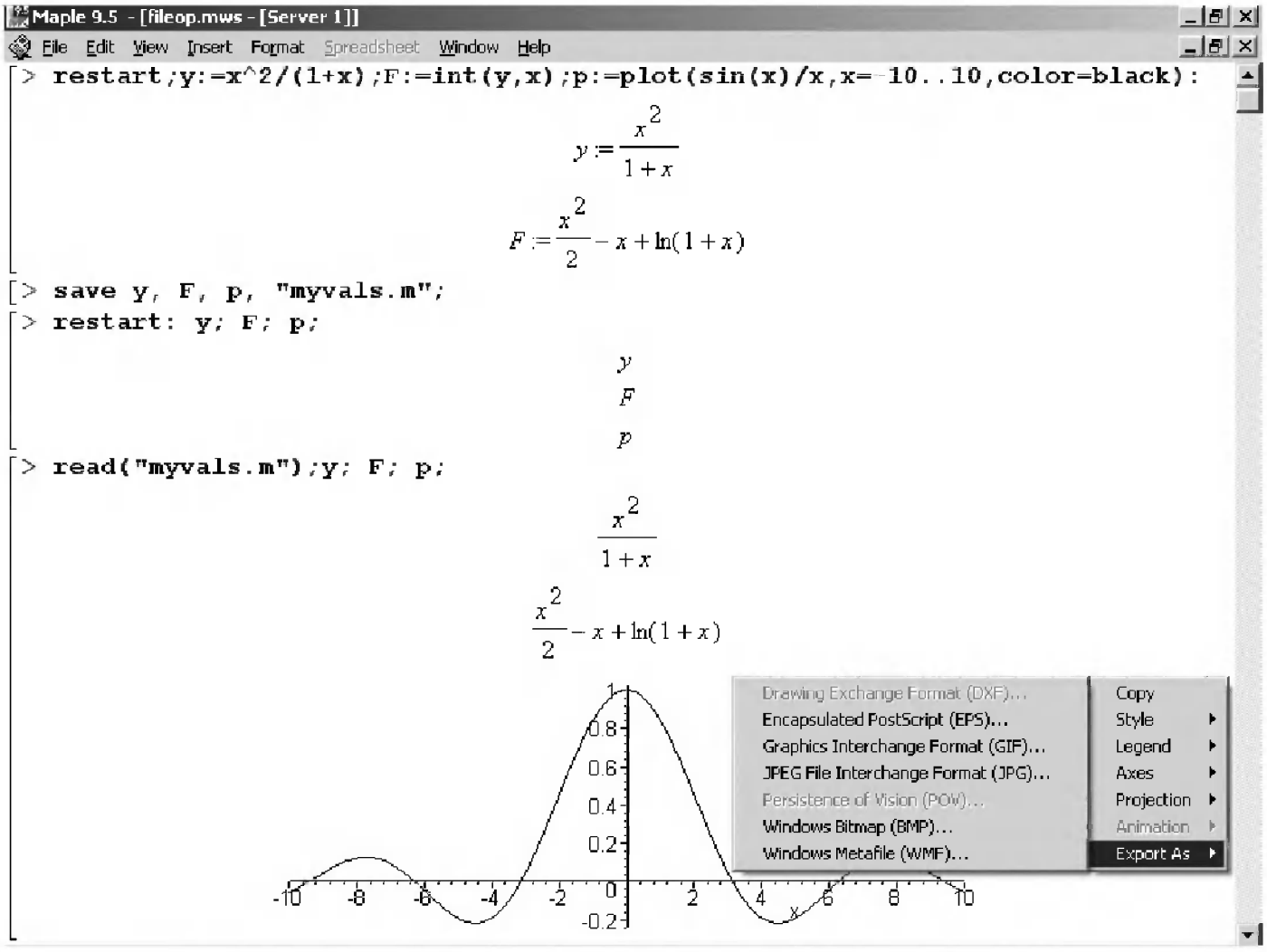


Рис. 2.10. Примеры работы с m-файлом

На этом рисунке вначале заданы три объекта y, F и p, представляющие собой обычную переменную с заданным значением, функцию, представляющую значение интеграла, и графический объект. Эти объекты записываются командой save в файл с именем myvals.m. Затем командой restart все определения объектов уничтожаются, поэтому вывод y, F и p просто повторяет имена этих неопределенных переменных.

В заключение командой read выполняется считывание объектов из файла myvals.m. Теперь вывод объектов дает их полное представление – для y и F в виде выражений, а для p – в виде графика. Рисунок 2.10 дает наглядное представление о возможностях применения m-файлов.

2.8.5. Запись в файлы графических объектов

Графические объекты могут быть записаны в файлы своих собственных форматов, например таких, как GIF, BMP, JPG и др. Удобнее всего для этого использовать контекстное меню правой клавиши мыши, показанное на рис. 2.10 в правой части графика. Команда **Export As...** открывает окно с перечнем возможных форматов графических файлов. После выбора нужного формата появляется стандартное окно сохранения файлов, в котором можно задать нужное имя файла и завершить операцию сохранения графического объекта в файле. В Maple предусмотрена возможность в формате GIF записывать рисунки с анимацией изображения.

2.9. Вывод в специальных форматах

2.9.1. Вывод в формате LaTeX

Maple имеет ряд средств для общения с другими программами. Часть из них, в основном относящаяся к обмену через файлы, уже была описана выше и в главе 1. Однако системы Maple 9/9.5/10 способны генерировать коды для прямого их включения в такие программы, причем не только математические. В ряде случаев вывод в специальных форматах полезен для оценки возможностей осуществления тех или иных вычислений или просто записи их в той или иной форме.

Для подготовки математических статей и книг широкое распространение получили редакторы TeX и LaTeX. Для подготовки выражений или файлов в *формате LaTeX* служит функция

```
latex(expr,filename)
```

Параметр `filename` не обязателен, если достаточно получить нужное выражение в ячейке вывода Maple:

```
> latex(a*x^2+b*x+c);
a{x}^{2}+bx+c
> latex(diff(x^n,x$2));
{\frac {{x}^{n}{n}^{2}}{{x}^{2}}}-{\frac {{x}^{n}n}{{x}^{2}}}
```

2.9.2. Вывод на языке Фортран

Язык Фортран вот уже многие десятилетия используется для программирования вычислительных задач. Накоплены обширные библиотеки решения таких задач на Фортране. Почитателей этого языка Maple порадует тем, что он позволяет готовить коды для программ на Фортране. Для этого вначале надо загрузить библиотечную функцию:

```
> with(codegen,fortran);
```

[*fortran*]

После этого может использоваться функция `fortran`:

```
fortran(expr,filename=str,optimized)
```

Два последних параметра не обязательны при выводе выражения `expr` в форме, присущей языку Фортран:

```
> fortran(a*x^2+b*x+c);
      t0 = a*x**2+b*x+c
> fortran(diff(x^n,x$2));
      t0 = x**n*n**2/x**2-x**n*n/x**2
```

Параметр `optimize` позволяет генерировать оптимизированные коды:

```
> fortran(a*x^2+b*x+c,optimized);
      t1 = x**2
      t4 = a*t1+b*x+c
```

При этом вычислительный процесс строится так, чтобы минимизировать число арифметических операций.

2.9.3. Вывод на языке C

Язык C (Си) также широко используется для решения вычислительных задач. Достаточно отметить, что сама система Maple создана на языке C. Для вывода на языке C вначале надо подключить соответствующую функцию:

```
> with(codegen,C);
```

[*C*]

Затем можно использовать функцию `C`:

```
C(expr,folename=str,optimized)
```

Например:

```
> C(diff(x^b,x$2));
      t0 = pow(x,1.0*b)*b*b/(x*x)-pow(x,1.0*b)*b/(x*x);
> C(diff(x^b,x$2),optimized);
      t1 = pow(x,1.0*b);
      t2 = b*b;
      t4 = x*x;
      t5 = 1/t4;
      t9 = t1*t2*t5-t1*b*t5;
```

Обширные возможности преобразования выражений в различные формы представляет функция `convert`. А функция `interface` позволяет управлять выводом.

2.10. Визуально-ориентированное создание документов

2.10.1. Вызов пакета Maplets

Начиная с Maple 8 в систему был введен новый пакет расширения Maplets, который обеспечивает построение *визуально-ориентированных* элементов интерфейса для документов системы. Этот пакет создан на основе применения средств языка Java, так что для его применения надо позаботиться, чтобы Java был инсталлирован на применяемом для работы с Maple компьютере.

О вызове пакета и его составе можно судить по приведенным ниже командам:

```
> restart; with(Maplets);
```

[*Display*, *Elements*, *Exemples*, *Tools*, *Utilities*]

Последний раздел Utilities был добавлен в реализацию Maple 9.5. Детальные данные о составе пакета можно получить, используя следующие команды:

```
> with(Maplets[Команда]);
```

Пакет позволяет выводить на экран множество диалоговых окон и иных средств интерфейса – начиная от простейших кнопок и заканчивая диалоговыми окнами вычисления интегралов и построения графиков по заданным функциям. Пакет основан на применении языка программирования Java, поэтому для его применения на ваш ПК необходима инсталляция Java.

2.10.2. Примеры создания визульно-ориентированного интерфейса

Рисунок 2.11 показывает вызов простейшего окна с запросом «Хочешь дальше?». Это окно используется для создания паузы в вычислениях – они возобновляются только после нажатия на клавишу с заданной надписью «Жми тут!». При этом управление передается следующей строке ввода. Нетрудно заметить, что в этом примере как надпись в окне, так и подпись на кнопке задаются пользователем.

```
> mymaplet := Maplet ([["Хочешь дальше?",
  Button("Жми тут!", Shutdown())]]):
  Maplets[Display](mymaplet);
```

Рис. 2.11. Создание кнопки для остановки и запуска вычислений

Более солидное действие производит функция вызова диалогового окна вычисления интегралов, показанная во фрагменте документа, представленного на рис. 2.12. Это окно вначале вызывает появление окна с запросом типа вычисляемого интеграла – определенного или неопределенного.

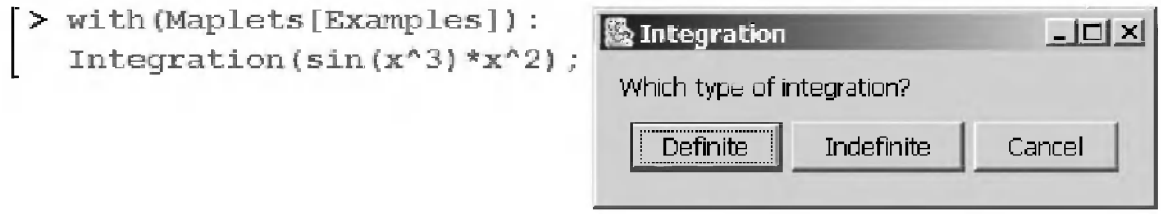


Рис. 2.12. Вызов окна задания интегралов

Задав, к примеру, вычисление неопределенного интеграла, можно получить окно с заданным интегралом. Но можно (см. рис. 2.13) задать в панели ввода и любое другое подынтегральное выражение, а также указать переменную интегрирования. Кнопка **Clear** очищает окно, а кнопка **Integrate** обеспечивает вычисление интеграла, что и показано на рис. 2.12. Если нажать кнопку **OK**, то вычисленное значение интеграла будет перенесено в строку вывода. А кнопка **Cancel** обеспечивает отказ от данной операции.

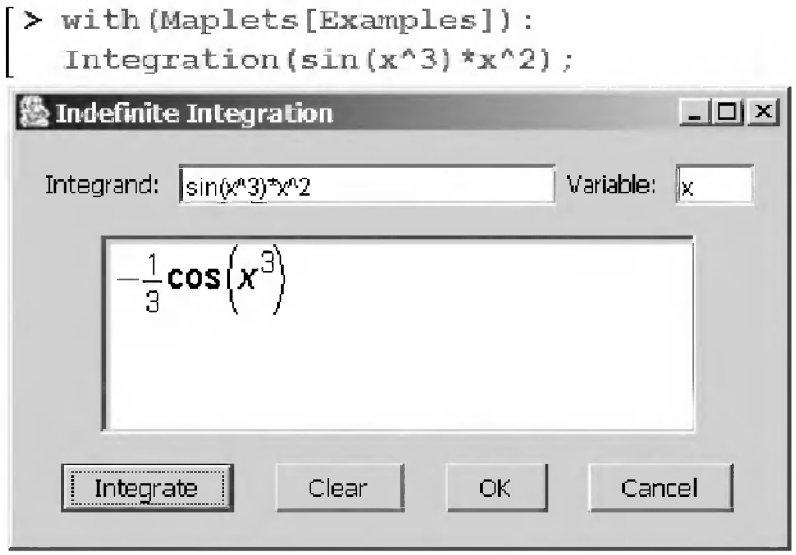


Рис. 2.13. Вывод окна задания и вычисления неопределенных интегралов

Еще один пример – рис. 2.14 – обеспечивает вывод диалогового окна построения графиков трехмерных объектов, представленных функцией двух переменных.

```
> with(Maplets[Elements]):
  maplet3d := Maplet([["Enter a function of 'x' and 'y':",
     TextField['TF3d']()], Plotter['PL1'](),
     [Button("Plot", Evaluate('PL1' = 'plot3d(TF3d, x = 0..10,
     y = 0..10)') ), Button("OK", Shutdown(['TF3d']))]]):
  result := Maplets[Display](maplet3d);
```

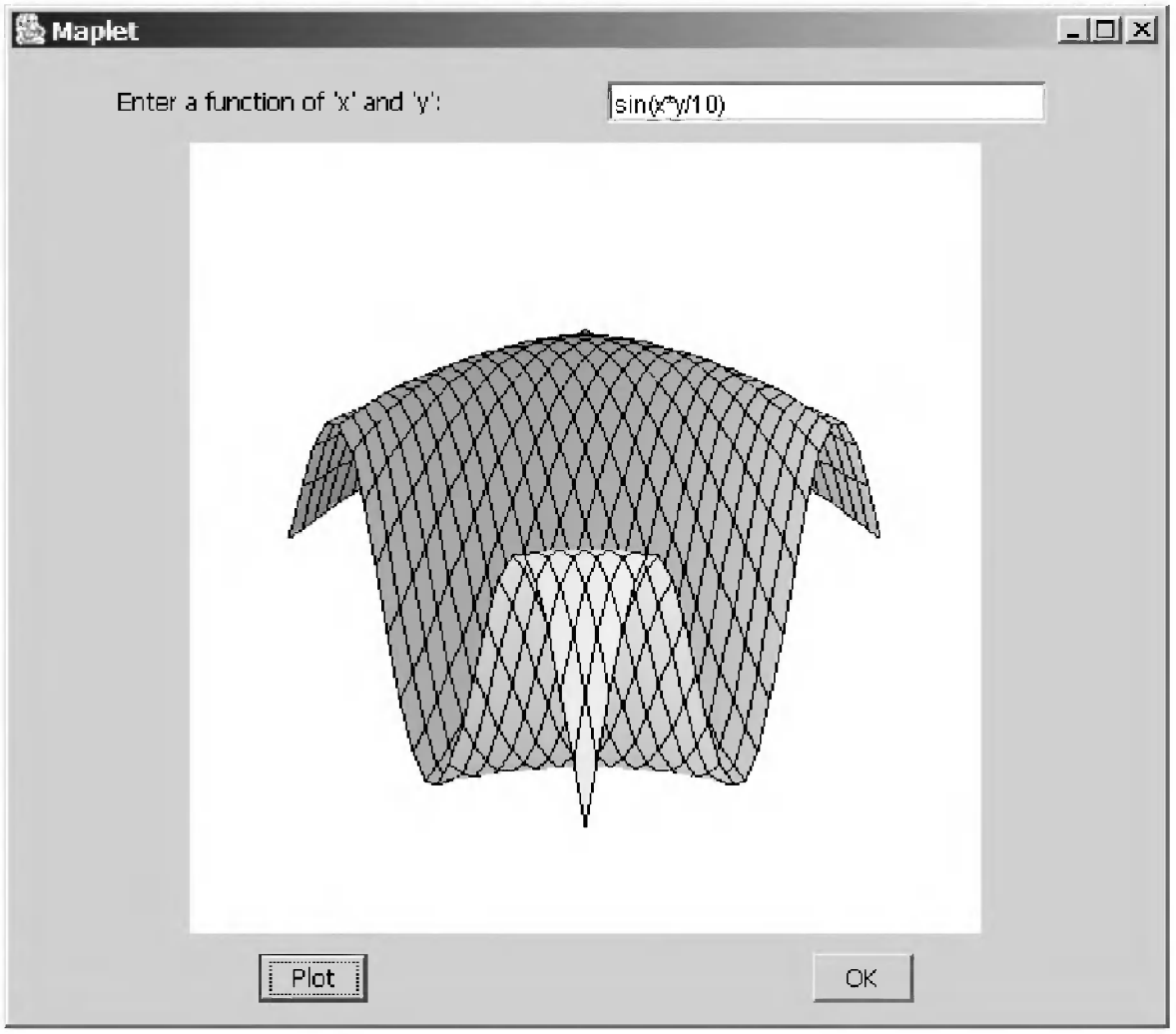


Рис. 2.14. Вызов и применение окна построения трехмерного графика заданной в его поле функции

2.10.3. Управление цветом

Пакет Maplets можно использовать для эффективного (и эффектного) управления цветом. Для этого достаточно использовать команду

```
  > with(Maplets[Examples]):
GetColor( 'title' = "Get Color" );
```

При исполнении этой команды появляется окно задания цвета, показанное на рис. 2.15. В этом окне имеются три вкладки для установки цвета в одной из трех цветовых систем: Swathes, HSB и RGB. Все они дают разные способы задания цве-

та в интерактивном режиме. Рисунок 2.15 демонстрирует наиболее распространенный способ задания цвета в системе RGB. При этом с помощью ползунковых регуляторов можно задать интенсивность каждой составляющей света: Red – красной, Green – зеленой и Blue – синей. В части окна Preview (Предварительный просмотр) можно наблюдать за изменением цвета текста, основы и пикселей.

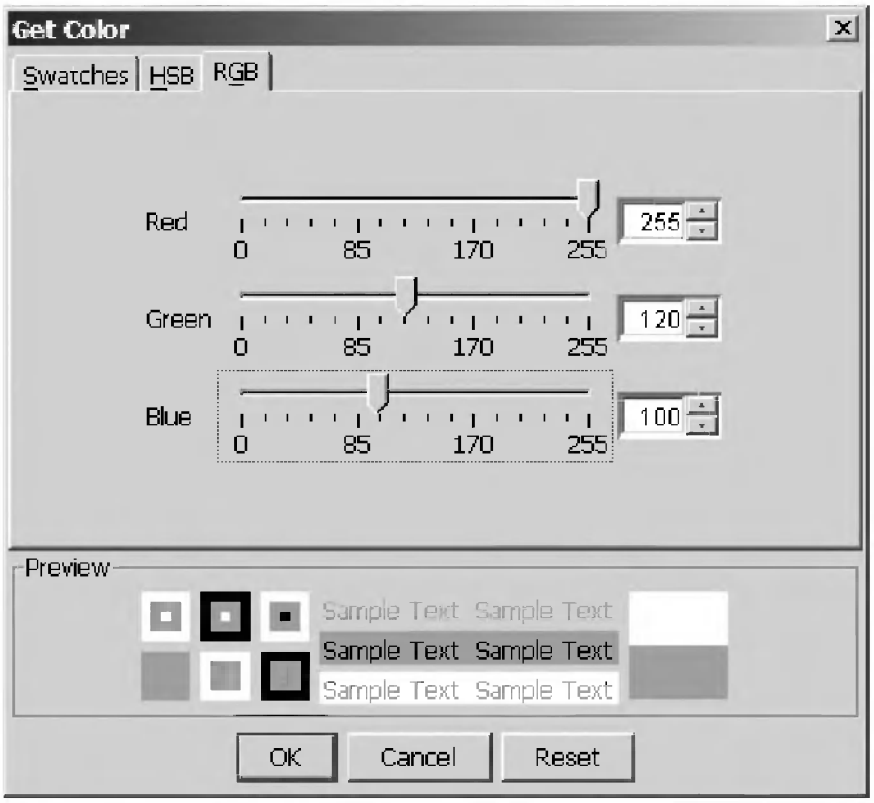


Рис. 2.15. Окно задания цвета

Если после установки подходящего цвета нажать кнопку OK, то будет сформирована строка с командами задания выбранного цвета. Для примера, показанного на рис. 2.15, эта строка имеет вид:

```
>
```

$$\mathrm{COLOR}\left(RGB,\ 1,\ \frac{8}{17},\ \frac{20}{51}\right)$$

Если использовать эту команду в любой графической функции, то объект (или часть объекта) будет окрашен в заданный цвет.

2.11. Типы данных в системе Mathematica

2.11.1. Работа с целыми числами

В Mathematica используются целые числа с различным основанием и десятичные числа с плавающей точкой (у нас запятой), представленные в различной нотации. Для вычисления чисел с произвольным *основанием* используется конструкция

```
Основание^^Число
```

Число должно быть записано по правилам записи чисел с соответствующим основанием. Для оснований более 10 для обозначений значений чисел используются буквы от a до z. Для шестнадцатеричных чисел используется их обычное обозначение, например:

```
16^^123abcde
 305839326
2^^1010111
 87
```

Для представления чисел с произвольным основанием *n* (до 32) используется функция **BaseForm[expr, n]** – возвращает выражение *expr* в форме числа с основанием *n*, которое указывается как подстрочный индекс.

Примеры на использование функции **BaseForm**:

```
BaseForm[87,2]
 1010111_2
BaseForm[305839326,16]
 123abcde_16
```

Для получения списков цифр различных целых чисел служит функция

```
IntegerDigits[n,b,len]
```

где n – число, b – основание и len – длина числовой последовательности, дополняемой слева нулями. Параметры b и len могут отсутствовать. Примеры применения этой функции представлены ниже:

```
IntegerDigits[1234]
 {1,2,3,4}
IntegerDigits[1234,2]
 {1,0,0,1,1,0,1,0,0,1,0}
IntegerDigits[10!,8]
 {1,5,6,5,7,4,0,0}
```

```
IntegerDigits[10!,10]
 {3,6,2,8,8,0,0}
IntegerDigits[10!,16]
 {3,7,5,15,0,0}
IntegerDigits[10!,10,12]
 {0,0,0,0,0,3,6,2,8,8,0,0}
```

Mathematica производит операции с числами изначально как с целыми. Однако установка значка разделительной точки означает, что число рассматривается как вещественное. Например, число 1 – целое число, но 1. – уже вещественное число. Для представления выражения expr в форме вещественного числа используется функция

N[expr] или **N[expr, число_цифр_результата]**

Примеры:

```
1/3
  1
  3
1./3
 0.333333
N[1/3]
 0.333333
N[2*Pi,50]
 6.2831853071795864769252867665590057683943 38
```

Mathematica имеет две системные переменные, позволяющие вывести значения максимально и минимально возможных значений чисел, с которыми оперирует система:

```
$MaxMachineNumber
  1.79769 × 10^308
$MinMachineNumber
  2.22507 × 10^-308
```

Обратите внимание на то, что функция **N[expr,m]** позволяет получить число с практическим любым числом цифр результата m. Разработчики последней версии Mathematica 4 утверждают, что это верно при количестве цифр результата до 1 млн, что с лихвой удовлетворяет требования подавляющего большинства расчетов и вычислений.

Функции **IntegerPart[x]** и **FractionalPart[x]** обеспечивают возврат целой и дробной частей вещественного числа x:

```
N[Pi]
 3.14159
IntegerPart[Pi]
 3
FractionalPart[Pi]
 - 3 + p
N[FractionalPart[Pi]]
 0.141593
```

Еще одна функция **RealDigits[x]** возвращает список реальных цифр результата и число цифр целой части x:

```
RealDigits[N[2*Pi]]
  {{6,2,8,3,1,8,5,3,0,7,1,7,9,5,8,6},1}
```

Есть множество и других функций для работы с вещественными числами. Они будут рассмотрены в дальнейшем. В Mathematica 4/5 функция **RealDigits** имеет расширенные формы, например **RealDigits[x,b,len,n]**. Для получения цифр мантиссы введена функция **MantissaExponent[x]** и **MantissaExponent[x,b]**.

2.11.2. Символьные данные и строки

Символьные данные в общем случае могут быть отдельными символами (например a, b, ..., z), строками (strings) и математическими выражениями expr (от expression – выражение), представленными в символьном виде.

Символьные строки задаются цепочкой символов в кавычках, например «sssss». В них могут использоваться следующие управляющие символы для строчных объектов:

\n новая линия (line feed);
\t табуляция.

Это иллюстрируется следующими примерами:

```
"Hello my friend!"
 Hello my friend!
"Hello\nmy\friend!"
 Hello
 my
 friend!
"Hello\tmy\tfriend!"
 Hello my friend!
```

Следует помнить, что управляющие символы не печатаются принтером и не отображаются дисплеем, а лишь заставляют их выполнять назначенные ими действия. Mathematica имеет множество функций для работы со строками, которые будут описаны в дельнейшем.

2.11.3. Выражения

Выражения в системе Mathematica обычно ассоциируются с математическими формулами, например:

Запись на языке Mathematica	Обычная математическая запись
2*Sin[x]	$2 \cdot \sin(x)$
2 Sin[x]	$2\sin(x)$
(a + b^2 + c^3) / (3*d – 4*e)	$(a+b2+c3)/(3d-4e)$
sqrt(2)	$\sqrt{2}$
Integrate[Sin[x],x]	$\int \sin(x)\,dx$

Для записи математических выражений используются как операторы, так и функции. Их особенности будут рассмотрены несколько позднее. А пока сразу отметим некоторые тонкости синтаксиса системы, используемого при записи арифметических операций:

- знак умножения может быть заменен пробелом;
- встроенные функции начинаются с большой буквы и обычно повторяют свое общепринятое математическое обозначение (за исключением тех, в названии которых есть греческие буквы, – они воспроизводятся латинскими буквами по звучанию соответствующих греческих букв);
- круглые скобки (...) используются для выделения частей выражений и задания приоритета их выполнения;
- параметры функций задаются в квадратных скобках [...];
- фигурные скобки используются при задании списков {...}.

2.11.4. Объекты и идентификаторы

В общем случае система Mathematica работает с *объектами*. Под ними подразумеваются математические выражения (expr), символы (symbols), строки из символов (strings), упомянутые выше числа различного типа, константы, переменные, графические и звуковые объекты и т. д.

Каждый объект характеризуется своим именем – *идентификатором*. Это имя должно быть уникальным, то есть единственным. Существуют следующие правила задания имен:

sssss – имя объекта, заданного пользователем;

Sssss – имя объекта, входящего в ядро системы;

$Sssss – имя системного объекта.

Итак, все объекты (например, функции), включенные в ядро, имеют имя, начинающееся с большой буквы (например, **Plus**, **Sin** или **Cos**). Относящиеся к системе объекты начинаются со знака $. Заданные пользователем объекты следует именовать прописными (малыми) буквами. Разумеется, под символами s...s подразумеваются любые буквы и цифры (но не специальные символы, такие как +, –, * и т. д.).

Полный список объектов, заданных в ядре системы, легко получить, используя команду ?* (ниже даны лишь начало и конец этого списка):

```
?*
 Abort
 AbortProtect
 Above
 Abs
 AbsoluteDashing
 ...................
 $Version
 $VersionNumber
```

Можно также получить список всех определений на заданную букву, используя команду **?S***, где S – любая буква латинского алфавита. Аналогичные воз-

можности представляет функция **Name["S"]**, например **Names["A*"]** дает список всех ключевых слов, начинающихся с символа A. Наконец, командой **?Name** можно вывести справку по любому определению с именем Name. Например, из нижеследующего

```
?Abs
  Abs[z] gives the absolute value
     of the real or complex number z. More...
```

ясно, что идентификатор Abs задает функцию **Abs[z]** для вычисления абсолютного значения комплексного числа.

С помощью выражения **?Name** можно проверить, является ли имя объекта name уникальным или оно уже использовано в системе:

```
?sin
  Information::notfound : Symbol sin not found. More...
?Sin
  Sin[z] gives the sine of z. More...
```

В первом случае ясно, что имя sin не использовано, а вот имя Sin уже зарезервировано – это функция вычисления синуса. Гиперссылка More..., введенная в Mathematica 5, позволяет получить дополнительную информацию переходом к нужному окну справки.

Всякий объект перед использованием должен быть определен (задан). Внутренние объекты уже заданы в ядре. Объекты пользователя последний задает в текстах своих документов (Notebooks). Объединенные в определенные группы документы принято называть пакетами применений. Они представлены файлами с расширением .ma, записываемыми в текстовом формате ASCII. Пакеты расширений представлены файлами с расширением .m.

2.11.5. Функции, опции, атрибуты и директивы

К важному типу объектов принадлежат *функции* – объекты, имеющие имя и список параметров, возвращающие некоторое значение в ответ на обращение к ним по имени с указанием в списке конкретных (фактических) значений параметров. В системах Mathematica 2/3/4 функции задаются в виде

Идентификатор_Функции[o1, o2, 03, ...],

где o1, 02, 03, ... – объекты – параметры, опции, математические выражения и т. д. Список входных параметров задается необычно – в **квадратных скобках**. В числе входных параметров могут быть специальные объекты – **опции**. Они задаются в виде:

Имя_опции->Значение_опции

Значением опции обычно является то или иное слово. Например, в функции построения графиков

```
Plot[sin[x],[x,0,20},Axes->None]
```

опция **Axes->None** указывает на то, что отменяется вывод координатных осей (Axes). Функция **Options[name]** выводит для функции с идентификатором name список всех возможных для нее опций. Некоторые функции, например **Sin**, могут вообще не иметь опций, другие, например **Solve**, могут иметь целый «букет» опций:

```
Options[Sin]
 {}
Options[Solve]
 {InverseFunctions→Automatic,MakeRules→False,Method→3,
  Mode→Generic,Sort→True,VerifySolutions→Automatic,
  WorkingPrecision→∞}
```

В последнем случае характер возвращаемого ими результата может сильно зависеть от значений опций. Назначение каждой опции мы рассмотрим в дальнейшем. В этой главе они нам пока не понадобятся.

Каждый объект может характеризоваться некоторой совокупностью своих свойств и признаков, называемых **атрибутами**. Функция **Attributes[Sin]** возвращает список всех атрибутов функции с именем name, например:

```
Attributes[Sin]
 {Listable,NumericFunction,Protected}
Attributes[Solve]
 {Protected}
```

Так, для функции синуса характерны три атрибута:

- **Listable** – указывает на применимость в списках и таблицах;
- **NumericFunction** – указывает на отношение к числовым функциям;
- **Protected** – указывает на то, что слово Sin защищено от какой-либо модификации.

Кроме того, в Mathematica имеется понятие функций – **директив**. Эти функции не возвращают значений, а указывают, как будут выполняться в дальнейшем функции, работа которых зависит от директив. Синтаксис директив-функций тот же, что и у обычных функций.

2.11.6. Применение констант

В системе Mathematica используются следующие поименованные константы:

- **ComplexInfinity** – комплексная бесконечность, которая представляет величину с бесконечным модулем и неопределенной комплексной фазой;
- **Degree** – число радиан в одном градусе, которое имеет числовое значение Pi/180;
- **E** – основание натурального логарифма с приближенным числовым значением 2.71828...;
- **EulerGamma** – постоянная Эйлера с числовым значением 0.577216...;
- **GoldenRatio** – константа со значением (1+Sqrt[5])/2, определяющая деление отрезка по правилу золотого сечения;
- **I** – представляет мнимую единицу Sqrt[–1];

- **Infinity** – «положительная» бесконечность (со знаком минус – «отрицательная» бесконечность);
- **Catalan** – константа Каталана 0.915966...;
- **Pi** – число, имеющее значение 3.14159... и дающее отношение длины окружности к ее диаметру.

Имеющие значение константы дают их в виде вещественных чисел:

```
{N[Degree], N[E], N[Pi]}
  {0.0174533,2.71828,3.14159}
{N[EulerGamma],N[GoldenRatio],N[Catalan]}
  {0.577216,1.61803,0.915966}
```

Константы в описываемой системе используются вполне естественно, так что от дальнейшего их описания можно воздержаться.

2.11.7. Физические константы и размерные величины

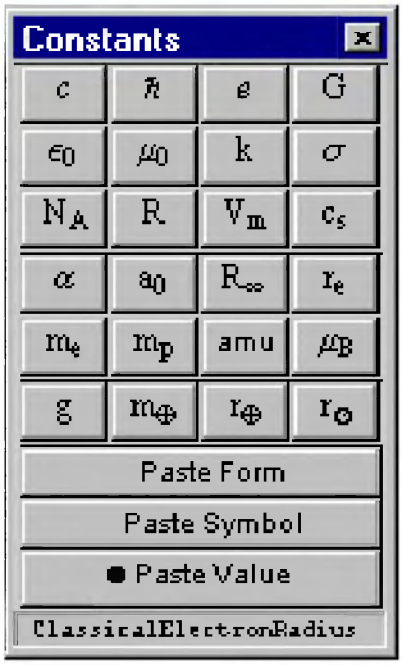


Рис. 2.16. Дополнительная палитра физических констант

Mathematica позволяет оперировать с *размерными величинами*, например **Meter** (метр), **Second** (секунда) и т. д. Последние могут стоять в числителе и в знаменателе выражений, представляющих размерные величины:

```
1 Meter
 Meter
5Meter
 5 Meter
0.5Second
 0.5 Second
```

Между значением размерной величины и единицей измерения знак умножения можно не ставить. Это видно из приведенных выше примеров.

Для облегчения ввода физических констант, представляющих собой размерные величины, в наборе файлов Mathematica в Интернете можно найти файл PhysicalConstants.nb. При его загрузке появляется дополнительная палитра физических констант, показанная на рис. 2.16.

Для ввода констант достаточно активировать соответствующую кнопку с нужной константой. Будет введено выражение, задающее константу.

2.11.8. Переменные в системе Mathematica 4/5

Переменными в системе Mathematica являются поименованные объекты, могущие в ходе выполнения документа неоднократно принимать различные значения – как численные, так и символьные. Имена переменных (*идентификаторы*)

должны быть уникальными, то есть не совпадать с именами директив, атрибутов, опций и функций в ядре системы. Имена переменных должны начинаться с буквы.

Заранее объявлять типы переменных не требуется. Тип определяется операцией присваивания переменной некоторого значения. Такой подход упрощает построение программ и естествен при использовании переменных в обычной математической литературе.

Без особых на то указаний переменные в системе Mathematica являются *глобальными*. Переменная появляется как действующий объект только после ее первого определения или задания. Определения переменных выполняются с помощью *операции присваивания*. Возможны типы присваивания:

x = value переменной x присваивается значение value;
x = y = value значение value присваивается переменным x и y;
x:=value отложенное присваивание переменной x значения value;
x =. с переменной x снимается определение.

Примеры (комментарий *In[...]* опущен):

- **g = Plot[Sin[x],{x,0,20}]** – переменной g присваивается значение в виде графического объекта;
- **y = 1 + x^2** – переменной y присваивается символьное значение в виде математического выражения (1 + x^2);
- **z = {1, 2, x, a + b}** – переменной z присваивается значение в виде списка, содержащего четыре элемента.

Различие в присваивании переменным значений с помощью знаков = и := иллюстрируют следующие примеры:

```
a=12
 12
b:=15
b
 15
```

Переменную или несколько переменных из списка можно сделать предполагаемыми с помощью функции-директивы:

Element[x, dom] или **Element[{x1, x2, x3,...}, dom]**

где опция dom задает область определения переменной или переменных в списке. Для Mathematica dom задает следующие типы переменных: Algebraics – алгебраическая, Boolean – логическая, Complexes – комплексная, Integers – целочисленная, Primes – простое число, Racional – рациональная, Reals – вещественная. Пример:

```
FullSimplify[{Re[Sin[x]],Re[ArcSin[x]],Sqrt[x^2]},Element[x,
Reals]]
  {Sin[x],Re[ArcSin[x]],Abs[x]}
```

Особо обратите внимание на то, что возможно *снятие с переменной определения* с помощью символа «**=.**». В символьной математике это очень полезная возможность, поскольку нередко переменные с одним и тем же именем в разных частях программы могут иметь разный смысл и представлять собой объекты, требующие значительных затрат памяти.

Учтите, что в Mathematica объекты сохраняются даже при использовании команды **New** при переходе к подготовке нового документа. Поэтому рекомендуется всякий раз снимать определения переменных, как только использование их завершается. Это предотвращает возникновение конфликтов между одноименными переменными и освобождает память.

2.11.9. Эволюция значений переменных и операции присваивания

Специфику математических выражений в системе Mathematica составляет возможность их изменения – *эволюции* – в соответствии с заложенными в ядро системы правилами математических преобразований. В итоге после эволюции значение выражения, которое присваивается переменной, может быть совсем иным, чем до эволюции. Поэтому в целом для определения переменных используют описанные ниже конструкции.

Основная функция **Set[lhs,rhs]** имеет аналогичные по действию упрощенные операторы:

- **lhs = rhs** – вычисляет правую часть rhs и присваивает ее значение левой части lhs. С этого момента lhs замещается на rhs всюду, где бы ни появилось;
- **{l1, l2, ...} = {r1, r2, ...}** – вычисляет ri и назначает полученные результаты соответствующим li.

Функция задержанного присваивания **SetDelayed[lhs,rhs]** может быть заменена аналогичными по действию операторами:

- **lhs := rhs** – назначает правой части rhs роль отложенного значения левой части lhs. При этом rhs содержится в невычисленной форме. Когда появляется lhs, оно заменяется на rhs, вычисляемое каждый раз заново.

При отложенном присваивании вывода нет, тогда как при обычном немедленном присваивании **lhs=rhs** значение rhs вычисляется немедленно и результат выводится в строку вывода.

Функция присваивания верхнего уровня **UpSet[lhs,rhs]** применяется в виде:

- **lhs^=rhs** – левой части lhs присваивает значение правой части hs и связывает это назначение с символами, которые появляются на первом уровне в lhs.

И наконец, задержанную функцию присваивания **UpSetDelayed[lhs,rhs]** может заменить оператор:

- **lhs^:=rhs** – величина rhs выполняет роль отложенного значения lhs, и связывается это присвоение с символами, которые появляются на первом уровне в lhs.

Отметим еще одну важную конструкцию:

- **SetOptions[s, name1->value1, name2->value2, ...]** – устанавливает для символа s указанные опции, определяемые по умолчанию.

Применение различных типов операций присваивания способствует большей гибкости системы. Различия между этими операциями с первого взгляда несущественны, но они принципиальны, и это станет понятно после более детального знакомства с символьными преобразованиями и практики работы с системой.

В Mathematica имеются системные переменные, значениями которых являются данные о системе и ее работе, например версия применяемой операционной системы, текущая дата, время в данный момент, машинная точность вычислений и т. д. Многие из таких переменных имеют отличительный знак $ перед своим именем.

2.11.10. Списки в системе Mathematica

Квадратные скобки в системе Mathematica используются для указания аргументов функций. Поэтому для создания списков применяются фигурные скобки:

- **{1,2,3}** – список из трех констант;
- **{a,b,c}** – список из трех переменных;
- **{1,a,x^2}** – список с разнотипными элементами.

Возможно создание списков в списках, например:

- **{{a,b,c},{d,e,f}}** – список из двух строк.

Примеры задания списков:

- **{1, 2, 3}** – список из трех целых чисел,
- **{a, b, c}** – список из трех символьных данных,
- **{1, a, x^2}** – список из разнотипных данных,
- **{{a,b},{c,d}}** – список, эквивалентный матрице $\begin{vmatrix} a & b \\ c & d \end{vmatrix}$;
- **{x^2+y^2, 2*Sin[x]}** – список из двух математических выражений.

Для создания списков ранжированных числовых значений используется функция **Range**:

- **Range[imax]** – генерирует список числовых элементов **{1, 2, …, imax}**;
- **Range[imin,imax]** – генерирует список числовых элементов **{imin,…,imax}**;
- **Range[imin,imax, di]** – генерирует список числовых элементов с шагом **di**.

Примеры на использование функции **Range**:

```
Range[5]
 {1, 2, 3, 4, 5}
Range[0,2,0.5]
 {0, 0.5, 1., 1.5, 2.}
```

Для выделения элементов списка list используются двойные квадратные скобки:

- **list[[i]]** – выделяет i-й элемент списка с его начала (если I < 0 – с конца);
- **list[[{i,j,...}]]** - выделяет i-й, j-й и т. д. элементы списка.

Ниже приведены примеры на выделение элементов списков.

Пример	Комментарий
l1:={1,2,3,4,5}	Задание исходного списка l1
l1[[3]]	Выделение третьего элемента с начала
3	

```
l1[[-2]]                    Выделение второго элемента с конца
4
l1[[{1,2,5}]]               Выделение первого, второго и пятого
{1, 2, 5}                    элементов
l2={{1,2,3},{4,5,6}}        Задание сдвоенного (двумерного)
{{1, 2, 3}, {4, 5, 6}       списка
TableForm[l2]               Вывод сдвоенного списка в табличной
1  2  3                     форме
4  5  6
l2[[2,3]]                   Выделение элемента сдвоенного списка
6
```

Для выделения заданного i-го элемента списка list используется также функция **Part[list,i]**. При I > 0 отсчет номеров элементов идет с начала списка, а при i < 0 – с его конца. Это правило поясняют примеры, показанные ниже:

```
L:={1,2,3,a,b,c}
{Part[L,2],Part[L,5],Part[L,6]}
 {2,b,c}
{Part[L,-2],Part[L,-5],Part[L,2]}
 {b,2,2}
```

2.11.11. Задание массивов, векторов и матриц

Для задания массивов, векторов и матриц в Mathematica используются следующие функции.

- **Array[f, n]** – генерирует список длиной n с элементами f[i].
- **Array[f, {n1, n2, ...}]** – генерирует массив с размером n1×n2×... в виде вложенных списков с элементами f[i1, i2, ...].
- **Array[f, dims, origin]** – генерирует список с размерностью dims, используя спецификацию индекса origin.
- **Array[f, dims, origin, h]** – использует заголовок h, а не List, для каждого уровня массива.

Здесь f – выражение, формирующее значения элементов массива. Примеры задания массивов и их вывода:

```
Ввод (In)                   Вывод (Out)
Y:=Array[Exp,4]
                            2   3   4
Y                           {E, E , E , E }
N[Y]                         {2.71828, 7.38906, 20.0855, 54.5982}
Array[f,{3,3}]              {{f[1, 1], f[1, 2], f[1, 3]},
                            {f[2, 1], f[2, 2], f[2, 3]},
                            {f[3, 1], f[3, 2], f[3, 3]}}
Array[Sin,3,0]              {0, Sin[1], Sin[2]}
Array[Sin,4,1,Plus]         Sin[1] + Sin[2] + Sin[3] + Sin[4]
Array[f,5,2,2]              2[f[2], f[3], f[4], f[5], f[6]]
```

В справке по системе Mathematica и в книгах [79–81] можно найти описание многих функций, предназначенных для расширенной работы со списками. Работу со сложными типами данных (массивами, матрицами и векторами) в системе Mathematica мы рассмотрим в главе 8.

Глава 3

Работа с математическими выражениями и функциями

Центральным понятием математики являются *математические выражения*, которые представляют записи вычислений на общепринятом математическом языке, с применением констант, переменных, операторов, операндов и функций [122–126]. В этой главе описана практика работы с выражениями, вычисляемыми с помощью встроенных в СКМ операторов и функций – как элементарных, так и специальных. Большинство примеров дано для работы в системах Maple 9.5/10/11 и Mathematica 5.1/5.2/6.

3.1. Работа с операторами

3.1.1. Операторы и операнды

Операторы – специальные символы, которые используются в записях математических выражений для указания того, какие виды операций должны в них выполняться и в каком порядке. Операторы обычно применяются совместно с данными, которые они обрабатывают и которые именуются *операндами*. Самыми распространенными являются арифметические операторы (+, –, * или пробел, / и ^).

Операнды могут быть числами, константами, переменными и математическими выражениями. К примеру, в выражении (2+3)+5 операторами являются знаки + и скобки ( ), а операндами – константы 2 и 3 для первого оператора сложения и выражение (2+3) и константа 5 для второго оператора сложения. Аналогично в выражении $(a+b)-c$ операндами будут переменные a, b и c.

В математических выражениях операторы имеют общепринятый *приоритет выполнения*, то есть определенный порядок выполнения операторов в сложном выражении. В порядке убывания он представлен ниже:

|| :- :: % & ! {^, @@} { ., *, &*, /, @, intersect} {+, -, union, minus} mod subset .. {<, <=, >, >=, =, <>, in} $ not and or xor implies -> , assuming :=

Для *изменения приоритета* операций в математических выражениях используются круглые скобки. Степень вложения скобок в современных системах компьютерной математики практически не ограничена.

3.1.2. Виды операторов

В Maple имеется пять основных *типов операторов*:

- `binary` – бинарные операторы (с двумя операндами);
- `unary` – унарные операторы (с одним операндом);
- `nullary` – нульарные операторы (без операнда – это одна, две и три пары кавычек);
- `precedence` – операторы старшинства (включая логические операторы);
- `functional` – функциональные операторы.

В Maple для просмотра операторов определенного вида и их свойств можно использовать следующую команду:

```
> ?operators[вид];
```

А для изучения примеров применения операторов нужно задать и исполнить команду

```
> ?operators[examples];
```

Команда

```
> ?define;
```

позволяет ознакомиться с функций `define`. С ее помощью можно определять новые операторы.

3.1.3. Применение бинарных (инфиксных) операторов

Бинарные операторы, именуемые также *инфиксными*, используются с двумя операндами, размещаемыми по обе стороны от оператора (см. табл. 3.1).

Таблица 3.1. Бинарные операторы Maple 9.5/10

Обозначение	Оператор
+	Сложение
–	Вычитание
*	Умножение
/	Деление
** или ^	Возведение в степень
mod	Остаток от деления
$	Оператор последовательности
.	Разделительная точка
@	Оператор композиции
@@	Повторение композиции
,	Разделитель выражений
:=	Присваивание
..	Задание интервала
,	Разделитель выражений
&*	Некоммутативное умножение
&<string>	Нейтральный оператор
||	Конкатенация (объединение)

Примеры вычисления выражений с бинарными операторами:

```
> [2^3,2**3];
```

$$[8, 8]$$

```
> 7 mod 5;
```

$$2$$

```
> [x@x,x@@x];
```

$$[x^{(2)}, x^{(x)}]$$

```
> [x$3,x$4];
```

$$[x, x, x, x, x, x, x]$$

```
> S:=`Hello`||` my `||`friend!`;
```

$$S := Hello\ my\ friend!$$

Оператор композиции @@ может использоваться для создания и вычисления сложных функций, содержащих цепные дроби. Ниже показано задание функции пользователя f и цепной дроби:

```
> f:=a->1/(1+a);(f@@3)(a);
```

$$f := a \to \frac{1}{a+1}$$

$$\frac{1}{\frac{1}{\frac{1}{a+1}+1}+1}$$

```
> f(5);
```

$$\frac{1}{6}$$

А вот еще один пример применения этого оператора для составления цепного радикала и вычисления ряда таких цепочек в цикле:

```
> f := x -> sqrt( 1 + x);
```

$$f := x \to \sqrt{1+x}$$

```
> f(f(f(0)));
```

$$\sqrt{1+\sqrt{2}}$$

```
> (f@@10)(x);
```

$$\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+}}}}}}}}}$$

```
> for k from 1 to 5 do (f@@k)(0) = evalf((f@@k)(0)); od;
```

$$1 = 1.$$

$$\sqrt{2} = 1.414213562$$

$$\sqrt{1+\sqrt{2}} + 1.553773974$$

$$\sqrt{1+\sqrt{1+\sqrt{2}}} = 1.598053182$$

$$\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{2}}}} = 1.611847754$$

Внимание! *У некоторых СКМ знак умножения может быть заменен на пробел. Например, ввод 2 3 4 в системе Mathematica даст результат 24. Проверьте это правило в той системе, которой вы отдаете предпочтение. Например, в Maple «этот номер» не проходит.*

3.1.4. Работа с множествами

Множества, относящиеся к первичным понятиям математики, не являются точно определенными математическими объектами [10–12]. Можно рассматривать, например, различные множества чисел, множества людей или деревьев и т. д. Будем считать, что они определяют группу неповторяющихся объектов. Для работы с множествами определены следующие бинарные операторы:

- `union` – включает первый операнд (множество) во второй;
- `intersect` – создает множество, содержащее общие для операндов элементы;
- `minus` – исключает из первого операнда элементы второго операнда.

В любом случае в результирующем множестве устраняются повторяющиеся элементы. Примеры вычисления выражений с этими операторами в системе Maple приведены ниже:

```
> {a,a,b,c,c,d} union {e,e,f,g};
```

$$\{f, g, a, b, e, c, d\}$$

```
> {a,a,b,c,c,d} intersect {a,c,e,e,f,g};
```

$$\{a, c\}$$

```
> {a,a,b,c,c,d} minus {a,d};
```

$$\{b, c\}$$

Напоминаем, что эти операторы заданы ключевыми словами.

3.1.5. Новый оператор in в множествах

В Maple 8 для удобства работы с множествами был добавлен новый оператор in. Он может использоваться в виде:

```
element in objs  или  element in SetOf( type )
```

где `element` – элемент множества или списка `objs`, `type` – тип выражения. Применение оператора in поясняют несколько примеров:

```
> a in {a, b, c, d};
```

$a \in \{b, a, c, d\}$

```
> evalb(%);
```

true

```
> 3 in {1,2,3} intersect ({1,2,6} minus {1,4,7});
```

$3 \in \{2\}$

```
> evalb(%);
```

false

3.1.6. Применение унарных арифметических операторов

Унарные операторы используются с одним операндом. Они могут быть *префиксными*, если оператор стоит перед операндом, и *постфиксными*, если он стоит после операнда. К унарным относятся семь операторов, приведенных в табл. 3.2.

Таблица 3.2. Унарные операторы

Обозначение	Оператор
+	Унарный плюс (префикс)
–	Унарный минус (префикс)
!	Факториал (постфикс)
.	Десятичная точка (префикс или постфикс)
$	Последовательность (префикс)
not	Логическое отрицание (префикс)
&string	Метка (префикс)

Примеры применения унарных операторов в Maple:

```
> [-x,x+(-x),x+(+x)];
```

$$[-x, 0, 2x]$$

```
> 20!;
```

$$2432902008176640000$$

```
> 2.34;
```

$$2.34$$

```
> 2*%;
```

$$4.68$$

```
> a$3;
```

$$a, a, a$$

3.1.7. Применение оператора % и команды history

В Maple оператор % обеспечивает подстановку на его место в строке ввода (или в выражении) последнего результата операции, %% – предпоследнего и %%% – третьего с конца и т. д. Есть еще одна иногда полезная возможность проследить за ходом частных вычислений внутри документа – применение команды-функции `history`.

Функция `history(expr)` или `history()` создает список переменных вида `Oi`, где индекс $i = 1, 2, 3, \ldots$ Этим переменным можно присваивать значения в диалоговом режиме и отслеживать результаты вычислений. Команда `off;`, вводимая после использования данной функции, завершает работу с ней. Ниже представлен диалог с применением функции `history`:

```
> history();

O1 := 2;
```

$$2$$

```
O2 := sin(1.);
```

$$0.8414709848$$

```
O3 := O1*O2;
```

$$1.682941970$$

```
O4 := off;

> %;
```

history

Функция `history` может применяться в качестве средства начальной отладки вычислений. Внутри фрагмента программы, заданного функцией `history`, можно задавать построения графиков.

3.1.8. Работа с логическими операторами

Логические операторы, именуемые также булевыми, указывают на логическую связь величин (или выражений). Они представлены рядом бинарных операторов, приведенных в табл. 3.3, и реализованы во всех СКМ.

Таблица 3.3. Бинарные логические операторы

Обозначение	Оператор
<	Меньше
<=	Меньше или равно
>	Больше
>=	Больше или равно
=	Равно
<>	Не равно
And	Логическое «и»
Or	Логическое «или»

Конструкции с этими операторами, такие как `x=y`, возвращают логическое значение – константу `true`, если условие выполняется, и `false`, если оно не выполняется. Кроме того, к логическим операторам относится унарный оператор `not` – он представляет логическое «нет». Для возврата логических значений выражений с этими операторами в Maple используется функция `evalb(условие)`, например:

```
> 5<2;
```

$$5 < 2$$

```
> evalb(%);
```

false

```
> evalb(3=3 and 4>2);
```

true

```
> evalb(x*y=y*x);
```

true

Логические операторы часто используются в управляющих структурах программ, составленных на языке программирования Maple. Такое их применение мы рассмотрим позже.

Mathematica

Для осуществления логических операций в системе Mathematica используются следующие *логические операторы*:

`==` равенство (например, a == b);
`!=` неравенство;
`>` больше (например, b > a);
`>=` больше или равно;
`<` меньше;
`<=` меньше или равно;

Как нетрудно заметить, тут есть явные отличия от обозначений данных операторов, принятых в системе Maple. Но к принципиальным их не отнесешь.

3.1.9. Применение операторов специальных типов

Операторы в Maple описывают операции по преобразованию данных, в частности выражений. Последние, в свою очередь, можно отнести к данным абстрактного типа. Могут быть описаны следующие типы операторов:

- неопределенные (`f`);
- нейтральные (`&`);
- процедурные;
- функциональные;
- композиционные (`@`).

Оператор относится к *неопределенным*, если он не был заранее определен. Такой оператор не выполняет никаких действий и просто повторяется в строке вывода:

```
> restart:f(1,2,a);
```

$$f(1, 2, a)$$

Композиционные операторы (на базе знака `@`) мы уже применяли. Другие типы операторов рассмотрены ниже.

3.1.10. Применение функциональных операторов

Функциональные операторы Maple-языка являются альтернативами функций и записываются в двух формах.

Нотация	Запись оператора
«arrow» (стрелочная)	`vars -> result`
«angle bracket» (в угловых скобках)	`<result | vars>`

Данные операторы могут использоваться для реализации подстановок. Например, запись `x -> x^2` означает подстановку `x^2` на место переменной `x`. Возможны и такие подстановки в множественной форме:

```
(x,y) -> x^2 + y^2
 x -> (2*x, 3*x^4)
(x,y,z) -> (x*y, y*z)
```

Функциональный оператор в Maple часто используется для задания функций пользователя. Эти операторы системно-ориентированные и в приведенном виде присущи только СКМ Maple.

3.1.11. Определение нейтральных операторов

Для создания *нейтральных операторов* (задаваемых пользователем и в момент задания не исполняемых), определяемых пользователем, служит знак амперсанда – `&`. Синтаксис нейтрального оператора следующий:

```
&name
```

Имя оператора строится по правилам задания допустимых идентификаторов. Также в качестве имени может быть использована последовательность (один и более) специальных символов. В последовательности специальных символов не должно быть букв, цифр, подчеркивания, а также следующих символов:

```
&  |  (  )  {  }  [  ]  :  ;  "  `  #  <перевод строки> <пробел>
```

Максимальная длина имени – 495 символов. Нейтральные операторы могут быть унарными и бинарными. Примеры задания бинарного нейтрального оператора приведены ниже:

```
> x&/y;
```

$$x \&/ y$$

```
> z+x&/y;
```

$$z + (x \&/ y)$$

```
> &/(x,y);
```

$$x \&/ y$$

```
> x&/y-&/(x,y);
```

$$0$$

3.1.12. Определение операторов с помощью оператора define

Большие возможности для создания операторов с заданными свойствами предоставляет специальный оператор `define`. Он записывается в следующей форме:

define(oper, property1, property2, ...)

Здесь `oper` – имя определяемого оператора, `property1`, `property2` и т. д. – наименования свойств. В принципе, оператор `define` позволяет создавать операторы с новыми свойствами, которые отсутствуют у операторов и функций, встроенных в систему. Могут быть указаны следующие свойства операторов:

- `unary` – унарный оператор;
- `binary` – бинарный оператор;
- `diff` – дифференциальный оператор;
- `linear` – линейный оператор;
- `multilinear` – множественный линейный оператор;
- `flat` – ассоциативный оператор, для которого $f(x,f(y,z)) = f(f(x,y),z) = f(x,y,z)$;
- `orderless` – коммутативный симметричный оператор, такой что $f(x,y) = f(y,x)$;
- `antisymmetric` – асимметричный оператор, такой что $f(x,y) = -f(y,x)$;
- `zero` – нулевой оператор (например, `V:=Vector(5,shape=zero)` задает вектор с 5 нулевыми элементами);
- `identity` – единичный оператор (например, `M:=Matrix(3,3,shape=identity)` задает единичную матрицу).

Следующий пример задает линейный оператор `L`:

```
> define(L,linear);
> L(a*x+b*x^2+c*x^3);
```

$$L(ax) + L(bx^2) + L(cx^3)$$

Для задания некоторых свойств операторов можно использовать уравнения и соотношения вида `f(x)=value`. Чтобы свойство выполнялось для всех аргументов (или некоторого класса аргументов), используется описание `forall`. Так, приведенный ниже пример задает оператор `F`, который вычисляет n-е число Фибоначчи ($n > 2$):

```
> restart;
> define(fib,fib(0)=1,fib(1)=1,fib(n::posint)=fib(n-1)+fib(n-2));
> fib(6);
```

13

```
> fib(10);
```

89

```
> fib(20);
```

10946

Обратите внимание на то, что команда `restart` снимает все определения переменных, а соотношения `fib(0)=1` и `fib(1)=1` задают начальные значения целочисленного массива чисел Фибоначчи, которые нужны для реализации обычного итерационного алгоритма их нахождения, – напоминаем, что очередное число Фибоначчи равно сумме двух предшествующих чисел Фибоначчи.

3.1.13. Укороченные операторы системы Mathematica

Спецификой систем Mathematica являются арифметические операторы с укороченной формой записи, объединяющие операцию присваивания с арифметической операцией:

Функция	Оператор	Назначение
Increnent[i]	**i++**	– увеличивает значение i на 1 до использования i в выражении;
Decrement[i]	**i—**	– уменьшает значение i на 1 до использования i в выражении;
PreIncrement[i]	**++i**	– увеличивает значение i на 1 после использования i в выражении;
PreDectement[i]	**—i**	– уменьшает значение i на 1 после использования i в выражении;
AddTo[x,d]	**x += dx**	– прибавляет dx к x и возвращает новое значение x;
SubstractFrom[x,dx]	**x -= dx**	– отнимает dx от x и возвращает новое значение x;
TimesBy[x,c]	**x *= c**	– умножает x на c и возвращает новое значение x;
DivideBy[x,c]	**x /= c**	– делит x на c и возвращает новое значение x.

Примеры выполнения укороченных арифметических операций:

Ввод (*In*)	**Вывод (*Out*)**
`i=0`	`0`
`++i; ++i; ++i`	`3`
`i=0; i++; i++; i++`	`2`
`i=5`	`5`
`-i`	`4`
`i=5`	`5`
`i-`	`5`
`i-`	`4`
`x=5`	`5`
`x+=0.5`	`5.5`
`x-=0.5`	`5.`
`x*=2`	`10.`
`x/=5`	`2.`

3.1.14. Операторы СКМ Mathcad

Очень удобно организованы ввод и применение операторов с системе Mathcad 11/12/13/14 и в ее предшествующих версиях. В приведенном ниже списке операторов, вводимых с клавиатуры, используются следующие обозначения:

- `A` и `B` – массивы векторов или матриц;
- `u` и `v` – векторы с действительными или комплексными элементами;
- `M` – квадратная матрица;
- `z` и `w` – действительные или комплексные числа;
- `x` и `y` – действительные числа;
- `m` и `n` – целые числа;
- `i` – диапазон переменных;
- `t` – любое имя переменной;
- `f` – функция;
- `X` и `Y` – переменные или выражения любого типа.

Список операторов представлен в табл. 3.4.

Таблица 3.4. Список операторов системы Mathcad

Оператор	Обозначение	Клавиши	Описание
Круглые скобки	(X)	'	Изменение приоритета операций
Нижний индекс	An	[	Задание индексированной переменной
Верхний индекс	A<n>	Ctrl+6	Выбор n-го столбца из массива A
Векторизация	$\bar{X}$	Ctrl+- (дефис)	Векторизация – выполнение заданной операции для всех элементов вектора или матрицы X
Факториал	n!	!	Факториал целого неотрицательного числа n
Сопряженное комплексное число	X	"	Сопряженное комплексное число
Транспонирование	AT	Ctrl+1	Транспонирование матрицы A
Возведение в степень	zw	^	Возведение числа z в степень w
Возведение в степень	Mn	^	Возведение в n-ю степень квадратной матрицы M (при n=–1 получение обратной матрицы)
Отрицание	-X	–	Умножение X на –1
Сумма вектора	Σv	Ctrl+4	Сумма элементов вектора v (возвращается скалярное значение)
Квадратный корень	$\sqrt{z}$	\	Квадратный корень
Корень n-й степени	$\sqrt[n]{z}$	Ctrl+\	Корень n-й степени из числа z

Таблица 3.4. Список операторов системы Mathcad (продолжение)

Оператор	Обозначение	Клавиши	Описание
Модуль комплексного числа	|z|	|	$\sqrt{\mathrm{Re}(z)^2+\mathrm{Im}(z)^2}$
Размер вектора	|v|	|	$\sqrt{v\cdot v}$, если все элементы в векторе v являются реальными, $\sqrt{v\cdot\overline{v}}$, если элементы в векторе v являются комплексными
Детерминант матрицы	|M|	|	Определитель квадратной матрицы M
Деление	X/z	/	Деление выражения X на скаляр z, не равный нулю (если X является массивом, то на z делится каждый элемент массива)
Умножение	X·Y	*	Произведение X и Y, если X и Y являются скалярами. Умножение каждого элемента Y на X, если Y является массивом, а X – скаляром. Скалярное произведение, если X и Y – векторы одинакового размера. Умножение матриц, если X и Y являются матрицами совместимых размеров
Векторное произведение	u·v	Ctrl+8	Векторное произведение векторов u и v
Суммирование для конечного ряда	$\sum_{i=m}^{n} X$	Ctrl+Shift+4	Сумма членов X для i=m,m+1,...,n, причем X может быть любым выражением
Произведение для конечного ряда	$\prod_{i=m}^{n} X$	Ctrl+Shift+3	Произведение членов X для i=m,m+1,...,n, где X может быть любым выражением
Суммирование для бесконечного ряда	$\sum_{i} X$	$	Сумма членов X бесконечного ряда
Произведение для бесконечного ряда	$\prod_{i} X$	#	Произведение членов X бесконечного ряда
Предел функции в заданной точке	$\lim_{x\to a} f(x)$	Ctrl+L	Предел функции f(x) при x, стремящемся к a (выполняется только в режиме символьных вычислений)
Предел функции слева от заданной точки	$\lim_{x\to a^-} f(x)$	Ctrl+B	Предел функции f(x) при x, приближающемся к a слева (выполняется только в режиме символьных вычислений)
Предел функции справа от заданной точки	$\lim_{x\to a^+} f(x)$	Ctrl+A	Предел функции f(x) при x, приближающемся к a справа (выполняется только в режиме символьных вычислений)
Определенный интеграл	$\int_{a}^{b} f(t)dt$	&	Определенный интеграл от подынтегральной функции f(t) с нижним (a) и верхним (b) пределами интегрирования

Таблица 3.4. *Список операторов системы Mathcad (окончание)*

Оператор	Обозначение	Клавиши	Описание
Неопределенный интеграл	$\int f(t)dt$	Ctrl+I	Вычисление в символьном виде неопределенного интеграла от подынтегральной функции **f(t)**
Производная заданной функции по переменной **t**	$\frac{d}{dt}f(t)$	?	Первая производная функции **f(t)** по переменной **t**
n-я производная заданной функции по переменной **t**	$\frac{d^n}{dt^n}f(t)$	Ctrl+?	***n***-я производная функции **f(t)** по переменной **t**
Сложение	**X+Y**	+	Скалярное, векторное или матричное сложение **X** и **Y**
Вычитание	**X–Y**	–	Скалярное, векторное или матричное вычитание **Y** из **X**
Перевод строки	**X +Y**	Ctrl+Enter	Перенос части выражения на следующую строку
Больше, чем	**x>>y**	>>	1, если **x>>y**, иначе 0
Меньше, чем	**x<<y**	<<	1, если **x<<y**, иначе 0
Больше или равно, чем	**x≥y**	Ctrl+0	1, если **x≥y**, иначе 0
Меньше или равно, чем	**x≤y**	Ctrl+9	1, если **x≤y**, иначе 0
Не равно	**z≠w**	Ctrl+3	1, если **z≠y**, иначе 0
Равно	**z=w**	Ctrl+=	1, если **z=w**, иначе 0
Логическое отрицание	**b** $\neg$	Ctrl+Shift+1	Инверсное значение булевого операнда **b**
Логическое умножение	**b1** $\wedge$ **b2**	Ctrl+Shift+7	Логическая 1, если **b1** и **b2** имеют значения логической единицы, иначе логический 0
Логическое сложение	**b1** $\vee$ **b2**	Ctrl+Shift+6	Логическая 1, если **b1** или **b2** имеет значение логической единицы, иначе логический 0
Логическое исключающее ИЛИ	**b1** $\otimes$ **b2**	Ctrl+Shift+5	Логический 0, если **b1=b2** (0=0 или 1=1), иначе логическая 1

3.2. Работа с математическими функциями

3.2.1. Понятие о функциях

Более двух сотен лет тому назад в обиход математиков пришло понятие *функции* как некоторой зависимости одной величины, например f или y, от другой величины – независимой переменной x или t. Функции стали обозначать как $f(x)$, $f(t)$, $y(x)$ и т. д. Могут быть и функции ряда переменных, например вида $f(x,y,z,...)$.

Часто функция определена на некотором отрезке от $x = a$ до $x = b$. В этом случае *область определения функции* принято записывать как $x \in [a,b]$. Иногда используют и иные обозначения. Например, если интервал определения функции задан как $a < x \le b$, то это записывают как $x \in (a,b]$.

Особого внимания требует определение функций в бесконечном интервале значений независимой переменной от $-\infty$ до $+\infty$. Мы будем обозначать его как **R,** то есть $\mathbf{R} = (-\infty, +\infty)$. Пусть, скажем, x принадлежит **R**, это записывается как $x \in \mathbf{R}$. Примером функции, определенной в **R**, является синусоидальная функция $\sin(x)$.

В процессе изучения функций мы иногда можем встретиться с понятиями о множествах или пространствах функций [126]. Пространство $L^P[\mathbf{R}]$ означает, что функция $f(x)$ удовлетворяет условию

$$\int_{-\infty}^{\infty} |f(x)|^p \, dx < \infty.$$

К примеру, часто используемое гильбертовое пространство $L^2[\mathbf{R}]$ означает, что $f(x)$ удовлетворяет условию

$$\int_{-\infty}^{\infty} |f(x)|^2 \, dx < \infty.$$

Могут встречаться и иные пространства, например $L^2(0,2\pi)$ означает, что $f(x)$ удовлетворяет условию

$$\int_{0}^{2\pi} |f(x)|^2 \, dx < \infty.$$

Практически можно считать, что в нашем случае $f(x)$ – любая кусочно-непрерывная функция. Она может быть периодической. Например, на периоде 2π

периодической будет функция, удовлетворяющая равенству $f(x) = f(x \pm 2\pi)$. Множество функций из $L^2(0, 2\pi)$ называют пространством 2π-периодических функций, интегрируемых с квадратом.

Примером такой функции является и комплексная синусоида $f(x) = \mathbf{e}^{ix} = \cos(x) + i\cdot\sin(x)$, где $i = \sqrt{-1}$ – мнимая единица. Синусоида является функцией, периодически продолжаемой на всю вещественную ось x. Однако пространству $L^2(\mathbf{R})$ синусоида не принадлежит, поскольку из условия определения этого пространства следует, что принадлежащие ему функции должны затухать при $x \in \mathbf{R}$. Синусоида таким свойством не обладает.

В современных СКМ *функция* – это имеющий уникальное имя (идентификатор) объект математического выражения, выполняющий некоторое преобразование своих входных данных, представленных списком *входных параметров*. Входные параметры изначально являются *формальными* и представляются именами некоторых переменных. Особенностью функции является *возврат ее значения* в ответ на обращение к функции по имени с указанием *фактических* параметров в списке параметров функций. Фактические параметры могут быть различными константами, определенными переменными и даже вычисляемыми математическими выражениями. Функция atan2(x,y) является примером функции, имеющей список из двух формальных параметров – x и y.

Как правило, в системах символьной математики принципиально важно, как записан фактический параметр. Например, число 1. или 1.0 является вещественным, на что указывает разделительная точка. Если число представлено в виде 1, то оно рассматривается как целое и как константа. Большинство систем символьной математики не вычисляет выражения вида sin(1) или sin(π/2), а выводит их в исходном виде. Это связано с тем, что такой вид дает о значении функции гораздо больше информации, чем просто ее вычисленное значение.

Функции обычно подразделяются на четыре типа:

- встроенные в ядро системы предопределенные, или внутренние, функции;
- функции пользователя, например вида f(x,y,z) := (x^2 + y^2)/z^2;
- библиотечные функции, вызываемые из пакетов или библиотек расширения системы, например sin(x) или ln(x);
- функции, заданные в виде программного модуля.

Кроме того, функции могут классифицироваться по характеру производимых ими преобразований входных параметров. Они делятся на алгебраические, тригонометрические, обратные тригонометрические, гиперболические, обратные гиперболические, специальные и т. д.

В сложных математических системах, например Maple или Mathematica, функции могут применяться со специальными *директивами* и *опциями*. Они могут задаваться как дополнительный параметр функции (Maple, Mathematica) или как специальное указание перед ее применением (Mathcad, MATLAB).

3.2.2. Математические выражения

Математические выражения – это сложные (комбинированные) объекты, которые состоят из операторов, операндов и функций со списками их параметров. В системах для численных расчетов математические выражения применяются в естественном виде, и в разборе их структуры нет особой необходимости. Исключение составляет разве что анализ скобок, меняющих приоритет выполнения операций в выражениях.

Иное дело – системы символьной математики. У них в ходе вычислений выражения *эволюционируют*, то есть изменяются по мере выполнения расчетов. Это может приводить к весьма неожиданным последствиям, например когда сложнейшее выражение упрощается к 0 или 1, а внешне совсем не страшное выражение разворачивается так, что не помещается в десятке страниц экрана.

3.2.3. Работа с элементарными функциями в системе Maple

Maple имеет полный набор *элементарных* математических функций [122–126]. Все они, кроме арктангенса двух аргументов, имеют один аргумент `x`, например `sin(x)`. Он может быть целым, рациональным, дробно-рациональным, вещественным или комплексным числом. В ответ на обращение к ним элементарные функции возвращают соответствующее значение. Поэтому они могут быть включены в математические выражения. Все описанные здесь функции называются *встроенными*, поскольку они реализованы в ядре системы.

Как правило, если аргументом функции является фундаментальная константа, целое или рациональное число, то функция выводится с таким аргументом без получения результата в форме действительного числа с плавающей точкой. Например:

```
> sin(1);
```

$$\sin(1)$$

```
> exp(1);
```

$$\mathbf{e}$$

```
> ln(Pi);
```

$$\ln(\pi)$$

```
> arcsin(1/2);
```

$$\frac{1}{6}\pi$$

Для получения подробной информации о некоторой произвольной функции <f> достаточно задать команду

```
> ? <f>
```

Например, команда

```
> ? sin
```

открывает окно справки по тригонометрическим функциям, включая функцию синуса.

Ввиду общеизвестности элементарных функций мы не будем обсуждать ни их свойства, ни допустимые для них пределы изменения аргумента. Эти вопросы отражаются в любом учебнике по элементарной математике.

В ядре Maple (и других СКМ) определены следующие *тригонометрические функции*: `sin` – синус; `cos` – косинус; `tan` – тангенс; `sec` – секанс; `csc` – косеканс; `cot` – котангенс. Все эти функции являются периодическими (с периодом 2π, кроме тангенса и котангенса, у которых период равен π) и определены для действительного и комплексного аргументов.

Многие свойства тригонометрических функций можно оценить, рассматривая их графики. Для построения таких графиков средствами Maple можно использовать функцию `plot`. Читатель может легко построить графики любых элементарных функций.

К *обратным тригонометрическим функциям* относятся: `arcsin` – арксинус; `arccos` – арккосинус; `arctan` – арктангенс; `arcsec` – арксеканс; `arccsc` – арккосеканс; `arccot` – арккотангенс. К этому классу функций принадлежит еще одна полезная функция:

arctan(y,x) = argument(x+I*y)

Она возвращает угол радиус-вектора в интервале от `-Pi` до `Pi` при координатах конца радиус-вектора `x` и `y` (см. пример ниже):

```
> arctan(2.,3);
```

.5880026035

Гиперболические функции представлены следующим набором: `sinh` – гиперболический синус; `cosh` – гиперболический косинус; `tanh` – гиперболический тангенс; `sech` – гиперболический секанс; `csch` – гиперболический косеканс; `coth` – гиперболический котангенс. В отличие от тригонометрических функций, гиперболические функции не являются периодическими.

С помощью функции преобразования `convert(f, exp)` можно перевести гиперболические функции в экспоненциальную форму:

```
> convert(sinh(x),exp);
```

$$\frac{1}{2}\mathbf{e}^{x} - \frac{1}{2}\frac{1}{\mathbf{e}^{x}}$$

```
> convert(tan(x),exp);
```

$$\frac{-I((\mathbf{e}^{(xI)})^2 - 1)}{(\mathbf{e}^{(xI)})^2 + 1}$$

К *обратным гиперболическим функциям* относятся: `arcsinh` – гиперболический арксинус; `arccosh` – гиперболический арккосинус; `arctanh` – гиперболический арктангенс; `arcsech` – гиперболический арксеканс: `arccsch` – гипербо-

лический арккосеканс: `arccoth` – гиперболический арккотангенс. Примеры применения:

```
> [arcsinh(1.),arccosh(1.),arctanh(1.)];
```

$$[.8813735870,\ 0.,\ \text{Float}(\infty) + \text{Float}(\text{undefined})I]$$

С помощью функции преобразования `convert(f, ln)` можно перевести гиперболические функции в логарифмическую форму:

```
> convert(arcsin(x),ln);
```

$$-I\ln(\sqrt{1-x^2}+x\,I)$$

```
> convert(arctan(x),ln);
```

$$\frac{-1}{2}I(-\ln(1-x\,I)+\ln(1+x\,I))$$

К *степенным* и *логарифмическим* относятся следующие функции системы Maple: `exp` – экспоненциальная функция; `ilog10` – целочисленный логарифм по основанию 10 (возвращает целую часть от логарифма по основанию 10); `ilog` – целочисленный логарифм (библиотечная функция, возвращающая целую часть от натурального логарифма); `ln` – натуральный логарифм; `log` – логарифм по заданному основанию (библиотечная функция); `log10` – логарифм по основанию 10; `sqrt` – квадратный корень.

Многие функции этой группы обычно определены для положительных значений аргумента. Однако введение комплексных чисел позволяет вычислять такие функции и для отрицательных значений аргумента. Несколько интересных примеров этого представлены ниже:

```
> restart:sqrt(-4);
```

$$2\,I$$

```
> simplify( sqrt( x^2 ) );
```

$$\text{csgn}(x)x$$

```
> ln( -1 );
```

$$\pi\,I$$

```
> simplify( log( exp(x) ) );
```

$$\ln(\mathbf{e}^{x})$$

```
> assume(x,positive);simplify( log( exp(x) ) );
```

$$x\sim$$

Обратите внимание на то, что в предпоследнем примере Maple отказался вычислить «очевидное» значение выражения, но сделал это после придания ей статуса предполагаемой переменной с только положительными значениями. После этого вычисления прошли «без сучка и задоринки».

3.2.4. Гармонический синтез пилы и меандра

Фундаментальная роль функций синуса и косинуса проявляется в решении задач *спектрального анализа и синтеза*. На рис. 3.1 показан пример гармонического синтеза двух периодов пилообразного колебания (сигнала) при суммировании 3,

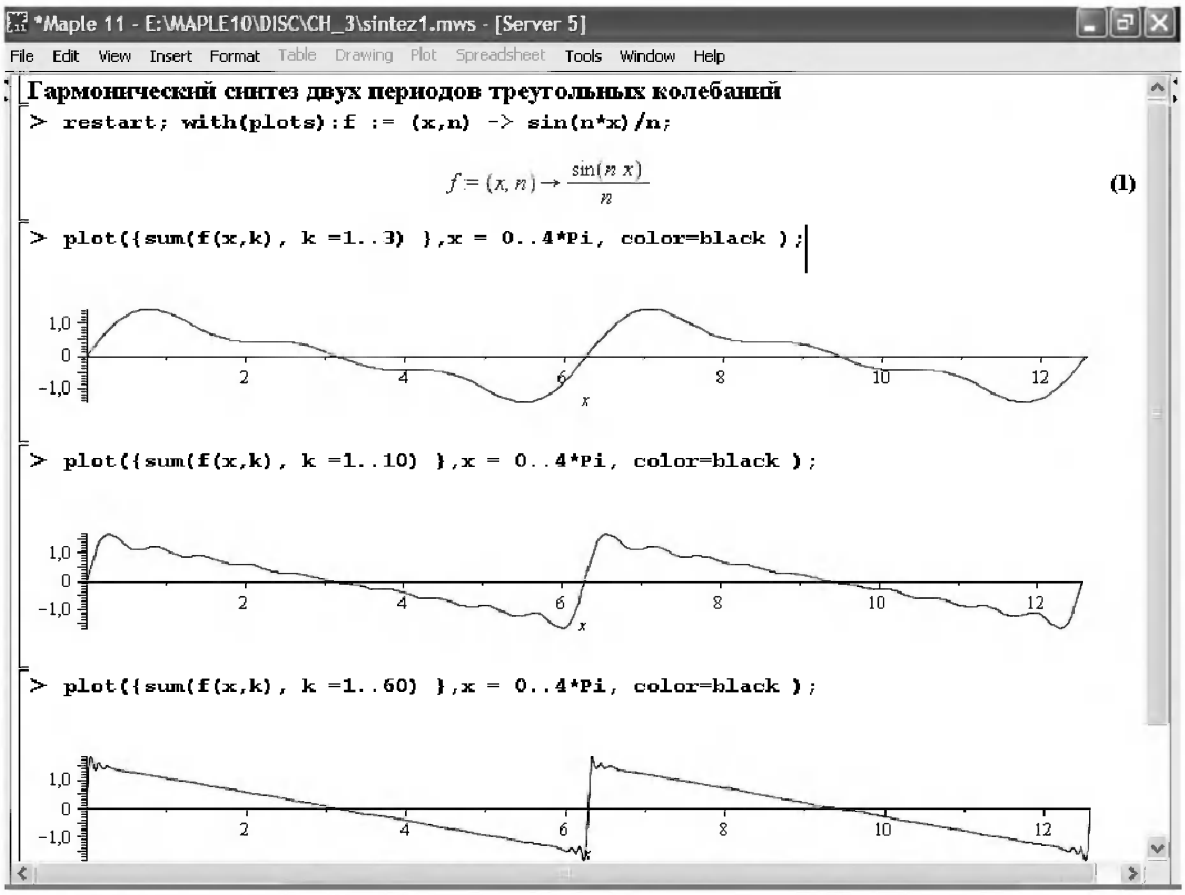


Рис. 3.1. Гармонический синтез треугольных колебаний по 3, 10 и 60 гармоникам

10 и 60 гармоник. Отчетливо видно, что по мере увеличения числа гармоник форма колебаний действительно приближается к треугольной. В условиях резкого ограничения числа гармоник в местах предполагаемого разрыва функции или ее производных наблюдаются характерные колебания – эффект Гиббса.

Колебания описанной формы получаются за счет синтеза всех гармоник, причем амплитуда гармоник равна 1/k, где k – номер гармоники.

А теперь рассмотрим синтез симметричных прямоугольных колебаний, получивших название меандра, – рис. 3.2. Для синтеза меандра надо использовать только нечетные гармоники.

Читатель, интересующийся вопросами гармонического синтеза сигналов, может опробовать в нем свои силы и синтезировать колебания и сигналы других форм.

3.2.5. Применение элементарных функций для моделирования сигналов

Элементарные функции можно использовать для моделирования сигналов различной формы – рис. 3.3. В этом рисунке опция `axes=NONE` убирает координатные оси. Обратите внимание, что смещение графиков отдельных функций вниз с целью устранения их наложения достигнуто просто прибавлением к значению каждой функции некоторой константы.

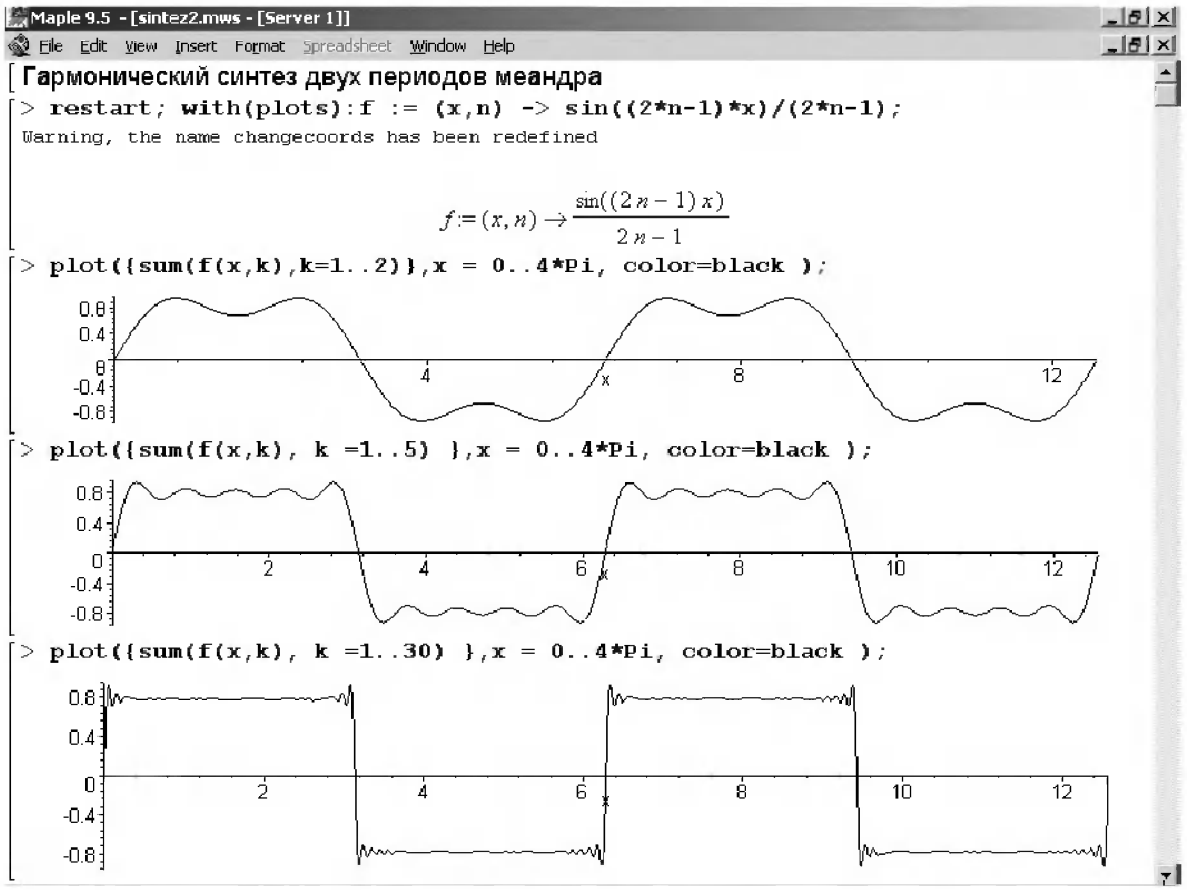


Рис. 3.2. Гармонический синтез меандра по 3, 9 и 30 гармоникам

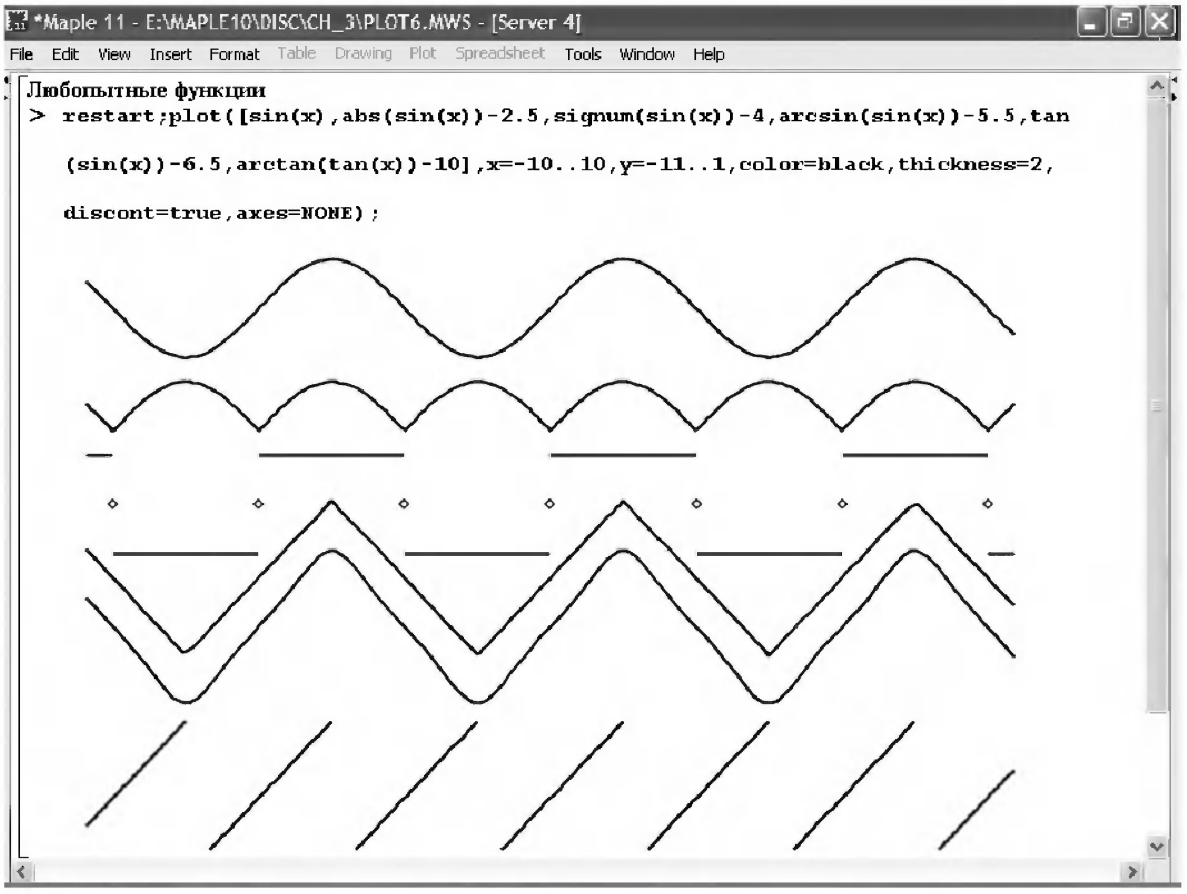


Рис. 3.3. Примеры моделирования сигналов с помощью комбинаций элементарных функций

Приведенные выше сигналы нередко можно формировать, используя функции с условиями, например функцию `signum`. Однако достоинство моделирования сигналов с помощью только элементарных функций заключается в том, что такие сигналы нередко могут обрабатываться аналитически, тогда как для функций с условиями это возможно далеко не всегда.

3.2.6. Некоторые целочисленные функции и факториал

Ниже представлены наиболее распространенные *целочисленные функции* Maple, используемые в теории чисел:

- `factorial(n)` – функция вычисления факториала (альтернатива – оператор `!`);
- `iquo(a,b)` – целочисленное деление `a` на `b`;
- `irem(a,b)` – остаток от деления `a` на `b`;
- `igcd(a b)` – наибольший общий делитель;
- `lcm(a,b)` – наименьшее общее кратное.

Факториал целого числа N задается как N=1*2*3*...*N при особых случаях 0!=1 и 1!=1.

Полезность возможности Maple вычислять факториалы больших чисел демонстрирует следующий пример. Пусть в группе студентов 25 человек, а в году 365 дней. Тогда вероятность того, что ни у кого из студентов дни рождения не совпадают, составит:

```
> p = 365!/(340!*25!*365^25);
> evalf(%);
```

$$p = 0.2780571566\ \ 10^{-25}$$

При обычной арифметике был бы получен 0 или сообщение о некорректности операций из-за превышения разрядной сетки компьютера.

А вот еще более характерный пример такого рода. Пусть вероятность рождения мальчика 0.515. Какова вероятность того, что среди 1000 новорожденных мальчиков будет не менее половины? Расчетная формула требует вычисления факториала 1000!:

```
> p = sum(1000!*.515^k*.485^(1000-k)/(k!*(1000-k)!), k=500..1000);
> evalf(%);
```

$$p = 0.8366435978$$

Если увеличить число рождаемых мальчиков до 10 или 100 тысяч, то такая задача может послужить хорошим тестом не только на работу с очень большими числами, но и на скорость вычислений. Решение подобных задач без применения аппарата точной арифметики больших чисел встречает немалые трудности, а подчас и просто невозможно.

3.2.7. Применение функций с элементами сравнения

В алгоритме вычисления ряда функций заложено сравнение результата с некоторым опорным значением. К таким *функциям с элементами сравнения* относятся: `abs` – абсолютное значение числа; `ceil` – наименьшее целое, большее или равное аргументу; `floor` – наибольшее целое, меньшее или равное аргументу; `frac` – дробная часть числа; `trunc` – целое, округленное в направлении нуля; `round` – округленное значение числа; `signum (x)` – знак x (-1 при $x < 0$, 0 при $x = 0$ и $+1$ при $x > 0$).

Для комплексного аргумента x эти функции определяются следующим образом:

- `trunc(x) = trunc(Re(x)) + I*trunc(Im(x));`
- `round(x) = round(Re(x)) + I*round(Im(x));`
- `frac(x) = frac(Re(x)) + I*frac(Im(x)).`

Для введения определения значения `floor(x)` от комплексного аргумента прежде всего запишем $a = \mathrm{Re}(x) - \mathrm{floor}(\mathrm{Re}(x))$ и $b = \mathrm{Im}(x) - \mathrm{floor}(\mathrm{Im}(x))$. Тогда $\mathrm{floor}(x) = \mathrm{floor}(\mathrm{Re}(x)) + I^*\mathrm{floor}(\mathrm{Im}(x)) + X$, где

$$X = \begin{cases} 0, & a+b<1, \\ 1, & a+b\geq 1\ 8\ a\geq b, \\ I, & a+b\geq 1\ 8\ a<b. \end{cases}$$

Наконец, функция `ceil` для комплексного аргумента определяется следующим образом:

ceil(x) = –floor(–x)

Примеры вычисления выражений с данными функциями представлены ниже:

```
> [ceil(Pi),trunc(Pi),floor(Pi),frac(Pi),round(Pi)];
```

$$[4, 3, 3, \pi - 3, 3]$$

```
> frac(evalf(Pi));
```

$$.141592654$$

```
> trunc(2.6+3.4*I);
```

$$2 + 3I$$

```
> [signum(-Pi),signum(0),signum(Pi)];
```

$$[-1, 0, 1]$$

Хотя функции этой группы достаточно просты, их нельзя относить к числу элементарных функций. Нередко их применение исключает возможность проведения символьных преобразований или их существенное усложнение.

3.2.8. Работа с функциями комплексного аргумента

Для комплексных чисел и данных, помимо упомянутых в предшествующем разделе, определен следующий ряд базовых функций: `argument` – аргумент комплексного числа; `conjugate` – комплексно-сопряженное число; `Im` – мнимая

часть комплексного числа; `Re` – действительная часть комплексного числа; `polar` – полярное представление комплексного числа (библиотечная функция). Примеры вычисления для этих функций:

```
> z:=2+3*I;
```

$$z := 2 + 3I$$

```
> [Re(z),Im(z),abs(z)];
```

$$[2, 3, \sqrt{13}]$$

```
> [argument(z),conjugate(z)];
```

$$\left[\arctan\left(\frac{3}{2}\right), 2 - 3\,I\right]$$

```
> readlib(polar);
```

proc(*r*::*algebraic*, *th*::*algebraic*) ... **end proc**

```
> polar(z);
```

$$\mathrm{polar}\left(\sqrt{13}, \arctan\left(\frac{3}{2}\right)\right)$$

```
> polar(-3.,Pi/2);
```

$$\mathrm{polar}\left(-3., \frac{1}{2}\pi\right)$$

В некоторых случаях полезна визуализация операций с комплексными числами. Для этого удобен пакет расширения plots, который позволяет представлять комплексные числа в виде стрелок на комплексной плоскости. Например, для иллюстрации операции умножения двух комплексных чисел

$$z_1 z_2 = r_1 r_2 e^{i(\theta_1 + \theta_2)}$$

можно использовать следующие графические построения (рис. 3.4):

```
   > with(plottools):
l1 := arrow([0,0], [1,2], .1, .3, .1, color=green):
l1a := arc([0,0],1.5,0..arctan(2),color=green):
   > l2 := arrow([0,0], [1,.8], .1, .3, .1, color=green):
l2a := arc([0,0],.75,0..arctan(.8),color=green):
   > l3 := arrow([0,0], [-.6,2.8], .1, .3, .1, color=black):
```

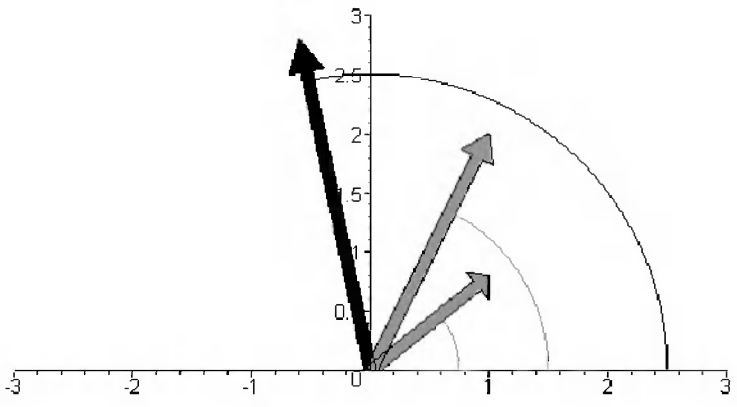


Рис. 3.4. Иллюстрация перемножения двух комплексных чисел

```
l3a := arc([0,0],2.5,0..arctan(2.8,-.6),color=black):
   > plots[display](l1,l2,l3,l1a,l2a,l3a, axes=normal,view=[-
3..3,0..3],scaling=constrained);
```

3.2.9. Построение графиков функций в Maplet-окне

При изучении графиков элементарных функций вне особенностей системы Maple полезно Maplet-приложение, окно которого представлено на рис. 3.5. Открывается это окно исполнением команды **Tools ⇒ Precalcus ⇒ Standard Functions...** при работе в стандартном интерфейсе Maple 9.5.

В окне в разделе определения функций Define Function имеется список элементарных функций, графики которых можно просматривать. Однако возможно построение и графиков простых функций более сложного вида, например x*sin(x) вместо sin(x), – это и иллюстрирует график, представленный на рис. 3.5.

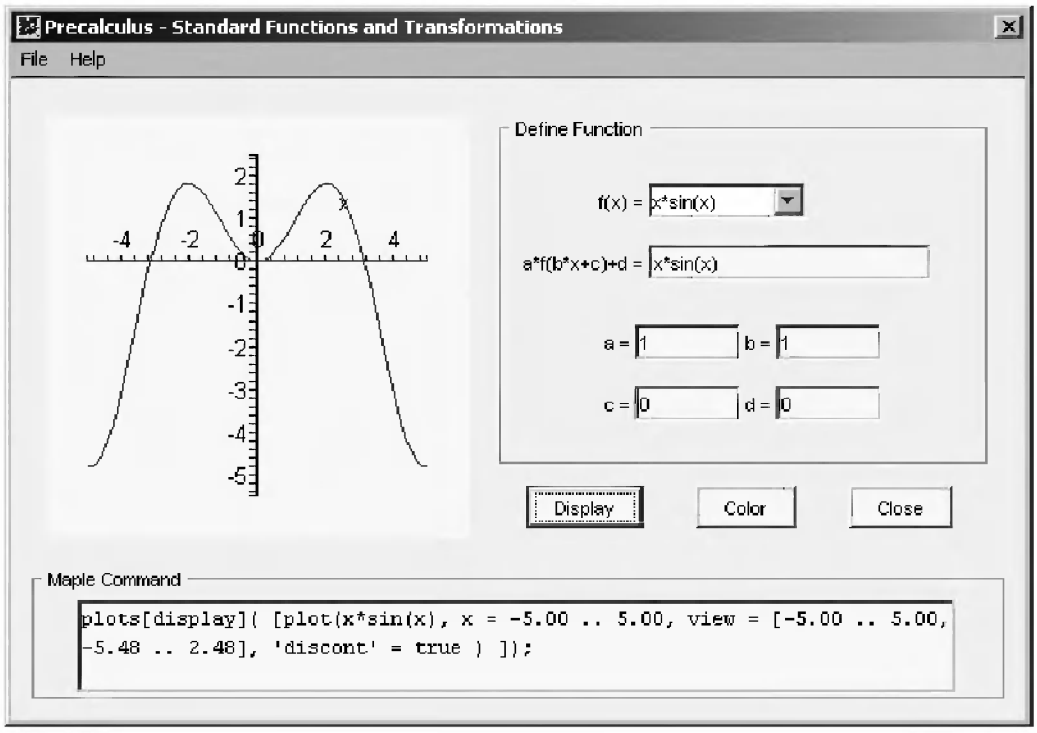


Рис. 3.5. Maplet-окно для изучения функций и построения их графиков

3.2.10. Арифметические и элементарные функции в системе Mathematica

Синтаксис арифметических и элементарных функций в системе Mathematica отличается от описанного выше для СКМ Maple. В частности, все функции начинаются с заглавной буквы, а список входных параметров задается в квадратных скобках – круглые используются при записи выражений.

Для выполнения арифметических действий в системах Mathematica 4/5 определены следующие *арифметические функции*:

- **Divide[x, y]** – возвращает результат деления x на y, эквивалентно выражению x y^–1.
- **Plus[x, y,...]** – возвращает сумму элементов списка.
- **PowerMod[a, b, n]** – возвращает **Mod[a^b, n]**. Для b<0 возвращает инверсию остатка.
- **Times[x,y,...]** – возвращает произведение аргументов x*y*...
- **Mod[m, n]** – возвращает остаток от деления m на n. Результат имеет такой же знак, как n.

Примеры на применение арифметических функций:

Ввод (*In*)	**Вывод (*Out*)**
`Divide[1.,3]`	`0.333333`
`Mod[123,20]`	`3`
`Mod[123,-20]`	`-17`
`Mod[-123,20]`	`17`
`Plus[2,3,4]`	`9`
`Times[2,3,4]`	`24`

Для обмена значениями переменных x и y можно использовать выражение:

```
{x,y}={y,x}
```

Следующие функции служат для приведения вещественных чисел к ближайшим целым по определенным правилам:

- **Ceiling[x]** – возвращает значение наименьшего целого числа, большего или равного *x*.
- **Floor[x]** – возвращает наибольшее целое число, не превышающее данного *x*.
- **Quotient[n, m]** – возвращает целое значение *n/m*, определяемое как **Floor[n/m]**.
- **Round[x]** – округляет *x* до ближайшего целого.

Хотя аргументами этих функций указано значение *x*, под ним можно понимать список вещественных чисел. Следующие примеры поясняют это и наглядно иллюстрируют правила приведения к целым числам:

Ввод (*In*)	**Вывод (*Out*)**
`Ceiling[{-.5.9,-5.1,5,5.1,5.9}]`	`{-5,-5,5,6,6}`
`Floor[{-.5.9,-5.1,5,5.1,5.9}]`	`{-6,-6,5,5,5}`
`Round[{-.5.9,-5.1,5,5.1,5.9}]`	`{-6,-5,5,5,6}`

Ряд функций обеспечивает нахождение делителей целых чисел и наименьшее общее кратное:

- **Divisors[n]** – возвращает список целочисленных делителей числа *n*.
- **DivisorSigma[k, n]** – возвращает сумму k-ых степеней положительных делителей числа *n*.
- **ExtendedGCD[n, m]** – возвращает расширенный наибольший общий делитель целых чисел *n* и *m*.
- **GCD[n1, n2, ...]** – возвращает наибольший общий делитель целых чисел *ni*.
- **LCM[n1, n2, ...]** – возвращает наименьшее общее кратное целых чисел *ni*.

Примеры на применение этих функций:

Ввод (*In*)	**Вывод (*Out*)**
`Divisors[123]`	`{1,3,41,123}`
`DivisorSigma[17,3]`	`129140164`
`ExtendedGCD[144,12]`	`{12,{0,1}}`
`GCD[144,12,6]`	`6`
`LCM[124,12,6]`	`372`

К целочисленным функциям систем Mathematica можно отнести функции вычисления факториала и двойного факториала:

- **Factorial[n]** или **n!** – возвращает значение факториала числа *n* (n!=n*(n-1)*...*3*2*1, причем 0!=1 и 1!=1).
- **Factorial2[n]** или **n!!** – возвращает значение двойного факториала числа *n*, равное n*(n-2)*(n-4)*....

Примеры на вычисление факториалов:

Ввод (*In*)	**Вывод (*Out*)**
`Factorial[10]`	`3628800`
`20!`	`2432902008176640000`
`10!!`	`3840`
`20!//N`	
`2.4329×10^18`	

Mathematica, как и Maple, способна вычислять факториалы больших чисел. Практически мгновенно вычисляются значения до 1000!, хотя результат при этом занимает несколько страниц на экране дисплея. Можно вычислить даже 10000!, но для этого потребуется время до нескольких минут (зависит от типа ПК и его скорости вычислений).

Следующие функции служат для получения простых чисел:

- **Prime[n]** – возвращает n-ое простое число.
  Пример: **Prime[5]** возвращает пятое простое число – 11. Всего несколько секунд надо системе для вычисления миллиардного простого числа: `Prime[10^9]` – `22801763489`.
- **PrimePi[x]** – возвращает количество простых чисел, не превышающих *x*.
  Пример: **PrimePi[10]** – возвращает 4.

Эти функции полезны при решении задач теории чисел.

Ввиду общеизвестности элементарных функций приведем лишь несколько таких функций, реализованных в системе Mathematica:

- **Abs[z]** – возвращает абсолютное значение для действительного числа и модуль для комплексного z.
- **ArcSin[z]** – возвращает арксинус комплексного аргумента z.
- **Exp[z]** – возвращает значение exp(z).
- **Log[b, z]** – возвращает логарифм по основанию b.
- **Sign[x]** – возвращает –1, 0 или 1 , если аргумент x соответственно отрицательный, нуль или положительный.
- **Sqrt[z]** – возвращает корень квадратный из аргумента z.
- **Tanh[z]** – возвращает гиперболический тангенс z.

В связи с общеизвестностью элементарных функций ограничимся приведением лишь нескольких примеров на их использование:

Ввод (*In*)	**Вывод (*Out*)**
Sqrt[2]	Sqrt[2]
Sqrt[2.]	1.41421
2*Sin[1]	2 Sin[1]
N[2*Sin[1]]	1.68294
Log[Exp[1]]	1
Simplify[Sin[x]/Cos[x]]	Tan[x]
ComplexExpand[Sin[a+b*I]]	Cos[b] Sin[a] + I Cos[a] Sinh[b]

3.3. Работа со специальными функциями

3.3.1. Обзор специальных математических функций

Специальные математические функции являются решениями дифференциальных уравнений, которые, как правило, невозможно представить через элементарные функции. Через такие функции нередко представляются и многие интегралы. Наиболее полное описание подобных функций дано в [125] и в ряде других книг [123, 124].

Функция Эйри формирует пару линейно независимых решений дифференциального уравнения вида:

$$\frac{d^2W}{dZ^2} - ZW = 0.$$

Связь между функцией Эйри и модифицированной функцией Бесселя выражается формулой:

$$Ai(Z) = \left[\frac{1}{\pi}\sqrt{\frac{Z}{3}}\right] K_{\frac{1}{3}}(\zeta),$$

где

$$\zeta = \frac{2}{3} Z^{\frac{3}{2}}.$$

Дифференциальное уравнение вида

$$z^2 \frac{d^2y}{dz^2} + z\frac{dy}{dz} + (z^2 - \nu^2)y = 0,$$

где ν – неотрицательная константа, называется уравнением Бесселя, и его решения известны как функция Бесселя. $\mathbf{J_\nu(z)}$ и $\mathbf{J_{-\nu}(z)}$ формируют фундаментальное

множество решений уравнения Бесселя для неотрицательных значений nn (так называемые *функции Бесселя* первого рода):

$$J_\nu(z)=\frac{z^\nu}{2}\sum_{k=0}^{\infty}\frac{\left(\frac{-z^2}{4}\right)^k}{k!\Gamma(\nu+k+1)},$$

где для гамма-функции используется следующее представление:

$$\Gamma(a)=\int_0^\infty e^{-t}t^{a-1}dt.$$

Второе решение уравнения Бесселя, линейно независимое от $\mathbf{J_\nu(z)}$, определяется как

$$Y_\nu(z)=\frac{J_\nu(z)\cos(\nu\pi)-J_{-\nu}(z)}{\sin(\nu\pi)}$$

и задает функции Бесселя второго рода $\mathbf{Y_\nu(z)}$.

Функции Бесселя третьего рода (функции Ханкеля) и функция Бесселя связаны следующим выражением:

$$H^{(1)}{}_\nu(z)=J_\nu(z)+iY_\nu(z),$$
$$H^{(2)}{}_\nu(z)=J_\nu(z)-iY_\nu(z).$$

Дифференциальное уравнение вида

$$z^2\frac{d^2y}{dz^2}+z\frac{dy}{dz}-(z^2+\nu^2)y=0,$$

где ν – неотрицательная константа – называется модифицированным уравнением Бесселя, и его решения известны как модифицированные функции Бесселя $\mathbf{I\nu(z)}$ и $\mathbf{I\text{-}\nu(z)}$. $\mathbf{K_\nu(z)}$ – второе решение модифицированного уравнения Бесселя, линейно независимое от $\mathbf{I_\nu(z)}$. $\mathbf{I_\nu(z)}$ и $\mathbf{K_\nu(z)}$ определяются как

$$I_\nu(z)=\frac{z^\nu}{2}\sum_{k=0}^{\infty}\frac{\left(\frac{z^2}{4}\right)^k}{k!\Gamma(\nu+k+1)}$$

и

$$K_\nu(z)=\left(\frac{\pi}{2}\right)\frac{I_{-\nu}(z)-I_\nu(z)}{\sin(\nu\pi)}.$$

Бета-функция определяется как

$$B(z,w)=\int_0^1 t^{z-1}(1-t)^{w-1}dt=\frac{\Gamma(z)\Gamma(w)}{\Gamma(z+w)},$$

где $\Gamma(z)$ – гамма-функция. Неполная бета-функция определяется интегральным выражением

$$I_x(z,w)=\frac{1}{B(z,v)}\int_0^x t^{z-1}(1-t)^{w-1}dt.$$

Эллиптические функции Якоби определяются интегралом

$$u=\int_0^{\varphi}\frac{d\theta}{(1-m\sin^2\theta)^{\frac{1}{2}}}$$

и $\mathrm{sn}(u)=\sin\phi, \mathrm{cn}(u)=\cos\phi, \mathrm{dn}(u)=(1-\sin^2\phi)^{\frac{1}{2}}, \mathrm{am}(u)=\phi$. В некоторых случаях при определении эллиптических функций используются модули **k** вместо параметра **m.** Они связаны выражением

$k^2 = m = \sin^2\alpha.$

Полные эллиптические интегралы первого и второго рода определяются следующим образом:

$$K(m)=\int_0^1\left[(1-t^2)(1-mt^2)\right]^{-\frac{1}{2}}dt=\int_0^{\frac{\pi}{2}}\frac{d\theta}{(1-msin^2\theta)^{\frac{1}{2}}},$$

$$E(m)=\int_0^1(1-t^2)^{-\frac{1}{2}}(1-mt^2)^{\frac{1}{2}}dt=\int_0^{\frac{\pi}{2}}(1-m\sin^2\theta)^{\frac{1}{2}}d\theta.$$

Функция ошибки (интеграл вероятности) определяется как

$$erf(X)=\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt.$$

erf(X) – возвращает значение функции ошибки для каждого элемента вещественного массива **X**.

Остаточная функция ошибки задается соотношением

$$erfc(X)=\frac{2}{\sqrt{\pi}}\int_x^{\infty} e^{-t^2}dt=1-erf(X).$$

Встречается и масштабированная остаточная функция ошибки. Эта функция определяется так:

$$erfcx(x)=e^{x^2}erfc(x).$$

Интегральная показательная функция определяется следующим образом:

$$E_1(x)=\int_x^{\infty}\frac{e^{-t}}{t}dt.$$

expint(X) – вычисляет интегральную показательную функцию для каждого элемента **X**.

Гамма-функция определяется выражением

$$\Gamma(a)=\int_0^x e^{-t}t^{a-1}dt.$$

Неполная гамма-функция определяется как

$$P(x,a)=\frac{1}{\Gamma(a)}\int_0^x e^{-t}t^{a-1}dt.$$

Перейдем к функциям, представляющим *ортогональные полиномы*. *Функция Лежандра* определяется следующим образом:

$$P_n^m=(-1)^m(1-x^2)^{\frac{m}{2}}\frac{d^m P_n(x)}{d\quad x^m},$$

где $P_n(x)$ – *полином Лежандра* степени n, определяется так:

$$P_n(x)=\frac{1}{2^n n!}\left[\frac{d^n(x^2-1)^n}{dx}\right].$$

3.3.2. Специальные математические функции системы Maple

Maple 9.5/10/11 имеет практически полный набор специальных математических функций:

- `AiryAi (Bi)` – функции Эйри;
- `AngerJ` – функция Ангера;
- `bernoulli` – числа и полиномы Бернулли;
- `BesselI (J, K, Y)` – функции Бесселя разного рода;
- `Beta` – бета-функция;
- `binomial` – биноминальные коэффициенты;
- `Chi` – интегральный гиперболический косинус;
- `Ci` – интегральный косинус;
- `csgn` – комплексная сигнум-функция;
- `dilog` – дилогарифм;
- `Dirac` – дельта-функция Дирака;
- `Ei` – экспоненциальный интеграл;
- `EllipticCE (CK, CPi, E, F, K, Modulus, Nome, Pi)` – эллиптические интегралы;
- `erf` – функция ошибок;
- `erfc` – дополнительная функция ошибок;
- `euler` – числа и полиномы Эйлера;
- `FresnelC (f, g, S)` – интегралы Френеля;
- `GAMMA` – гамма-функция;
- `GaussAGM` – арифметико-геометрическое среднее Гаусса;
- `HankelH1 (H2)` – функции Ганкеля;
- `harmonic` – частичная сумма серии гармоник;
- `Heaviside` – функция Хевисайда;
- `JacobiAM (CN, CD, CS, DN, DC, DS, NC, ND, NS, SC, SD, SN)` – эллиптические функции Якоби;

- `JacobiTheta1 (2, 3, 4)` – дзета-функции Якоби;
- `JacobiZeta` – зет-функция Якоби;
- `KelvinBer (Bei, Her, Hei, Ker, Kei)` – функции Кельвина;
- `Li` – логарифмический интеграл;
- `lnGAMMA` – логарифмическая гамма-функция;
- `MeijerG` – G-функция Мейджера;
- `pochhammer` – символ Похгамера;
- `polylog` – полилогарифмическая функция;
- `Psi` – дигамма-функция;
- `Shi` – интегральный гиперболический синус;
- `Si` – интегральный синус;
- `Ssi` – синусный интеграл смещения;
- `StruveH (L)` – функции Струве;
- `surd` – неглавная корневая функция;
- `LambertW` – W-функция Ламберта;
- `WeberE` – E-функция Вебера;
- `WeierstrassP` – P-функция Вейерштрасса;
- `WeierstrassPPrime` – производная P-функции Вейерштрасса;
- `WeierstrassZeta` – зета-функция Вейерштрасса;
- `WeierstrassSigma` – сигма-функция Вейерштрасса;
- `Zeta` – зета-функция Римана и Гурвица.

На рис. 3.6 даны примеры применения ряда специальных функций. Обратите особое внимание на первый пример. Он показывает, как средствами системы Maple задается определение функций Бесселя. Показано, что функции Бесселя являются решениями заданного на рис. 3.6 дифференциального уравнения второго порядка. Maple 9.5/10 способен вычислять производные и интегралы от специальных функций.

Еще несколько примеров работы со специальными функциями представлены на рис. 3.7. Как видно из приведенных примеров, на экране монитора можно получить математически ориентированное представление специальных функций, обычно более предпочтительное, чем представление на Maple-языке или в текстовом формате. Записи функций при этом выглядят как в обычной математической литературе.

На рис. 3.7 показаны примеры разложения специальных функций в ряды и применения функции `convert` для их преобразования. Любопытно отметить, что в двух первых примерах вывод оказался иным, чем в предшествующих версиях Maple. Да и в них вывод для этих примеров отличался. Это говорит о непрерывной работе разработчиков над алгоритмами символьных вычислений и необходимости переработки примеров при переходе от одной версии Maple к другой.

Много информации о поведении специальных функций дает построение их графиков. На рис. 3.8 показано построение семейства графиков функций Бесселя `BesselJ` разного порядка и гамма-функции. Эти функции относятся к числу наиболее известных. Если читателя интересуют те или иные специальные функции, следует прежде всего построить и изучить их графики.

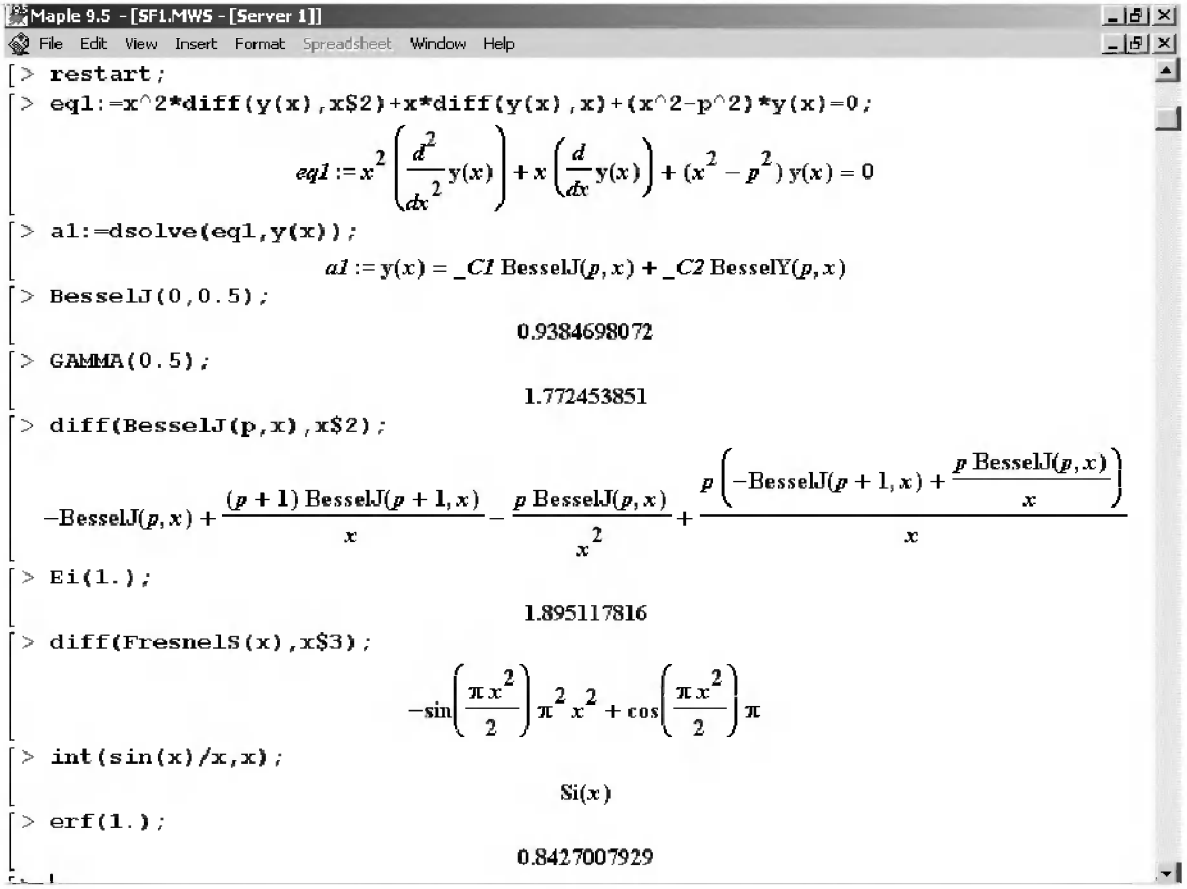


Рис. 3.6. Примеры применения специальных функций

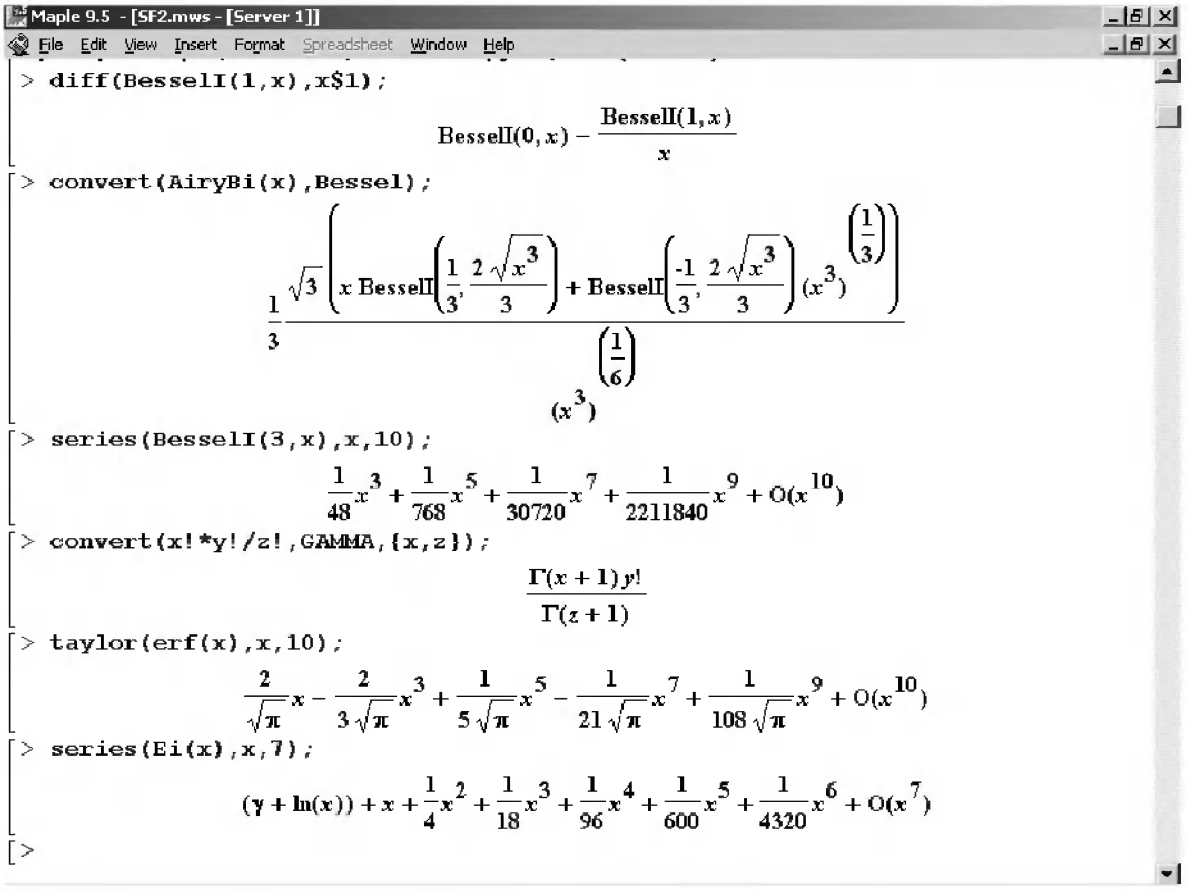


Рис. 3.7. Примеры работы со специальными математическими функциями

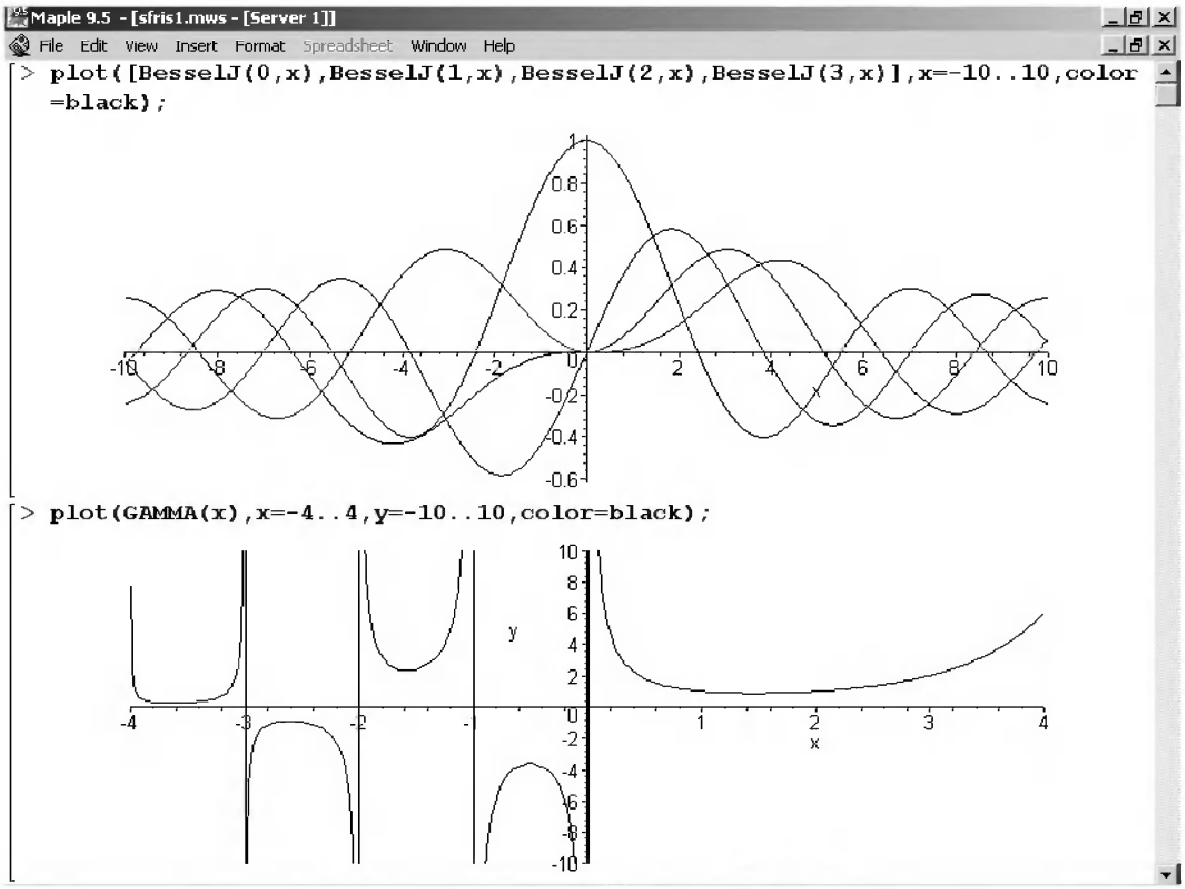


Рис. 3.8. Графики функций Бесселя и гамма-функции

3.3.3. Консультант по функциям

Математикам, серьезно работающим с функциями, большую помощь может оказать имеющийся в составе Maple 9.5/10 консультант по функциям, вводимый командой:

```
FunctionAdvisor()              FunctionAdvisor(topics, quiet)
FunctionAdvisor(Topic, function, quiet)
```

Здесь: `topics` – строковый параметр, задающий вывод тематической информации, `quiet` – строковый параметр, указывающий на вывод вычислительных данных, `Topic` – задание темы и `function` – задание имени функции или класса функций.

Команда `FunctionAdvisor()` выводит правила применения консультанта по функциям. Следующие примеры показывают вывод определений функций Бесселя:

```
> FunctionAdvisor(describe, Bessel);
```

BesselI = *Modified Bessel function of the first kind*,
BesselJ = *Bessel function of the first kind*,
BesselK = *Modified Bessel function of the second kind*,
BesselY = *Bessel function of the second kind*

```
> FunctionAdvisor(describe, BesselJ);
```

BesselJ = *Bessel function of the first kind*

В следующем примере выводится информация о представлении функции синуса в виде ряда, представленного суммой его членов:

```
> FunctionAdvisor( sum_form, sin);
```

$$\left[\sin(z)=\sum_{\_kl=0}^{\infty}\frac{(-1)^{\_kl}\ z^{(2\_kl+1)}}{(2\_kl+1)!},\ \text{with no restrictions on } (z)\right]$$

Еще один пример показывает вывод интегрального представления синусного интеграла Френеля:

```
> FunctionAdvisor( integral_form, FresnelS);
```

$$\left[FrenselS(z)=\int_0^z \sin\left(\frac{\pi\_kl^2}{2}\right)d\ \_kl,\ \text{with no restrictions on } (z)\right]$$

3.4. Специальные функции в системе Mathematica

По обилию специальных математических функций лидирует система Mathematica. Ниже представлено описание таких функций в системах Mathematica с примерами их вычислений.

3.4.1. Ортогональные многочлены

Mathematica имеет следующие функции, возвращающие значения ортогональных многочленов:

- **ChebyshevT[n, x]** – Чебышева n-ой степени первого рода.
- **CyebyshevU[n, x]** – Чебышева n-ой степени второго рода.
- **HermiteH[n, x]** – Эрмита n-ой степени.
- **JacobiP[n, a, b, x]** – Якоби n-ой степени.
- **GegenbauerC[n, m, x]** – Гегенбауэра.
- **LaguerreL[n, x]** – Лагерра n-ой степени.
- **LaguerreL[n, a, x]** – обобщенного Лагерра n-ой степени.
- **LegendreP[n, x]** – n-го порядка Лежандра.
- **LegendreP[n, m, x]** – присоединенного полинома Лежандра.
- **LegendreQ[n, z]** – n-го порядка функция Лежандра второго рода.
- **LegendreQ[n, m, z]** – присоединенная функция Лежандра второго рода.
- **LegendreType** – опция для **LegendreP** и **LegendreQ**; она указывает выборы разрывов кривой для функций Лежандра на комплексной плоскости.

Следующие примеры иллюстрируют работу с ортогональными многочленами:

Ввод (*In*)	**Ввод (*Out*)**
`ChebyshevT[5,0.2]`	`0.84512`
`HermiteH[4,3]`	`876`
`JacobiP[3,1,2,0.2]`	`-0.256`

```
GegenbauerC[3,1,x]                    3
                                -4 x + 8 x
                                    3       5
                                15 x  - 70 x + 63 x
LegendreP[5,x]                  ------------------
                                          8
LegendreQ[2,0.2]                -0.389202
```

Важно отметить, что при указании конкретного значения параметра n и символьном значении параметра x функции этой группы возвращают присущие им представления через степенные многочлены с соответствующими коэффициентами.

3.4.2. Интегральные показательные и родственные им функции

К другой известной группе специальных функций относятся интегральные показательные и родственные им функции:

- **CoshIntegral[x]** – гиперболический интегральный косинус.
- **CosIntegral[x]** – интегральный косинус Ci(x).
- **Erf[z]** – функция ошибок (интеграл вероятности).
- **Erf[z0, z1]** – обобщенная функция ошибок erf(z1) – erf(z0).
- **Erfc[z]** – дополняющая функция ошибок 1 – erf(z).
- **Erfi[z]** – мнимое значение функции ошибок –i erf(iz).
- **ExpIntegralE[n, z]** – интегральная показательная функция E(n,z).
- **ExpIntegralEi[z]** – интегральная показательная функция Ei(z).
- **LogIntegral[z]** – интегральный логарифм li(z).
- **SinhIntegral[x]** – интегральный гиперболический синус.
- **SinIntegral[x]** – интегральный синус Si(x).

Примеры на применение этих функций:

Ввод (*In*)	**Ввод (*Out*)**
`CosIntegral[1.]`	`0.337404`
`Erf[2.+I*3.]`	`-20.8295 + 8.68732 I`
`Erf[2.,3.]`	`0.00465564`
`ExpIntegralE[3,1.]`	`0.109692`
`ExpIntegralEi[1.]`	`1.89512`
`LogIntegral[2.+3.*I]`	`2.3374 + 2.51301 I`
`SinIntegral[1.]`	`0.946083`

Следует обратить внимание на то, что большая часть из этих функций может иметь комплексный аргумент.

3.4.3. Гамма- и полигамма-функции

Широко используются гамма-функция и относящиеся к ней родственные функции:

- **Gamma[a]** – Эйлеровая гамма-функция.
- **Gamma[a, z]** – неполная гамма-функция.
- **Gamma[a, z0, z1]** – обобщенная неполная гамма-функция Gamma(a, z0) – Gamma(a, z1).

- **GammaRegularized[a, z]** – регуляризованная неполная гамма-функция Q(a, z) = Gamma(a,z)/Gamma(a).
- **GammaRegularized[a, z0, z1]** – обобщенная неполная гамма-функция Q(a,z0) – Q(a, z1);
- **LogGamma[z]** – логарифм Эйлеровой гамма-функции.
- **PolyGamma[z]** – дигамма-функция psi(z).
- **PolyGamma[n, z]** – n-ая производная от дигамма-функции.

Приведем примеры на вычисление этих функций:

Ввод (*In*)	**Вывод (*Out*)**
`Gamma[0.5]`	`1.77245`
`Gamma[1,2.+3.*I]`	`-0.133981 - 0.0190985 I`
`Gamma[1,2.,3.]`	`0.0855482`
`GammaRegularized[1,2.+3.I,4.+6.*I]`	`-0.139176 - 0.0366618 I`
`LogGamma[0.5]`	`0.572365`
`LogGamma[2.+3.*I]`	`-2.09285 + 2.3024 I`
`PolyGamma[1]`	`-EulerGamma`
`PolyGamma[1.]`	`-0.577216`
`PolyGamma[2.+3.*I]`	`1.20798 + 1.10413 I`

Как видно из этих примеров, данный класс функций (как и многие другие) определен в общем случае для комплексного значения аргумента.

3.4.4. Функции Бесселя

Функции Бесселя широко используются в анализе и моделировании волновых процессов. В системе Mathematica к этому классу относятся следующие функции:

- **BesselI[n, z]** – модифицированная функция Бесселя первого рода I(n, z).
- **BesselJ[n, z]** – функция Бесселя первого рода J(n,z).
- **BesselK[n, z]** – модифицированная функция Бесселя второго рода K(n, z).
- **BesselY[n, z]** – функция Бесселя второго рода Y(n, z).

Соотношения между этими функциями хорошо известны – см. [125] и справочную базу данных системы Mathematica. Следующие примеры показывают вычисление функций Бесселя:

Ввод (*In*)	**Вывод (*Out*)**
`BesselI[0,1.]`	`1.26607`
`BesselI[3,1.]`	`0.0221684`
`BesselI[1,2.+3.*I]`	`-1.26098 + 0.780149 I`
`BesselJ[2,2.+3.*I]`	`1.25767 + 2.31877 I`
`N[BesselJ[1,0.5]]`	`0.242268`
`N[BesselJ[1,2+I*3]]`	`3.78068 - 0.812781 I`

3.4.5. Гипергеометрические функции

Класс гипергеометрических функций представлен в системе Mathematica следующими встроенными в ядро функциями:

- **HypergeometricU[a, b, z]** – конфлюэнтная (вырожденная) гипергеометрической функции U(a, b, z).

- **Hypergeometric0F1[a, z]** – гипергеометрическая функция $F_0(a, z)$.
- **Hypergeometric1F1[a, b, z]** – вырожденная гипергеометрической функции Куммера $F_1(a; b; z)$.
- **Hypergeometric2F1[a, b, c, z]** – гипергеометрическая функция $F_2(a, b; c; z)$.

Следующие примеры показывают вычисления гипергеометрических функций:

Ввод (*In*)	**Вывод (*Out*)**
`Hypergeometric0F1[2.,1.]`	`1.59064`
`Hypergeometric0F1[2.,2.+3.*I]`	`1.22457+2.51372 I`
`Hypergeometric1F1[1.,2.,2.+3.*I]`	`-1.03861+2.07929 I`
`Hypergeometric2F1[1.,2.,3.,2.+3.*I]`	`0.0291956+0.513051 I`

Следует отметить, что число этих функций в ядре новых версий даже несколько сокращено по сравнению с предшествующими версиями. Убраны довольно редко используемые функции, в имени которых имеется слово Regularized.

3.4.6. Эллиптические интегралы и интегральные функции

В ядро системы Mathematica входят эллиптические функции и функции вычисления эллиптических интегралов:

- **EllipticE[m]** – полный эллиптический интеграл E(m).
- **EllipticE[phi, m]** – эллиптический интеграл второго рода E(phi|m).
- **EllipticExp[u, {a, b}]** – обобщенный экспоненциал, связанный с эллиптической кривой y^2 = x^3 + a x^2 + b x.
- **EllipticExpPrime[u, {a, b}]** – производная по первому аргументу **EllipticExp[u, {a, b}]**.
- **EllipticF[phi, m]** – эллиптический интеграл первого рода F(phi|m).
- **EllipticK[m]** – полный эллиптический интеграл первого рода K(m).
- **EllipticLog[{x, y}, {a, b}]** – обобщенный логарифм, связанный с эллиптической кривой y^2 = x^3+ a x^2 + b x.
- **EllipticNomeQ[m]** – возвращает значение q = Exp[-PiEllipticK[1-m]/EllipticK[m]].
- **EllipticPi[n, phi, m]** – эллиптический интеграл третьего рода Pi(n; phi|m).
- **EllipticPi[n, m]** – полный эллиптический интеграл Pi(n|m).
- **EllipticTheta[i, z, q]** – эллиптическая дзета-функция: theta_i(z,q), где i = 1, 2, 3, или 4.
- **EllipticThetaC[u, m]** – эллиптическая дзета-функция Невилла theta_c(u, m).
- **EllipticThetaD[u, m]** – эллиптическая дзета-функция Невилла theta_d(u, m).
- **EllipticThetaN[u, m]** – эллиптическая дзета-функция Невилла theta_n(u, m).
- **EllipticThetaPrime[i, z, q]** – производная по второму аргументу эллиптической дзета-функции theta_i(z, q), где i=1,2,3, или 4.
- **EllipticThetaS[u, m]** – эллиптическая дзета-функция Невилла theta_s(u, m).
- **FresnelC[x]** – интеграл Френеля C(x).

- **FresnelS[x]** – интеграл Френеля S(x).
- **InverseJacobi**[v, m]** – обратная эллиптическая функция Якоби с обобщенным названием **. Возможны следующие наименования для **: **CD** , **CN**, **CS**, **DC**, **DN**, **DS**, **NC**, **ND**, **NS**, **SC**, **SD** и **SN**.
- **JacobiAmplitude[u, m]** – амплитуда для эллиптических функций Якоби.
- **Jacobian** – опция для FindRoot; может применяться для указания якобиана системы функций, для которых ищется корень.
- **Jacobi**[u, m]** – эллиптическая функция Якоби с обобщенным именем **, которое может принимать значения **CD** , **CN**, **CS**, **DC**, **DN**, **DS**, **NC**, **ND**, **NS**, **SC**, **SD** и **SN**.
- **JacobiSymbol[n, m]** – символ Якоби от n и m.
- **JacobiZeta[phi, m]** – дзета-функция Якоби Z(phi|m).
- **WeierstrassP[u, g2, g3]** – эллиптическая функция Вейерштрасса P.
- **WeierstrassPPrime[u, g2, g3]** – производная эллиптической функции Вейерштрасса P' по переменной u.

Приведем примеры на использование некоторых из этих функций:

Ввод (*In*)	**Вывод (*Out*)**
`EllipticE[Pi,0.1]`	`3.06152`
`EllipticF[Pi/2,0.1]`	`1.61244`
`EllipticPi[Pi,0.1]`	`-0.0266412 - 1.09088 I`
`EllipticK[0.1]`	`1.61244`
`FresnelC[1.0]`	`0.779893`
`FresnelS[1.0]`	`0.438259`
`JacobiCD[1,0.2]`	`0.605887`
`JacobiZeta[Pi,0.5]`	`0`
`WeierstrassPPrime[1.,2.,3.]`	`-1.31741`

Эллиптические функции (интегралы) широко используются в практике выполнения оптических расчетов и в астрофизике.

3.4.7. Функции Эйри

Функции Эйри представляют независимые решения линейного дифференциального уравнения $w'' - zw = 0$. В Mathematica эти функции представлены следующим набором:

- **AiryAi[z]** – возвращает значение функции Эйри Ai(z).
- **AiryAiPrime[z]** – возвращает значение производной функции Эйри Ai'(z).
- **AiryBi[z]** – возвращает значение функции Эйри Bi(z).
- **AiryBiPrime[z]** – возвращает производную функции Эйри Bi'(z).

Ниже представлены примеры на вычисление функций Эйри:

Ввод (*In*)	**Вывод (*Out*)**
`AiryAi[2.+3.*I]`	`0.00810446 + 0.131178 I`
`AiryBi[2.+3.*I]`	`-0.396368 - 0.569731 I`
`AiryBiPrime[2.+3.*I]`	`0.349458 - 1.10533 I`

3.4.8. Бета-функция и относящиеся к ней функции

Класс бета-функций, имеющих специальное интегральное представление (см. [125]), в Mathematica представлен следующим набором:

- **Beta[a, b]** – Эйлерова бета-функция B(a, b).
- **Beta[z, a, b]** – неполная бета-функция.
- **Beta[z0, z1, a, b]** – обобщенная неполная бета-функция Beta[z1, a, b] – Beta[z0, a, b].
- **BetaRegularized[z,a,b]** – регуляризированная неполная бета-функция I(z,a,b) = Beta[z, a, b]/Beta[a, b].
- **BetaRegularized[z0, z1, a,b]** – регуляризированная обобщенная неполная бета-функция I(z1,a,b) – I(z0, a, b).

Примеры на вычисление этих функций представлены ниже:

Ввод (*In*)	**Вывод (*Out*)**
`Beta[1.,2.]`	`0.5`
`Beta[1.,2.,3.]`	`0.0833333`
`Beta[2.+3.*I,4.+6.*I,1,2]`	`4. - 12. I`
`BetaRegularized[0.1,1,2]`	`0.19`

3.4.9. Специальные числа и полиномы

Для вычисления специальных чисел и полиномов служит следующая группа функций:

- **BernoulliB[n]** – n-ое число Бернулли.
- **BernoulliB[n, x]** – полином Бернулли n-ой степени.
- **Binomial[n, m]** – биномиальный коэффициент.
- **Cyclotomic[n, x]** – циклотомический полином порядка n по x.
- **EulerE[n]** – n-ое число Эйлера.
- **EulerE[n, x]** – n-ый полином Эйлера.
- **EulerPhi[n]** – Эйлерова функция сумм phi(n) – количество положительных целых чисел, не превосходящих n и взаимно простых с n.
- **Fibonacci[n]** – n-ое число Фибоначчи.
- **Fibonacci[n,x]** – полином Фибоначчи $F_n(x)$.
- **Multinomial[n1, n2, ...]** – мультиномиальный коэффициент (n1+n2+...)!/(n1! n2! ...).
- **NBernoulliB[n]** – численное значение n-го числа Бернулли.
- **NBernoulliB[n, d]** – n-ое число Бернулли до d-цифровой точности представления.
- **Pochhammer[a, n]** – символ Похгамера.
- **StirlingS1[n, m]** – число Стирлинга первого рода.
- **StirlingS2[n, m]** – число Стирлинга второго рода.

Ниже представлены примеры на вычисление данных функций:

Ввод (*In*)	**Вывод (*Out*)**
`N[BernoulliB[2]]`	`0.166667`
`BernoulliB[2,0.1]`	`0.0766667`
`Binomial[6,4]`	`15`
`Cyclotomic[5,0.2]`	`1.2496`
`EulerE[2]`	`-1`
`EulerE[2,0.1]`	`-0.09`
`EulerPhi[2]`	`1`
`Fibonacci[10]`	`55`
`Fibonacci[6,x]`	$3\,x + 4\,x^3 + x^5$
`Pochhammer[1,3]`	`6`
`StirlingS1[8,4]`	`6769`

3.4.10. Другие специальные функции СКМ Mathematica

Помимо описанных выше функций, в ядро системы входит также ряд других, менее распространенных функций:

- **ArithmeticGeometricMean[a, b]** – арифметико-геометрическое среднее значение аргументов a и b.
- **IncludeSingularTerm** – опция для LerchPhi и Zeta, определяющая, следует ли включать члены вида (k + a)^-s при k + a = 0.
- **InverseErf[s]** – инверсная функция ошибок.
- InverseErfc[s] – инверсная дополнительная функция ошибок.
- **InverseGammaRegularized[a,s]** – инверсная регуляризированная неполная гамма-функция.
- **InverseBetaRegularized[s,a,b]** – инверсная регуляризированная неполная бета-функция.
- **InverseSeries[s]** – берет ряд s, порождаемый директивой **Series**, и возвращает ряд для функции, обратной по отношению к функции, представленной рядом s.
- **InverseSeries[s, y]** – обратный ряд по переменной y.
- **InverseWeierstrassP[{P, P'}, g2, g3]** – возвращает величину u такую, что P= WeierstrassP[u, g2, g3] и P'= WeierstrassPPrime[u, g2, g3]. Следует заметить, что P и P' не являются независимыми.
- **JordanForm[A]** – возвращает { S, J}, где A = S. J. **Inverse[S]** и J является канонической формой Жордана A.
- **LerchPhi[z, s, a]** – трансцендентная функция Лерча Phi(z, s, a).
- **MathieuC[*a*, *q*, *z*]** и **MathieuS[*a*, *q*, *z*]** – функции Матье.
- **MathieuCPrime[*a*, *q*, *z*]** и **MathieuSPrime[*a*, *q*, *z*]** – производные от функций Матье.

- **MathieuCharacteristic**[*r*, *q*]** – характеристическая функция Матье (** имеет значения A, B и Exponent).
- **MeijerG[{{a_1,...a_n},{a_{n+1},..a_p}},{{b1,....bm},{b_{m+1},..,b_q}},z]** – Мейджера G-функция.
- **MoebiusMu[n]** – значение функции Мебиуса mu(n).
- **PolyLog[n, z]** – n-ая полилогарифмическая функция от z.
- **RiemannSiegelTheta[t]** – аналитическая функция theta(t), удовлетворяющая уравнению **RiemannSiegelZ[t] = Exp[I RiemannSiegelTheta[t]] Zeta[1/2 + I t]**. Аргумент t не обязательно должен быть вещественным, но если является таковым, тогда **RiemannSiegelTheta[t]] = Im[LogGamma[1/4 + I t/2]] – t Log[Pi]/2**.
- **RiemannSiegelZ[t]** – возвращает значение **Exp[I RiemannSiegelTheta[t]] Zeta[1/2 + I t]**.
- **SphericalHarmonicY[l, m, theta, phi]** – сферическая гармоника Ylm(theta, phi).
- **Zeta[s]** – дзета-функция Римана zeta(s).
- **Zeta[s, a]** – возвращает значение обобщенной дзета-функции Римана.

Ниже даны примеры на использование ряда из этих функций:

Ввод (*In*)	**Вывод (*Out*)**
`LerchPhi[2.+3.*I,1,2]`	`0.0145978 + 0.256525 I`
`InverseErf[0.1]`	`0.088856`
`InverseErfc[0.1]`	`1.16309`
`InverseGammaRegularized[1, 0.5]`	`0.693147`
`InverseBetaRegularized[0.5, 1, 2]`	`0.292893`
`MathieuC[1,2,0.1]`	`0.196600+0.879889 I`
`MathieuS[1,2,0.1]`	`0.133005 – 0.0297195 I`
`MathieuCharacterisyicaA[1.5,2.]`	`2.85238`
`MeijerG[{{1,1},{}},{{1},{0}},x]`	`Log[1+x]`
`MoebiusMu[3]`	`-1`
`NBernoulliB[2]`	`0.166667`
`NBernoulliB[1,5]`	`-0.5`
`PolyLog[2,2.+3.*I]`	`-0.280988 + 3.01725 I`
`RiemannSiegelTheta[1.]`	`-1.76755`
`RiemannSiegelZ[1.]`	`-0.736305`
`SphericalHarmonicY[0.1,0.5,Pi/3,Pi/2]`	`0.195671 + 0.195671 I`
`Zeta[0.1]`	`-0.603038`
`Zeta[0.1,0.5]`	`-0.0432821`

В Mathematica 4 были добавлены новые встроенные функции **StruveH[n,z]** и **StruveL[n,z]**, вычисляющие функции Струве порядка n для комплексного аргумента z. Примечательно, что все специальные функции могут участвовать в символьных преобразованиях, порой довольно сложных. С десяток новых, но достаточно редко применяемых специальных функций добавлено в ядро системы Mathematica 6.

3.5. Специальные математические функции в СКМ Mathcad

Наряду с элементарными функциями в системе Mathcad содержится и ряд встроенных специальных математических функций. Их применение расширяет возможности системы при решении сложных математических задач.

3.5.1. Встроенные в ядро Mathcad специальные функции

В ядро системы Mathcad встроены функции Бесселя и гамма-функция. Функции Бесселя с вещественным аргументом в системе Mathcad представлены следующим набором:

- `J0(x)` – функция Бесселя первого рода нулевого порядка;
- `I0(x)` – модифицированная функция Бесселя первого рода нулевого порядка;
- `Y0(x)` – функция Бесселя второго рода нулевого порядка;
- `K0(x)` – модифицированная функция Бесселя второго рода нулевого порядка;
- `J1(x)` – функция Бесселя первого рода первого порядка;
- `I1(x)` – модифицированная функция Бесселя первого рода первого порядка;
- `Y1(x)` – функция Бесселя второго рода первого порядка;
- `K1(x)` – модифицированная функция Бесселя второго рода первого порядка;
- `Jn(n,x)` – функция Бесселя первого рода *n*-го порядка;
- `In(n,x)` – модифицированная функция Бесселя первого рода *n*-го порядка;
- `Yn(n,x)` – функция Бесселя второго рода *n*-го порядка;
- `Kn(n,x)` – модифицированная функция Бесселя второго рода *n*-го порядка.

Эти функции есть во всех вариантах поставки Mathcad, начиная с версии Mathcad 8. Читателю рекомендуется построить графики функций Бесселя, если они его интересуют.

Другой широко распространенной специальной функцией, вычисление которой (причем как при действительном, так и при комплексном аргументе **z**) предусмотрено в системе Mathcad, является гамма-функция **GG(z)**. Она широко применяется в статистических расчетах, в которых используется также функция ошибок **erf(x)**, называемая еще интегралом вероятности.

Наличие встроенных в систему этих наиболее распространенных математических функций расширяет ее возможности. Многие другие специальные математические функции могут быть определены через перечисленные встроенные функции или заданы своим интегральным либо дифференциальным представлением.

3.5.2. Дополнительные специальные функции

В систему Mathcad 8.0 PRO было введено около 50 новых функций. Среди них имеется ряд специальных математических функций.

Дополнительные функции Бесселя:

- `Ai(x)` – функция Эйри первого рода;
- `bei(n,x)` – мнимая часть функции Бесселя-Кельвина порядка `n`;
- `ber(n,x)` – действительная часть функции Бесселя-Кельвина порядка `n`;
- `Bi(x)` – функция Эйри второго рода;
- `js(n,x)` – сферическая функция Бесселя первого рода целого порядка `n (nii—200)` в точке `x (x>0)`;
- `ys(n,x)` – сферическая функция Бесселя второго рода порядка `n (nii—200)` в точке `x (x>0)`.

Гипергеометрические функции:

- `dhypergeom(m,n,M,N)` – гипергеометрическая функция;
- `fhyper(a,b,c,x)` – гипергеометрическая функция Гаусса в точке `x` с параметрами `a`, `b` и `c`;
- `mhyper(a,b,x)` – конфлюэнтная гипергеометрическая функция в точке `x` с параметрами `a` и `b`.

Ортогональные полиномы:

- `Her(n,x)` – полином Эрмита степени `n` с аргументом `x`;
- `Jac(n,a,b,x)` – полином Якоби степени `n` в точке `x` с параметрами `a` и `b`;
- `Lag(n,x)` – полином Лагерра степени `n` в точке `x`;
- `Leg(n,x)` – полином Лежандра степени `n` в точке `x`;
- `Tcheb(n,x)` – полином Чебышева первого рода степени `n` в точке `x`;
- `Ucheb(n,x)` – полином Чебышева второго рода степени `n` в точке `x`.

Неполная бета-функция:

- `ibeta(a,x,y)` – неполная бета-функция параметров `a`, `x` и `y`.

Появление этих функций делает систему Mathcad более удобной в применении, поскольку исключает задание данных функций пользователем. Все эти функции включены в состав систем последующих версий Mathcad. Так, в новейшую версию Mathcad 14 введены функции вычисления производных функций `Ai` и `Bi`.

3.5.3. Дополнительные неактивные функции

При загрузке символьного процессора (от системы Maple) система Mathcad распознает ряд дополнительных специальных функций:

- `FresnelC(x)` – интеграл Френеля $C(x)$;
- `FresnelS(x)` – интеграл Френеля $S(x)$;
- `Ci(x)` – интегральный косинус;

- `Si(x)` – интегральный синус;
- `Ei(x)` – интегральная показательная функция;
- `dilog(x)` – дилогарифм;
- `Psi(n,x)` – n-я производная пси-функции и т. д.

Выражения с неактивными функциями Mathcad размещает в буфере обмена, и их можно извлечь оттуда для просмотра.

3.5.4. Альтернативные функции с новой нормировкой в Mathcad

В Mathcad 11 был добавлен ряд новых функций со специальной нормировкой. Эти функции относятся к классу функций Бесселя, и их можно найти в списке функций Бесселя в окне, которое открывается активизацией кнопки $f(x)$. В Mathcad 12 к ним добавились также альтернативные функции Эйри из этой же группы. Имена альтернативных функций состоят из имени основной функции и расширения .sc. Например, если основная функция I0(z), то альтернативная будет I0.sc(z). В справке и в окне кнопки $f(x)$ указываются нормирующие множители для альтернативных функций. Читатель может легко построить графики основных и альтернативных функций и сравнить их между собой.

В Mathcad поддерживается приличное число специальных математических функций, которое к тому же растет от версии к версии. Например, новые функции вычисления производных функций Эйри включены в новейшую версию Mathcad 14. Однако многие специальные математические функции по-прежнему в систему не включены. В этом нет особой нужды, поскольку они легко вычисляются через встроенные операторы и функции системы.

3.6. Специальные функции других СКМ

3.6.1. Специальные функции в Derive

Средства Derive позволяют вычислять большинство из специальных математические функций, описанных выше, с помощью библиотек (в ядро Derive определения функций не включены). Поэтому перед использованием специальных функций необходима предварительная загрузка соответствующих файлов. Представлены следующие классы специальных функций:

- интегральные показательные функции (файл exp_int.mth);
- интегралы Френеля (файл fresnel.mch);
- функции Бесселя и Эйри (файл bessel.mth);
- гипергеометрические функции (файл hipergeo.mth);
- эллиптические интегралы (файл elliptic.mth);
- ортогональные полиномы (файл orth_pol.mth);
- дзета-функция, полилогарифм и дилогарифм (файл zeta.mth).

Библиотеки Derive содержат богатый набор специальных математических функций, который вполне удовлетворит потребности в численных вычислениях таких функций, возникающие у большинства научных работников и инженеров. Их определения в указанных файлах дают прекрасные примеры функционального программирования в области вычислений специальных математических функций. Однако, увы, символьные операции с этими функциями невозможны.

3.6.2. Специальные математические функции системы MuPAD

Система MuPAD не отличается обилием специальных математических функций. Ниже перечислены входящие в ее состав специальные функции:

- **bernulli(n[,expr])** – числа и полиномы Бернулли;
- **besselJ(v,x)** – функции Бесселя первого рода;
- **besselY(v,x)** – функции Бесселя второго рода;
- **beta(x,y)** – бета-функция;
- **binomial(n,k)** – биномиальная функция n!/k!(n-k)!;
- **dilog(x)** – дилогарифм;
- **dirac(expr)** – функция Дирака;
- **eint(x)** – экспоненциальный интеграл;
- **erf(x)** – функция ошибок;
- **erfc(x)** – дополнительная функция ошибок;
- **fact(n)** – факториал n!;
- **gamma(x)** – гамма-функция;
- **heaviside(expr)** – функция Хевисайда;
- **igamma(expr1,expr2)** – обратная гамма-функция;
- **psi(x)** – пси-функция;
- **Si(x)** – интегральный синус;
- **zeta(expr)** – дзета-функция Римана.

Для вычисления большинства из этих функций надо использовать функцию **float(x)**, преобразующую аргумент x в число. Ниже представлены примеры на вычисления специальных функций системы MuPAD:

```
bernoulli(10);
                                5/66
bernoulli(4,x);
                      2      3    4
                     x  - 2·x  + x  - 1/30
float(besselJ(0.5,1));
                          0.6713967071
float(besselY(3,0.5));
                          -42.0594943
beta(10,2);
                             1/110
binomial(10,5);
                              252
```

```
dirac(10.5);
                                    0
eint(1);
                                 eint(1)
float(%);
                              0.2193839343
float(erf(1)),float(erfc(1));
                       0.8427007929, 0.157299207
float(igamma(1,2)),float(psi(1));
                      0.1353352832, -0.5772156649
float(Si(1)),float(zeta(0.1));
                      0.9460830703, -0.6030375198
diff(gamma(x),x);
                             psi(x)·gamma(x)
diff(dirac(x),x);
                            diff(dirac(x), x)
int(gamma(x),x);
                             int(gamma(x), x)
```

Внимание! *Набор специальных математических функций системы MuPAD довольно ограничен. Однако следует отметить, что большинство отсутствующих функций легко задаются с помощью имеющихся в MuPAD средств, хотя это и связано с дополнительными затратами времени пользователя. С некоторыми функциями возможны символьные операции.*

3.7. Расширенные возможности Maple в работе с выражениями

3.7.1. Ввод выражений

Фактически Maple – это система для манипулирования математическими выражениями. *Выражение* в системе Maple – объект, вполне соответствующий сути обычного математического выражения. Оно может содержать операторы, операнды и функции с параметрами.

Выражения в Maple могут оцениваться и изменяться в соответствии с заданными математическими законами и правилами преобразований. Например, функция упрощения выражений `simplify` способна упрощать многие математические выражения, записанные в качестве ее параметра:

```
> simplify(sin(x)^2+cos(x)^2);
                                    1
> simplify((x^2-2*x*a+a^2)/(x-a));
                                  x - a
```

В Maple для ввода выражений и текстовых комментариев служат два соответствующих типа строк ввода. Переключение типа текущей строки ввода осуществляется клавишей **F5**. Строка ввода математических выражений имеет отличительный символ **>**, а строка ввода текстов такого признака не имеет.

В строке ввода могут располагаться несколько выражений. Фиксаторами (указанием, что выражение окончено) их могут быть символы **;** (точка с запятой) и **:** (двоеточие). Символ «**;**» фиксирует выражение и задает вывод результатов его вычисления. А символ «**:**» фиксирует выражение и блокирует вывод результатов его вычисления. Фиксаторы выполняют также функцию разделителей выражений, если в одной строке их несколько.

Ввод выражения оканчивается нажатием клавиши **Enter**. При этом маркер ввода (жирная мигающая вертикальная черта) может быть в любой позиции строки. Если надо перенести ввод на новую строку, следует нажимать клавиши **Shift** и **Enter** совместно. С помощью одного, двух или трех знаков **%** (в реализациях до Maple V R5 это был знак прямых кавычек ") можно вызывать первое, второе или третье выражение с конца сессии, например:

```
> a:b:c:
> %%%;
```

$$a$$

Особая роль при вводе выражений принадлежит знакам прямого апострофа (одиночного ' или двойного ''). Заключенное в такие знаки выражение освобождается от одной пары (закрывающего и открывающего знаков '):

```
> ''factor(a^2+2*a*b^2+b^2)'';
```

$$\text{`factor}(a^2 + 2ab^2 + b^2)\text{'}$$

```
> %;
```

$$\text{factor}(a^2 + 2ab^2 + b^2)$$

```
> factor(a^2+2*a*b+b^2);
```

$$(a + b)^2$$

Некоторые другие возможности обрамления выражений апострофами мы рассмотрим позже. Наиболее важная из них – временная отмена выполненного ранее присваивания переменным конкретных значений.

Для завершения работы с текущим документом достаточно исполнить команду `quit`, `done` или `stop`, набранную в строке ввода (со знаком **;** в конце).

3.7.2. Оценивание выражений

Встречая выражение, Maple *оценивает* его, то есть устанавливает возможность его вычисления. Если выражение – скалярная (неопределенная) переменная, то ее значение будет выведено в ячейке вывода. Для переменных более сложных типов выводится не их значение, а просто повторяется имя переменной. Просто повторяются также имена неопределенных переменных.

Для оценивания выражений различного типа существует группа функций, основные из которых перечислены ниже:

- `eval(array)` – возвращает вычисленное содержимое массива `array`;
- `evalf(expr, n)` – вычисляет `expr` и возвращает вычисленное значение в форме числа с плавающей точкой, имеющего `n` цифр после десятичной точки;
- `evalhf(expr)` – вычисляет `expr` и возвращает вычисленное значение с точностью, присущей оборудованию данного компьютера;
- `evalf(int(f, x=a..b))` – оценивает и возвращает значение определенного интеграла `int(f,x=a..b)`;
- `evalf(Int(f, x=a..b))` – оценивает и возвращает значение определенного интеграла, заданного инертной функцией `Int(f,x=a..b)`;
- `evalf(Int(f, x=a..b, digits, flag))` – аналогично предыдущему, но возвращает значение интеграла с заданным параметром `digits` числом цифр после десятичной точки и со спецификацией метода вычислений `flag`;
- `evalm(mexpr)` – вычисляет значение матричного выражения `mexpr` и возвращает его;
- `evalb(bexpr)` – вычисляет и возвращает значения логических условий;
- `evalc(cexpr)` – вычисляет значение комплексного выражения;
- `evalr(expr, ampl)` – оценивает и возвращает значения интервальных выражений (функция должна вызываться из библиотеки);
- `shake(expr, ampl)` – вычисляет интервальное выражение.

Для функции `evalf` параметр `n` является необязательным, при его отсутствии полагается `n=10`, то есть вещественные числа выводятся с мантиссой, имеющей десять цифр после десятичной запятой.

В выражении `expr` могут использоваться константы, например `Pi`, `exp(1)`, и функции, такие как `exp`, `ln`, `arctan`, `cosh`, `GAMMA` и `erf`. В матричном выражении `mexpr` для функции `evalm` могут использоваться операнды в виде матриц и матричные операторы `&*`, `+`, `–` и `^`. В комплексных выражениях `cexpr` наряду с комплексными операндами вида `(a + I*b)` могут использоваться многие обычные математические функции:

```
Sin       cos       tan       csc       Sec       cot
Sinh      cosh      tanh      csch      Sech      coth
Arcsin    arccos    arctan    arccsc    arcsec    arccot
Arcsinh   arccosh   arctanh   arccsch   arcsech   arccoth
Exp       ln        sqrt      ^         Abs       conjugate
Polar     argument  signum    csgn      Re        Im
Ei        LambertW  dilog     surd
```

Примеры применения функций оценивания даны ниже:

```
> A:=[[1,2],[3,4]];
```

$$A := [[1, 2], [3, 4]]$$

```
> eval(A);
```

$$[[1, 2], [3, 4]]$$

```
> evalf(sin(1));
```

$$.8414709848$$

```
> evalf(sin(2)^2+cos(2)^2,20);
```

$$1.0000000000000000000$$

```
> evalm(20*A+1);
```

$$\begin{bmatrix} 21 & 40 \\ 60 & 81 \end{bmatrix}$$

```
> evalb(1<3);
```

$$true$$

```
> readlib(shake):evalr(min(2,sqrt(3)));
```

$$\sqrt{3}$$

```
> evalr(abs(x));
```

$$\text{INTERVAL}(\text{INTARVAL}(\ , 0\ ..\ \infty), -\text{INTARVAL}(\ , -\infty\ ..\ 0))$$

```
> shake(Pi,3);
```

$$\text{INTERVAL}(3.1102\ ..\ 3.1730)$$

В дальнейшем мы многократно будем применять функции оценивания для демонстрации тех или иных вычислений.

3.7.3. Последовательности выражений

Maple может работать не только с одиночными выражениями, но и с *последовательностями выражений*. Последовательность выражений – это ряд выражений, разделенных запятыми и завершенных фиксатором:

```
> a,y+z,12.3,cos(1.0);
```

$$a, y + z, 12.3, .5403023059$$

Для автоматического формирования последовательности выражений применим специальный оператор `$`, после которого можно указать число выражений или задать диапазон формирования выражений:

```
> f$5;
```

$$f, f, f, f, f$$

```
> $1..5;
```

$$1, 2, 3, 4, 5$$

```
> (n^2)$5;
```

$$n^2, n^2, n^2, n^2, n^2$$

```
> (n^2)$n=0..5;
```

$$0, 1, 4, 9, 16, 25$$

```
> V1[i]$i=1..5;
```

$$V1, V2, V3, V4, V5$$

Для создания последовательностей выражений можно использовать также функцию `seq`:

```
> seq(sin(x),x=0..5);
```

$$0, \sin(1), \sin(2), \sin(3), \sin(4), \sin(5)$$

```
> seq(f1(1.),f1=[sin.cos,tan]);
```

$$.8414709848\ .\ .5403023059, 1.557407725$$

```
> sin(1.0),cos(1.0),tan(1.0);
```

$$.8414709848\ .\ .5403023059, 1.557407725$$

3.7.4. Вывод выражений

При выполнении порой даже простых операций результаты получаются чрезвычайно громоздкими. Для повышения наглядности выражений Maple выводит их с выделением общих частей выражений и с присваиванием им соответствующих меток. Метки представлены символами `%N`, где `N` – номер метки.

Помимо меток, при *выводе результатов* вычислений могут появляться и другие специальные объекты вывода, например корни `RootOf`, члены вида $O(x^n)$, учитывающие погрешность при разложении функций в ряд, и обозначения различных специальных функций, таких как интегральный синус, гамма-функция и др. Примеры такого вывода приведены ниже:

```
> solve(x^7-x^2-1,x);
```

$$\frac{1}{2}+\frac{1}{2}I\sqrt{3},\ \frac{1}{2}-\frac{1}{2}I\sqrt{3},\ \mathrm{RootOf}(\_Z^5+\_Z^4-\_Z^2-\_Z-1,\ index=1),$$
$$\mathrm{RootOf}(\_Z^5+\_Z^4-\_Z^2-\_Z-1,\ index=2),$$
$$\mathrm{RootOf}(\_Z^5+\_Z^4-\_Z^2-\_Z-1,\ index=3),$$
$$\mathrm{RootOf}(\_Z^5+\_Z^4-\_Z^2-\_Z-1,\ index=4),$$
$$\mathrm{RootOf}(\_Z^5+\_Z^4-\_Z^2-\_Z-1,\ index=5)$$

```
> taylor(sin(x),x,5);
```

$$x-\frac{1}{6}x^3+\mathrm{O}(x^5)$$

```
> int(sin(x)/x,x); simplify(N!);
```

$$\mathrm{Si}(x)$$
$$\Gamma(N+1)$$

Часто встречаются также знаки ~ для отметки предполагаемых переменных, постоянные интегрирования и другие специальные обозначения.

3.7.5. Работа с частями выражений

Выражения (`expr`) или уравнения (`eqn`) обычно используются как сами по себе, так и в виде равенств или неравенств. В последнем случае объекты с выражениями имеют левую и правую *части*. Для простейших манипуляций с выражениями полезны следующие функции:

- `cost(a)` – возвращает число сложений и умножений в выражении `a` (функция пакета codegen);
- `lhs(eqn)` – выделяет левую часть `eqn`;
- `rhs(eqn)` – выделяет правую часть `eqn`;
- `normal(expr)` – дает нормализацию (сокращение) `expr` в виде дроби;
- `numer(expr)` – выделяет числитель `expr`;
- `denom(expr)` – выделяет знаменатель `expr`.

Ввиду очевидности действия этих функций ограничимся наглядными примерами их применения:

```
> with(codegen,cost):
> cost(x^3+b^2-x);
```

$$2\ additions + 3\ multiplications$$

```
> lhs(sin(x)^2+cos(x)^2=1);
```

$$\sin(x)^2 + \cos(x)^2$$

```
> rhs(sin(x)^2+cos(x)^2=1);
```

$$1$$

```
> normal(2/4+3/6+6/12);
```

$$\frac{3}{2}$$

```
> f:=5*(a-b)^2/(a^2-2*a*b-b^2);
```

$$f := 5\frac{(a-b)^2}{a^2-2ab-b^2}$$

```
> numer(f);
```

$$5(a-b)^2 +$$

```
> denom(f);
```

$$a^2 - 2ab - b^2$$

Обратите внимание на то, что в старых версиях (до Maple 7) загрузка библиотечной функции `cost` выполнялась иначе – командой `readlib(cost)`. Это обстоятельство может служить причиной неверной работы документов, созданных в старых версиях Maple, в среде последующих версий Maple.

3.7.6. Работа с уровнями вложенности выражений

В общем случае выражения могут быть многоуровневыми и содержать объекты, расположенные на разных *уровнях вложенности*. Приведем две функции для оценки уровней выражений и списков:

- `nops(expr)` – возвращает число объектов первого уровня (операндов) в выражении `expr`;
- `op(expr)` – возвращает список объектов первого уровня в выражении `expr`;
- `op(n,expr)` – возвращает *n*-ый объект первого уровня в выражении `expr`.

Ниже представлены примеры применения этих функций:

```
> nops(a+b/c);
```

$$2$$

```
> op(a+b/c);
```

$$a, \frac{b}{c}$$

```
> op(1,a+b/c);
```

$$a$$

```
> op(2,a+b/c);
```

$$\frac{b}{c}$$

Рекомендуется просмотреть и более сложные примеры на применение этих функций в справке.

3.7.7. Преобразование выражений в тождественные формы

Многие математические выражения имеют различные *тождественные формы*. Основной функцией для такого преобразования является функция `convert`:

```
convert(expr, form, arg3, ...)
```

Здесь `expr` – любое выражение, `form` – наименование формы, `arg3, ...` – необязательные дополнительные аргументы.

`convert` – простая и вместе с тем очень мощная функция. Ее мощь заключается в возможности задания множества параметров. Их полный перечень (76 параметров) можно найти в справке по функции `convert`. Приведем примеры применения функции `convert`:

```
> convert(123,binary);
```

$$1111011$$

```
> convert([a,b,c,d],'+');
```

$$a+b+c+d$$

```
> f:=seq(x[i]^n,i=1..4);
```

$$f := x1^n, x2^n, x3^n, x4^n$$

```
> convert(1.234567,fraction);
```

$$\frac{50737}{41097}$$

```
> convert(1/7,float);
```

$$.1428571429$$

```
> convert(binomial(m,n),factorial);
```

$$\frac{m!}{n!(m-n)!}$$

```
> convert([[1,2],[3,4],[5,6]],table);
```

$$\mathrm{table}([(1,1)=1,(2,1)=3,(2,2)=4,(3,1)=5,(3,2)=6,(1,2)=2])$$

```
> s:=taylor(sin(x),x,8);
```

$$s := x - \frac{1}{6}x^3 + \frac{1}{120}x^5 - \frac{1}{5040}x^7 + \mathrm{O}(x^8)$$

```
> p:=convert(s,polynom);
```

$$p := x - \frac{1}{6}x^3 + \frac{1}{120}x^5 - \frac{1}{5040}x^7$$

```
> convert(p,float);
```

$$x - .1666666667x^3 + .008333333333x^5 - .0001984126984x^7$$

```
> f:=(x^4+x)/(x^2-1);
```

$$f := \frac{x^4 + x}{x^2 - 1}$$

```
> convert(f,parfrac,x);
```

$$x^2 + 1 + \frac{1}{x - 1}$$

```
> s:=series(f,x,5);
```

$$s := -x - x^3 - x^4 + O(x^5)$$

```
> convert(s,polynom);# Удаление члена ряда, описывающего погрешность
```

$$-x - x^3 - x^4$$

Функция преобразования `convert` является одной из самых мощных функций Maple. С ее помощью можно получить множество различных форм выражения.

3.7.8. Преобразование выражений

Еще одним мощным средством преобразования выражений является функция `combine`. Она обеспечивает *объединение показателей степенных функций* и преобразование тригонометрических и некоторых иных функций. Эта функция может записываться в трех формах:

```
combine(f)    combine(f, n)    combine(f, n, opt1, opt2, ...)
```

Здесь `f` – любое выражение, множество или список выражений; `n` – имя, список или множество имен; `opt1`, `opt2`, ... – имена параметров. Во втором аргументе можно использовать следующие функции:

```
@@        abs             arctan       conjugate    exp
ln        piecewise       polylog      power        product
Psi       radical         range        signum       trig
```

Примеры применения функции `combine` представлены ниже:

```
> combine(exp(2*x)^2,exp);
```

$$e^{(4x)}$$

```
> combine(sin(x)*cos(x));
```

$$\frac{1}{2}\sin(2x)$$

```
> combine(Int(x,x=a..b)-Int(x^2,x=a..b));
```

$$\int_a^b -x^2 + x \, dx$$

Эти примеры, естественно, не исчерпывают возможности функции `combine` в преобразовании выражений.

3.7.9. Контроль за типами объектов

В ходе манипуляций с объектами важное значение имеет *контроль за типами* объектов. Одной из основных функций, обеспечивающих такой контроль, является функция `whattype(object)`, возвращающая тип объекта, например `string`,

integer, float, fraction, function и т. д. Могут также возвращаться данные об операторах. Примеры применения этой функции даны ниже:

```
> whattype(2+3);
```

integer

```
> whattype(Pi);
```

symbol

```
> whattype(123./5);
```

float

```
> whattype(1/3);
```

fraction

```
> whattype(sin(x));
```

function

```
> whattype([1,2,3,a,b,c]);
```

list

```
> whattype(a^b);
```

^

```
> whattype(1+2+3=4);
```

=

С помощью функции type(object,t) можно выяснить, относится ли указанный объект к соответствующему типу t, например:

```
> type(2+3,integer);
```

true

```
> type(sin(x),function);
```

true

```
> type(hello,string);
```

false

```
> type("hello",string);
```

true

```
> type(1/3,fraction);
```

true

При успешном соответствии типа объекта указанному (второй параметр) функция type возвращает логическое значение true, в противном случае – false.

Для более детального анализа объектов может использоваться функция hastype(expr, t), где expr – любое выражение и t – наименование типа подобъекта.

Эта функция возвращает логическое значение true, если подобъект указанного типа содержится в выражении expr. Примеры применения этой функции даны ниже:

```
> hastype(2+3,integer);
```

true

```
> hastype(2+3/4,integer);
```

false

```
> hastype(2*sin(x),function);
```

true

```
> hastype(a+b-c/d,'+');
```

true

Еще одна функция – `has(f,x)` – возвращает логическое значение `true`, если подобъект `x` содержится в объекте `f`, и `false` в ином случае:

```
> has(2*sin(x),2);
```

true

```
> has(2*sin(x),'/');
```

false

```
> has(2*sin(x),3-1);
```

true

Следует отметить, что соответствие подобъекта выражения указанному подобъекту понимается в математическом смысле. Так, в последнем примере подобъект «3 – 1», если понимать его буквально, в выражении 2*sin(x) не содержится, но Maple-язык учитывает соответствие 3 – 1 = 2, и потому функция `has` в последнем примере возвращает `true`.

3.8. Работа с подстановками

3.8.1. Функциональные преобразования подвыражений

Нередко бывает необходимо заменить некоторое подвыражение в заданном выражении на функцию от этого подвыражения, то есть осуществить *подстановку*. Средства для обеспечения подстановок есть во всех СКМ. Так, для этого в Maple 9.5 можно воспользоваться функцией `applyop`:

- `applyop(f, i, e)` – применяет функцию `f` к i-му подвыражению выражения `e`;
- `applyop(f, i, e, ..., xk, ...)` – применяет функцию `f` к i-му подвыражению выражения `e` с передачей необязательных дополнительных аргументов `xk`.

Ниже даны примеры применения этой функции:

```
> restart;applyop(sin,2,a+x);
```

$$a + \sin(x)$$

```
> applyop(f,1,g,2,a+b);
```

$$\mathrm{f}(g, 2, a + b)$$

```
> applyop(f,{2,3},a+x+b);
```

$$a + \mathrm{f}(x) + \mathrm{f}(b)$$

```
> applyop(f,{1,2},x/y+z);
```

$$\mathrm{f}\left(\frac{x}{y}\right) + \mathrm{f}(z)$$

```
> p:=y^2-2*y-3;
```

$$p := y^2 - 2y - 3$$

```
> applyop(f,{2,3},p);
```

$$y^2 + \mathrm{f}(-2y) + \mathrm{f}(-3)$$

```
> applyop(abs,{[2,1],3},p);
```

$$y^2 + 2y + 3$$

3.8.2. Функциональные преобразования элементов списков

Еще две функции, реализующие операции подстановки, указаны ниже:

```
map(fcn, expr, arg2, …, argn)
map2(fcn, arg1, expr, arg3, …, argn)
```

Здесь `fcn` – процедура или имя, `expr` – любое выражение, `argi` – необязательные дополнительные аргументы для `fcn`.

Первая из этих функций позволяет приложить `fcn` к операндам выражения `expr`. Приведенные ниже примеры иллюстрируют использование функции `map`:

```
> f:=x->x^2;
```

$$f := x \to x^2$$

```
> map(f,[1,2,3]);
```

$$[1, 4, 9]$$

```
> map(f,[x,y,z]);
```

$$[x^2, y^2, z^2]$$

```
> map(x->x^n,[1,2,3]);
```

$$[1, 2^n, 3^n]$$

```
> L:=[1,2,3,4];
```

$$L := [1, 2, 3, 4]$$

```
> map(proc(x,y) x*y+1 end,[1,2,3,4],2);
```

$$[3, 5, 7, 9]$$

```
> map(int,L,x);
```

$$[x, 2x, 3x, 4x]$$

```
> map(F,[1,2,3],x,y,z);
```

$$[\mathrm{F}(1, x, y, z), \mathrm{F}(2, x, y, z), \mathrm{F}(3, x, y, z)$$

Из этих примеров нетрудно заметить, что если второй параметр функции `map` – список, то функция (первый параметр) прикладывается к каждому элементу списка, так что возвращается также список. Из последнего примера видно, что если за вторым параметром идет перечисление аргументов, то они включаются в список параметров функции.

Функции `map2` отличается иным расположением параметров. Ее действие наглядно поясняют следующие примеры:

```
> map2(w,g,{a,b,c});
```

$$\{\mathrm{w}(g, a), \mathrm{w}(g, b), \mathrm{w}(g, c)\}$$

```
> map2(op,1,[a+b+i,c+d+k,e+f+j]);
```

$$[a, c, e]$$

```
> map2(op,3,[a+b+i,c+d+k,e+f+j]);
```

$$[i, k, j]$$

```
> map2(diff,[sin(x),cos(x),x^n],x);
```

$$\left[\cos(x), -\sin(x), \frac{x^n n}{x}\right]$$

При решении некоторых задач оптимизации возникает необходимость в создании выражений с множителями Лагранжа. Для этого можно использовать список из трех элементов выражения, заключенный в угловые скобки:

```
> e := <x^2, -sqrt(16-x^2), 5>;
```

$$e := x^2\,\mathbf{e}_x - \sqrt{-x^2+16}\,\mathbf{e}_y + 5\,\mathbf{e}_z$$

3.8.3. Подстановки с помощью функций add, mul и seq

Заметим, что операции, подобные описанным выше, Maple реализует и с рядом других функций. Ограничимся примерами на подстановки с помощью функций сложения `add`, умножения `mul` и создания последовательностей `seq`:

```
> add(i^2,i=[a,b,c]);
```

$$a^2 + b^2 + c^2$$

```
> add(i^2,i=[1,2,3]);
```

$$14$$

```
> mul(x-i,i=0..4);
```

$$x(x-1)(x-2)(x-3)(x-4)$$

```
> mul(x^i,i=0..4);
```

$$x^{10}$$

```
> seq(w(x,y,z),i={1,2,3});
```

$$w(x,y,z), w(x,y,z), w(x,y,z)$$

```
> seq(int(x^i,x),i={1,2,3,4});
```

$$\frac{1}{2}x^2, \frac{1}{3}x^3, \frac{1}{4}x^4, \frac{1}{5}x^5$$

3.8.4. Подстановки с помощью функций subs и subsop

Подстановки в общем случае служат для замены одной части выражения на другую. Частными видами подстановок являются такие виды операций, как замена одной переменной на другую или замена символьного значения переменной ее численным значением. Основные операции подстановки выполняют следующие функции:

- `subs(x=a,e)` – в выражении `e` заменяет подвыражение `x` на подвыражение `a`;
- `subs(s1,…,sn,e)` – в выражении `e` заменяет одни подвыражения на другие, выбирая их из списков `s1`, ..., `sn` вида `x=a`;

- `subsop(eq1, eq2, …, eqi, …,eqn, e)` – в выражении `e` заменяет указанные в `eqi` операнды другими, указанными в правой части равенств `eqi` вида `ni=ei`, где `ni` – номер операнда, `ei` – выражение для замены.

Все эти функции возвращают измененное после подстановки выражение. Ниже показаны примеры применения функций подстановок:

```
> subs(a=b,b^2-2*a*b-b^2);
```

$$-2b^2$$

```
> subs(a=2,b=1,b^2-2*a*b-b^2);
```

$$-4$$

```
> subs(c=a-b,(a^2-2*a*b+b^2)/c);
```

$$\frac{a^2-2ab+b^2}{a-b}$$

```
> normal(%);
```

$$a-b$$

```
> subs(a=sin(x),b=cos(x),a^2+b*b);
```

$$\sin(x)^2+\cos(x)^2$$

```
> simplify(%);
```

$$1$$

```
> subsop(1=sin(x),(1+cos(x))/b);
```

$$\frac{\sin(x)}{b}$$

```
> subsop(2=sin(x),(1+cos(x))/b);
```

$$(1+\cos(x))\sin(x)$$

```
> subsop(1=sin(x),2=sin(x),(1+cos(x))/b);
```

$$\sin(x)^2$$

Поэтому полезно контролировать получаемые в результате подстановок выражения на их корректность.

Одним из важных применений подстановок является *проверка правильности решений* уравнений и систем уравнений. Ниже дан пример такой проверки:

```
> eqs:={x+y+z=6,y/x=z-1,z-x=2};
```

$$esq := \{x+y+z=6, z-x=2, \frac{y}{x}=z-1\}$$

```
> res:=solve(eqs,{x,y,z});
```

$$res := \{z=-2, y=12, x=-4\}, \{y=2, z=3, x=1\}$$

```
> subs(res,eqs);
```

$$\{2=2, 6=6, -3=-3\}$$

Решение верно, поскольку у всех уравнений значение левой части совпадает со значением правой части.

3.8.5. Подстановки правил и подвыражений

Для применения некоторого правила или списка правил `rule` к некоторому выражению `expr` используется функция `applyrule(rule, expr)`. Применение этой функции достаточно очевидно:

```
> restart:applyrule(f(a::integer*x)=a*f(x),f(2*x)+g(x)-p*f(x));
```

$$2\,\mathrm{f}(x) + \mathrm{g}(x) - p\,\mathrm{f}(x)$$

```
> applyrule(x^2=y,f(x^2,ln(cos(x)+2*x^2)));
```

$$\mathrm{f}(y, \ln(\cos(x) + 2y))$$

```
> applyrule(b+c=x,f(a+b+c+d));
```

$$\mathrm{f}(x + a + d)$$

Эта функция более мощная, чем subs, но она не выполняет математические вычисления, подобно тому, как это делает функция `algsubs(a = b, f, v, options)` с необязательными двумя последними параметрами. Проанализируйте следующие примеры

```
> algsubs( a^2=0, exp(2-a+a^2/2-a^3/6));
```

$$\mathrm{e}^{(2-\mathrm{a})}$$

```
> applyrule(a^2=0, exp(2-a+a^2/2-a^3/6));
```

$$\mathrm{e}^{(2-a-1/6a^3)}$$

и различие между этими функциями подстановки станет ясным.

3.8.6. Функции сортировки и селекции

Для выполнения *сортировки* служит функция `sort`, применяемая в одной из следующих форм:

`ort(L)` **`sort(L, F)`** **`sort(A)`** **`sort(A, V)`**

Здесь `L` – список сортируемых значений, `F` – необязательная булева процедура с двумя аргументами, `A` – алгебраическое выражение, `V` – необязательные дополнительные переменные.

Примеры применения этих функций:

```
> restart;
> sort(y*x^2+x*y+y-x^2+x^4*y^5);
```

$$x^4y^5 + x^2y - x^2 + xy + y$$

```
> sort((y+z+x)/(y-x-z),{x,y});
```

$$\frac{x+y+z}{-x+y-z}$$

```
> names:=["Peter","Anna","Vladimir","Ivan"];
```

$$names := [\text{"Peter"}, \text{"Anna"}, \text{"Vladimir"}, \text{"Ivan"}]$$

```
> sort(names);
```

$$[\text{"Anna"}, \text{"Ivan"}, \text{"Peter"}, \text{"Vladimir"}]$$

Если функция сортировки меняет порядок расположения членов в выражении (или порядок расположения выражений), то другая функция – `select` – служит для выделения требуемого выражения:

`select(f, e)` **`select(f, e, b1, …, bn)`**

Как бы обратной ей по действию служит функция `remove`, устраняющая заданные выражения:

`remove(f, e)` **`remove(f, e, b1, …, bn)`**

В этих функциях `f` – процедура, возвращающая логическое значение, `e` – список, множество, сумма, произведение или функция, `b1`, ..., `bn` – необязательные дополнительные аргументы. Применение функций очевидно.

Maple имеет также оператор селекции `A[expr]` Выражения A. Его действие поясняют следующие примеры:

```
> restart;
> S:=[a+b*c,x^2,c,1,2,3];
```

$$S := [a + b\,c, x^2, c, 1, 2, 3]$$

```
> S[1];
```

$$a + b\,c$$

```
> S[1..2];
```

$$[a + b\,c, x^2]$$

```
> S[-2..-1];
```

$$[2, 3]$$

```
> S[3..3];
```

$$[c]$$

```
> S[3..2];
```

$$[\ ]$$

3.9. Символьные преобразования выражений

3.9.1. Упрощение выражений – simplify

Функция `simplify` – одна из самых мощных в системах символьной математики. Она предназначена для *упрощения* математических выражений. В системе Maple функция упрощения используется в следующем виде:

- `simplify(expr)` – возвращает упрощенное выражение `expr` или повторяет его, если упрощение в рамках правил Maple невозможно;
- `simplify(expr, n1, n2, ...)` – возвращает упрощенное выражение `expr` с учетом параметров с именами `n1`, `n2`, ... (в том числе заданных списком или множеством);
- `simplify(expr,assume=prop)` – возвращает упрощенное выражение `expr` с учетом всех условий, представленных равенством или списком равенств.

Функция `simplify` – многоцелевая. Она обеспечивает упрощение математических выражений, выполняя следующие типовые действия (для простоты обозначим их как преобразрвание `->`):

- комбинируя цифровые подвыражения (`3*x*5->15*x, 10*x/5->2*x`);
- приводя подобные множители в произведениях (`x^3*a*x->a*x^4`);
- приводя подобные члены в суммах (`5*x+2+3*x->8*x+2`);
- используя тождества, содержащие нуль (`a+0->a, x-0->x`);
- используя тождества, содержащие единицу (`1*x->x`);

- распределяя целочисленные показатели степени в произведениях (`(3*x*y^3)^2->9*x^2*y^6`);
- сокращая `expr` на наибольший общий полиномиальный или иной множитель;
- понижая степень полиномов там, где это возможно;
- используя преобразования, способные упростить выражения.

Иногда функция `simplify` неспособна выполнить возможные упрощения. В этом случае ей надо подсказать, в какой области ищутся упрощения и где можно найти соответствующие упрощающие преобразования. С этой целью в функцию `simplify` можно включать дополнительные параметры. В качестве параметров могут задаваться имена специальных математических функций и указания на область действия упрощений: `BesselI`, `BesselJ`, `BesselK`, `BesselY`, `Ei`, `GAMMA`, `RootOf`, `LambertW`, `dilog`, `exp`, `ln`, `sqrt`, `polylog`, `pg`, `pochhammer`, `trig` (для всех тригонометрических функций), `hypergeom`, `radical`, `power` и `atsign` (для операторов).

Полезен также параметр `symbolic`, задающий формальные символьные преобразования для многозначных функций, например таких, как квадратный корень:

```
> g:=sqrt(x^2);
```

$$g := \sqrt{x^2}$$

```
> simplify(g);
```

$$\mathrm{csgn}(x)x$$

```
> simplify(g,assume=real);
```

$$|x|$$

```
> simplify(g,assume=positive);
```

$$x$$

```
> simplify(g,symbolic);
```

$$x$$

Но чуть иначе:

```
> g:=sqrt(x^y);
```

$$g := \sqrt{x^y}$$

```
> simplify(g);
```

$$\sqrt{x^y}$$

```
> simplify(g,assume=real);
```

$$\sqrt{x^y}$$

```
> simplify(g,assume=positive);
```

$$x^{\left(\frac{y}{2}\right)}$$

```
> simplify(g,symbolic);
```

$$x^{\left(\frac{y}{2}\right)}$$

Возможно также применение функции `simplify` в форме `simplify [<name>]`, где `<name>` – одно из следующих указаний: `atsign`, `GAMMA`, `hypergeom`, `power`, `radical`, `RootOf`, `sqrt`, `trig`.

Ниже даны примеры применения функции `simplify`:

```
> simplify(4^(1/2)+3);
```

$$5$$

```
> simplify((x^y)^z+3^(3),power);
```

$$(x^y)^z + 27$$

```
> simplify(sin(x)^2+cos(x)^2,trig);
```

$$1$$

```
> e:=cos(x)^5+sin(x)^4+2*cos(x)^2-2*sin(x)^2-cos(2*x);;
```

$$e := \cos(x)^5 + \sin(x)^4 + 2\cos(x)^2 - 2\sin(x)^2 - \cos(2x)$$

```
> simplify(e);
```

$$\cos(x)^5 + \cos(x)^4$$

```
> simplify(GAMMA(n+4)/GAMMA(n),GAMMA);
```

$$n(n+1)(n+2)(n+3)$$

```
> r:=RootOf(x^2-2=0,x):
> simplify(r^2,RootOf);
```

$$2$$

```
> simplify(ln(x*y),power,symbolic);
```

$$\ln(x) + \ln(y)$$

```
> e:=(-5*b^2*a)^(1/2);
```

$$e := \sqrt{-5b^2a}$$

```
> simplify(e,radical,symbolic);
```

$$b\sqrt{5}\sqrt{-a}$$

Действие функции `simplify` существенно зависит от областей определения переменных. В следующем примере упрощение выражения не произошло, поскольку результат этой операции неоднозначен:

```
> restart;
> simplify(sqrt(x^4*y^2));
```

$$\sqrt{x^4y^2}$$

Однако, определив переменные как реальные или положительные, можно легко добиться желаемого упрощения:

```
> simplify(sqrt(x^4*y^2),assume=positive);
```

$$x^2y$$

```
> simplify(sqrt(x^4*y^2),assume=real);
```

$$x^2|y|$$

С помощью равенств можно задать свои правила преобразования, например:

```
> eq:=x^2+2*x*y+y^2;
```

$$eq := x^2 + 2xy + y^2$$

```
> simplify(eq,{x=1});
```

$$y^2 + 2y + 1$$

```
> simplify(eq,{x^2=x*y,y^2=1});
```

$$3xy + 1$$

```
> simplify(eq,{x,y});
```

$$0$$

Обратите внимание на то, что указание в списке равенств только левой части равенства означает, что правая часть принимается равной нулю. Если функция `simplify` не способна выполнить упрощение выражения `expr`, то она просто его повторяет. Это сигнал к применению опций, уточняющих преобразования.

3.9.2. Расширение выражений – expand

Для *расширения* или *раскрытия* выражения используется функция

`expand(expr, expr1, expr2, …, exprn)`

где `expr` – расширяемое выражение, `expr1`, `expr2`, ..., `exprn` – необязательные подвыражения – опции. Имеется также инертная форма данной функции – `Expand(expr)`. Кроме того, возможно применение операторной конструкции `frontend(expans,[expr])`.

Функция `expand` раскладывает рациональные выражения на простые дроби, полиномы на полиномиальные разложения, она способна раскрыть многие математические функции, как элементарные, так и специальные. С помощью дополнительных аргументов `expr1`, `expr2`, ..., `exprn` можно задать расширение отдельных фрагментов в `expr`. Примеры применения функции `expand` приведены ниже:

```
> expand((x+2)*(x+3)*(x+4));
```

$$x^3 + 9x^2 + 26x + 24$$

```
> expand(sin(2*x));
```

$$2\sin(x)\cos(x)$$

```
> expand(sin(x+y));
```

$$\sin(x)\cos(y) + \cos(x)\sin(y)$$

```
> expand([(a+b)*(a-b),tan(2*x)]);
```

$$\left[a^2 - b^2, 2\frac{\tan(x)}{1-\tan(x)^2}\right]$$

```
> expand((a+d)*(b+d)*(c+d));
```

$$abc + abd + adc + ad^2 + dbc + d^2b + d^2c + d^3$$

```
> expand((x+1)*(y+z));
```

$$xy + xz + y + z$$

```
> expand((x+1)*(y+z), x+1);
```

$$(x+1)y + (x+1)z$$

```
> frontend(expand,[(a+b)^3]);
```

$$a^3 + 3a^2b + 3ab^2 + b^3$$

3.9.3. Разложение целых и рациональных чисел – ifactor

Для *разложения* целых или рациональных чисел на множители в виде простых чисел служит функция

`ifactor(n)` или **`ifactor(n,method)`**,

где n – число, method – параметр, задающий метод разложения. Другая библиотечная функция, ifactors(n), возвращает результат разложения в форме вложенных списков:

```
> ifactor(123456789);
```

$$(3)^2\ (3803)\ (3607)$$

```
> ifactor(12!/20!);
```

$$\frac{1}{(2)^8\,(3)^3\,(5)^2\,(7)\,(13)\,(17)\,(19)}$$

```
> ifactor(100/78);
```

$$\frac{(2)\,(5)^2}{(3)\,(13)}$$

```
> readlib(ifactors):
> ifactors(100/78);
```

$$[1, [[2, 1], [5, 2], [3, -1], [13, -1]]]$$

3.9.4. Разложение выражений (факторизация) – factor

Для алгебраических выражений функция *факторизации* записывается в вычисляемой и невычисляемой (инертной) формах:

```
factor(a)          Factor(a)
factor(a,K)        Factor(a,K)
```

Здесь a – полином с несколькими переменными, K – необязательное алгебраическое расширение. Для получения результата от инертной формы функции факторизации надо использовать функцию вычисления evala или evalgf.

Главная цель факторизации – это нахождение максимального числа независимых сомножителей выражения, линейных по заданным переменным с коэффициентами наиболее простой формы.

Ниже представлены примеры применения функции factor:

```
> factor(a^2+2*a*b+b^2);
```

$$(a+b)^2$$

```
> factor(a^2-2*a*b-b^2);
```

$$a^2-2ab-b^2$$

```
> p:=expand((x-1)*(x-2)*(x-3)*(x-4));
```

$$p := x^4-10x^3+35x^2-50x+24$$

```
> factor(p);
```

$$(x-1)(x-2)(x-3)(x-4)$$

```
> factor(x^5-2,2^(1/5));
```

$$(x-2^{(1/5)})(x^4+x^3 2^{(1/5)}+x^2 2^{(2/5)}+x 2^{(3/5)}+2^{(4/5)})$$

```
> alias(alpha=RootOf(x^2-2));
```

$$\alpha$$

```
> factor(x^2-2,alpha);
```

$$(x+\alpha)(x-\alpha)$$

```
> factor(x^3-y^3);
```

$$(x-y)(x^2+xy+y^2)$$

```
> factor(x^3-y^3,(-2)^(1/2));
```

$$(x-y)(x^2+xy+y^2)$$

```
> factor(x^3-y^3,(-3)^(1/2));
```

$$\frac{1}{4}(2x+y-y\sqrt{-3})(2x+y+y\sqrt{-3})(x-y)$$

```
> factor(x^3-3,complex);
```

$$(x+.7211247852+1.249024766\,I)(x+.7211247852-1.249024766\,I)$$
$$(x-1.442249570)$$

3.9.5. Комплектование по степеням – collect

Еще одна функция общего назначения – `collect` – служит для *комплектования* выражения `expr` по степеням указанного фрагмента `x` (в том числе множества либо списка). Она задается в одной из следующих форм:

```
collect(a, x)    collect(a, x, form, func)
```

Во второй форме этой функции дополнительно задаются параметры `form` (форма) и `func` (функция или процедура). Параметр `form` может иметь два значения: `recursive` (рекурсивная форма) и `distributed` (дистрибутивная форма). Параметр `func` позволяет задать имя функции, по которой будет идти комплектование `expr`.

Примеры применения функции `collect` представлены ниже:

```
> collect(x+x^3-2*x,x);
```

$$-x+x^3$$

```
> collect(x+2*y^3+x+3+x^3*y,recursive,x);
```

$$\mathrm{x}(2x+2y^3+3+x^3y)$$

```
> collect(x+2*y^3+x+3+x^3*y,distributive,y);
```

$$\mathrm{y}(2x+2y^3+3+x^3y)$$

```
> f:=a*exp(x)-exp(x)*x-x;
```

$$f:=a\mathrm{e}^x-\mathrm{e}^x x-x$$

```
> collect(f,exp(x));
```

$$(a-x)\mathrm{e}^x-x$$

```
> p:=x*y+a*x*y+y*x^2-a*y*x^2+x+a*x;
```

$$p:=xy+axy+yx^2-ayx^2+x+ax$$

```
> collect(p,[x,y],recursive);
```

$$(1-a)yx^2+((1+a)y+1+a)x$$

```
> collect(p,[x,y],distributed);
```

$$(1+a)x+(1+a)xy+(1-a)yx^2$$

3.9.6. Работа с пакетом рациональных нормальных форм RationalNormalForms

В Maple 7/8 вошел пакет рациональных нормальных форм RationalNormalForms:

```
> with(RationalNormalForms);
```

[*AreSimilar*, *IsHypergeometricTerm*, *MinimalRepresentation*, *PolinomialNormalForm*, *RationalCanonicalForm*]

Этот пакет обеспечивает следующие возможности:

- конструирование полиномиальных нормальных форм рациональных функций;
- конструирование рациональных канонических форм для рациональных функций;
- конструирование минимальных представлений для гипергеометрических термов.

Ввиду очевидности названий функций этого пакета ограничимся примерами его применения:

```
> F := (n^2-2)*(3*n+3)!/((n+3)!*(2*n+5)!);
```

$$F := \frac{(n^2-2)(3n+3)!}{(n+3)!\,(2n+5)!}$$

```
> IsHypergeometricTerm(F,n,'certificate');
```

true

```
> certificate;
```

$$\frac{3(n^2+2b-1)(n+2)(3n+5)(3n+4)}{2(n+4)(2n+7)(n+3)(n^2-2)}$$

```
> (z,r,s,u,v) := RationalCanonicalForm[1](certificate,n);
```

$$z, r, s, u, v := \frac{27}{4}, \left(n+\frac{5}{3}\right)\left(n+\frac{4}{3}\right), \left(n+\frac{7}{2}\right)(n+4), n^2-2, n+2$$

```
> MinimalRepresentation[1](F,n,k);
```

$$\frac{1}{60}\,\frac{\left(\frac{27}{4}\right)^n (n^2-2)\left(\prod_{k=0}^{n-1}\frac{\left(k+\frac{5}{3}\right)\left(k+\frac{4}{3}\right)}{\left(k+\frac{7}{2}\right)(k+4)}\right)}{n+2}$$

3.10. Работа с выражениями в системе Mathematica

3.10.1. Полная форма выражений

В СКМ Mathematica выражения expr часто представляются в *полной форме*, при которой основные арифметические операции задаются не операторами, а только соответствующими функциями. Выражениями, представленными в полной форме, оперирует символьное ядро системы.

Приведем примеры полной формы основных математических выражений:

Выражение expr	Полная форма expr	Комментарий
x+y+z	**Plus[x,y,z]**	Сложение
x y z	**Times[x,y,z]**	Умножение
x^n	**Power[x,n]**	Возведение в степень
{a,b,c}	**List[a,b,c]**	Создание списка
a->b	**Rule[a,b]**	Подстановка
a=b	**Set[a,b]**	Установка

Для вывода выражения expr в полной форме используется функция

```
FullForm[expr]
```

Примеры перевода выражений в его полную форму:

```
1 + x^2 + (y+z)^2 + 2
```

$$3 + x^2 + (y + z)^2$$

```
FullForm[%]
 Plus[3,Power[x,2],Power[Plus[y,z],2]]
Integrate[a*Sin[b*x]*Exp[-c*x],x]
```

$$-\frac{a\,E^{-cx}\,(b\,\text{Cos}[bx] + c\,\text{Sin}[bx])}{(-Ib+c)\,(Ib+c)}$$

```
FullForm[%]
 Times[-1,a,Power[Plus[Times[Complex[0,-1],b],c], -
1],Power[Plus[Times[Complex[0,1],b],c],-1],Power[E,Times[-
1,c,x]],Plus[Times[b,Cos[Times[b,x]]],Times[c,Sin[Times[b,x]] ]]]
```

Для определения типа выражения служит функция

```
Head[expr]
```

Применительно к числовым выражениям она возвращает тип результата, например:

```
1 + 2 + 3
 6
Head[%]
 Integer
Head[123/12345]
 Rational
Head[2*0.25]
 Real
```

Следующие примеры поясняют действие функции **Head** при символьных выражениях:

Head[f[x,y,z]	возвращает	f
Head[a+b+c]	возвращает	Plus
Head[x^n]	возвращает	Power
Head[{a,b,c}]	возвращает	List

и т. д. Другая пара примеров показывает применение Head в списках с разнородными выражениями:

```
{Head[1+2],Head[a*b],Head[5/7],Head[1+3*I],Head[Exp[2]]}
 {Integer,Times,Rational,Complex,Power}
{Integer,Rational,Real,Complex,Symbol,f,List,Plus,Times}
```

Обратите внимание на второй пример – при нем функция Head используется к каждому выражению списка, что дает более компактную запись.

3.10.2. Основные формы выражений

Возможны четыре основные формы записи выражений:

f[x,y]	Стандартная форма для f[x,y].
f @ x	Префиксная форма для f[x].
x // f	Постфиксная форма для f[x].
x ~ f ~ y	Инфиксная форма для f[x,y].

Примеры на применение этих форм (ввод – слева, вывод – справа):

F[x_]=2*x^2	$2x^2$
F[a]	$2a^2$
a//F	$2a^2$
f[x_,y_]=x^2+y^2	$a^2 + x^2$
f[a,b]	$2a^2$
a~f~b	$2a^2$

Можно использовать ту или иную форму выражений в зависимости от класса решаемых математических задач.

3.10.3. Части выражений и работа с ними

Сложные выражения состоят из **частей**, которые в Mathematica могут интерпретироваться различным образом:

Тип части	**Зависимость**	**Пример**
Function	От аргументов или параметров	**Exp[x], f[x,y]**
Command	От аргументов или параметров	**Expand[(x-1)^2]**
Operator	От операндов	**x+y+z, a=b**
Head	От элементов	**{a,b,c}**
Object type	От контекста	**RGBColor[r,g,b]**

Работа с частями выражений напоминает работу со списками. Для выделения любой заданной части выражений используются функция **Part** или двойные квадратные скобки:

Part[expr,n] или **expr[[n]]**	Выделяет n-ую часть выражения с начала.
expr[[-n]]	Выделяет n-ую часть выражения с конца.
expr[[n1,n2,...]]	Выделяет части выражения и показывает их в форме дерева.
expr[[{n1,n2,...}]]	Дает комбинацию нескольких частей выражения.

Приведем примеры на использование этих средств:

```
f:=a+b*x^2+c*x^3
Part[f,2]                    bx^2
Part[f,3]                    cx^3
f[[1]]                        a
f[[3]]                       cx^3
f[[-1]]                      cx^3
```

Нередко выражения рассматриваются как возможные значения переменных. В этом случае используются операторы *присваивания* переменным заданных значений. Mathematica имеет два типа присваивания – с помощью символов **:=** и **=**. Они отличаются временем исполнения выражения, следующего за этими символами. Знак **:=** используется для *задержки присваивания* до вычисления правой части, например:

```
f[x_] := % + 2 x
```

Вывода здесь нет. Продолжим наш эксперимент:

```
1 + y^2
 1 + y^2
g[x_] = % + 2 x
 1 + 2x + y^2
```

Теперь вывод есть, так как % определено в виде выражения 1 + y^2 и при задании g[x_] использован оператор присваивания немедленного исполнения. Далее:

```
2+z
 2+z
{f[a],g[a]}
 {2 + 2a + z, 1 + 2a + y^2}
```

Следующие функции возвращают особые части выражения:

- **Denominator[expr]** – возвращает знаменатель выражения expr.
- **Nominator[expr]** – возвращает числитель выражения expr.
- **First[expr]** – возвращает первый элемент в expr.
- **Last[expr]** – возвращает последний элемент в expr.
- **Rest[expr]** – возвращает expr с удаленным первым элементом.

Примеры применения этих функций:

```
Denominator[(x+1)/(x^2+2*x+3)]      3 + 2x + x^2
expr=a*b+c-d                        a b+c-d
First[expr]                         a b
Last[expr]                          -d
Rest[expr]                          c-d
```

Работа с выражениями, умение их преобразовывать и выделять нужные их фрагменты является важнейшей частью культуры символьных преобразований.

3.10.4. Удаление элементов выражения

Иногда возникает необходимость в удалении части выражения. Для этого используются следующие функции:

- **Delete[expr, n]** – удаляет элемент на позиции n в выражении expr. Если n отрицательно, позиция отсчитывается с конца.
- **Delete[expr, {i, j, ...}]** – стирает часть выражения на позиции {i,j, ...}.
- **Delete[expr, {{i1, j1, ...}, {i2, j2, ...}, ...}]** – удаляет части выражения в нескольких указанных позициях.
- **DeleteCases[expr, pattern]** – исключает все элементы выражения expr, которые совпадают с образцом pattern.
- **DeleteCases[expr, pattern, levspec]** – исключает все части выражения expr на уровнях, указанных levspec, и соответствующих образцов pattern.

Следующие примеры иллюстрируют применение этих функций:

```
expr=a*b+c-d                a b+c-d
Delete[expr,1]              c-d
Delete[expr,3]              a b+c
Delete[expr,{{1},{3}}]      c
DeleteCases[expr,a*b]       c-d
DeleteCases[expr,c,1]       a b-d
```

Обратите внимание на то, что в общем случае выражения могут быть многоуровневыми. Уровень задается спецификацией **levspec**.

3.10.5. Другие манипуляции с выражениями

В процессе преобразований выражений с ними возможны и иные манипуляции. Наиболее важные из них выполняются следующими функциями:

- **Append[expr, elem]** – возвращает expr с дополнением elem.
- **AppendTo[s, elem]** – добавляет elem к значению s и переустанавливает s в новое значение.
- **Apply[f, expr, levelspec]** – возвращает expr, замещая заголовки в тех частях expr, которые указаны спецификацией уровня levelspec.
- **Cancel[expr]** – возвращает expr с сокращением общих множителей числителя и знаменателя.
- **Cases[expr, pattern, levelspec]** – возвращает список всех частей выражения expr на уровнях, указанных спецификацией levelspec и соответствующих шаблону pattern.
- **Chop[expr]** – в выражении expr задает нулевой малую мнимую часть.
- **Chop[expr,tol]** – присваивает значение 0 приближенным вещественным числам в выражении expr, абсолютные величины которых меньше tol.
- **Replace[expr, rules]** – возвращает expr с подстановкой, заданной правилом или списком правил rules.

- **ReplaceAll** – используется в виде: **expr /. rules** – возвращает expr с подстановками, заданными правилом или списком правил.
- **ReplaceHeldPart[expr, Hold[new], n]** – возвращает выражение, в котором n-ая часть expr заменена на new.
- **ReplaceHeldPart[expr, Hold[new], {i, j, ...}]** – заменяет часть на позиции {i, j, ...}.
- **ReplaceHeldPart[expr, Hold[new], {{i1,j1, ...}, {i2,j2, ...}, ...}]** – замещает части в нескольких позициях на new.
- **ReplacePart[expr, new, n]** – возвращает выражение, в котором n-ая часть expr замещена на new.
- **ReplacePart[expr, new, {i, j, ...}]** – замещает часть на позиции {i, j, ...}.
- **ReplacePart[expr, new, {{i1, j1, ...}, {i2, j2,...}, ...}]** – замещает части в нескольких позициях на new.
- **ReplaceRepeated** – применяется в виде: **expr //. rules** – неоднократно выполняет замещения до тех пор, пока expr не перестанет изменяться.

Действие некоторых этих функций достаточно очевидно и поясняется следующими примерами:

```
Append[a+c,b]                                  a+b+c
x={a,b,c}                                      {a,b,c}
AppendTo[x,15]                                 {a,b,c,15}
x                                              {a,b,c,15}
Apply[f,a^2+b^2,2]                             f[a,2]+f[b,2]
Cancel[(z-1)^2/(z-1)]                          -1+z
Cases[{a,3.5,2,5,"HELLO"},_Integer]            {2,5}
Exp[N[- π I]]                                  -1.-1.22461×10-16ⅈ
Chop[%]                                        -1.
Exp[N[- π I]]                                  -1.-1.22461×10-16ⅈ
Exp[N[- π I]]                                  -1.-1.22461×10-16ⅈ
Chop[%,1*10^-10]                               -1.
Replace[s^2,s^2->a]                            a
s^2/.s->a                                      a²
```

3.10.6. Контроль выражений

При создании программного обеспечения на языке Mathematica, а иногда и в ходе диалоговой работы с системой необходим контроль за некоторыми **свойствами выражений**. Следующие функции обеспечивают такой контроль:

- **AtomQ[expr]** – возвращает True, если выражение expr не может быть разложено на подвыражения и является атомарным, и возвращает False в противном случае.
- **ByteCount[expr]** – возвращает количество байтов, используемых системой Mathematica для внутреннего хранения выражения expr.
- **FreeQ[expr, form]** – вырабатывает значение True, если в выражении expr отсутствует подвыражение, совпадающее с form, в противном случае дает False.

- **FreeQ[expr, form, levelspec]** – тестирует только части выражения на уровнях, указанных levelspec.

Следующие примеры показывают действие этих функций:

```
AtomQ[{a}]              False
AtomQ[2+3/4]            True
ByteCount[1+3/4]        64
ByteCount[1.+3./4]      16
FreeQ[a*x^b,a]          False
FreeQ[a*x^b+c,b,2]      True
```

3.10.7. Приложение имени функции к выражению или его части

Функции в системе Mathematica характеризуются именем (обобщенно f) и выражением expr, задающим функциональную зависимость. Следующие функции позволяют прикладывать имя функции к выражению или к частям выражения:

- **Apply[f, expr]** – замещает заголовок выражения expr на f.
- **Nest[f, expr, n]** – возвращает выражение, полученное n-кратным приложением f к expr.
- **Map[f, expr]** – прикладывает f к каждому элементу на первом уровне в expr.
- **Map[f, expr, levelspec]** – применяет f к частям expr, указанным с помощью levelspec.
- **MapAll[f,expr]** – прикладывает f ко всем частям выражения expr.

Приведем примеры на действие этих функций:

```
Apply[f,{a,b,x}]            f[a,b,x]
Nest[f,x,3]                 f[f[f[x]]]
s[x_,y_,z_]:=x+y+b
N[Apply[s,{1,2,a}]]         3. +b
Map[f,{a,b,c}]              {f[a],f[b],f[c]}
MapAll[f,a*x+b]             f[f[b]+f[f[a] f[x]]]
MapAll[f,{a,b,c}]           f[{f[a],f[b],f[c]}]
```

3.10.8. Укороченная форма функций

Из описания указанных функций вытекает, что они наряду с полной формой могут задаваться *укороченной формой*:

```
f @ expr            f[expr]
f @@ expr           Apply[f, expr]
f /@ expr           Map[f, expr]
f //@ expr          MapAll[f, expr]
```

Смысл укороченных выражений очевиден:

```
f@{a,b,c}           f[{a,b,c}]
f@@{a,b,c}          f[a,b,c]
```

```
f/@{a,b,c}          {f[a],f[b],f[c]}
f//@{a,b,x}         f[{f[a],f[b],f[x]}]
```

Укороченная форма функций может оказаться полезной для уменьшения записи алгоритмов и программ.

3.10.9. Выделение заданного аргумента в функциях

Функция **Slot[n]**, или в укороченной форме #n, представляет n-ый аргумент функции. Это иллюстрируют следующие примеры:

```
(5*Slot[1]+Slot[2]*Slot[3]^2)&[a,b,c]        5a + bc^2
 #1^#2&[a,b]                                 a^b
```

Объект # эквивалентен #1, а #0 – заголовку абстрактной функции. Так что F[#.#2]&F[a,b] эквивалентно F[a,b].

Функция **SlotSequence[n]**, или в укороченной форме ##n, где n = 1, 2, ..., представляет порядок применения формальных аргументов к абстрактной функции. Так что объект ##n определяет последовательность аргументов, начиная с n-го:

```
(Times[5,##2]+Times[##2,##3^2])&[a,b,c]      5bc + bc^3
```

Представленные средства обеспечивают работу с функциями на абстрактном уровне.

3.10.10. Подстановки в функциях

Интересные возможности связаны с использованием **подстановок** при определении функций. Система допускает использование таких подстановок:

f[x] = value и **f[x_] = value**

Поясним это примерами:

```
f[x]=u              u
f[x]+f[y]           u+f[y]
f[x_]=x^2           x^2
f[x]+f[y]           u + y^2
Clear[f]
f[x]+f[y]           f[x]+f[y]
```

Как нетрудно заметить из этих примеров, подстановки в функциях могут существенно изменить исходную функциональную зависимость. А потому важной областью их применения является модификация функций.

3.10.11. Рекурсивные функции

Использование подстановок при определении функции позволяет легко реализовать *рекурсивные* алгоритмы, то есть алгоритмы, при которых очередной шаг вычислений основан на определенном преобразовании предшествующих шагов.

Примером может служить задание функции вычисления факториала fact[n], представленное ниже:

Операция	Комментарий
`fact[n_] := n*fact[n-1]`	Задана рекурсивная функция факториала
`fact[1] := 1`	Выполнена инициализация функции
`fact[3]`	Вычислено значение 3!
6	
`fact[10]`	Вычислено значение 10!
3628800	

При необходимости использование знака вопроса перед именем функции позволяет вывести текст декларации (определения) функции.

3.10.12. Дополнительные примеры на работу с функциями

Приведем еще ряд примеров на действие функций **Apply**, **Map** и **Nest**:

```
s[x_,y_,z_]:=x+y+b
N[Apply[s,{1,2,a}]]          3. + b
Map[f,{a,b,c}]               {f[a], f[b], f[c]}
N[Map[Exp,{1,2,3}]]          {2.71828, 7.38906, 20.0855}
```

Большинство из описанных операций для работы с функциями могут использоваться и для операций со списками. Порой это резко упрощает запись алгоритмов вычислений для данных, представленных списками, поскольку дает общее определение функций для произвольного числа их параметров. Примерами могут служить определения следующих статистических функций (см. примеры ниже).

Вычисление среднего для элементов списка:

```
Meanl[list_] := Apply[Plus, list] / Length[list]  /; VectorQ[list]
&& Length[list] > 0
```

Вычисление среднего геометрического для списка:

```
GeometricMean[list_] := Apply[Times, list^(1/Length[list])]  /;
VectorQ[list] && Length[list] > 0
```

Вычисление гармонического среднего для списка:

```
HarmonicMean[list_] := Length[list] / Apply[Plus, 1/list] /;
VectorQ[list] && Length[list] > 0
```

Обратите внимание, что при задании первой функции Mathematica предупреждает о том, что введенный идентификатор list опасно напоминает зарезервированный идентификатор **List**. Все приведенные выше функции не имеют смысла, если список пустой. Поэтому в них введен контроль за такой ситуацией.

Теперь можно выполнить расчеты по этим формулам:

```
data={1,2,3,4}               {1,2,3,4}
Mean[data]                   5/2
```

GeometricMean[data] $2^{3/4}3^{1/4}$

HarmonicMean[data] $\frac{48}{25}$

3.10.13. Инверсные функции

Инверсными функциями называют функции, полученные в результате обращения заданных функций. Например, для функции Sin[x] инверсной будет ArcSin[x] и т. д. Следующие функции обеспечивают представление инверсных функций:

- **InverseFunction[f]** – представляет функцию, обратную f, определенную таким образом, что **InverseFunction[f][y]** – возвращает значение x, для которого f[x] равно y. Для функции нескольких переменных **InverseFunction[f]** представляет обращение по первому аргументу.
- **InverseFunction[f, n]** – представляет обращение по n-му аргументу.
- **InverseFunction[f, n, tot]** – представляет обращение по n-му аргументу, когда имеется всего tot аргументов.

Следующие примеры иллюстрируют работу с этими функциями:

```
InverseFunction[Sin]
ArcSin
%[x]
ArcSin[x]
Composition[f, g, h]
Composition[f,g,h]
InverseFunction[Composition[%, q]]
 Composition[q^(-1), h^(-1), g^(-1), f^(-1)]
```

Обратите внимание на то, что в этих примерах фигурируют заголовки функций, например, для получения инверсной функции от Sin[x] следует использовать в качестве аргумента функции **InverseFunction[f]** вместо f соответственно Sin.

3.10.14. Задание математических отношений

В математике можно найти множество примеров *математических отношений*. Например, хорошо известно такое отношение для логарифма и экспоненциальной функции:

$$\log(\exp(x)) = x$$

Не обременяя себя поиском действительно новых закономерностей (порой на это может не хватить жизни, да и везет не каждому ученому), зададим приведенную закономерность для введенных по-новому функций log и exp. Центральным моментом тут является введение новых имен функций, которые начинаются с малых букв, а не с больших, как у встроенных функций Log и Exp. Поэтому система воспринимает log и exp как новые функции.

Итак, «новую» закономерность вводим следующим образом:

```
log[exp[x_]]:=x
```

```
General::spell1 :
  Possible spelling error : new symbol name "log" is
    similar to existing symbol "Log". More...

  General::spell1 :
    Possible spelling error : new symbol name "exp" is
      similar to existing symbol "Exp". More...
```

Система на всякий случай сообщает о рискованности эксперимента – символы log и exp похожи на зарезервированные имена функций **Log** и **Exp**. Проигнорировав это предупреждение, проверим данную закономерность в работе:

```
log[exp[15]]
15
log[exp[y^2+1]]
```

$$1 + x^2$$

Итак, наша «новая» закономерность работает. Можно ввести, скажем, и такое известное отношение:

```
log[x_^n_]:=n*log[x]
```

Проверим, какие отношения нами заданы для функции log:

```
?log
 Global`log
 log[exp[x_]] := x
 log[x_^n_] := nlog[x]
```

Проверим введенные правила, например:

```
log[(1+x)^5]
 5 log[1+x]
```

Рассмотрим еще пару примеров на задание «новых» математических правил. В первом примере задано правило: логарифм произведения равен сумме логарифмов сомножителей:

```
log[x_*y_]:=log[x]+log[y]
```

Любопытно, что эта закономерность действует при любом числе сомножителей:

```
log[a*b*c*d*e]
 log[a]+log[b]+log[c]+log[d]+log[e]
```

Второй пример иллюстрирует задание объекта, ассоциированного со списком.

```
a/:a[x_]+a[y_]:=a[x+y]
a[x]+a[y]+a[z]
 a[x+y+z]
```

Введенные здесь обозначения x_, y_ и n_ означают *образцы*, на место которых могут подставляться произвольные выражения.

3.10.15. Упрощение выражений и функция Simplify

Mathematica, как и Maple, имеет функцию упрощения выражений:

Simplify[expr] – исполняет последовательность алгебраических преобразований над выражением expr и возвращает простейшую из найденных форм (обычно это бывает нормальная форма выражений).

Приведем нескольких примеров ее применения:

- комбинирование числовых подвыражений, например:
  Simplify[6 x 2] возвращает 12 x,
- приведение подобных множителей у произведений, например:
  Simplify[x^3 y x^5] возвращает x^8 y,
- приведение подобных членов суммы, например:
  Simplify[x + 12 + 4 x] возвращает 5 x + 12,
- упрощение тождеств, содержащих 0 или 1, например:
  Simplify[2+0] возвращает 2
  Simplify[1*x] возвращает x
- распределение целочисленных показателей степени в произведениях, например:
  Simplify[(5 x^2 y^3)^2] возвращает $5\,x^5\,y$,
- сокращение на наибольший полиномиальный делитель, например:
  Simplify[(x^2 – 2 x y + y^2)/(x^2 – y^2)] возвращает $\frac{-y}{x+y}$,
- разложение полиномов и понижение степени выражений, например:
  Simplify[(x + 1)^2 – x^2] возвращает 2 x + 1
- приведение общих знаменателей к выражениям с пониженной степенью или с исключением сокращаемых переменных, например:
  Simplify[2 x / (x^2 – 1) – 1/(x + 1)] возвращает 1/(x + 1).

3.10.16. Функция полного упрощения FullSimplify

Фунция **FullSimplify**, области применения которой в Mathematica заметно расширены, обладает заметно большими возможностями, чем функция **Simplify**. В частности, она обеспечивает упрощение выражений, содержащих специальные математические функции:

```
FullSimplify[Gamma[x]*x*(x+1)*(x+2)*(x+n)]
  (n+x) Gamma[3+x]
Simplify[Tan[x],ComplexityFunction→(Count[{#1},_Tan,∞]&)]
  Tan[x]
FullSimplify[Tan[x],ComplexityFunction→(Count[{#1},_Tan,∞]&)]
                  ⅈ (-1 + ℯ^(ⅈ x)) (1 + ℯ^(ⅈ x))
 -------------------------------------------------
                        1 + ℯ^(2 ⅈ x)
```

Как видно из этих примеров, функция **FullSimplify** обеспечивает упрощение даже в том случае, когда функция **Simplify** пасует. Неплохо упрощаются тригонометрические функции, особенно при использовании опции **ComplexityFunction**, подсказывающей путь упрощения.

3.10.17. Раскрытие и расширение выражений

Ниже представлены основные функции системы Mathematica, возвращающие результаты с раскрытием и расширением выражений:

- **ComplexExpand[expr]** – раскрывает expr, полагая все переменные вещественными.
- **ComplexExpand[expr, {x1, x2, ...}]** – раскрывает expr, полагая переменные соответствующими какому-либо действительному MuPAD.
- **FunctionExpand[expr]** – раскрывает выражения expr, содержащие специальные функции.
- **Expand[expr]** – раскрывает произведения и положительные целые степени в expr.
- **Expand[expr, patt]** – не выполняет расширение для тех элементов expr, которые не содержат соответствующие шаблону patt члены.
- **ExpandAll[expr]** – раскрывает все произведения и целочисленные степени в любой части expr.
- **ExpandAll[expr, patt]** – исключает из операции расширения те части expr, которые не содержат соответствующие шаблону patt члены.
- **ExpandDenominator[expr]** – раскрывает произведение и степени, которые присутствуют в выражении expr в роли знаменателей.
- **ExpandNumerator[expr]** – раскрывает произведения и степени в числителе выражения expr.
- **PowerExpand[expr]** – раскрывает вложенные степени, степени произведений, логарифмы от степеней и логарифмы от произведений. Осторожно используйте **PowerExpand** , так как эта функция не реагирует на разрывный характер выражения expr.

Приведем примеры на операции расширения выражений **Expand**:

```
Expand[(x-a)*(x-b)*(x-c)]
 -abc + abx + acx +
  bcx - ax^2 - bx^2 - cx^2 + x^3
Simplify[%]
 -(a-x) (-b+x) (-c+x)
Expand[(Sin[x]+Cos[x])/(Cos[x]*Sin[x])]
 Csc[x]+Sec[x]
Simplify[%]
 Csc[x]+Sec[x]
Expand[2*Cos[x]^2,Trig->True]
 2Cos[x]^2
Simplify[%]
 2Cos[x]^2
```

Полезно знать, что не всегда последовательное применение функций **Expand** и **Simplify** дает исходное выражение. Нередко получается новое выражение, порой представляющее определенную ценность. При операциях с тригонометрическими выражениями часто нужно использовать опцию **Trig->True**, намечая тригонометрический путь решения.

Приведем примеры на другие функции расширения выражений:

```
ExpandAll[Sin[2*Cos[x]],Trig->True]
 Cos[Cos[x]+i Sin[x]] Sin[Cos[x]-i Sin[x]]+Cos[Cos[x]-i Sin[x]]
Sin[Cos[x]+i Sin[x]]
Simplify[%]
 Sin[2 Cos[x]]
ExpandNumerator[(1+x)^2/x]
 1 + 2 x + x^2
 -------------
       x
ExpandDenominator[(1-x)^2/(1+x)^2]
   (1 - x)^2
 -------------
 1 + 2 x + x^2
ComplexExpand[Sin[a+I*b]]
 Cosh[b] Sin[a]+i Cos[a] Sinh[b]
ComplexExpand[(a+b I)/(x+d I)]
   b d          a x          a d          b x
 ------- + ------- + i (- ------- + -------)
 d^2 + x^2   d^2 + x^2    d^2 + x^2   d^2 + x^2
Simplify[%]
 - i a + b
 ---------
   d - i x
PowerExpand[Sqrt[a^2*b*c]]
 a Sqrt[b] Sqrt[c]
FunctionExpand[Gamma[4,x]]
 E^-x x^3 + 3 (E^-x x^2 + 2 (E^-x + E^-x x))
FunctionExpand[Zeta[3,2+x]]
 1   (-2 (1      1          )    )
 - (    (---- + --------- ) - PolyGamma[2, x])
 2   (   3        3         )    )
     (  x      (1 + x)      )    )
```

Разумеется, этими примерами далеко не исчерпываются возможности данной группы функций. Рекомендуется опробовать примеры из справочной системы данных Mathematica и свои собственные примеры.

3.10.18. Функция комплектования Collect

К операциям, расширяющим выражения, относятся также функции:

- **Collect[expr, x]** – выполняет приведение общих членов выражения по степеням переменной x.
- **Collect[expr, {x1, x2, ...}]** – приведение общих членов выражения по степеням переменных x1, x2, ...

Эта операция особенно полезна, если результат можно представить в виде степенных многочленов. Проиллюстрируем это следующими примерами:

```
Collect[(x-1)*(x-2)*(x^2-9),x]
 -18 + 27x - 7x^2 - 3x^3 + x^4
Collect[a*x^2+b*x*y+c*y+d*y^2,x]
 ax^2 + cy + bxy + dy^2
Collect[a*x^2+b*x*y+c*y+d*y^2,y]
 ax^2 + (c + bx)y + dy^2
(5+x^2)*(x-1)*x
 (-1 + x) x (5 + x^2)
Collect[%,x]
 -5x + 5x^2 - x^3 + x^4
```

Другой пример показывает применение функции **Collect** к выражению с двумя переменными:

```
Collect[(x-1)*(y-3)*(x-2)*(y-2)*(x-1),y,x]
 yx[-5(-2 + x)(-1 + x)^2] +
  y^2x[(-2 + x)(-1 + x)^2] + x[6(-2 + x)(-1 + x)^2]
```

Разумеется, как и в случае упрощения выражений, их расширение не является однозначной операцией и предполагает наличие определенных условностей. Опытный пользователь, используя опции функций, обычно без труда может получить результат в нужной форме.

3.10.19. Функции преобразования тригонометрических выражений

Хотя представленные выше функции иногда применимы для тригонометрических выражений, для последних есть ряд специальных функций, дающих более надежные результаты в ходе преобразований тригонометрических функций. В названии этой группы функций имеется слово Trig. Начнем со следующей функции:

TrigExpand[expr] – обеспечивает расширение выражения expr, содержащего тригонометрические и гиперболические функции.

Представленные ниже примеры иллюстрируют работу этой функции:

```
TrigExpand[Sin[a+b]]
 Cos[b] Sin[a]+Cos[a] Sin[b]
TrigExpand[Cos[3*x]]
 Cos[x]^3 - 3Cos[x]Sin[x]^2
TrigExpand[Sinh[2*x]]
 2 Cosh[x] Sinh[x]
TrigExpand[Sin[2 ArcCoth[t]]]
 2 Cos[ArcCoth[t]] Sin[ArcCoth[t]]
```

Следующая пара функций:

- **TrigToExp[expr]** – преобразует тригонометрические выражения к экспоненциальному виду;
- **ExpToTrig[expr]** – преобразует экспоненциальные выражения в тригонометрические.

Примеры на применение этих функций:

```
TrigToExp[Cos[z]]
```

$$\frac{e^{-i z}}{2} + \frac{e^{i z}}{2}$$

```
ExpToTrig[%]
 Cos[z]
f:=Sinh[z]+Cosh[z]
TrigToExp[f]
```

$$e^{z}$$

```
ExpToTrig[%]
 Cosh[z]+Sinh[z]
TrigToExp[Sin[x]/Cos[y]]
```

$$\frac{i\left(e^{-i x} - e^{i x}\right)}{e^{-i y} + e^{i y}}$$

```
ExpToTrig[%]
 Sec[y] Sin[x]
```

Приведем еще две функции:

- **TrigFactor[expr]** – раскладывает на простые множители тригонометрическое выражение expr;
- **TrigFactorList[expr]** – раскладывает тригонометрическое выражение expr на листы с термами выражения.

Следующие примеры показывают применение этих функций:

```
TrigFactor[expr]
 Sin[a + b]^3
TrigFactorList[expr]
 {{1,1},{Sin[a+b],3}}
TrigExpand[Cosh[Sin[x*y]]]
```

$$\begin{aligned}&\mathrm{Cos}\left[\frac{1}{2}\mathrm{Cos}[x\,y] - \frac{1}{2}\,i\,\mathrm{Sin}[x\,y]\right]\\ &\quad \mathrm{Cos}\left[\frac{1}{2}\mathrm{Cos}[x\,y] + \frac{1}{2}\,i\,\mathrm{Sin}[x\,y]\right] +\\ &\mathrm{Sin}\left[\frac{1}{2}\mathrm{Cos}[x\,y] - \frac{1}{2}\,i\,\mathrm{Sin}[x\,y]\right]\\ &\quad \mathrm{Sin}\left[\frac{1}{2}\mathrm{Cos}[x\,y] + \frac{1}{2}\,i\,\mathrm{Sin}[x\,y]\right]\end{aligned}$$

```
TrigFactorList[%]
 {{1,1},{Cosh[Sin[x y]],1}}
```

TrigReduce[expr] – упрощает выражения с произведениями тригонометрических функций.

Примеры на применение этой функции:

```
TrigReduce[2*Sin[x]*Cos[y]]
 Sin[x-y]+Sin[x+y]
TrigReduce[Cosh[x]*Tanh[x]]
 Sinh[x]
```

```
TrigReduce[Sin[x]^2+Cos[x]^2]
```
$$1$$
```
TrigReduce[Sin[x]*Cos[x]]
```
$$\frac{1}{2}\,\text{Sin}[2\,x]$$
```
TrigReduce[Sinh[x/y]^3]
```
$$\frac{1}{4}\left(-3\,\text{Sinh}\left[\frac{x}{y}\right]+\text{Sinh}\left[\frac{3\,x}{y}\right]\right)$$

Применение этих функций расширяет круг задач, решаемых с применением символьных преобразований.

3.10.20. Функции и директивы для работы с полиномами

Над полиномами в Mathematica можно выполнять обычные арифметические операции: сложение, вычитание, умножение и деление. Это иллюстрируют следующие примеры (p1 и p2 – полиномы от одной переменной x):

```
p1:=x^3+2*x^2+3*x+4
p2:=x^2-1
p1+p2
```
$$3+3x+3x^2+x^3$$
```
p1-p2
```
$$5+3x+x^2+x^3$$
```
Expand[p1*p2]
```
$$-4-3x+2x^2+2x^3+2x^4+x^5$$
```
p1/p2
```
$$\frac{4+3\,x+2\,x^2+x^3}{-1+x^2}$$
```
Simplify[(x^5+2*x^4+2*x^3+2*x^2-3*x-4)/(x^2-1)]
```
$$4+3x+2x^2+x^3$$

Разложение чисел, математических выражений и особенно полиномов на простые множители является столь же распространенной операцией, что и функции `Simplify`, `Collect` и `Expand`. Имеется целый ряд функций, в названии которых есть слово Factor и которые выполняют разложения:

- **Factor[poly]** – выполняет разложение полинома над целыми числами.
- **Factor[poly, Modulus->p]** – выполняет разложение полинома по модулю простого p.
- **FactorInteger[n]** – возвращает список простых множителей целого числа n вместе с их показателями степеней. Опция **FactorComplete** позволяет указать, следует ли выполнять полное разложение.
- **FactorList[poly]** – возвращает список множителей полинома с их показателями степени. Опция **Modulus->p** позволяет представить множители полинома по модулю простого p.

- **FactorSquareFree[poly]** – записывает полином в виде произведения множителей, свободных от квадратов. Опция **Modulus->p** позволяет представить разложение полинома по модулю простого p.
- **FactorSquareFreeList[poly]** – возвращает список множителей полинома, свободных от квадратов, вместе с показателями степени. Может использоваться опция **Modulus->p**.
- **FactorTerms[poly]** – извлекает полный (общий) числовой множитель в poly.
- **FactorTermsList[poly]** – возвращает лист всех общих числовых множителей полинома poly.

Ниже представлен ряд примеров на применение этих функций:

```
Factor[x^3-6*x^2+11*x-6]
 (-3+x) (-2+x) (-1+x)
Factor[x^3-6*x^2+21*x-52]
 (-4 + x)(13 - 2x + x^2)
Factor[x^5+8*x^4+31*x^3+80*x^2+94*x+20,Modulus->3]
 (1 + x)/(2 + x)
FactorList[x^4-1,Modulus->2]
 {{1,1},{1+x,4}}
FactorSquareFree[(x^2+1)*(x^4-1)]
 (-1 + x^2)(1 + x^2)
FactorSquareFree[(x^2+1)*(x^4-1),Modulus->2]
 (1+x)^4
FactorSquareFreeList[(x^2+1)*(x^4-1),Modulus->2]
 {{1,1},{1+x,6}}
FactorTerms[2*x^2+4*x+6]
 2(3 + 2x + x^2)
FactorTermsList[2*x^2+4*x+6]
 {2, 3 + 2x + x^2}
FactorInteger[123456789]
 {{3,2},{3607,1},{3803,1}}
FactorList[x^4-1]
 {{1, 1}, {-1 + x, 1}, {1 + x, 1}, {1 + x^2, 1}}
FactorSquareFreeList[(x^2+1)*(x^4-1)]
 {{1, 1}, {-1 + x^2, 1}, {1 + x^2, 2}}
```

Обычно функция `Factor` выявляет внутреннюю суть полинома, раскладывая его множители, содержащие корни полинома. Однако в ряде случаев корни полинома удобнее получать в явном виде с помощью уже рассмотренной функции **Roots**.

Функция `Factor` может работать с опцией **Trig->True**:

```
Factor[Csc[x]+Sec[x],Trig->True]
 Csc[x] Sec[x] (Cos[x]+Sin[x])
Factor[Sin[3*x],Trig->True]
 (1+2 Cos[2 x]) Sin[x]
```

3.10.21. Другие функции для работы с полиномами

Имеется множество функций, большей частью достаточно очевидных для знакомого с математикой пользователя, – функций для работы с полиномами:

- **Decompose[poly, x]** – выполняет разложение полинома, если это возможно, на более простые полиномиальные множители.
- **GroebnerBasis[{poly1, poly2, ...}, {x1, x2, ...}]** – возвращает список полиномов, которые образуют базис Гробнера для идеала, порожденного полиномами polyi.
- **PolynomialDivision[p, q, x]** – возвращает список частного и остатка, полученных делением полиномов p и q от x.
- **PolynomialGCD[poly1, poly2, ...]** – возвращает наибольший общий делитель ряда полиномов poly1, poly2, ... С опцией **Modulus->p]** функция возвращает GCD по модулю простого p.
- **PolynomialLCM[poly1, poly2, ...]** – возвращает наименьшее общее кратное полиномов poly1, poly2, ... С опцией **Modulus->p]** функция возвращает LCM по модулю простого p.
- **PolynomialMod[poly, m]** – возвращает полином poly, приведенный по модулю m.
- **PolynomialMod[poly, {m1, m2, ...}]** – выполняет приведение по модулю всех mi.
- **PolynomialQ[expr, var]** – выдает значение True, если expr является полиномом от var, иначе дает False.
- **PolynomialQ[expr, {var1, ...}]** – проверяет, является ли expr полиномом от vari.
- **PolynomialQuotient[p, q, x]** – возвращает частное от деления p и q как полиномов от x, игнорируя какой-либо остаток.
- **PolynomialRemainder[p, q, x]** – возвращает остаток от деления p на q как полиномов от x.
- **Resultant[poly1, poly2, var]** – вычисляет результат полиномов poly1 и poly2 по переменной var. С опцией **Modulus->p]** функция вычисляет результат по модулю простого p.

Работа с этими функциями сводит операции с многочленами к типовым алгебраическим операциям.

3.10.22. Расширенные операции с выражениями

В этом параграфе сосредоточено описание основных функций для расширенных операций с выражениями. Для этого служат следующие функции:

- **AlgebraicRules[eqns, {x1, x2, ...}]** – формирует множество алгебраических правил, которые замещают переменные, ранее находившиеся в списке xi, на

более поздние в списке, соответствующем уравнению (или уравнениям) eqns.

- **Apart[expr]** – возвращает expr, записывая заново рациональное выражение как сумму членов с минимальными знаменателями.
- **Apart[expr, var]** – аналогична **Apart[expr]**, но все переменные, кроме var, интерпретируются как константы.
- **ApartSquareFree[expr, var]** – возвращает expr как сумму членов со знаменателями, свободными от квадратов, по переменной var.
- **Catch[expr]** – возвращает аргумент первого Throw, генерируемого при вычислении expr.
- **Check[expr, failexpr]** – вычисляет expr и возвращает его результат, если только не вырабатывались сообщения, иначе вычисляет и возвращает failexpr.
- **Check[expr, failexpr, s1::t1,s2::t2, ...]** – выполняет контроль только для указанных сообщений.
- **CheckAbort[expr, failexpr]** – вычисляет expr, возвращая failexpr в случае прерывания.
- **Coefficient[expr, form]** – возвращает коэффициент перед form в полиномиальном выражении expr.
- **Coefficient[expr, form, n]** – возвращает коэффициент перед form^n в выражении expr.
- **CompoundExpression** – применяется в виде:
- **expr1; expr2; ...** – вычисляет expri по очереди, возвращая последнее как результат.
- **Depth[expr]** – возвращает максимальное число индексов, требуемых для указания любой части выражения expr, плюс единица.
- **Dimensions[expr]** – возвращает список размерностей выражения expr.
- **Dimensions[expr, n]** – возвращает список размерностей expr, пониженного до уровня n.
- **Edit[expr___]** – предоставляет на редактирование выражения expr.
- **Evaluate[expr]** – вычисляет выражение expr безусловно, то есть даже если оно оказывается аргументом функции, чьи атрибуты определяют его невычисляемым.
- **Exponent[expr, form]** – возвращает максимальную степень, с которой form присутствует в expr.
- **Exponent[expr, form, h]** – применяет h к множеству показателей степеней (экспонент), с которыми form обнаруживается в выражении expr.
- **FlattenAt[expr, {i, j, ...}]** – выравнивает часть выражения expr на позиции {i, j, ...}.
- **FlattenAt[expr, {{i1, j1, ...}, {i2, j2, ...}, ...}]** – выравнивает части выражения expr в нескольких позициях.
- **HeldPart[expr, pos]** – извлекает (удаляет) часть или несколько частей, указанных при помощи pos, и помещает их в Hold.
- **Hold[expr]** – содержит expr в невычисленном виде.

- **HoldForm[expr]** – выводит выражение expr, сохраняя при этом его в невычисленной форме.
- **LeafCount[expr]** – возвращает общее (полное) число неделимых подвыражений в expr.
- **Length[expr]** – возвращает число элементов в expr.
- **Level[expr, levelspec]** – возвращает список всех подвыражений выражения expr на уровнях, указанных параметром levelspec.
- **Level[expr, levelspec, f]** – относит f к списку подвыражений.
- **Literal[expr]** – является эквивалентом expr в смысле совпадения формы, но содержит expr в непреобразованном виде.
- **LogicalExpand[expr]** – выполняет расширение выражений, содержащих логические связи, как, например, && и ||.
- **MapAt[f, expr, n]** – применяет f к элементу на позиции n в expr. Если n отрицательно, позиция отсчитывается с конца.
- **MapAt[f, expr, {i, j,...}]** – применяет f к частям expr на позиции {i,j,...}.
- **MapAt[f, expr, {{i1, j1, ...}, {i2, j2, ...}, ...}]** – применяет f к частям expr в ряде позиций.
- **MapIndexed[f, expr]** – применяет f к элементам expr, возвращая спецификацию части каждого элемента в качестве второго аргумента в f.
- **MapIndexed[f, expr, levspec]** – применяет f ко всем частям expr на уровнях, указанных с помощью levspec.
- **Order[expr1, expr2]** – возвращает 1, если expr1 предшествует expr2 в канонической последовательности, и дает –1, если expr1 стоит после expr2 в каноническом порядке. Результатом будет 0 , если expr1 тождественно expr2.
- **Postfix[f[expr]]** – выводит с f[expr], заданной по умолчанию в постфиксной форме: expr // f.
- **Postfix[f[expr], h]** – выводит в виде exprh.
- **Prepend[expr, elem]** – возвращает expr, к которому предварительно добавлен elem.
- **Print[expr1, expr2, ...]** – выводит на экран дисплея выражения expri и затем дает перевод строки. Может использоваться для создания диалога.
- **Return[expr]** – возвращает величину expr из функции.
- **Run[expr1, expr2, ...]** – создает выводимую форму выражений expri, разделенных пробелами, и выполняет ее как внешнюю команду операционной системы.
- **RunThrough[«command», expr]** – выполняет внешнюю команду и возвращает результат вычисления expr.
- **Scan[f, expr]** – вычисляет f, применяемую к каждому элементу expr по очереди.
- **Scan[f, expr, levelspec]** – применяет f к частям выражения expr, указанным с помощью levelspec.
- **SequenceForm[expr1, expr2, ...]** – печатает в виде текстовой конкатенации (объединения) печатных форм выражений expri.

- **SetAccuracy[expr, n]** – дает вариант expr, в котором все числа должны быть установлены с точностью n цифр.
- **SetPrecision[expr, n]** – вырабатывает вариант expr, в котором все числа установлены с точностью представления n цифр.
- **Share[expr]** – меняет способ внутреннего хранения выражения expr, что минимизирует объем используемой памяти.
- **Through[expr, h]** – выполняет преобразование всюду, где h появляется в заголовке выражения expr.
- **Together[expr]** – приводит члены суммы к общему знаменателю и сокращает множители в полученном результате.
- **Variables[expr]** – возвращает список всех независимых переменных в выражении.
- **With[{x = x0, y = y0, ...}, expr]** – указывает, что в случае обнаружения в выражении **expr** символов **x, y, ...** они должны быть заменены на **x0, y0,...**.
- **Write[channel, expr1, expr2, ...]** – записывает в указанный выходной канал **channel** последовательно (один за другим) выражения **expri**, завершаемые **newline** (переводом строки).
- **WriteString[channel, expr1, expr2, ...]** – превращает expri в символьные цепочки, а затем последовательно записывает их в указанный выходной канал **channel**.

К сожалению, объем этого раздела не позволяет привести примеры на все эти функции. Поэтому приведем лишь отдельные примеры работы с некоторыми из этих функций:

```
Apart[(x^4+1)/(x^2-1)]
```

$$1+\frac{1}{-1+x}+x^2-\frac{1}{1+x}$$

```
Apart[(x^3-y^3-1)/(x^2-y),y]
```

$$x^4+x^2y+y^2+\frac{1-x^3+x^6}{-x^2+y}$$

```
Cancel[(x^2-1)/(x-1)]
 1+x
Denominator[(x^2-x-1)/(x-1)]
 -1+x
Numerator[(x^2-x-1)/(x-1)]
 -1 - x + x^2
Depth[x^3+x^2+x+1]
 3
Dimensions[x^3-2*x^2+1]
 {3}
Evaluate[1+1+Sin[1]]
 2+Sin[1]
Head[Sin[x]]
 Sin
```

Обилие функций для работы с математическими выражениями позволяет с помощью системы Mathematica решать самые серьезные задачи символьной математики (компьютерной алгебры).

3.10.23. Поддержка кусочных функций в Mathematica

В реализациях системы Mathematica 5.1/5.2/6 существенно расширена поддержка кусочных функций PieceWise. Это относится как к обычным операциям символьных преобразований, так и к вычислению интегралов, решению уравнений и иным операциям математического анализа. Ограничимся простым примером, представленным на рис. 3.9.

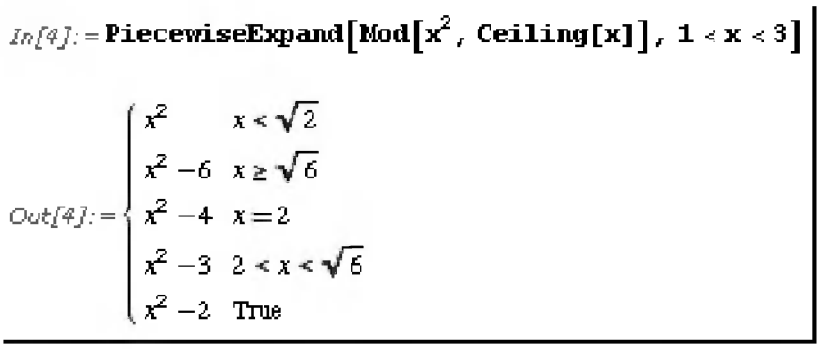


Рис. 3.9. Пример операции, порождающей решение в форму PieceWise функции

Здесь использована функция расширения выражения PiecewiseExpand. Подобные функции есть и для других видов символьных преобразований.

3.11. О работе с выражениями и функциями в других СКМ

Разумеется, и другие СКА имеют подобные функции, хотя и в более скромном наборе. В Mathcad, вплоть до версии 13 включительно, используется ядро Maple, поэтому функции этих систем практически идентичны. В Mathcad 14 применено новое ядро – от системы MuPAD. Derive и MuPAD имеют свои ядра (куда менее мощные, чем у Maple и Mathematica). Поэтому результаты преобразований у них иногда могут отличаться от приведенных для Maple и Mathematica. Чаще всего эти различия не существенны и не принципиальны.

В любой СКМ и СКА работа с выражениями и функциями требует от пользователя повышенного внимания. Особенно это касается функций упрощения, точная формулировка действия которых в описаниях систем чаще всего не указывается.

Глава 4

Практика математического анализа

Математический анализ [123–126] – одна из самых благодатных областей применения систем компьютерной алгебры. В этой главе описано решение с помощью СКА наиболее важных задач математического анализа. Как и ранее, основной системой для этого взята система Maple, но возможности других систем также будут представлены. Особое внимание в этой главе уделено визуализации записи исходных выражений и результатов вычислений, а также проверке последних.

4.1. Вычисление сумм последовательностей

4.1.1. Основные функции для вычисления сумм последовательностей

Вычисление *суммы* членов некоторой последовательности $f(k)$ при изменении целочисленного индекса k от значения m до значения n с шагом +1, то есть выражения

$$\sum_{k=m}^{n} f(k) = f(m) + f(m+1) + \ldots + f(n-1) + f(n),$$

является достаточно распространенной операцией математического анализа. Для вычисляемой и инертной форм сумм последовательностей в Maple служат следующие функции:

```
sum(f,k);          sum(f,k=m..n);          sum(f,k=alpha);
Sum(f,k);          Sum(f,k=m..n);          Sum(f,k=alpha).
```

Здесь `f` – функция, задающая члены суммируемого ряда, `k` – индекс суммирования, `m` и `n` – целочисленные пределы изменения `k`, `alpha` – `RootOf`-выражение. Значение `n` может быть принято бесконечным для вычисления бесконечных сумм (многие такие суммы сходятся). В этом случае для `n` используется обозначение ∞ или `infinity`. Допустимо (а зачастую рекомендуется с целью исключения преждевременной оценки суммы) заключение `f` и `k` в прямые кавычки – например, `sum('f','k'=m..n)`. Рекомендуется все примеры проверять после команды `restart`, убирающей предыдущие определения `f` и `k`.

> ***Внимание!*** *При вычислении сумм (и произведений) последовательностей надо строго соблюдать прямой (нарастающий) порядок задания значений индексной переменной суммы. Нарушение этого порядка чревато грубыми ошибками. Так что правила о том, что при измени порядка суммируемых или умножаемых членов последовательности сумма и произведения не меняются, в данном случае не поддерживаются на программном уровне.*

4.1.2. Последовательности с заданным числом членов

Простейшими являются суммы последовательностей с фиксированным числом членов. Ниже даны примеры применения этих функций:

```
> restart;k:=2;
```

$$k := 2$$

```
> Sum(k^2,k=1..4);
```

$$\sum_{2=1}^{4} 4$$

```
> sum(k^2,k=1..4);
Error, (in sum) summation variable previously assigned, second
argument evaluates to 2 = 1 .. 4
> sum('k^2','k'=1..4);
```

$$30$$

```
> sum(1/i,i=1..100);
```

$$\frac{14466636279520351160221518043104131447711}{2788815009188499086581352357412492142272}$$

```
> evalf(%);
```

$$5.187377518$$

Обратите внимание, что во втором примере система отказалась от вычисления, а в третьем даже выдала сообщение об ошибке, связанное с тем, что переменной k перед вычислением сумм было ранее присвоено численное значение 2. После заключения выражения и переменной индекса k в прямые кавычки ошибка исчезла, поскольку такая операция означает, что переменной изначально придается неопределенное значение.

4.1.3. Суммы с известным пределом

Особый класс образуют последовательности, у которых предел задается в общем виде значением переменной:

```
> restart;
> sum(k,k=1..n);
```

$$\frac{(n+1)^2}{2} - \frac{n}{2} - \frac{1}{2}$$

```
> sum(i/(i+1),i=0..n);
```

$$n + 1 - \Psi(n+2) - \gamma$$

```
> sum(k*binomial(n,k),k=0..n);
```

$$\frac{2^n n}{2}$$

Некоторые из таких сумм выражаются через специальные математические функции – см., например, второй пример.

4.1.4. Суммы бесконечных рядов

Многие суммы *бесконечных рядов* сходятся к определенным численным или символьным значениям, и система Maple способна их вычислять при задании индекса k без указания пределов изменения. Это поясняют следующие примеры:

```
> restart;
> sum(-exp(-k),k);
```

$$\frac{\mathbf{e}}{(-1+\mathbf{e})\mathbf{e}^{k}}$$

```
> sum(k*a^k,k);
```

$$\frac{a^{k}(k\,a-k-a)}{(a-1)^{2}}$$

```
> Sum(1/i^2,i=1..infinity)=sum(1/i^2,i=1..infinity);
```

$$\sum_{i=1}^{\infty}\frac{1}{i^{2}}=\frac{\pi^{2}}{6}$$

```
> Sum(1/n!,n=1..infinity)=sum(1/n!,n=1..infinity);
```

$$\sum_{n=1}^{\infty}\frac{1}{n!}=\mathbf{e}(1-\mathbf{e}^{(-1)})$$

```
> Sum(1/i^2,i)=sum(1/i^2,i);
```

$$\sum_{i}\frac{1}{i^{2}}=-\Psi(1,i)$$

4.1.5. Двойные суммы

Могут встречаться множественные суммы по типу «сумма в сумме». Ограничимся приведением примера *двойной суммы*, имеющей аналитическое значение:

```
> Sum( Sum(k^2, k = 1..m), m = 1..N);   factor( simplify( value(%)));
```

$$\sum_{m=1}^{N}\left(\sum_{k=1}^{m}k^{2}\right)$$

$$\frac{N(N+2)(N+1)^{2}}{12}$$

При конкретном значении N такую сумму нетрудно вычислить подстановкой:

```
> subs( N = 100, %);
```

$$8670850$$

4.1.6. Пакет вычисления специальных сумм sumtools

Возможности вычисления *специальных сумм* существенно расширяются при использовании инструментального пакета вычисления специальных сумм sumtools. При его вызове выводится список функций пакета:

```
> with(sumtools);
```
[*Hypersum, Sumtohyper, extended_gosper, gosper, hyperrecursion, hypersum, hyperterm, simpcomb, sumrecursion, sumtohyper*]

Назначение некоторых функций данного пакета перечислено ниже:

- `hypersum(U, L, z, n)` и `Hypersum(U, L, z, n)` – вычисление гипер-сумм;
- `sumtohyper(f, k)` и `Sumtohyper(f, k)` – преобразование сумм в гипер-суммы;
- `extended_gosper(f, k)`, `extended_gosper(f, k=m..n)` и `extended_gosper(f, k, j)` – реализация расширенного алгоритма Госпера;
- `gosper(f, k)` и `gosper(f, k=m..n)` – реализация алгоритма Госпера;
- `hyperrecursion(U, L, z, s(n))` – реализация гиперрекурсионного алгоритма;
- `hyperterm(U, L, z, k)` и `Hyperterm(U, L, z, k)` – ввод гипергеометрического терма.

4.1.7. Примеры вычисления специальных сумм

Приведем примеры на вычисление специальных сумм с помощью функций пакета sumtools:

```
> extended_gosper(k*(k/2)!,k=1..n);
```
$$2\left(\frac{n}{2}+\frac{1}{2}\right)!+2\left(\frac{n}{2}+1\right)!-2\left(\frac{1}{2}\right)!-2\ 1!$$

```
> hyperrecursion([-n,a],[b],1,f(n));
```
$$(-n+a-b+1)\,\mathrm{f}(n-1)+(n+b-1)\,\mathrm{f}(n)$$

```
> simpcomb(binomial(n,k));
```
$$\frac{\Gamma(n+1)}{\Gamma(n-k+1)\,\Gamma(k+1)}$$

```
> sumrecursion(binomial(n,k)^3,k,f(n));
```
$$-8(n-1)^2\,\mathrm{f}(n-2)-(7n^2-7n+2)\,\mathrm{f}(n-1)+\mathrm{f}(n)n^2$$

Из этих примеров применение функций данного пакета достаточно очевидно.

4.1.8. Вычисление сумм в Mathematica и в других СКМ

В других СКМ также имеются средства для вычисления сумм последовательностей. В системах Mathematica 4/5 для вычисления сумм используется следующая функция:

```
Sum[f, {i, imax}] Sum[f, {i, imin, imax}]  Sum[f, {i, imin, imax, di}]
Sum[f, {i, imin, imax}, {j, jmin, jmax}, ...]
```

Пр отсутствии параметра шага di он принимается равным 1. Приведем примеры вычисления сумм в этой системе:

```
Sum[x^n/n!, {n, 1, 9, 2}]
```

$$x+\frac{x^3}{6}+\frac{x^5}{120}+\frac{x^7}{5040}+\frac{x^9}{362880}$$

```
Sum[x^i y^j, {i, 1, 4}, {j, 1, i}]
```

$$x\,y+x^2\,y+x^3\,y+x^4\,y+x^2\,y^2+x^3\,y^2+x^4\,y^2+x^3\,y^3+x^4\,y^3+x^4\,y^4$$

```
Sum[i^3/(i+1), {i, 1, n}]
```

$$\frac{1}{3}\,(3-3\,\mathrm{EulerGamma}+2\,n+n^3)-\mathrm{PolyGamma}[0,\,2+n]$$

В Mathematica есть также функция **NSum** для вычисления *сумм в численном виде*. Она аналогична выражению **N[Sum[...]]**, где функция **N** вычисляет любое выражение в численном виде, если такой результат возможен.

В Mathcad функций суммирования нет, но есть *оператор суммирования* в панели Calculus – см. рис. 1.23. Примеры его применения показаны на рис. 1.24 и 1.28. Возможно вычисление множественных сумм. В Derive и MuPAD также есть операторы и функции суммирования, но они слабо поддерживают вычисление тех сумм, результатом которых являются специальные математические функции.

4.2. Вычисление произведений членов последовательностей

4.2.1. Основные функции для произведения членов последовательностей

Аналогичным образом для *произведений* членов $f(i)$ некоторой последовательности, например вида

$$\prod_{i=m}^{n} f(i)=f(m)f(m+1)\cdots f(n-1)f(n),$$

используются следующие функции:

```
product(f,k);  product(f,k=m..n);  product(f,k=alpha);
Product(f,k);  Product(f,k=m..n);  Product(f,k=alpha).
```

Обозначения параметров этих функций и их назначение соответствуют приведенным для функций вычисления сумм. Это относится, в частности, и к применению одиночных кавычек для `f` и `k`.

4.2.2. Примеры вычисления произведений членов последовательностей

Примеры применения функций вычисления произведений даны ниже:

```
> restart;
> Product(k^2,k=1..5)=product(k^2,k=1..5);
```

$$\prod_{k=1}^{5} k^2 = 14400$$

```
> Product(k^2,k)=product(k^2,k);
```

$$\prod_{k} k^2 = \Gamma(k)^2$$

```
> product(a[k],k=1..5);
```

$$a1a2a3a4a5$$

```
> f:=[1,2,3,4,5];
```

$$f := [1, 2, 3, 4, 5]$$

```
> product(f[k],k=1..4);
```

$$24$$

```
> Product(n+k,k=1..m)=product(n+k,k=1..m);
```

$$\prod_{k=1}^{m} (n+k) = \frac{(n+m+1)}{(n+1)}$$

```
> product(k,k=RootOf(x^3-9));
```

$$9$$

Как и в случае вычисления сумм, вычисление произведений возможно как в численной, так и в аналитической форме – разумеется, если таковая существует. Это показывает следующий пример:

```
> Product(2/I,I=1..infinity)=product(2/I,I=1..infinity);
```

$$\prod_{i=1}^{\infty} \left(\frac{2}{i}\right) = 0$$

Нетрудно понять, что при i, стремящемся к бесконечности, перемножаемые члены последовательности стремятся к нулю, а потому к нулю стремится и их произведение.

4.2.3. Вычисление произведений в Mathematica и в других СКМ

Операции вычисления произведений в Mathematica 4/5 представлены следующими вполне очевидными функциями:

```
Product[f, {i, imax}]      Product[f, {i, imin, imax}]
Product[f, {i, imin, imax, di}]
Product[f, {i, imin, imax}, {j, jmin, jmax}, ...]
```

Нижеприведенные примеры иллюстрируют вычисление произведения в символьном виде:

$$\prod_{i=1}^{5}\left(x+i^2\right)$$

```
(1+x) (4+x) (9+x) (16+x) (25+x)
```

$$\prod_{k=1}^{m}(n+k)$$

```
Pochhammer[1+n,m]
```

В Mathematica есть и функция вычисления произведений в численном виде – NProduct.

Mathcad вычисляет произведения с помощью оператора – см. рис. 1.28 снизу. Операторы и функции вычисления произведений есть в системах Derive и MuPAD, но набор вычисляемых сумм намного меньше, чем в Maple и Mathematica.

4.3. Вычисление производных

4.3.1. Определение производной и полного дифференциала

Если $f(x)$ – непрерывная функция аргумента x, то *производная* этой функции

$$f'(x)=\frac{df(x)}{dx}=\lim_{\Delta x\to 0}\frac{f(x+\Delta x)-f(x)}{\Delta x}. \qquad (4.1)$$

Если речь идет о вычислении численного значения производной, то оно производится в некоторой точке $x = x0$. Как известно, значение производной геометрически характеризуется наклоном касательной к графику $f(x)$ в точке $x = 0$.

Помимо производной, часто встречается понятие дифференциала

$$df(x) = f'(x)\cdot\Delta x,$$

то есть произведения производной функции на приращение ее аргумента $\Delta x \to 0$.

Непрерывность функции не является достаточным признаком того, что она имеет производную. Не все такие функции имеют производные во всех точках. В принципе, возможны даже непрерывные функции, вообще не имеющие производных. Но это – математическая экзотика. Разрывные функции в точках разрыва не имеют производных, хотя возможны производные слева и справа от точек разрыва.

Производная от производной $f(x)$, то есть функция $f''(x)$, называется *производной второго порядка*. Могут быть производные третьего, четвертого и т. д. – словом, *производные высшего порядка*. Все математические системы способны вычислять такие производные, как и первую производную $f'(x)$, от функции $f(x)$. Довольно часто встречаются функции ряда переменных, например $f(x,y,z,...)$. В этом случае может идти речь о *частных производных* по переменным x, y, z, ...

Системы символьной математики позволяют вычислять производные как символьной, так и в численной форме.

4.3.2. Функции дифференцирования diff и Diff Maple

Для вычисления производных Maple имеет следующие основные *функции дифференцирования*:

```
diff(a, x1, x2, …, xn)          diff(a, [x1, x2, …, xn])
Diff(a, x1, x2, …, xn)          Diff(a, [x1, x2, …, xn])
```

Здесь `a` – дифференцируемое алгебраическое выражение, в частности функция `f(x1, x2, …, xn)` ряда переменных, по которым производится дифференцирование. Функция `Diff` является инертной формой вычисляемой функции `diff` и может использоваться для естественного воспроизведения производных в документах.

Первая из этих функций (в вычисляемой и в инертной формах) вычисляет частные производные для выражения `a` по переменным `x1`, `x2`, ..., `xn`. В простейшем случае `diff(f(x),x)` вычисляет первую производную функции $f(x)$ по переменной x. При n, большем 1, вычисления производных выполняются рекурсивно, например `diff(f(x), x, y)` эквивалентно `diff(diff (f(x), x), y)`. Оператор `$` можно использовать для вычисления производных высокого порядка. Для этого после имени соответствующей переменной ставится этот оператор и указывается порядок производной. Например, выражение `diff(f(x),x$4)` вычисляет производную 4-го порядка и эквивалентно записи `diff(f(x),x,x,x,x)`. А `diff(g(x,y),x$2,y$3)` эквивалентно `diff(g(x,y),x,x,y,y,y)`.

Примеры визуализации и вычисления производных:

```
> restart;
> Diff(a*sin(b*x),x)=diff(a*sin(b*x),x);
```

$$\frac{\partial}{\partial x} a \sin(bx) = a\cos(bx)b$$

```
> Diff([sin(x),x^n,exp(a*x)],x)=diff([sin(x),x^n, exp(a*x)],x);
```

$$\frac{\partial}{\partial x}[\sin(x), x^n, \mathbf{e}^{(ax)}] = \left[\cos(x), \frac{x^n n}{x}, a\,\mathbf{e}^{(ax)}\right]$$

```
> Diff(a*x^n,x$3)=diff(a*x^n,x$3);
```

$$\frac{\partial^3}{\partial x^3} a\,x^n = \frac{a\,x^n n^2}{x^3} - \frac{3a\,x^n n^2}{x^3} + \frac{2a\,x^n n}{x^3}$$

```
> Diff([x^2,x^3,x^n],x)=diff([x^2,x^3,x^n],x);
```

$$\frac{\partial}{\partial x}[x^2, x^3, x^n] = \left[2x, 3x^2, \frac{x^n n}{x}\right]$$

Как видно из приведенных примеров, функции вычисления производных могут использоваться с параметрами, заданными списками. Приведенные ниже при-

меры показывают эти возможности и иллюстрируют дифференцирование функции пользователя для двух переменных:

```
> restart; f(x,y):=cos(x)*y^3;
```

$$f(x, y) := \cos(x)y^3$$

```
> Diff(f(x,y),x)=diff(f(x,y),x);
```

$$\frac{\partial}{\partial x}\cos(x)y^3 = -\sin(x)y^3$$

```
> Diff(f(x,y),y)=diff(f(x,y),y);
```

$$\frac{\partial}{\partial x}\cos(x)y^3 = 3\cos(x)y^2$$

```
> Diff(f(x,y),x,y)=diff(f(x,y),x,y);
```

$$\frac{\partial^2}{\partial y\,\partial x}\cos(x)y^3 = -3\sin(x)y^2$$

```
> Diff(f(x,y),x$4)=diff(f(x,y),x$4);
```

$$\frac{\partial^4}{\partial x^4}\cos(x)y^3 = \cos(x)y^3$$

Получаемые в результате дифференцирования выражения могут входить в другие выражения. Можно задавать их как функции пользователя и строить графики производных.

4.3.3. Дифференциальный оператор D

Для создания функций с производными может также использоваться *дифференциальный оператор* `D`. Порой он позволяет создавать более компактные выражения, чем функции `diff` и `Diff`. Дифференциальный оператор можно записывать в следующих формах: `D(f)` или `D[i](f)`, где параметр `f` – выражение или имя функции, `i` – положительное целое число, выражение или последовательность. Оператор `D(f)` просто вычисляет имя производной от `f`, поскольку в этой форме он эквивалентен `unnaply(diff(f(x),x),x)`. В форме `D(f)(x)` этот оператор подобен `diff(f(x),x)`.

Приведем примеры дифференцирования функций, заданных только именами, и функций с одним параметром:

```
> restart;
> D(exp^2+cos^2+tan+GAMMA);
```

$$2\exp^2 - 2\sin\cos + 1 + \tan^2 + \Psi\Gamma$$

```
> D(ln);
```

$$a \to \frac{1}{a}$$

```
> D[1](sin*cos);
```

$$\cos^2 - \sin^2$$

Следующий пример показывает дифференцирование функции пользователя `fun` с применением дифференциального оператора `D` и функции `diff`:

```
> fun:=(x)->sin(x^2);
```

$$fun := x \to \sin(x^2)$$

```
> D(fun)=diff(fun(x),x);
```

$$(x \to 2\cos(x^2)x) = 2\cos(x^2)x$$

Дифференциальный оператор можно применять и для дифференцирования функций нескольких переменных по заданной переменной:

```
> f:=(x,y,z)->x*exp(y)+ln(z);
```

$$f := (x, y, z) \to x\,\mathbf{e}^y + \ln(z)$$

```
> D[1](f);
```

$$(x, y, z) \to \mathbf{e}^y$$

```
> D[2](f);
```

$$(x, y, z) \to x\,\mathbf{e}^y$$

```
> D[3](f);
```

$$(x, y, z) \to \frac{1}{z}$$

Пример применения дифференциального оператора для функции `f`, заданной программным объектом-процедурой, представлен ниже:

```
> restart;
> f:=proc(x,b,n) local i,d,s;
> s:=0;
> for i from n by -1 to 0 do s:=s*x+b[i] od;
> s
> end:
> D[1](f);
```

proc(x, b, n)
local i, s, sx;
    $sx := 0$;
    $s := 0$;
    for i **from** n **by** -1 **to** 0 **do** $sx := sx \times x + s$; $s := s \times x + b[i]$ **end do**;
    sx
end proc

Этот пример показывает реализацию *схемы Горнера* для полинома `b` степени `n` от переменной `x`. При этом применение оператора дифференцирования возвращает процедуру. Ряд интересных возможностей по вычислению производных предоставляет пакет расширения `student`.

4.3.4. Maplet-вычислитель производных Derivatives

При обучении основам математического анализа удобны обучающие средства на основе *Maplet-технологии*. Эти новые средства размещены в позиции Tools меню системы Maple 9.5 при ее применении в стандартном виде. Команда **Tools** ⇒ **Tutors** ⇒ **Calculus-Single Veriables** ⇒ **Derivatives...** открывает окно Maple-вычислителя производных, показанное на рис. 4.1.

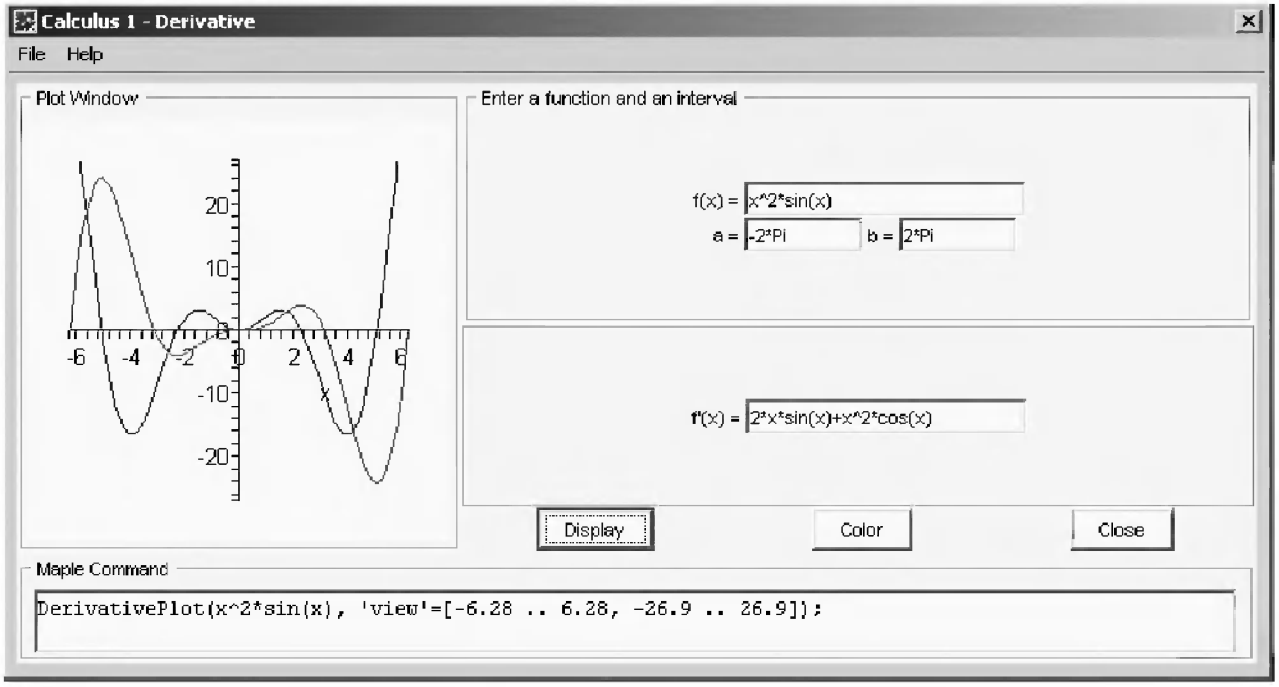


Рис. 4.1. Окно Maplet-вычислителя производных

В окне можно в интерактивном режиме задать выражение для функции $f(x)$, вычислить производную $f'(x)$ и, нажав кнопку Dispay, получить графики заданной функции и ее производной в заданных пределах изменения x от a до b. При закрытии окна графики появляются в текущей строке вывода системы Maple 9.5.

4.3.5. Maplet-инструмент по методам дифференцирования

Инструмент `Differentiate Methods...` служит для иллюстрации методов аналитического дифференцирования – см. рис. 4.2.

Здесь можно задать выполнение всех шагов дифференуирования сразу по всем шагам (кнопка `All Steps`) или запустить дифференцирование раздельно по шагам (кнопка `Start`). С помощью кнопки `Hint` можно вызвать советы по дифференцированию и применить их активизацией кнопки `Apply Hint` в поле Differentiate Rules (Правила дифференцирования). Пример на рис. 4.2 показывает дифференцирование функции f(x)=sin(x)*exp(-x). Представлены шаги дифференцирования и конечный результат.

4.3.6. Дифференцирование в Mathematica 4/5

СКМ Mathematica 4/5 ничуть не уступает в средствах символьного дифференцирования системам класса Maple. Mathematica имеет следующие *функции дифференцирования*:

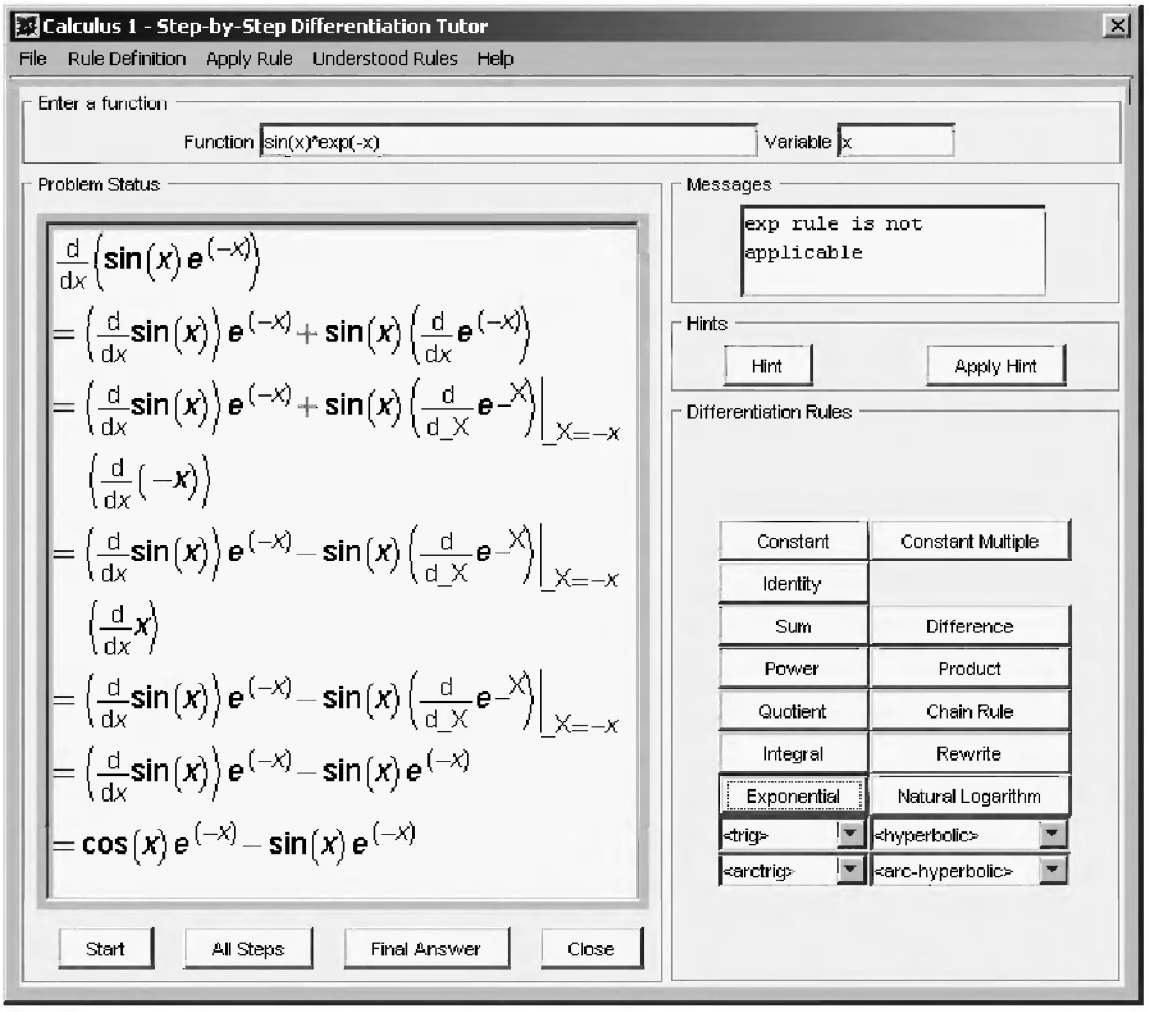


Рис. 4.2. Окно Maplet-инструмента по методам дифференцирования

- **D[f, x]** – возвращает частную производную функции f по переменной x.
- **D[f, {x, n}]** – возвращает частную производную n-го порядка по x.
- **D[f, x1, x2, ...]** – возвращает смешанную производную.
- **Dt[f, x]** – возвращает обобщенную производную функции f по переменной x.
- **Dt[f]** – возвращает полный дифференциал f.

Для функции **D** существует опция **NonConstants**, которая возвращает список объектов, находящихся в неявной зависимости от переменных дифференцирования. По умолчанию список пустой. Аналогично для функции **Dt** имеется опция Constant (по умолчанию возвращает пустой список). На практике применение данных опций встречается редко.

Существует еще одна функция:

- **Derivative[n1, n2, ...][f]** – основная (общая) форма представления функции, полученной в результате дифференцирования f n1 раз по первому аргументу, n2 раз по второму аргументу и т. д.

К примеру, **Derivative[2][x*y]** возвращает (x y)'' а **Derivative[2,3][x*y]** соответственно $(x\ y)^{(2,3)}$.

Следующие примеры показывают применение функции D для вычисления производной в аналитическом виде:

```
D[Exp[x/b],x]
```

$$\frac{e^{\frac{x}{b}}}{b}$$

```
D[Log[3*x/4],x]
```

$$\frac{1}{x}$$

```
D[x^n,{x,4}]
```

$$(-3 + n)(-2 + n)(-1 + n)nx^{-4+n}$$

```
D[BesselJ[2,x],x]
```

$$\frac{1}{2}(\text{BesselJ}[1, x] - \text{BesselJ}[3, x])$$

Приведенные ниже примеры иллюстрируют вычисление производных от первого до третьего порядка включительно для функции f[x], заданной пользователем:

```
f[x]:=x/(1+x^2)
D[f[x],{x,1}]
```

$$2\left(-\frac{2x^2}{(1+x^2)^2} + \frac{1}{1+x^2}\right)$$

```
D[%,x]
```

$$\frac{8x^3}{(1+x^2)^3} - \frac{6x}{(1+x^2)^2}$$

```
D[f[x],{x,2}]
```

$$-\frac{4x}{(1+x^2)^2} + x\left(\frac{8x^2}{(1+x^2)^3} - \frac{2}{(1+x^2)^2}\right)$$

```
D[D[D[f[x],x],x],x]
```

$$-\frac{48x^4}{(1+x^2)^4} + \frac{48x^2}{(1+x^2)^3} - \frac{6}{(1+x^2)^2}$$

Из последнего примера видно, что для вычисления высших производных возможно последовательное применение функции D.

В палитре `Basic Input` можно найти шаблоны для вычисления частных производных. Вот примеры их применения:

```
∂x,x,yExp[xy]
```

$$2e^{xy}y + e^{xy}xy^2$$

```
∂x,x,yLn[xy]
```

$$2y\text{Ln}''[xy] + xy^2\text{Ln}^{(3)}[xy]$$

Использование функции **Dt** для вычисления *обобщенных производных* демонстрируют примеры, приведенные ниже:

```
Dt[x^n,x]
```

$$x^n\left(\frac{n}{x}+\text{Dt[n, x] Log[x]}\right)$$

```
Dt[x*Sin[x],x]
 (x Cos[x]+Sin[x])
Dt[Exp[x/5],x]
```

$$\frac{1}{5}\ e^{x/5}$$

```
Dt[a*x^2+b*x+c,x]
 (b + 2ax + x^2Dt[a,x] + xDt[b,x] + Dt[c,x]
Dt[BesselJ[2,x],x]
```

$$\frac{1}{2}\ \text{(BesselJ[1, x] - BesselJ[3, x])}$$

Обратите внимание на то, что порой результаты для одного и того же дифференцируемого выражения у функций **D** и **Dt** заметно отличаются. Это вполне закономерно вытекает из различных определений данных функций.

4.3.7. Дифференцирование в системе Mathcad и в других СКА

В Mathcad для вычисления производных используются шаблоны – см., например, рис. 1.23. Derive и MuPAD имеют свои ядра и также вычисляют производные для многих выражений. Но в части представления производных через специальные математические функции они заметно уступают Maple и Mathematica.

В Derive для вычисления производной используется простое специальное окно. В этом окне надо задать дифференцируемую функцию, выбрать переменную, по которой надо вычислить производную, и указать порядок вычисляемой производной. Нажатие кнопки **Simplify** ведет к вычислению производной. Derive имеет также следующие функции для вычисления производных:

DIF (u,x) и **DIF (u,x,n)**.

Эти функции можно вводить в панели **Author**. Например, введя

```
DIF (SIN(a x^2),x),
```

получим на экране выражение

```
d          2
-- SIN (a x ),
dx
```

которое упрощается командой **Basic** в позиции **Simplify** главного меню к окончательному результату, представленному ниже:

```
                2
2 a x COS (a x )
```

В системе MuPAD функция дифференцирования обозначается как **diff**. Синтаксис ее следующий:

- **`diff(f(x),x[$n])`** – возвращает n-ую производную $f(x)$, при отсутствии указания $n (n=1,2,3,...) возвращает первую производную.

Система MuPAD имеет дифференциальный оператор D, представляемый в формах:

`D(expr)` или **`D([integer...],expr)`.**

Свой дифференциальный оператор есть в MuPAD и для полиномов:

`Dpoly(pol)` или **`Dpoly([integer...],pol)`.**

Наличие этих операторов означает претензию разработчиков системы MuPAD на ее близость к более продвинутым математическим системам, таким как Maple. Пока, однако, различие по возможностям этих систем весьма значительно.

4.4. Вычисление интегралов

4.4.1. Определение интегралов

Интегральное исчисление зародилось из практической необходимости вычисления площадей, объемов и центров тяжести различных фигур. Если есть некоторая функция $f(x)$, то *определенный интеграл* вида

$$\int_a^b f(x)dx$$

дает значение площади, ограниченной вертикалями a и b, именуемыми пределами интегрирования, кривой $f(x)$ и осью абсцисс X. Под площадью надо понимать ее алгебраическое значение, то есть разность между площадью над осью X и под ней. В этом случае ясно, что определенный интеграл может иметь как положительные, так и отрицательные значения.

Если $f(x)\mathrm{d}x$ есть дифференциал функции $F(x)$, то

$f(x)\mathrm{d}x=\mathrm{d}F(x)$.

Функцию $F(x)$ называют *первообразной* функции $f(x)$. Наиболее общий вид первообразной функции $f(x)$ называют *неопределенным интегралом* и обозначают как

$$\int f(x)dx.$$

В состав этого выражения включена некоторая *произвольная постоянная* C, подчеркивающая, что для одной и той же $f(x)$ существует масса первообразных, описываемых одной и той же линией, но смещенных по вертикали на произвольную постоянную. Например, для $f(x) = \sin(x)$ имеем

$$\int \sin(x)dx = -\sin(x)+C.$$

Соответственно определенный интеграл определяется как

$$\int_a^b f(x)dx = F(b)-F(a).$$

В некоторых системах символьной математики (в новых версиях Mathcad, например) возможно вычисление определенных интегралов несколькими методами (выбираются из меню). Кроме того, возможна оптимизация вычислений, которая заключается в том, что если для определенного интеграла удается вычислить первообразную, то вычисления ведутся по ней, то есть по аналитическому выражению.

4.4.2. Вычисление неопределенных интегралов в Maple

Для вычисления неопределенных и определенных интегралов Maple предоставляет следующие функции:

```
int(f,x);     int(f,x=a..b);      int(f,x=a..b,continuous);
Int(f,x);     Int(f,x=a..b);      Int(f,x=a..b,continuous);
```

Здесь `f` – подынтегральная функция, `x` – переменная, по которой выполняются вычисления, `a` и `b` – нижний и верхний пределы интегрирования, `continuous` – необязательное дополнительное условие.

Maple старается найти аналитическое значение интеграла с заданной подынтегральной функцией. Если это не удается (например, для «неберущихся» интегралов), то возвращается исходная запись интеграла. Ниже приведены примеры визуализации и вычисления неопределенных интегралов:

```
> Int(a*x^n,x)=int(a*x^n,x);
```

$$\int ax^n dx = \frac{ax^{(n+1)}}{n+1}$$

```
> Int(sin(x)/x,x)=int(sin(x)/x,x);
```

$$\int \frac{\sin(x)}{x} dx = \mathrm{Si}(x)$$

```
> Int(ln(x)^3,x);
```

$$\int \ln(x)^3 dx$$

```
> value(%);
```

$$\ln(x)^3 x - 3x\ln(x)^2 + 6x\ln(x) - 6x$$

```
> Int(1/x,x)=int(1/x,x);
```

$$\int \frac{1}{x} dx = \ln(x)$$

Обратите внимание, что в аналитическом представлении неопределенных интегралов опущена произвольная постоянная *C*. Возможно вычисление сумм интегралов и интегралов сумм, а также интегралов от полиномов:

```
> Sum(Int(x^i,x),i=1..5);
```

$$\sum_{i=1}^{5} \int x^i dx$$

```
> value(%);
```

$$\frac{1}{2}x^2+\frac{1}{3}x^3+\frac{1}{4}x^4+\frac{1}{5}x^5+\frac{1}{6}x^6$$

```
> Int(Sum(x^i,i=1..5),x);
```

$$\int\sum_{i=1}^{5}x^i\,dx$$

```
> value(%);
```

$$\frac{1}{2}x^2+\frac{1}{3}x^3+\frac{1}{4}x^4+\frac{1}{5}x^5+\frac{1}{6}x^6$$

```
> P(x):=a*x^3+b*x^2+c*x+d;
```

$$P(x) := ax^3 + bx^2 + cx + d$$

```
> int(P(x),x);
```

$$\frac{ax^4}{4}+\frac{bx^3}{3}+\frac{cx^2}{2}+dx$$

Maple успешно берет большинство справочных интегралов. Но следует помнить, что вполне может найтись интеграл, который окажется «не по зубам» и Maple.

4.4.3. Конвертирование и преобразование интегралов

В некоторых случаях Maple не может вычислить интеграл. Тогда он просто повторяет его. С помощью функций `taylor` и `convert` можно попытаться получить аналитическое решение в виде полинома умеренной степени, что демонстрирует следующий характерный пример:

```
> int(exp(sin(x)),x);
```

$$\int e^{\sin(x)}\,dx$$

```
> convert(taylor(%,x=0,8),polynom);
```

$$x+\frac{1}{2}x^2+\frac{1}{6}x^3-\frac{1}{40}x^5-\frac{1}{1680}x^7+\frac{1}{720}x^8$$

Естественно, что в этом случае решение является приближенным, но оно все же есть, и с ним можно работать, например можно построить график функции, представляющей данный интеграл.

4.4.4. Вычисление определенных интегралов в Maple

Для вычисления определенных интегралов используются те же функции `int` и `Int`, в которых надо указать пределы интегрирования, например `x=a...b`, если интегрируется функция переменной x. Это поясняется приведенными ниже примерами:

```
> Int(sin(x)/x,x=a..b)=int(sin(x)/x,x=a..b);
```

$$\int_a^b \frac{\sin(x)}{x}dx = \mathrm{Si}(b) - \mathrm{Si}(a)$$

```
> Int(sin(x)/x,x=0..1.)=int(sin(x)/x,x=0..1.);
```

$$\int_0^{1.} \frac{\sin(x)}{x}dx = .9460830704$$

```
> Int(x*ln(x),x=0..1)=int(x*ln(x),x=0..1);
```

$$\int_0^1 x\ln(x)dx = \frac{-1}{4}$$

```
> Int(x*exp(-x),x=0..infinity)=int(x*exp(-x), x=0..infinity);
```

$$\int_0^{\infty} x\,\mathbf{e}^{(-x)}dx = 1$$

```
> Int(1/(x^2+6*x+12),x=-infinity..infinity);
```

$$\int_{-\infty}^{\infty} \frac{1}{x^2+6x+12}dx$$

```
> value(%);
```

$$\frac{1}{3}\pi\sqrt{3}$$

Как видно из этих примеров, среди значений пределов может быть бесконечность, обозначаемая как `infinity`.

4.4.5. Каверзные интегралы и визуализация результатов интегрирования

Иногда приходится сталкиваться с примерами вычисления «каверзных» интегралов. Рассмотрим следующий интеграл:

```
> Int(x^20*exp(-x),x=0..1)=int(x^20*exp(-x),x=0..1);
```

$$\int_0^1 x^{20}\mathbf{e}^{(-x)}dx = -6613313319248080001\,\mathbf{e}^{(-1)} + 2432902008176640000$$

В полученном выражении фигурируют большие числа, и потому для правильного приближенного решения (в виде вещественного числа в научной нотации) нужно заведомо использовать аппарат точной арифметики и ни в коем случае не доверяться на погрешность, заданную по умолчанию. Например, система Maple 6 при задании погрешности по умолчанию вычисляла значение этого интеграла также как 0, тогда как Maple 9/9.5/10/11 «поумнела» уже настолько, что дает значение 0.01835046770.

Maple позволяет наглядно проиллюстрировать характер промежуточных вычислений подобных интегралов:

```
> int(x^20*exp(-x),x);
```

$$-x^{20}\mathbf{e}^{(-x)} - 20x^{19}\mathbf{e}^{(-x)} - 380x^{18}\mathbf{e}^{(-x)} - 6840x^{17}\mathbf{e}^{(-x)} - 116280x^{16}\mathbf{e}^{(-x)}$$
$$- 1860480x^{15}\mathbf{e}^{(-x)} - 27907200x^{14}\mathbf{e}^{(-x)} - 390700800x^{13}\mathbf{e}^{(-x)}$$
$$- 5079110400x^{12}\mathbf{e}^{(-x)} - 60949324800x^{11}\mathbf{e}^{(-x)}$$
$$- 670442572800x^{10}\mathbf{e}^{(-x)} - 6704425728000x^{9}\mathbf{e}^{(-x)}$$
$$- 60339831552000x^{8}\mathbf{e}^{(-x)} - 482718652416000x^{7}\mathbf{e}^{(-x)}$$
$$- 3379030566912000x^{6}\mathbf{e}^{(-x)} - 20274183401472000x^{5}\mathbf{e}^{(-x)}$$
$$- 101370917007360000x^{4}\mathbf{e}^{(-x)} - 405483668029440000x^{3}\mathbf{e}^{(-x)}$$
$$- 1216451004088320000x^{2}\mathbf{e}^{(-x)} - 2432902008176640000x\mathbf{e}^{(-x)}$$
$$- 2432902008176640000\mathbf{e}^{(-x)}$$

Нетрудно заметить, что решение распадается на множество слагаемых, соответствующих общеизвестному *интегрированию по частям*. В каждом слагаемом имеются большие числа, и потому *принципиально необходимо* применение *арифметики высокой точности* (или разрядности). Maple 9/9.5/10, как СКА, такими средствами обладает.

Продолжим изучение данного «каверзного» интеграла. Опробуем силы Maple на интеграле более общего вида, где конкретный показатель степени заменен на обобщенный – n. Здесь нас ожидает приятный сюрприз – Maple с легкостью выдает аналитическое решение для данного определенного интеграла:

```
> y:=(n)->int(x^n*exp(-x),x=0..1);
```

$$y := n \to \int_0^1 x^n \mathbf{e}^{(-x)} dx$$

```
> y(n);
```

$$\frac{\mathbf{e}^{(-1/2)}\,\mathrm{WhittakerM}\left(\frac{1}{2}n, \frac{1}{2}n+\frac{1}{2}, 1\right)}{n+1}$$

```
> y(20);
```

$$-661331331924808001\mathbf{e}^{(-1)} + 2432902008176640000$$

```
> evalf(%,30);
```

$$.01835046770$$

```
> y(20.);
```

$$0.$$

Однако радоваться несколько преждевременно. Многие ли знают, что это за специальная функция – `WhittakerM`? Но хуже другое – Maple при конкретном $n = 20$ дает грубо неверное решение – 0 (почему – уже объяснялось). Забавно, что при этом сама по себе функция `WhittakerM` вычисляется для $n = 20$ без проблем:

```
> WhittakerM(10,10.5,1);
```

$$.6353509348$$

А теперь присмотритесь к новому результату вычисления злополучного интеграла. Оказывается, он уже не содержит больших чисел, свойственных прямому решению! Зная значение `WhittakerM` с погрешностью по умолчанию, можно уверенно вычислить приближенное численное значение интеграла с той же погрешностью, уже не прибегая к арифметике высокой точности:

```
> (exp(-.5)*WhittakerM(10,10.5,1))/21;
```

.01835046770

Как торжество Maple приведем график зависимости значений данного интеграла от показателя степени n при его изменении от 0 до 50 – рис. 4.3. Плавный ход графика показывает, что в вычислении данного интеграла нет никаких признаков неустойчивости решения при изменении n, если соблюдать правило выбора достаточно малой погрешности вычислений.

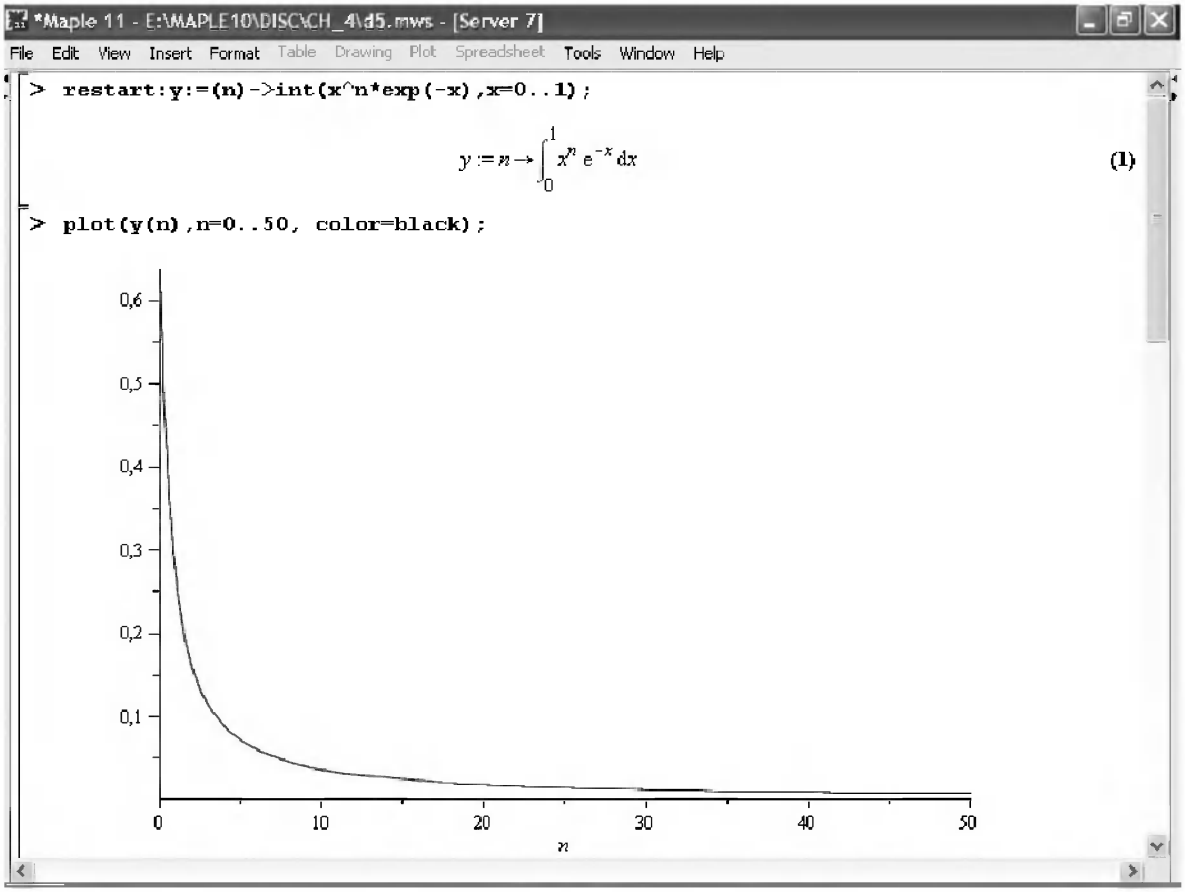


*Рис. 4.3. Значение интеграла от x^n*exp(-x) как функция n*

Наличие у функции особых (сингулярных) точек нередко затрудняет выполнение с ней ряда операций, таких как численное интегрирование. В этом случае могут помочь соответствующие параметры. Например, вычисление в Maple 8/9 следующего интеграла дает явно неудобное выражение в виде набора значений, разных для разных интервалов изменения a:

```
> int(1/(x+a)^2,x=0..2);
```

$$\frac{2+2\left(\begin{cases}0 & a\le -2\\ \infty & a<0\\ 0 & 0\le a\end{cases}\right)a+\left(\begin{cases}0 & a\le -2\\ \infty & a<0\\ 0 & 0\le a\end{cases}\right)a^2}{(2+a)a}$$

Этот интеграл расходится, поскольку при $x = -a$ подынтегральная функция устремляется в бесконечность, что и показывает приведенное выражение. График зависимости значения интеграла от параметра a имеет подозрительный вид.

Это как раз тот случай, когда надо обратить особое внимание на результаты, полученные системой Maple. А теперь покажем, как выглядит этот пример при его решении в системе Maple 9.5, – рис. 4.4. Обратите внимание на «провал» графика в средней части.

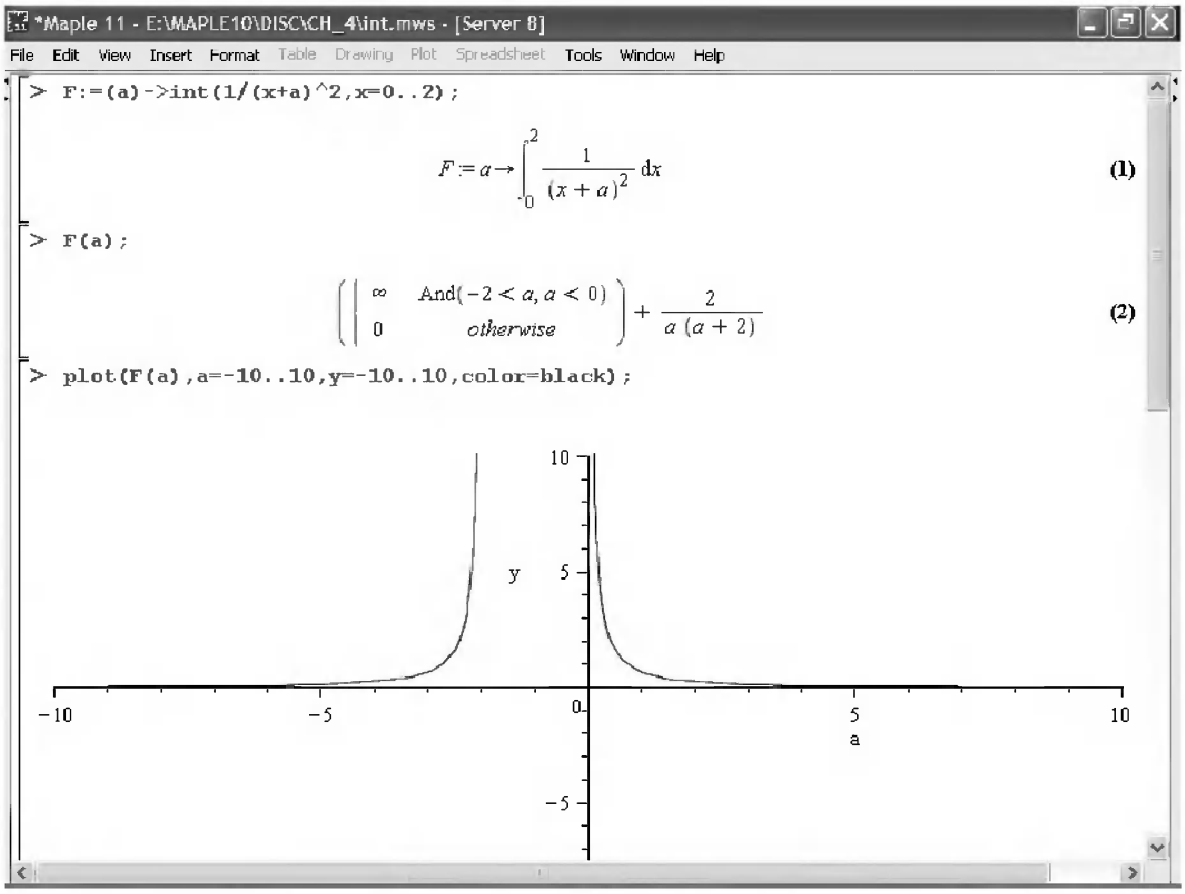


Рис. 4.4. Построение графика зависимости значений интеграла с подынтегральной функцией 1/(x+a)^2 от параметра a

Интересно, что если в нашем случае применить параметр `continuous` (в апострофах) при вычислении интеграла, можно получить более простое выражение:

```
> int(1/(x+a)^2,x=0..2,'continuous');
```

$$\frac{2}{(2+a)a}$$

Рисунок 4.5 показывает это решение с двумя важными дополнениями – оно представляется функцией пользователя, а ее график строится при изменении a от –10 до 10. «Провал» в средней части графика уже отсутствует.

Приведем еще один пример «каверзного» интеграла довольно простого вида:

```
> int(1/x^3,x=-1..2);
```

undefined

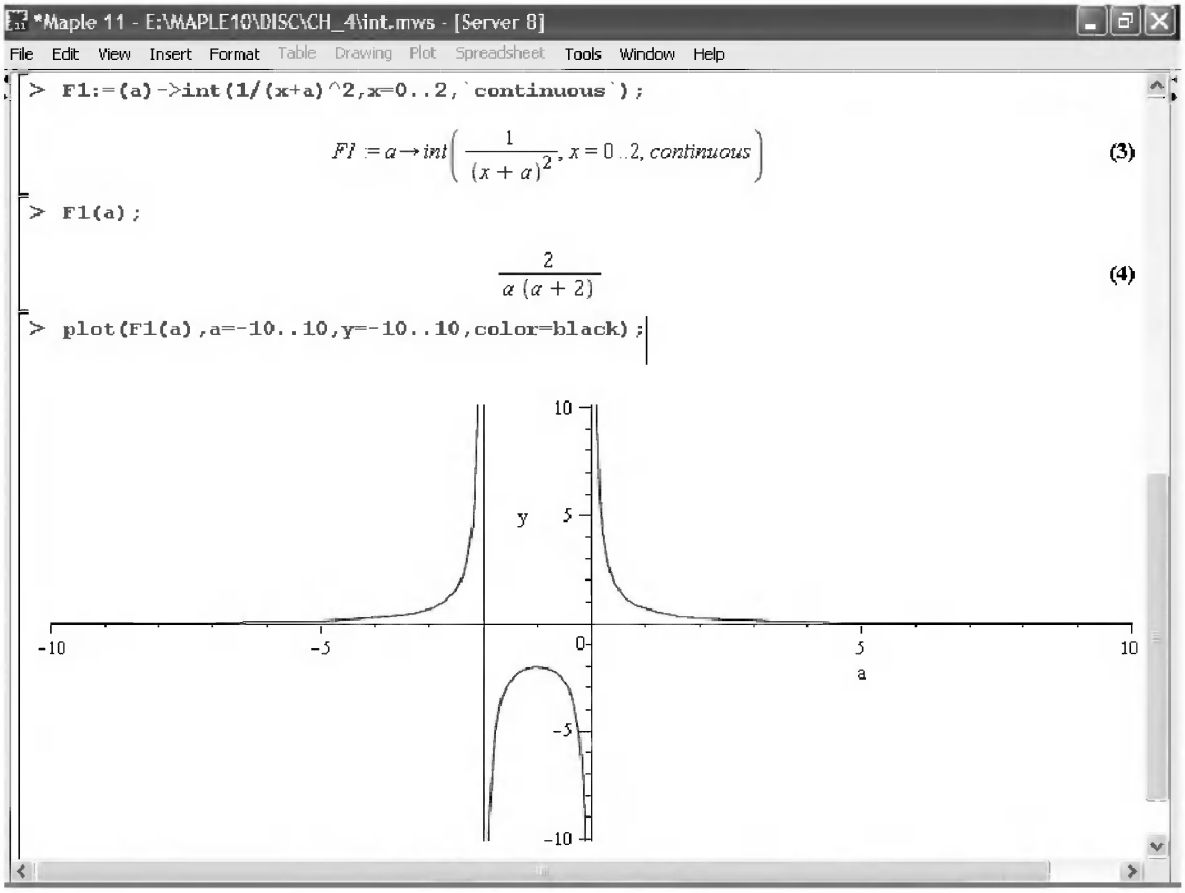


Рис. 4.5. Зависимость значения интеграла с подынтегральной функцией 1/(x+a)^2 и пределами от 0 до 2 от параметра a

Этот интеграл не берется вообще, так что Maple 9 совершенно справедливо об этом и сообщает. Но введение параметра `CauchyPrincipalValue` позволяет получить численное значение интеграла:

```
> int(1/x^3,x=-1..2,'CauchyPrincipalValue');
```

$$\frac{3}{8}$$

Возьмем еще один наглядный пример – вычисление интеграла от синусоидальной функции при произвольно больших пределах, но кратных 2π! Очевидно, что при этом (учитывая равность площадей положительной и отрицательной полуволн синусоиды) значение интеграла будет равно 0. Например:

```
> int(sin(x),x=-1000*pi..1000*pi);
```

$$0$$

Однако распространение этого правила на бесконечные пределы интегрирования является грубейшей ошибкой. Интеграл такого рода уже *не сходится*, и Maple дает соответствующий результат:

```
> int(sin(x),x=-infinity..infinity);
```

undefined

Возьмем, к примеру, широко распространенную функцию $y(t) = \exp(-t)\sin(2\pi t)$, которую неточно именуют «затухающей синусоидой». Неточно потому, что $y(t)$

уже синусоидой не является. Построим ее график и вычислим определенный интеграл от этой функции с пределами от 0 до ∞ (рис. 4.6).

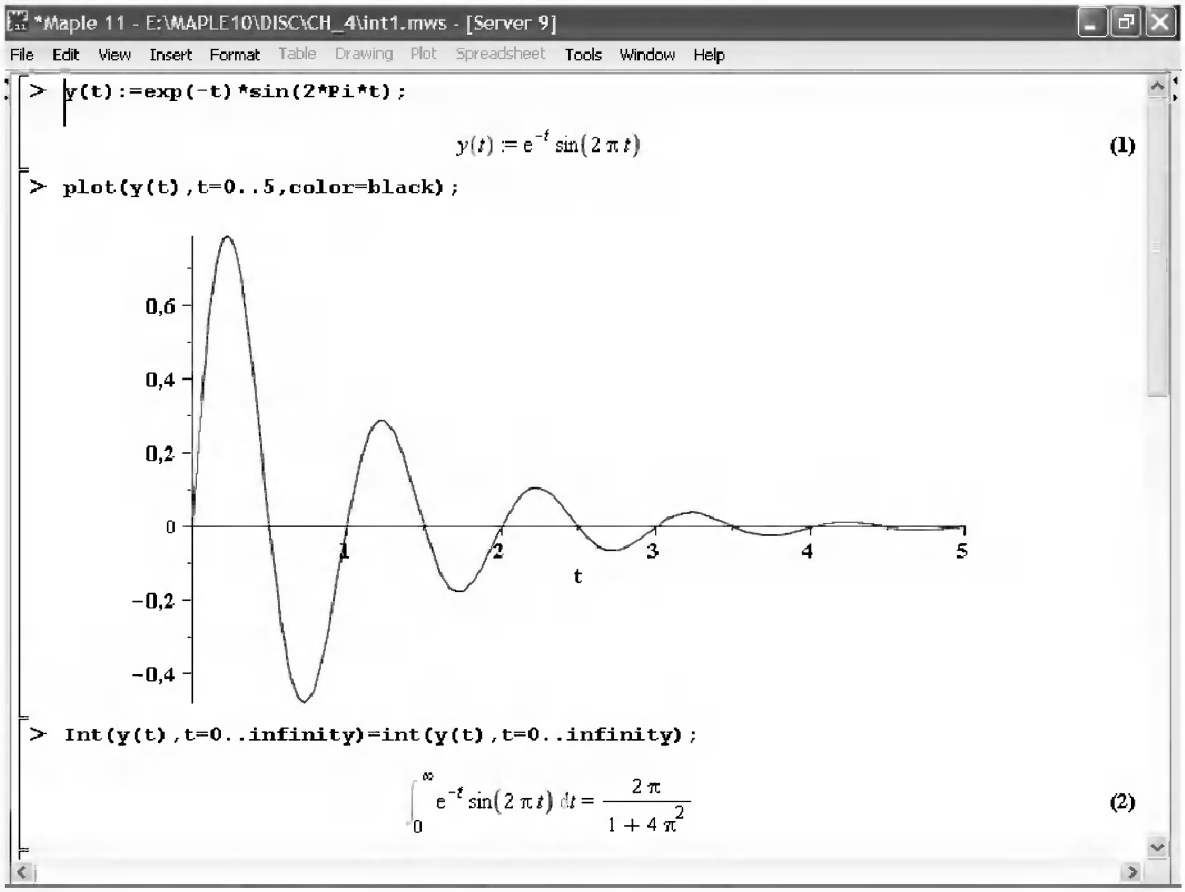


Рис. 4.6. График «затухающей синусоиды» и интеграл от нее с пределами от 0 до ∞

С первого взгляда на график видно, что каждая положительная полуволна функции явно больше последующей отрицательной полуволны. А осцилляции функции быстро затухают, и через десяток-другой периодов значение функции становится исчезающе малым. Вот почему Maple уверенно вычисляет интеграл с такой подынтегральной функцией. Говорят, что этот интеграл сходится!

Теперь возьмем антипод этой функции – «синусоиду с экспоненциально нарастающей до стационарного значения 1 амплитудой»:

$Y(t) = (1 - \exp(-t))\sin(2\pi t)$.

Ее график и попытки вычисления интеграла с такой подынтегральной функцией приведены на рис. 4.7.

Обратите внимание на то, что здесь прямое вычисление интеграла к успеху не привело, хотя из графика функции видно, что каждая положительная полуволна в близкой к $t = 0$ области явно больше по амплитуде, чем последующая отрицательная полуволна. Однако, в отличие от предыдущей функции, при больших значениях аргумента данная функция вырождается в обычную синусоиду с неизменной (и равной 1) амплитудой. Вот почему Maple честно отказывается вычислять несходящийся интеграл от такой функции.

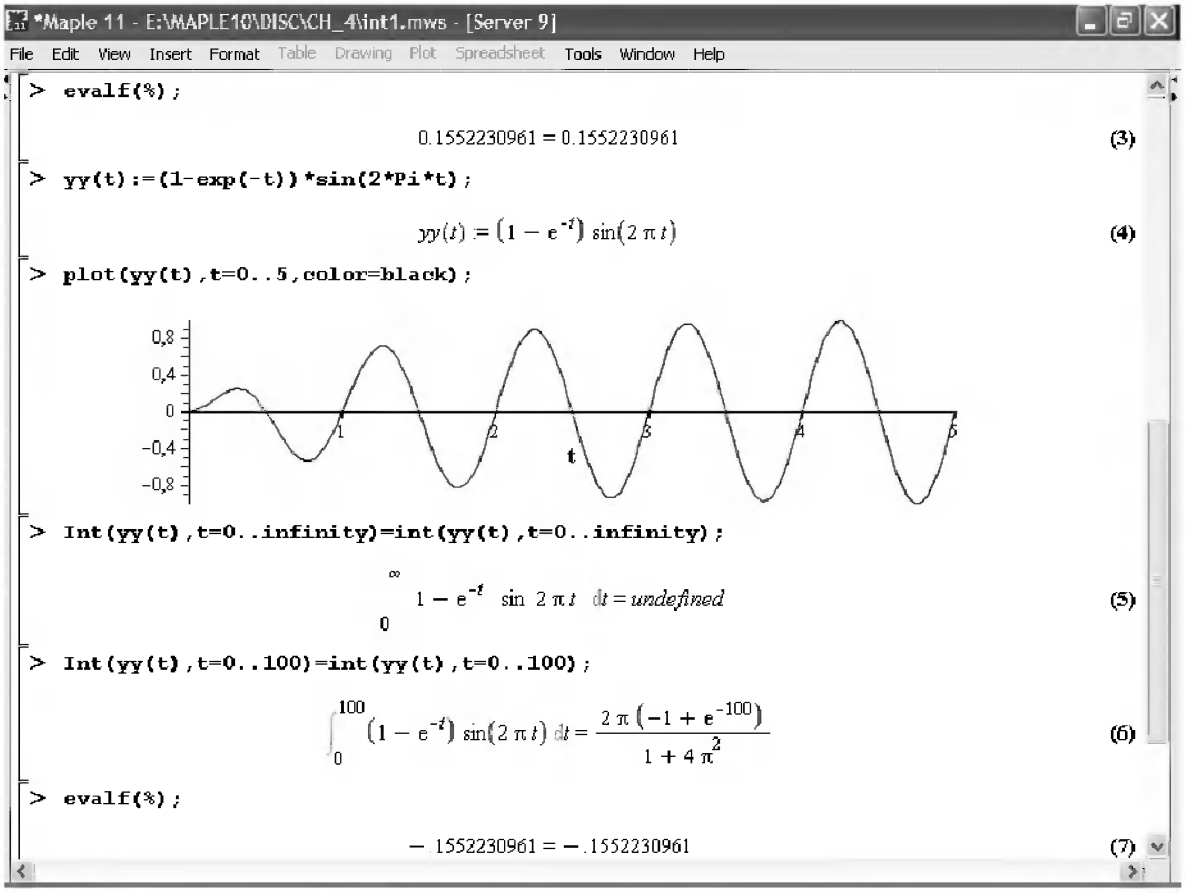


Рис. 4.7. График «экспоненциально нарастающей синусоиды» и интеграл от нее с пределами от 0 до ∞

4.4.6. Вычисление несобственных интегралов первого рода

Несобственными интегралами называют интегралы, у которых хотя бы один из пределов или подынтегральная функция устремляются в бесконечность.

Соответственно различают несобственные интегралы первого и второго родов.

Новые версии Maple существенно продвинулись в направлении решения многих несобственных интегралов, которые в старых версиях не брались. Это видно из благополучного решения ряда таких несобственных интегралов первого рода:

```
> Int(sin(x)/x^2,x=1..infinity);
```

$$\int_1^{\infty} \frac{\sin(x)}{x^2}\,dx$$

```
> value(%);evalf(%);
```

$$\sin(1) - \mathrm{Ci}(1)$$
$$0.5040670619$$

```
> Int(sin(x)^2,x=0..infinity);
```

$$\int_0^{\infty} \sin(x)^2\,dx$$

```
> value(%);
```

$$\infty$$

```
> Int( exp(-t^2)*t^2, t=-infinity..infinity );
```

$$\int_{-\infty}^{\infty} \mathbf{e}^{(-t^2)} t^2 dt$$

```
> value(%);
```

$$\frac{\sqrt{\pi}}{2}$$

```
> Int(exp(-t)/t^(1/3), t=0..infinity);
```

$$\int_{0}^{\infty} \frac{\mathbf{e}^{(-t)}}{t^{(1/3)}} dt$$

```
> value(%);
```

$$\Gamma\left(\frac{2}{3}\right)$$

```
> Int(exp(-t)*ln(t)/t^2,t=1..infinity);
```

$$\int_{1}^{\infty} \frac{\mathbf{e}^{(-t)} \ln(t)}{t^2} dt$$

```
> value(%);
```

$$\frac{1}{2} - \frac{\pi}{12} - \frac{(1-\gamma)^2}{2} + \frac{1}{2}\,\mathrm{hypergeom}([1,1,1],[2,2,3],-1)$$

```
> evalf(%);
```

$$0.0506523094$$

Для подавляющего большинства интегралов результат вычислений с применением функций `Int` и `int` оказывается абсолютно идентичным. Однако есть и исключения из этого правила. Например, следующий интеграл благополучно очень быстро вычисляется функцией `Int` с последующей `evalf`:

```
> Int(cos(x)/(x^4+x+1),x=-infinity..infinity);
```

$$\int_{-\infty}^{\infty} \frac{\cos(x)}{x^4 + x + 1} dx$$

```
> evalf(%);
```

$$1.878983562$$

Однако в Maple 9 функция `int` вместо числа возвращает «страшное» выражение. Увы, но функция `evalf(%)`, примененная после него, к более простому выражению не приводит. А Maple 9.5 при вычислении этого интеграла просто «зависла» и спустя минуту так и не выдала результата. Просмотрите график подынтегральной функции, и эти «фокусы» станут вам понятны. Вам должно быть ясно, что в исключительных случаях результаты интегрирования могут оказаться разными в различных реализациях даже СКМ одного класса.

4.4.7. Вычисление несобственных интегралов второго рода

К несобственным интегралам второго рода относятся интегралы, имеющие в пределах интегрирования особенности подынтегральной функции. При этом сами пределы могут быть и конечными. Некоторые интегралы не имеют в среде Maple 9.5 общего решения, но исправно вычисляются для частных случаев (см. ниже для n неопределенного и конкретного n=6):

```
> Int(1/sqrt(1-x^n),x=0..1);
```

$$\int_0^1 \frac{1}{\sqrt{1-x^n}}\,dx$$

```
> value(%);
  Definite integration: Can't determine if the integral is
  convergent.
  Need to know the sign of -> n
  Will now try indefinite integration and then take limits.
```

$$\frac{\sqrt{\pi}\ \left(\frac{1}{n}\right)}{n\ \left(\frac{1}{2}+\frac{1}{n}\right)}$$

```
> Int(1/sqrt(1-x^6),x=0..1)=evalf(int(1/sqrt(1-x^6), x=0..1));
```

$$\int_0^1 \frac{1}{\sqrt{1-x^6}}\,dx = 1.214325324$$

Приведем тройку примеров, требующих для обычных вычислений заметных усилий, но прекрасно выполняемых системой Maple:

```
> Int((x-1)/ln(x),x=0..1)=int((x-1)/ln(x),x=0..1);
```

$$\int_0^1 \frac{x-1}{\ln(x)}\,dx = \ln(2)$$

```
> Int(ln(1-x)/x,x=0..1)=int(ln(1-x)/x,x=0..1);
```

$$\int_0^1 \frac{\ln(1-x)}{x}\,dx = -\frac{\pi}{6}$$

```
> Int(exp(-x)*sin(x)/x,x=0..infinity)=int(exp(-x)*sin(x)/x,
x=0..infinity);
```

$$\int_0^\infty \frac{e^{(-x)}\sin(x)}{x}\,dx = \frac{\pi}{4}$$

Однако не стоит думать, что так бывает всегда. Следующий интеграл дает иногда весьма подозрительный результат:

```
> Int(1/(x^2*(sqrt(x^2-9))),x=0..infinity);
```

$$\int_0^{\tilde{\infty}} \frac{1}{x^2\sqrt{x^2-9}}\,dx$$

```
> value(%);
```

$$-\infty I$$

Это наглядный пример, когда Maple 9.5 ошибается, несмотря на заверения его создателей о том, что эта система прошла полную сертификацию на вычисления интегралов. Выполнив тригонометрическую подстановку t=3*/cos(q) и проведя некоторые преобразования, найдем интеграл в системе Maple 8:

```
> Int(1/(t^2*(sqrt(t^2-9))),t=3..x)=int(1/(t^2*(sqrt(t^2-9))),
t=3..x);
```

$$\int_3^x \frac{1}{t^2\sqrt{t^2-9}}\,dt = \frac{\sqrt{x^2-9}}{9x}$$

Увы, Maple 9.5/10/11 вычислять данный интеграл не желает – он его просто повторяет. Но и в Maple 8, и в Maple 9.5/10/11 нужное значение определяется пределом этого выражения при x, стремящемся к бесконечности:

```
> Int(1/(x^2*(sqrt(x^2-9))),x=0..infinity)=
value(Limit(rhs(%),x=infinity));
```

$$\int_0^{\tilde{\infty}} \frac{1}{x^2\sqrt{x^2-9}}\,dx = \frac{1}{9}$$

Этот пример наглядно показывает, что иногда полезны аналитические преобразования, выполняемые пользователем.

4.4.8. Интегралы с переменными пределами интегрирования

Встречается ряд специальных видов интегралов. Один из них – интеграл с переменным верхним пределом, представленный в виде:

$$\int_a^{y(x)} f(x)dx.$$

В данном случае верхний предел представлен функцией $y(x)$. В принципе, переменными могут быть оба предела. Интеграл с переменными пределами в общем случае есть функция переменной x – см. рис. 4.8.

На этом рисунке построены также графики подынтегральной функции (это наклонная прямая) и функции, которую задает интеграл.

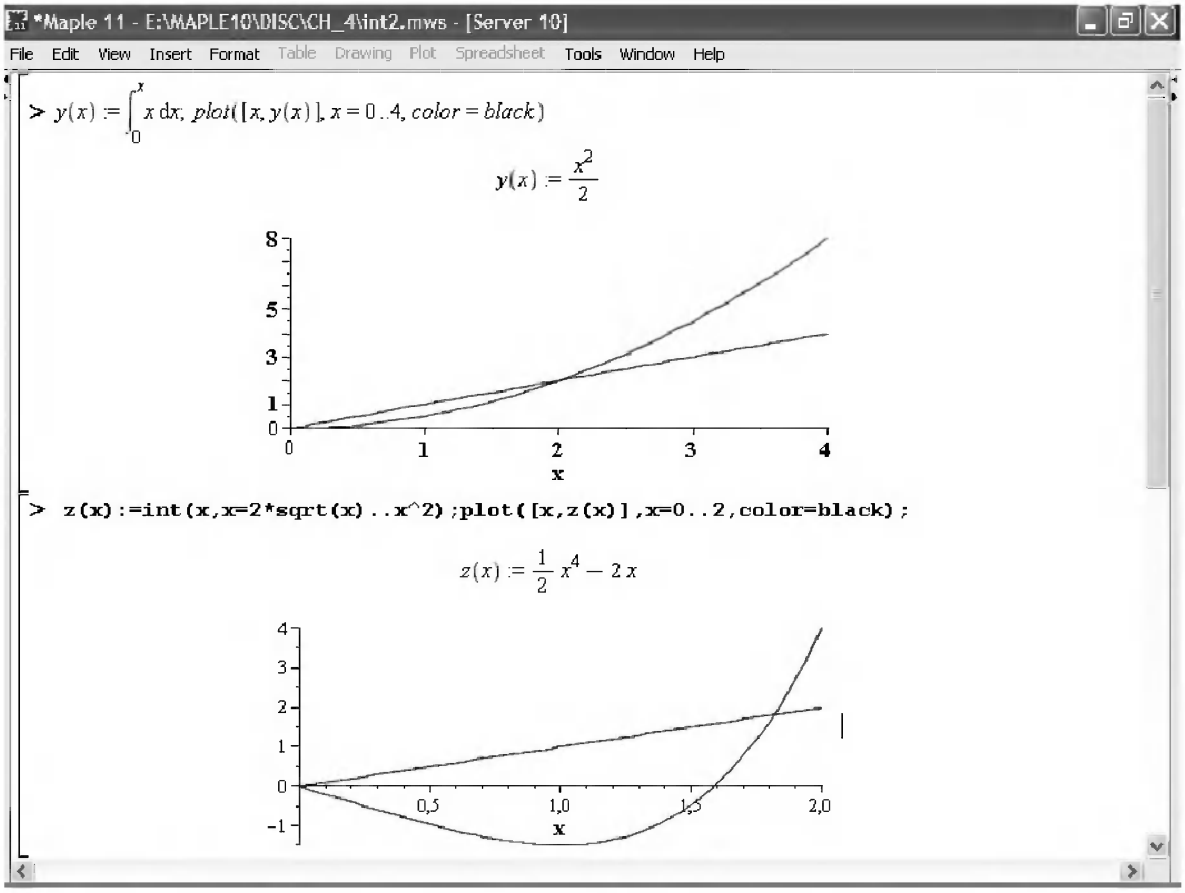


Рис. 4.8. Примеры интегралов с переменными пределами интегрирования

4.4.9. Вычисление кратных интегралов

Функции `int` и `Int` могут использоваться для вычисления *кратных интегралов*, например двойных и тройных. Для этого функции записываются многократно:

```
> restart;
> Int(Int(Int((x^2+y^2)*z, x=0..a), y=0..a), z=0..a);
```

$$\int_0^a \int_0^a \int_0^a (x^2+y^2)z\,dx\,dy\,dz$$

```
> value(%);
```

$$\frac{a^6}{3}$$

```
> Int(Int(2-x-y, x=sqrt(y)..y^2), y=0..1);
```

$$\int_0^1 \int_{\sqrt{y}}^{y^2} 2-x-y\,dx\,dy$$

```
> value(%);
```

$$\frac{-11}{30}$$

```
> I1:=
```

$$\int_0^{\frac{\pi}{2}} \int_0^{\frac{\pi}{4}} \int_0^{4\cos(w)} p^2 \sin(w)\,dp\,dw\,dq$$

$$I1 := -\frac{8}{3}\cos\left(\frac{1}{4}\pi\right)^4\pi + \frac{8}{3}\pi$$

```
> evalf(I1);
```

$$-2.666666667\cos(.2500000000\pi)^4\pi + 2.66666666\pi$$

Обратите внимание на нечеткую работу функции `evalf` в последнем примере. Эта функция уверенно выдает значение `evalf(Pi)` в форме вещественного числа с плавающей точкой, но отказывается вычислить значение интеграла, в которое входит число `Pi`. Данный пример говорит о том, что отдельные недостатки у Maple все же есть, как и поводы для ее дальнейшего совершенствования. Вот только вряд ли на этом стоит «зацикливаться».

4.4.10. О вычислении некоторых других интегралов

Maple открывает большие возможности в вычислении криволинейных, поверхностных и объемных интегралов. Нередко такие интегралы довольно просто заменяются на интегралы с переменными пределами интегрирования, что и иллюстрируют приведенные ниже примеры.

Пусть требуется вычислить объем фигуры, ограниченной координатными плоскостями и плоскостью $x + y + z = 1$. Он, с учетом равенства $z = 1 - x - y$, задается интегралом

$$V = \iint_R (1 - x - y)d\mathrm{A},$$

который заменяется следующим интегралом:

```
> Int(Int(1-x-y,y=0..1-x),x=0..1)=int(int(1-x-y,y=0..1-x),x=0..1);
```

$$\int_0^1\int_0^{1-x} 1 - x - y\,dy\,dx = \frac{1}{6}$$

Последний, как видно, легко вычисляется.

Теперь вычислим массу указанной фигуры, которая задается тройным интегралом:

$$m = \iiint_V k \cdot x \cdot y \cdot z\,dV.$$

Здесь k – константа, характеризующая удельную площадь вещества. Этот интеграл также сводится к легко решаемому в Maple 9/9.5:

```
> m=Int(Int(Int(k*x*y*z,z=0..1-x-y),y=0..1-x),x=0..1);
```

$$m = \int_0^1\int_0^{1-x}\int_0^{1-x-y} k\,x\,y\,z\,dz\,dy\,dx$$

```
> value(%);
```

$$m = \frac{k}{720}$$

Специальные средства для вычисления подобных интегралов имеет пакет расширения VectorCalculus, который описывается в конце этой главы.

4.4.11. Maplet-демонстрация построения графика первообразной

В составе самоучителей Maple 9.5 есть раздел Antiderivative, который иллюстрирует технику построения первообразной при интегрировании. Для доступа к окну этого инструмента (рис. 4.9) достаточно исполнить команду **Tools ⇒ Tutors ⇒ Calculus-Single Variables ⇒ Antiderivative...**.

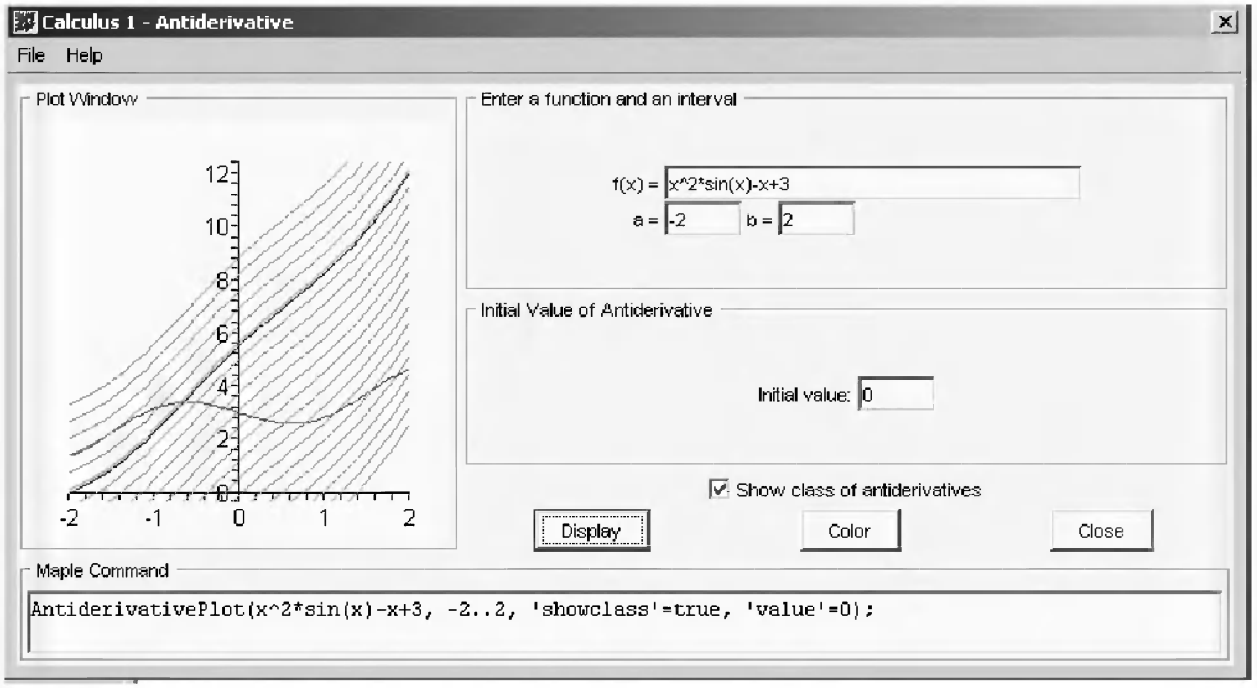


Рис. 4.9. Окно Maplet-демонстрации графиков функций и их первообразных

Окно Maplet-демонстрации интегрирования позволяет задать подынтегральную функцию и построить ее график и график первообразной, представляющей неопределенный интеграл. В окне *a* и *b* – это не пределы интегрирования, а пределы изменения *x* при построении графиков. Опция Show class of antiderivatives позволяет построить графики множества первообразных с выделением графика первообразной функции для заданного начального значения Initial Value. По завершении работы с окном демонстрации графики выводятся в документ Maple 9.5 (рис. 4.10). Обратите внимание на вывод целого семейства первообразных.

4.4.12. Maplet-демонстрация методов интегрирования

Для демонстрации методов пошагового интегрирования имеется Maplet-инструмент Step-by-step Integration Tutor. Для вызова его окна (рис. 4.11) нужно исполнить команду (в стандартном варианте интерфейса): **Tools ⇒ Tutors ⇒ Calculus-Single Variables ⇒ Antiderivative...**.

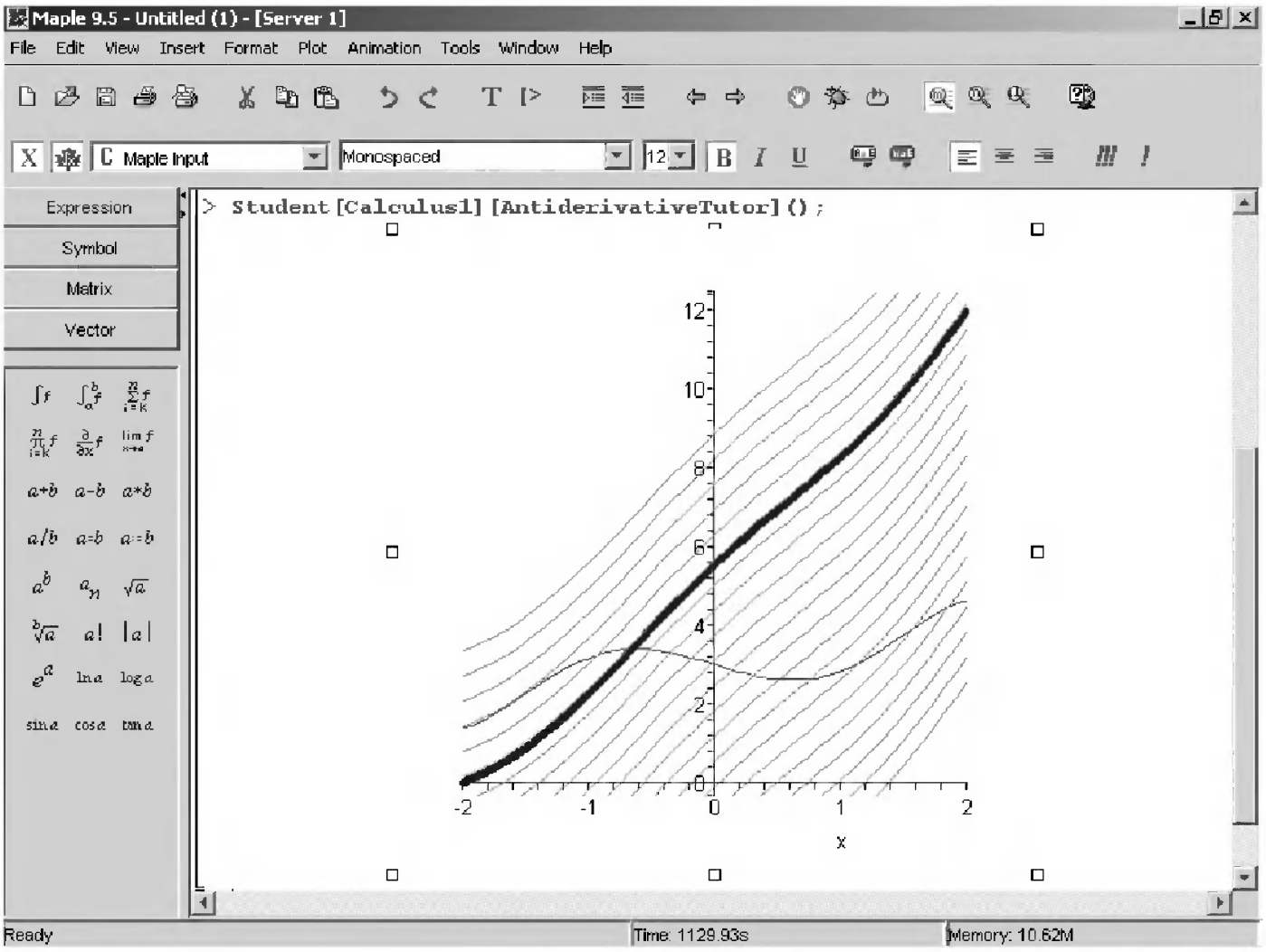


Рис. 4.10. Графики исходной функции и первообразных в окне документа Maple 9.5

Нетрудно заметить, что это окно практически аналогично окну для демонстрации методов пошагового дифференцирования, описанному в разделе 4.3.4 (рис. 4.2). Maplet позволяет задавать подынтегральную функцию и пределы интегрирования и по шагам (автоматически или вручную) вычислять интегралы (рис. 4.12).

4.4.13. Интегрирование в Mathematica и в других СКМ

Для интегрирования в системе Mathematica используются следующие функции:

- **Integrate[f, x]** – возвращает первообразную (неопределенный интеграл) подынтегральной функции f по переменной x.
- **Integrate[f,{x, xmin, xmax}]** – возвращает значение определенного интеграла с пределами a=xmin до b=xmax.
- **Integrate[f,{x, xmin, xmax},{y, ymin, ymax},...]** – возвращает значение кратного интеграла с пределами от xmin до xmax по переменной x, от ymin до ymax по переменной y и т. д. (кратность реально не ограничена).

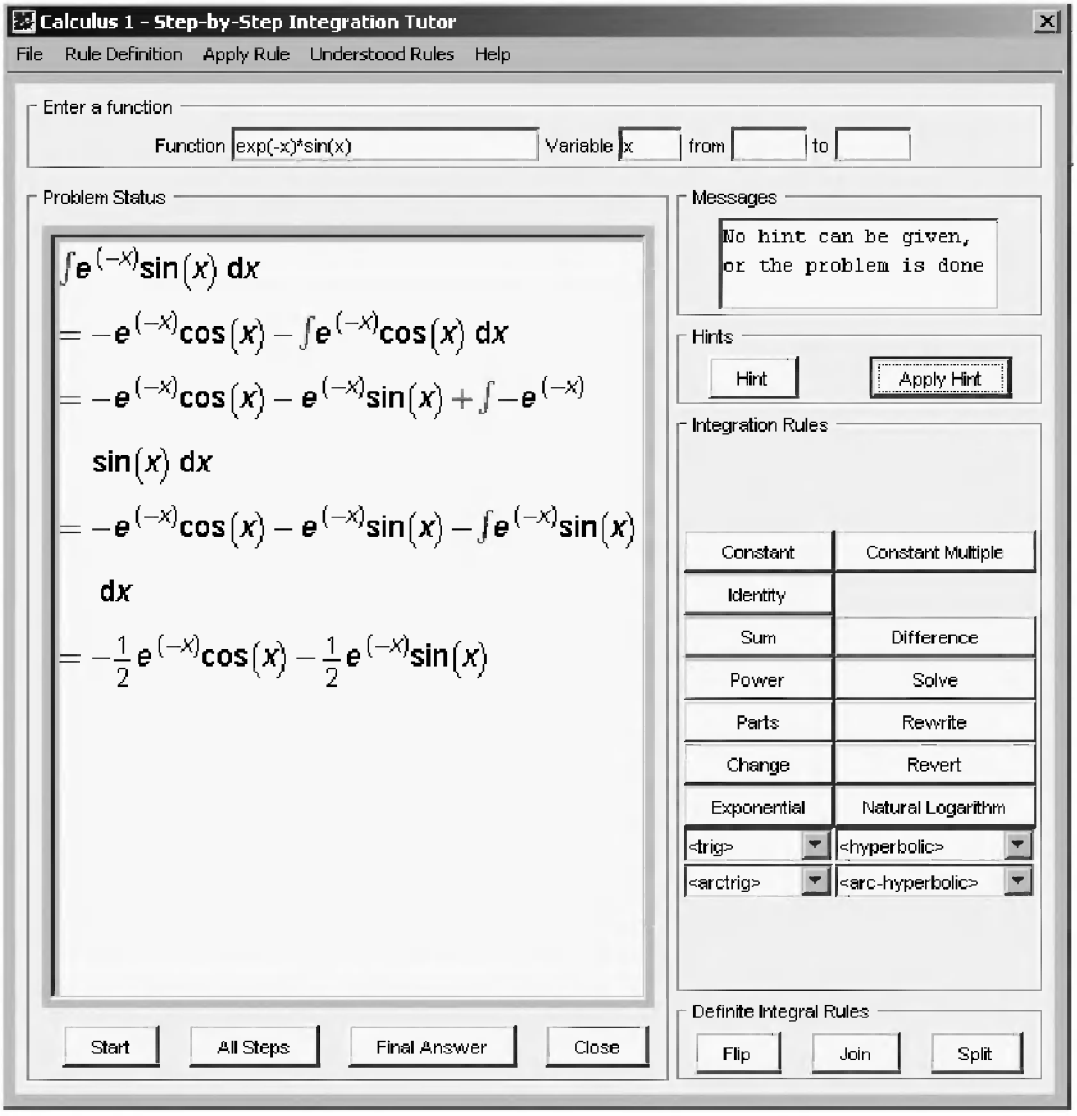


Рис. 4.11. Окно Maplet-демонстрации методов пошагового интегрирования

```
> Student[Calculus1][IntTutor]();
```

$$\int e^{(-x)} \sin(x)\, dx = -\frac{1}{2} e^{(-x)} \cos(x) - \frac{1}{2} e^{(-x)} \sin(x)$$

Рис. 4.12. Пример вывода результата работы с Maplet-инструментом по методам интегрирования

Обычно функция **Integrate** применяется прямо, но она имеет три характерные опции:

```
Options[Integrate]
{Assumptions¦$Assumptions,GenerateConditions→Automatic,
PrincipalValue→False}
```

Для обозначения бесконечных пределов используется константа **Infinity**. Пределы могут задаваться как константами, так и функциями. Ввод возможен в виде функции или оператора интегрирования с панели Basic Input.

Особый интерес, естественно, вызывает применение функции **Integrate** для вычисления в символьном виде заданных пользователем неопределенных интегралов. Для подавляющего большинства интегралов результаты, получаемые при их вычислении в системе Maple и Mathematica, оказываются абсолютно идентичными. Но вычисление «каверзных» интегралов порой оказывается неожиданным. Приведем несколько примеров:

```
In[1]:= Integrate[Sin[x]/x, {x, 0, Infinity}]
```

$$\frac{\pi}{2}$$

```
In[2]:= Integrate[Sin[x]/x^2, {x, 0, Infinity}]
```

```
Integrate::idiv : Integral of Sin[x]/x^2 does not converge on {0, ∞}. More...
```

```
In[3]:= Integrate[(E^(-t^2))*Sin[t^2], {t, 0, Infinity}]
```

$$\frac{\sqrt{\pi}\,\mathrm{Sin}\left[\frac{\pi}{8}\right]}{2\,2^{1/4}}$$

```
In[4]:= N[%]
```

```
Out[4]= 0.285185
```

Тут во втором примере Mathematica 5 отказалась вычислять интеграл, который Maple вычисляет. Зато в третьем примере Mathematica 5 выдала более простой результат, чем дает Maple 9.5, но численные значения интеграла совпадают.

В системах Mathcad и MATLAB (с пакетом Symbolic Math Toolbox) вычисление интегралов основано на возможностях ядра Maple той или иной версии (как правило, последней на момент выпуска этих систем). В связи с этим возможности вычисления интегралов у них тождественны описанным для системы Maple. Однако форма вывода – текстовая.

Система Derive позволяет вычислять с помощью функций:

- **INT (u,x)** – вычисление первообразной интеграла с подынтегральной функцией u по переменной x;
- **DIF (u,x,-n)** – вычисление первообразной n-го порядка;
- **INT (u,x,a,b)** – вычисление определенного интеграла с пределами изменения x от a до b.

Обратите внимание, что функция **DIF** при записи значения n со знаком «–» возвращает первообразную, а не производную. Данные функции можно вводить в окне Author.

Файл утилит int_apps.mth Derive содержит около 30 специальных функций интегрирования. В основном это интегральные функции из области геометрии – вычисление дуг и дуговых интегралов, расчет плоскости геометрических фигур и их объема, вычисления центра масс и др. Ввиду частного характера этих функций они детально не рассматриваются – их можно найти в справочной базе данных системы и в справочнике [16].

Интегрирование в системе MuPAD представлено функцией

- **int(f(x),x)** – вычисление неопределенного интеграла;
- **int(f(x), x=a..b)** – вычисление определенного интеграла с пределами a и b.

Для вычисления определенного интеграла в численном виде применяется функция float совместно с функцией int:

`float(int(f(x),x=a..b))` или `float(hold(int)(f(x),x=a..b))`.

Применение этих функций очевидно.

> ***Внимание!*** *На уровне ядра системы Derive и MuPAD имеют типовые функции вычисления неопределенных и определенных интегралов в символьной форме и в численном виде. Довольно большая библиотека утилит интегрирования имеется в составе системы Derive. В основном она содержит средства вычисления интегралов из области геометрических расчетов. Есть и функции для интегральных преобразований Фурье и Лапласа.*

4.5. Вычисление пределов функций

4.5.1. Определение предела функции

Пределом функции $f(x)$ называют то ее значение b, к которому функция неограниченно приближается в точке $x = a$ (предел в точке) или слева либо справа от нее. Пределы обозначаются как:

Предел в точке *a*	**Предел слева от точки *a***	**Предел справа от точки *a***
$\lim\limits_{x \to a} f(x) = b$	$\lim\limits_{x \to a-} f(x) = b$	$\lim\limits_{x \to a+} f(x) = b$

При этом подразумевается, что функция $f(x)$ определена на некотором промежутке, включающем точку $x = a$, и во всех точках, близких к ней слева и справа. В последнем случае предел вычисляется для $x = a - h$ или $x = a + h$ при h, стремящемся к нулю. Пределом может быть число, математическое выражение и положительная или отрицательная бесконечность. Последнее соответствует расширенному представлению о пределах.

4.5.2. Функции вычисления пределов в Maple

Для вычисления *пределов* функции f в точке $x = a$ используются следующие функции:

```
limit(f,x=a);          limit(f,x=a,dir);
Limit(f,x=a);          Limit(f,x=a,dir);
```

Здесь `f` – алгебраическое выражение, `x` – имя переменной, `dir` – параметр, указывающий на направление поиска предела (`left` – слева, `right` – справа, `real` – в области вещественных значений, `complex` – в области комплексных значений). Значением `a` может быть бесконечность (как положительная, так и отрицательная).

Примеры применения этих функций для вычисления пределов в точке приведены ниже:

```
> Limit(sin(x)/x,x=0)=limit(sin(x)/x,x=0);
```

$$\lim_{x\to 0}\frac{\sin(x)}{x}=1$$

```
> Limit(1-exp(-x),x=infinity)=limit(1-exp(-x),x=infinity);
```

$$\lim_{x\to\infty}1-\mathbf{e}^{(-x)}=1$$

```
> Limit(exp(x),x=infinity)=limit(exp(x),x=infinity);
```

$$\lim_{x\to\infty}\mathbf{e}^{x}=\infty$$

```
> Limit(exp(-x),x=infinity)=limit(exp(-x),x=infinity);
```

$$\lim_{x\to\infty}\mathbf{e}^{(-x)}=0$$

```
> Limit((x-sin(x))/x^3,x=0)=limit((x-sin(x))/x^3,x=0);
```

$$\lim_{x\to 0}\frac{x-\sin(x)}{x^3}=\frac{1}{6}$$

4.5.3. Графическая иллюстрация вычисления пределов с двух сторон

Рисунок 4.13 показывает вычисление пределов функции $\tan(x)$ в точке $x = \pi/2$, а также слева и справа от нее. Для указания направления используются опции `right` (справа) и `left` (слева). Видно, что в самой точке предел не определен (значение `undefined`), а пределы справа и слева уходят в бесконечность.

Показанный на рис. 4.13 график функции $\tan(x)$ наглядно подтверждает существование пределов справа и слева от точки $x = \pi/2$ и отсутствие его в самой этой точке, где функция испытывает разрыв от значения $+\infty$ до $-\infty$.

4.5.4. Maplet-инструмент для иллюстрации методов вычисления пределов

Для демонстрации методов пошагового вычисления пределов имеется Maplet-инструмент Step-by-step Limit Tutor. Для вызова его окна (рис. 4.14) нужно исполнить команду (в стандартном варианте интерфейса): **Tools ⇒ Tutors ⇒ Calculus-Single Variables ⇒ Limit...**.

Нетрудно заметить, что это окно практически аналогично окну для демонстрации методов пошагового дифференцирования, описанному в разделе 4.3.4 (рис. 4.2). В связи с этим подробное описание средств и этого инструмента можно

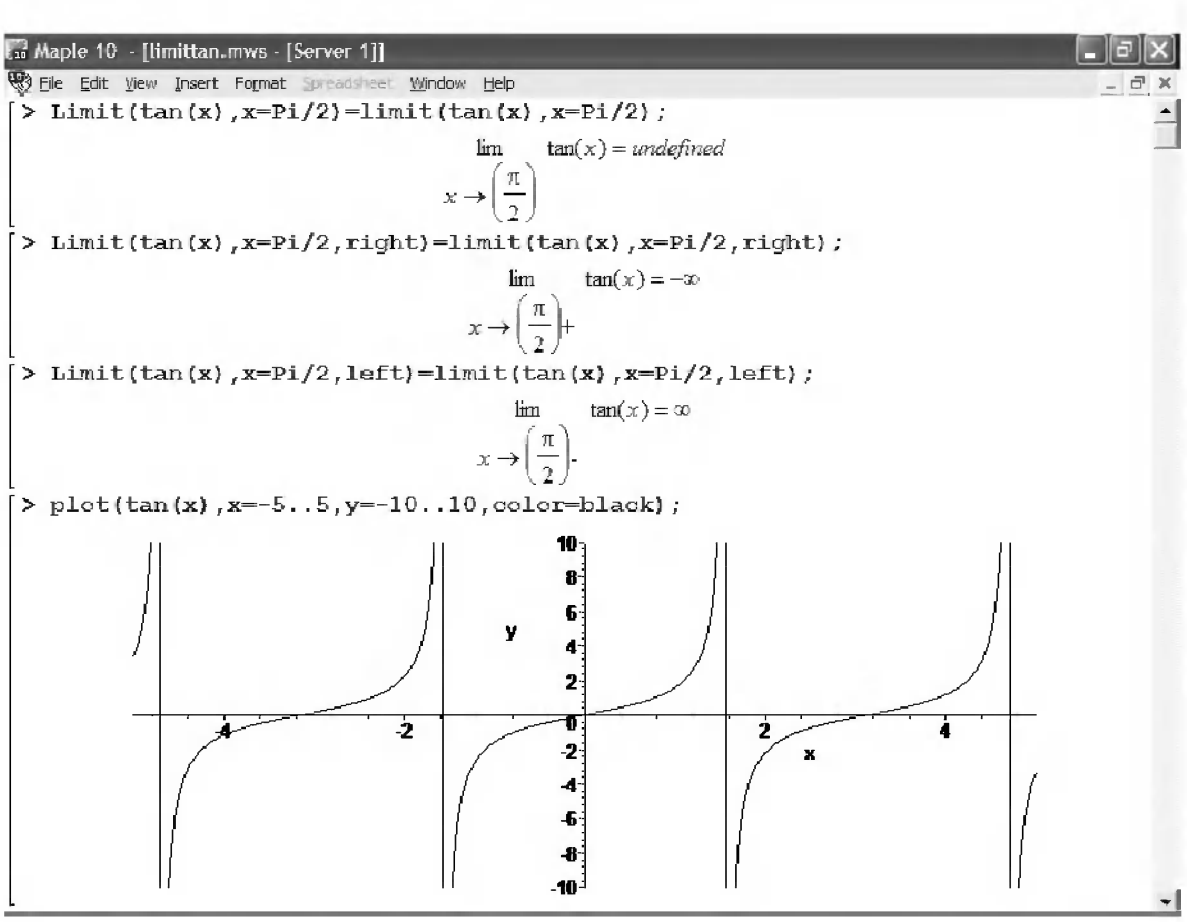


Рис. 4.13. Пример вычисления пределов функции tan(x) и построение ее графика

опустить. Отметим лишь, что он позволяет задавать функцию и значение x и по шагам (автоматически или вручную) вычислять пределы. По окончании работы с окном соответствующий предел и результат его вычисления появляются в окне документа – рис. 4.15.

4.5.5. Вычисление пределов в Mathematica и в других СКМ

Вычисление пределов в системе Mathematica – вполне рядовая операция. Она реализуется следующей функцией:

- **Limit[expr, x->x0]** – ищет значение предела выражения expr при x, стремящемся к $x0$.

При работе с этой функцией могут задаваться следующие опции:

- **Direction** – указывает направление, в котором происходит приближение к пределу. Опция используется в виде Direction -> –1 (или +1).
- **Analytic** – указывает, следует ли неопознанные функции интерпретировать как аналитические.
- **Automatic** – представляет значение опции, которое должно быть выбрано встроенной функцией автоматически.

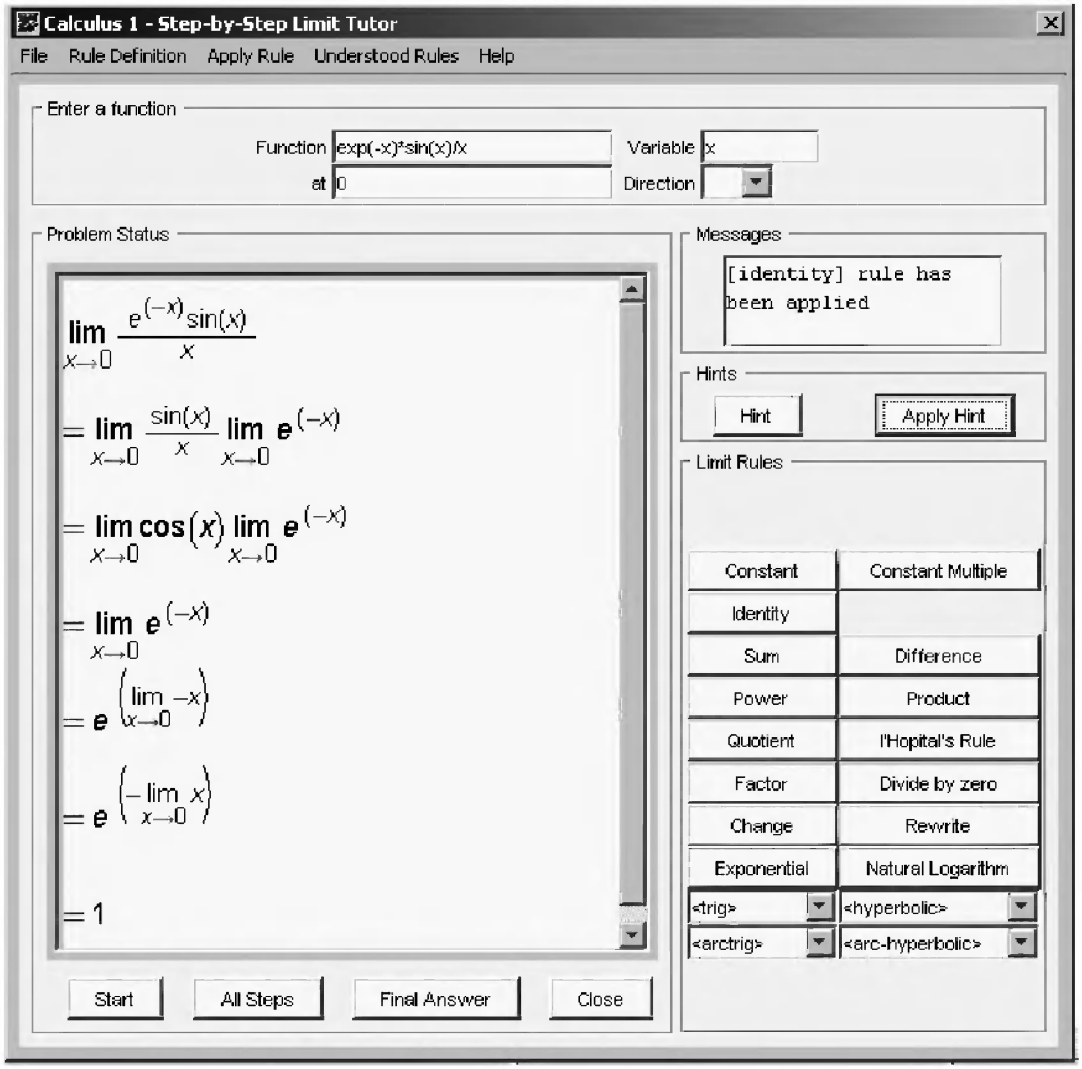


Рис. 4.14. Окно Maplet-демонстрации методов пошагового вычисления пределов

```
> Student[Calculus1][LimitTutor]();
```

$$\lim_{x \to 0} \left(\frac{e^{(-x)} \sin(x)}{x} \right) = 1$$

Рис. 4.15. Пример вывода результата работы с Maplet-инструментом по методам вычисления пределов

Ниже приведены примеры на вычисление пределов функций:

```
Limit[1/Log[x]-1/(x-1),x->1]
1/2
f[x,y]:=(a*x^2+b*x*y+c*y^2)/(d*x^2+e*x*y+f*y^2)
Limit[Limit[f[x,y],y->0],x->0]-Limit[Limit[f[x,y],
x->0],y->0]
a    c
- - -
d    f
```

```
Limit[(x^x-a^a)/(a^x-x^a),x->a]
1 + Log[a]
------
-1 + Log[a]
Limit[Sin[n*x]/x,x->0]
n
Limit[(x^2+x+2)/(x^2-2*x-3),x->3,Direction->-1]
Infinity
Limit[(x^2+x+2)/(x^2-2*x-3),x->3,Direction->+1]
-Infinity
Limit[(x^2+x+2)/(x^2-2*x-3),x->3]
Infinity
Limit[(x^2+1)/(2*x^2+3),x->Infinity]
1
-
2
```

Спецификой вычисления пределов в системе Mathcad является возможность этого только в режиме символьных вычислений. Для этого используются оператор lim и оператор символьного вывода >. Пример вычисления предела представлен в конце рис. 1.24.

Derive имеет функции для вычисления пределов:

- **LIM (u,x,a)** – предел функции u по переменной x в точке a;
- **LIM (u,x,a,1)** – предел функции u по переменной x выше точки a;
- **LIM (u,x,a,-1)** – предел функции u по переменной x ниже точки a.

Их можно задавать в окне **Author** системы Derive. Так, если задать

```
LIM (((x+h)^2-x^2)/((x+h)-x,h,0),
```

то на экране эта функция предстанет как

```
            2    2
     (x + h)  - x
lim -------------
      x + h - x
```

и упрощается к $2x$ (производная от x^2).

Если функция $u(x)$ разрывная, то пределы сверху и снизу могут отличаться. Для указания этого вводится четвертый аргумент – в общем случае это положительное число для вычисления предела сверху и отрицательное для вычисления предела снизу (удобно вводить соответственно 1 или –1). Бесконечность указывается как inf, например:

```
LIM ( a x/(x+1), x, inf)
```

изображается как

```
         a x
 lim    -----
x->oo   x + 1
```

и упрощается к a.

Вычисление предела часто бывает более предпочтительным, чем подстановка значений переменных с помощью опции **Manage Substitute**. Например, для функ-

ции **x/SIN(x)** подстановка $x = 0$ дает неопределенность вида 0/0 и упрощается к сообщению ?, тогда как

```
        x
 lim ------
x->0 SIN(x)
```

упрощается к 1.

В системе MuPAD используется функция *limit*, в общем виде записываемая в виде:

limit(f[,id=expr][,dir]),

где `f` – функция, `id` – идентификатор переменной, `dir` – направление, по которому ищется предел. Направление указывается опциями Left (предел слева) и Right (предел справа). Поясним применение этой функции несколькими примерами:

```
limit(sin(x)/x,x=0);
1
limit(1/(x-2),x=2);
undefined
limit(1/(x-2),x=2,Left);
-infinity
limit(1/(x-2),x=2,Right);
infinity
limit(1/x,x=0),limit(1/x,x=0,Left),limit(1/x,x=0,Right);
undefined, -infinity, infinity
```

Возможности вычисления пределов в системах Derive и MuPAD вполне достаточны для учебных целей, но заметно уступают таковым у систем Maple и Mathematica.

4.6. Разложение функций в ряды

4.6.1 Определение рядов Тейлора и Маклорена

Пусть задана задача – представление функции $f(x)$ в виде степенного многочлена $F(x)$ в окрестности заданной на оси абсцисс точки $x = x0$. Такое разложение было впервые рассмотрено Тейлором и получило название *ряда Тейлора* [68, 69]:

$$F(x) = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \ldots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n.$$

Если разложение выполняется относительно точки x=0, его принято называть *рядом Маклорена*:

$$F(x) = f(x_0) + \frac{f'(x_0)}{1!}x + \frac{f''(x_0)}{2!}x^2 + \ldots + \frac{f^{(n)}(x_0)}{n!}x^n.$$

Операции разложения в ряд Тейлора и Маклорена настолько распространены, что включены в набор функций практически всех СКМ и СКА.

4.6.2. Разложение в степенной ряд в системе Maple

Для разложения функции или выражения `expr` в обычный степенной ряд служат функции

`series(expr, eqn)` и **`series(expr, eqn, n)`**.

Здесь `eqn` – условие (например, в виде `x=a`) или имя переменной (например, `x`) и `n` – необязательное и неотрицательное целое число, задающее число членов ряда (при его отсутствии оно по умолчанию берется равным 6, но может переустанавливаться системной переменной `Order`). Если в качестве `eqn` задано имя переменной, то это соответствует разложению по этой переменной в области точки с ее нулевым значением. Задав `eqn` в виде `x=x0`, можно получить разложение по переменной x в окрестности точки $x = x_0$.

Разложение получается в форме степенного многочлена, коэффициенты которого задаются рациональными числами. Остаточная погрешность задается членом вида `O(x)^n`. При точном разложении этот член отсутствует. В общем случае для его удаления можно использовать функцию `convert`. Ниже представлены примеры разложения различных выражений в ряд:

```
> series(sinh(x),x=0);
```

$$x+\frac{1}{6}x^3+\frac{1}{120}x^5+O(x^6)$$

```
> series(sinh(x),x=1,3);
```

$$\left(\frac{1}{2}\mathbf{e}-\frac{1}{2}\frac{1}{\mathbf{e}}\right)+\left(\frac{1}{2}\mathbf{e}+\frac{\frac{1}{2}}{\mathbf{e}}\right)(x-1)+\left(\frac{1}{4}\mathbf{e}-\frac{1}{4}\frac{1}{\mathbf{e}}\right)(x-1)^2+O((x-1)^3)$$

```
> series(sinh(x),x=1.0,3);
```

$$1.175201193 + 1.543080635(x - 1.0) + .5876005967(x - 1.0)^2 + O((x - 1.0)^3)$$

```
> series(2*x^2-x+1,x=1,10);
```

$$2 + 3(x - 1) + 2(x - 1)^2$$

```
> f(x):=sin(x)/x;
```

$$f(x) := \frac{\sin(x)}{x}$$

```
> series(f(x),x=0,10);
```

$$1-\frac{1}{6}x^2+\frac{1}{120}x^4-\frac{1}{5040}x^6+\frac{1}{362880}x^8+O(x^9)$$

```
> convert(%,polynom);
```

$$1-\frac{1}{6}x^2+\frac{1}{120}x^4-\frac{1}{5040}x^6+\frac{1}{362880}x^8$$

```
> s:=series(ln(x),x=2,4);
```

$$s := \ln(2) + \frac{1}{2}(x-2) - \frac{1}{8}(x-2)^2 + \frac{1}{24}(x-2)^3 + O((x-2)^4)$$

```
> evalf(convert(s,polynom));
```

$$-.3068528194 + .5000000000x - .1250000000(x - 2.)^2 + .04166666667(x - 2.)^3$$

Здесь видно, что член, обозначающий погрешность, отсутствует в тех разложениях, которые точны, например в разложениях степенных многочленов. Результат разложения имеет особый формат, и для преобразования его в обычный полином (например, для построения графика разложения) следует использовать функцию `convert`.

4.6.3. Разложение в ряды Тейлора и Маклорена в Maple

Для разложения в *ряд Тейлора* используется функция `taylor(expr, eq/nm, n)`. Здесь `expr` – разлагаемое в ряд выражение, `eq/nm` – равенство (в виде `x=a`) или имя переменной (например, `x`), `n` – необязательный параметр, указывающий на порядок разложения и представленный целым положительным числом (при отсутствии указания порядка он по умолчанию принимается равным 6). При задании `eq/nm` в виде `x=a` разложение производится относительно точки $x = a$. При указании `eq/nm` в виде просто имени переменной разложение ищется в окрестности нулевой точки, то есть фактически вычисляется *ряд Маклорена*.

Ниже представлены примеры применения функции `taylor`:

```
> taylor(1-exp(x),x=1,4);
```

$$(1-\mathbf{e}) - \mathbf{e}(x-1) - \frac{1}{2}\mathbf{e}(x-1)^2 - \frac{1}{6}\mathbf{e}(x-1)^3 + O(x-1)^4$$

```
> convert(%,polynom);
```

$$2\frac{x}{\sqrt{\pi}} - \frac{2}{3}\frac{x^3}{\sqrt{\pi}} + \frac{\frac{1}{5}x^5}{\sqrt{\pi}}$$

```
> taylor(sinh(x),x,10);
```

$$x + \frac{1}{6}x^3 + \frac{1}{120}x^5 + \frac{1}{5040}x^7 + \frac{1}{362880}x^9 + O(x^{10})$$

```
> taylor(int(sin(x)/x,x),x);
```

$$x - \frac{1}{18}x^3 + \frac{1}{600}x^5 + O(x^6)$$

```
> taylor(erf(x),x);
```

$$2\frac{1}{\sqrt{\pi}}x - \frac{2}{3}\frac{1}{\sqrt{\pi}}x^3 + \frac{1}{5}\frac{1}{\sqrt{\pi}}x^5 + O(x^6)$$

Не все выражения (функции) имеют разложение в ряд Тейлора. Ниже дан пример такого рода:

```
> taylor(1/x+x^2,x,5);
Error, does not have a taylor expansion, try series()
> series(1/x+x^2,x,10);
```

$$x^{-1} + x^2$$

```
> taylor(1/x+x^2,x=1,5);
```

$$2 + x - 1 + 2(x-1)^2 - (x-1)^3 + (x-1)^4 + O((x-1)^5)$$

Здесь Maple отказалась от вычисления ряда Тейлора в окрестности точки $x = 0$ (по умолчанию) и предложила воспользоваться функцией `series`. Однако эта функция просто повторяет исходное разложение. В то же время в окрестности точки $x = 1$ ряд Тейлора вычисляется.

Для разложения в ряд Тейлора функций нескольких переменных используется библиотечная функция `mtaylor`:

```
mtaylor(f, v)  mtaylor(f, v, n)  mtaylor(f, v, n, w)
```

Здесь `f` – алгебраическое выражение, `v` – список имен или равенств, `n` – необязательное число, задающее порядок разложения, `w` – необязательный список целых чисел, задающих «вес» каждой из переменных списка `v`. Эта функция должна вызываться из библиотеки Maple 9 с помощью команды `readlib`.

Для получения только коэффициента при k-ом члене ряда Тейлора можно использовать функцию `coeftayl(expr,var,k)`. Если `expr` – функция нескольких переменных, то `k` должен задаваться списком порядков коэффициентов.

4.6.4. Пример документа – разложения синуса в ряд

Полезно сочетать разложение выражений (функций) в ряд Тейлора с графической визуализацией такого разложения – рис. 4.16. Поскольку выбрано разложение относительно точки $x = 0$, то полученный ряд является рядом Маклорена.

Можно буквально в считанные секунды попробовать изменить число членов ряда или диапазон изменения переменной x, что и показано на рис. 4.17. При этом легко убедиться в том, что при больших x поведение ряда не имеет ничего общего с поведением разлагаемой в ряд функции – в частности, нет и намека на периодичность разложения, которая присуща тригонометрической функции $\sin(x)$.

На рис. 4.18 представлено уже истинное разложение синуса в ряд Тейлора в окрестности смещенной от нуля точки $x = 1$. При смещении точки, относительно которой ведется разложение, выражение для ряда Тейлора существенно изменяется. В нем, во-первых, появляются члены четных степеней, а во-вторых, фигурирует аргумент вида $(x-1)^n$.

Нетрудно заметить, что даже при представлении такой «простой» функции, как $\sin(x)$, приемлемая погрешность представления одного периода достигается при числе членов ряда Тейлора порядка 10 и более. Однако существенное повышение порядка ряда нецелесообразно из-за резкого возрастания вычислительных погрешностей. Впрочем, если задать достаточно большое число верных цифр результатов, то в Maple можно использовать ряды с гораздо большим числом членов.

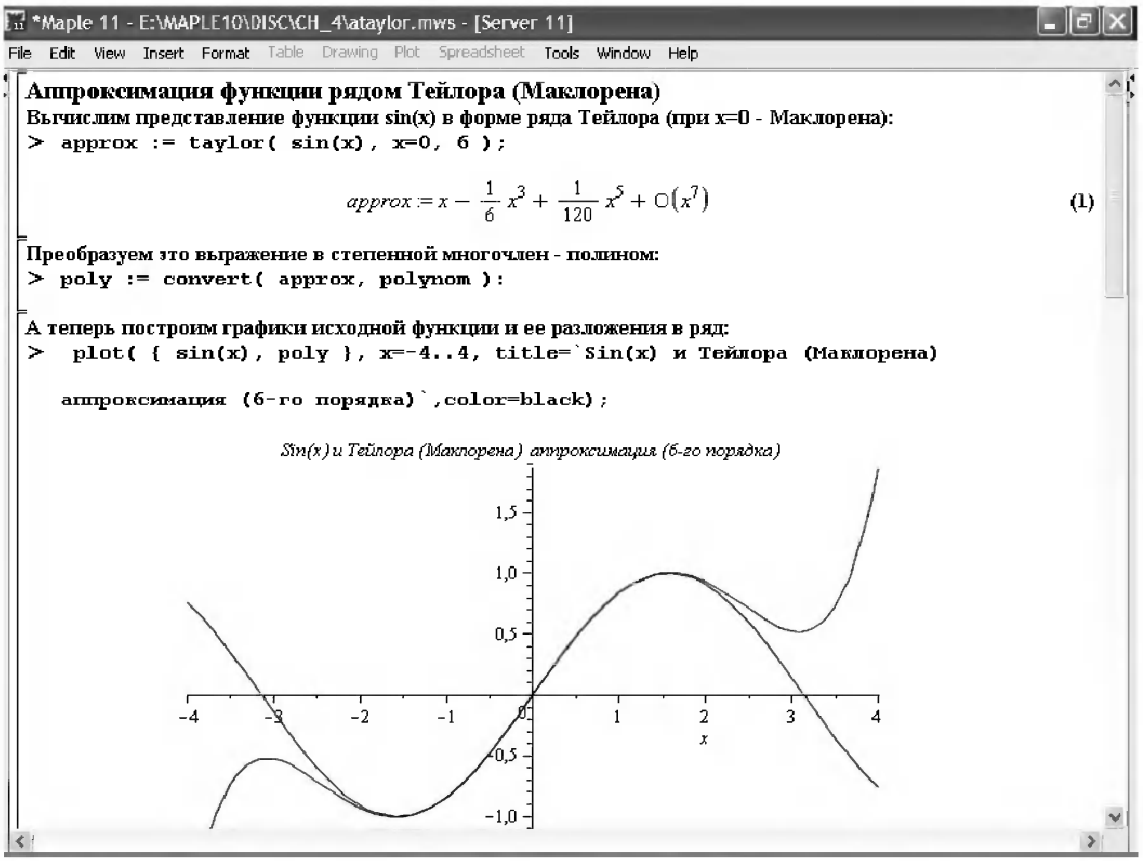


Рис. 4.16. Разложение функции sin(x) в ряд Маклорена шестого порядка и построение ее графика

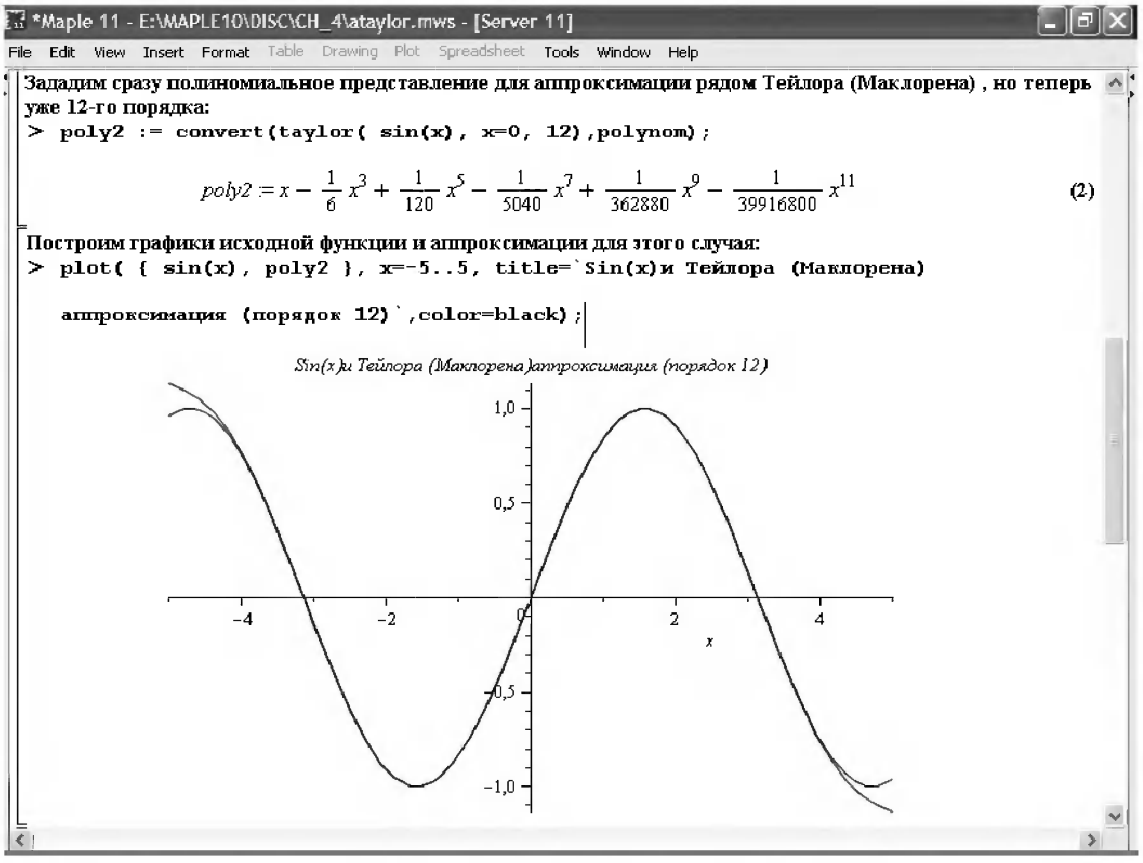


Рис. 4.17. Разложение функции sin(x) в ряд Маклорена 12-го порядка и построение ее графика (продолжение документа рис. 4.16)

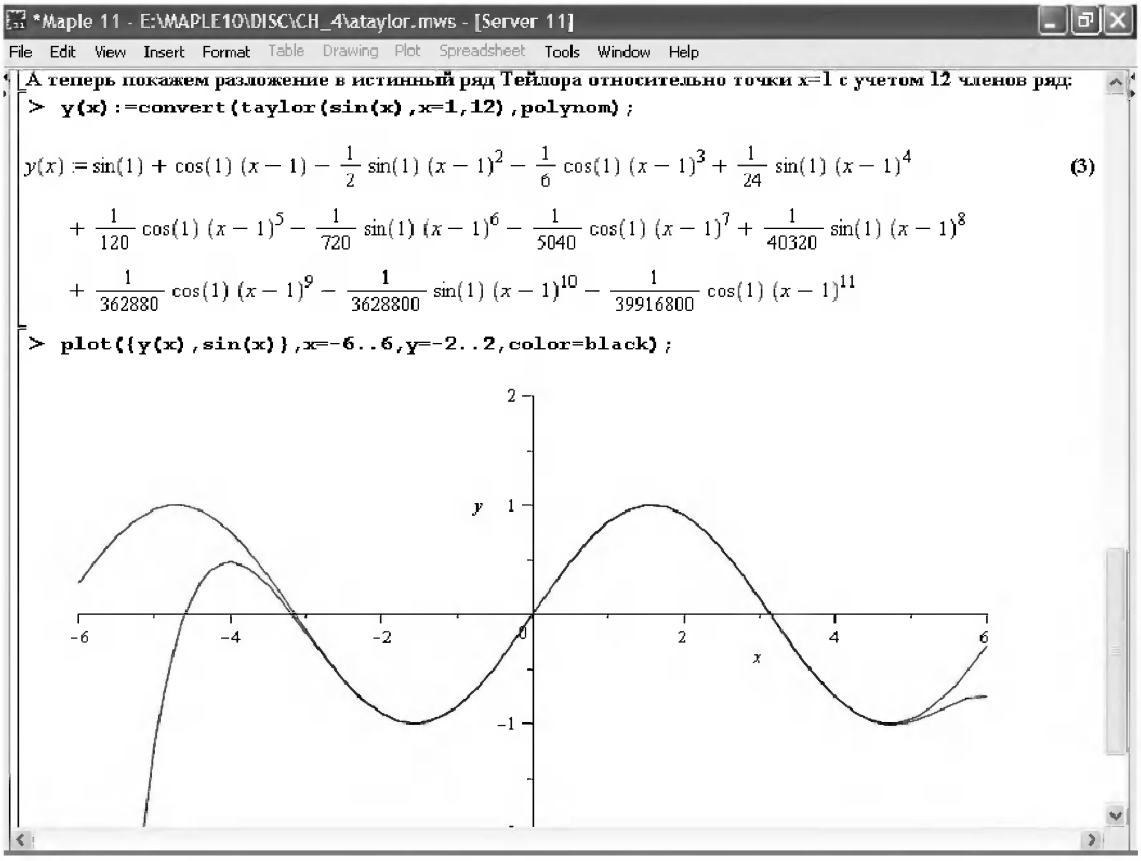


Рис. 4.18. Разложение функции sin(x) в ряд Тейлора 12-го порядка относительно точки x = 1 и построение ее графика (конец документа рис. 4.16)

Кроме того, серьезным недостатком аппроксимации рядом Тейлора является непредсказуемое поведение полинома вдали от точки, относительно которой задается представление. Это хорошо видно на всех трех приведенных выше примерах.

Помимо указанных выше разложений в ряд, Maple имеет множество функций для иных разложений. Например, в пакете numapprox имеется функция `laurent(expr,var,n)`, позволяющая получить разложение в ряд Лорана, функция `chebyshev(expr, eq/nm, eps)` дает разложение в форме полиномов Чебышева и т. д.

4.6.5. Пакет вычисления степенных разложений powseries

Степенные разложения часто используются в математических расчетах для приближенного представления разнообразных функций и обеспечения единообразия такого представления. В пакете powseries, вызываемом командой

```
> with(powseries):
```

задано определение этих функций:

- `compose(a,b)` — объединяет ряды a и b;
- `evalpow(expr)` — вычисляет выражение `expr` и возвращает его в виде ряда;

- `inverse(p)` – инвертирует ряд `p`;
- `multconst(p,const)` – умножает ряд `p` на константу `const`;
- `multiply(a,b)` – умножает ряд `a` на ряд `b`;
- `negative(p)` – возвращает аддитивный обратный по отношению к `p` ряд;
- `powadd(a,b,...)` – складывает ряды `a`, `b`, ...;
- `powcreate(expr)` – создает ряд для выражения `expr`;
- `powpoly(pol,var)` – создает ряд для полинома `pol` по переменной `var`;
- `powsolve(sys)` – создает ряд для решения дифференциальных уравнений `sys`;
- `quotient(a,b)` – возвращает частное для `a` и `b` в виде ряда;
- `reversion(a)` – дает обратное к композиции разложение ряда `a`;
- `subtract(a,b)` – дает разность рядов `a` и `b`.

В выражении `expr` могут использоваться операторы +, –, *, / и ^. С ними могут комбинироваться встроенные функции и функции пользователя, например $f(g)$.

Назначение большинства этих функций очевидно из их названий – они возвращают соответствующую функцию (указанную после слова `pow` в имени) в виде разложения в ряд или полинома. Например, `powexp` раскладывает выражения с экспоненциальными функциями в ряд.

Получаемые функциями ряды представляются в специальном формате. Поэтому для их применения в обычном виде необходимо использовать функцию `tpsform` в следующих видах:

- `tpsform(p, var, order)` – преобразует ряд `p` в обычную форму с заданием порядка `order`;
- `tpsform(p, var)` – преобразует ряд `p` в обычную форму с порядком, заданным переменной `Order`.

Здесь `p` – имя степенного ряда, `var` – переменная, относительно которой записан ряд, `order` – порядок ряда. Если параметр `order` не указан, используется значение глобальной переменной `Order`.

4.6.6. Примеры выполнения степенных разложений

Ниже даны примеры, иллюстрирующие технику работы со степенными разложениями:

```
> p1:=powexp(sin(x));
```

$$p1 := \mathbf{proc}(powparm) \ldots \mathbf{end\ proc}$$

```
> p2:=powexp(cos(x));
```

$$p2 := \mathbf{proc}(powparm) \ldots \mathbf{end\ proc}$$

```
> tpsform(p1,x);
```

$$1 + x + \frac{1}{2}x^2 - \frac{1}{8}x^4 - \frac{1}{15}x^5 + O(x^6)$$

```
> tpsform(p2,x);
```

$$e-\frac{1}{2}ex^2+\frac{1}{6}ex^4+O(x^6)$$

```
> a := powseries[powexp](x):
> b := powseries[tpsform](a, x, 5);
```

$$b := 1+x+\frac{1}{2}x^2+\frac{1}{6}x^3+\frac{1}{24}x^4+O(x^5)$$

```
> c := powadd( powpoly(1+x^2+x,x), powlog(1+x) ):
> d := tpsform(c, x, 6);
```

$$d := 1+2x+\frac{1}{2}x^2+\frac{1}{3}x^3-\frac{1}{4}x^4+\frac{1}{5}x^5+O(x^6)$$

4.6.7. Maplet-иллюстрация аппроксимации рядом Тейлора в ряд

Для демонстрации разложения аналитической функции в ряд имеется Maplet-инструмент Taylor Approximation. Для вызова его окна (рис. 4.19) нужно исполнить команду (в стандартном варианте интерфейса): **Tools ⇒ Tutors ⇒ Calculus-Single Variables ⇒ Taylor Approximation ...**.

Это окно позволяет задавать функцию и значение x в точке разложения. По окончании работы с окном соответствующий предел и результат его вычисления

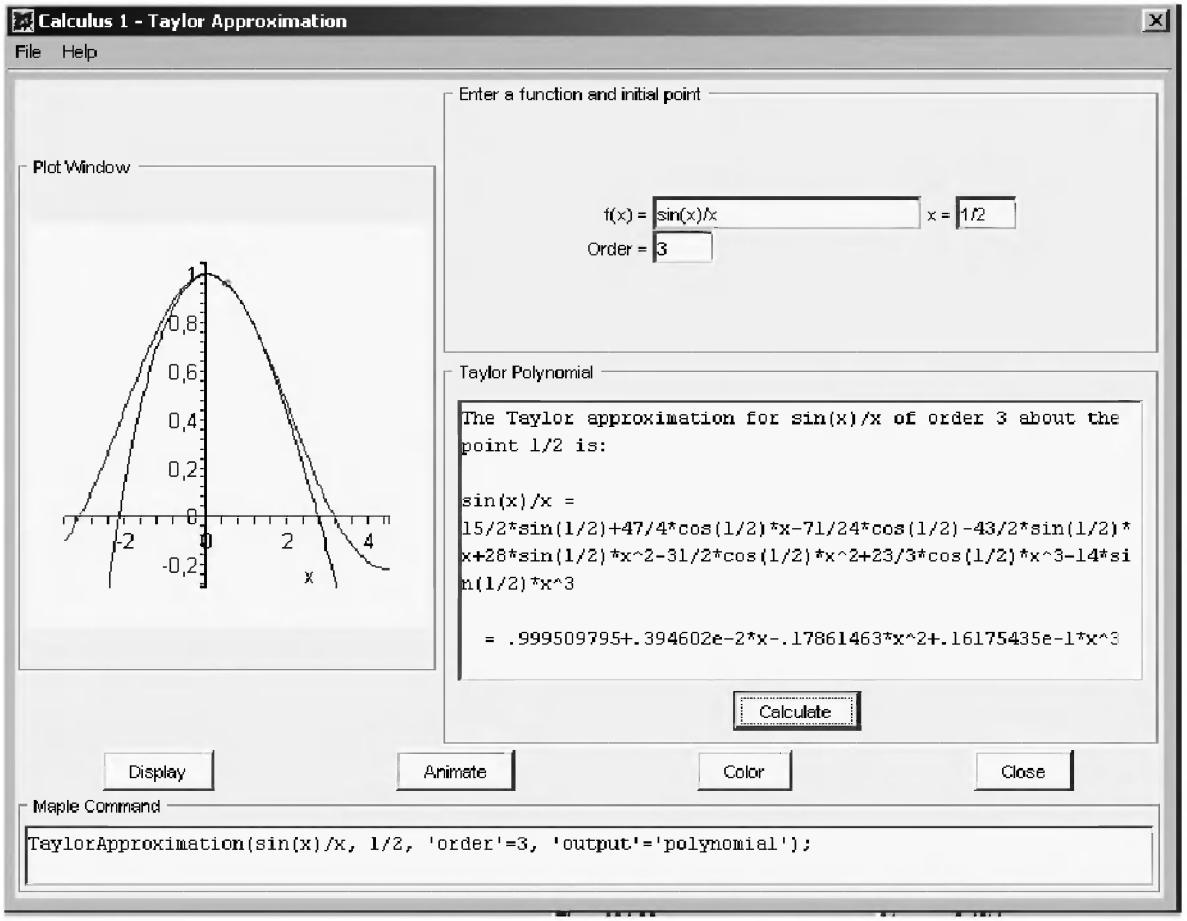


Рис. 4.19. Окно Maplet-демонстрации аппроксимации функции рядом Тейлора

появляются в окне документа. Можно просматривать постепенное улучшение приближения по мере увеличения порядка метода в режиме анимации.

4.6.8. Разложение в ряд в Mathematica и в других СКМ

Для разложения в ряд используются следующие функции системы Mathematica:

- **Series[f, {x, x0, n}]** – выполняет разложение в степенной ряд функции f в окрестности точки x=x0 по степеням (x-x0)^n.
- **Series[f, {x, x0, nx}, {y, y0, ny}]** – последовательно ищет разложения в ряд по переменным y, затем x.
- **SeriesCoefficient[s,n]** – возвращает коэффициент при переменной n-ой степени ряда s.
- **SeriesData[x, x0, {a0, a1, ...}, nmin, nmax, den]** – представляет степенной ряд от переменной *x* в окрестности точки x0. Величины ai являются коэффициентами степенного ряда. Показатели степеней (x-x0) представлены величинами nmin/den, (nmin+1)/den, ..., nmax/den.

Примеры разложения функций в степенной ряд:

```
Series[y[x],{x,0,5}]
```

$$y[0]+y'[0]\,x+\frac{1}{2}y''[0]\,x^2+\frac{1}{6}y^{(3)}[0]\,x^3+\frac{1}{24}y^{(4)}[0]\,x^4+\frac{1}{120}y^{(5)}[0]\,x^5+O[x]^6$$

```
Series[Sinh[x],{x,0,10}]
```

$$x+\frac{x^3}{6}+\frac{x^5}{120}+\frac{x^7}{5040}+\frac{x^9}{362880}+O[x]^{11}$$

Результат разложения имеет особый тип данных – *ряд* с остаточной погрешностью. Если нужно убрать член с остаточной погрешностью, то можно использовать функцию **Collect[expr]**, например:

```
Collect[Series[Sin[x],{x,0,5}],x]
```

$$x-\frac{x^3}{6}+\frac{x^5}{120}$$

Другая полезная функция **Normal[expr]** также отбрасывает член с остаточной погрешностью:

```
Normal[Series[Sin[x+y],{x,0,3},{y,0,3}]]
```

$$x-\frac{x^3}{6}+\left(1-\frac{x^2}{2}\right)y+\left(-\frac{x}{2}+\frac{x^3}{12}\right)y^2+\left(-\frac{1}{6}+\frac{x^2}{12}\right)y^3$$

Для представления выходных данных в стандартном формате ввода надо снять защиту от модификации у функции преобразования **InputForm**:

```
Unprotect[InputFotm]
InputForm[Normal[Series[Sin[x+y],{x,0,3},{y,0,3}]]
```

Это даст выход в форме:

```
{}
x*y-(x^3+y^3)/6
```

Теперь можно составить функцию пользователя, задающую вычисление данного разложения в ряд:

```
F[x_,y_]:=x*y-(x^3+y^3)/6
```

Здесь мы получили функцию пользователя, представляющую разложение функции *sin(x+y)* по двум переменным – *x* и *y*. Нетрудно проверить, что эта функция и впрямь вычисляется:

```
F[0.1,0.2]
  0.0199987
```

Малые системы компьютерной алгебры Derive и MuPAD также прекрасно справляются с разложением функций в ряды Тейлора и Маклорена. Derive, к примеру, имеет команду разложения в ряд Тейлора **Taylor Series…** в позиции **Calculus** главного меню. Эта команда выводит простое окно для задания разлагаемой функции, переменной по которой идет разложение, точки $x0=a$ и порядка степенного многочлена.

Как обычно, кнопка **Simplify** выводит в текущую строку документа результат разложения, а кнопка **OK** задает функцию разложения в ряд Тейлора вида:

```
TAYLOR(f(x),x,a,n),
```

где $a = x0$ – точка на оси *X*, относительно которой выполняется разложение, n – порядок степенного многочлена.

В системе MuPAD для разложения функций (выражений) в ряд используется следующая функция:

- **series(expr,id[,n][,dir])** – возвращает разложение в ряд выражения expr относительно переменной с идентификатором id при числе членов ряда n и направлении dir со значениями Left или Right. По умолчанию $n=6$ и используется ряд Маклорена.
- **series(expr,id[=expr_1][,n][,dir])** – возвращает разложение в ряд выражения expr относительно переменной с идентификатором id со значением expr_1, при числе членов ряда *n* и направлении dir со значениями Left или Right.

Запись результата в MuPAD получается в классической форме – с членом O, учитывающим остаточную погрешность.

> ***Внимание!*** *Возможности малых систем компьютерной алгебры Derive и MuPAD позволяют легко и удобно решать задачи на разложения в ряды Тейлора и Маклорена функций и строить их графики вместе с графиками разложения. Однако, как и в системе Mathcad, требуются ручные манипуляции при смене разлагаемой функции или числа членов степенного ряда, представляющего функцию.*

4.7. Визуализация приложений математического анализа

Любая СКМ имеет возможности для визуализации различных приложений математического анализа. Особое внимание этому уделено в системе Maple, где с помощью Maplet-средств созданы самоучители, обеспечивающие наглядное представление приложений математического анализа. Ниже рассмотрены как эти средства, так и средства популярной системы Mathcad.

4.7.1. Суммы Римана и приближение интегралов

Есть два основных способа вычисления определенных интегралов в численном виде:

- на основе сумм Римана (варианты метода прямоугольников);
- на основе приближения подынтегральной функции той или иной зависимостью.

Оба метода реализуются Maplet-инструментом Approximate Integration. Для вызова окна этого инструмента (рис. 4.20) нужно исполнить команду (в стандартном варианте интерфейса): **Tools ⇒ Tutors ⇒ Calculus-Single Variables ⇒ Approximate Integration ...**. Совершенно аналогичное окно выводит команда **Tools ⇒ Tutors ⇒ Calculus-Single Variables ⇒ Rieman summs...**.

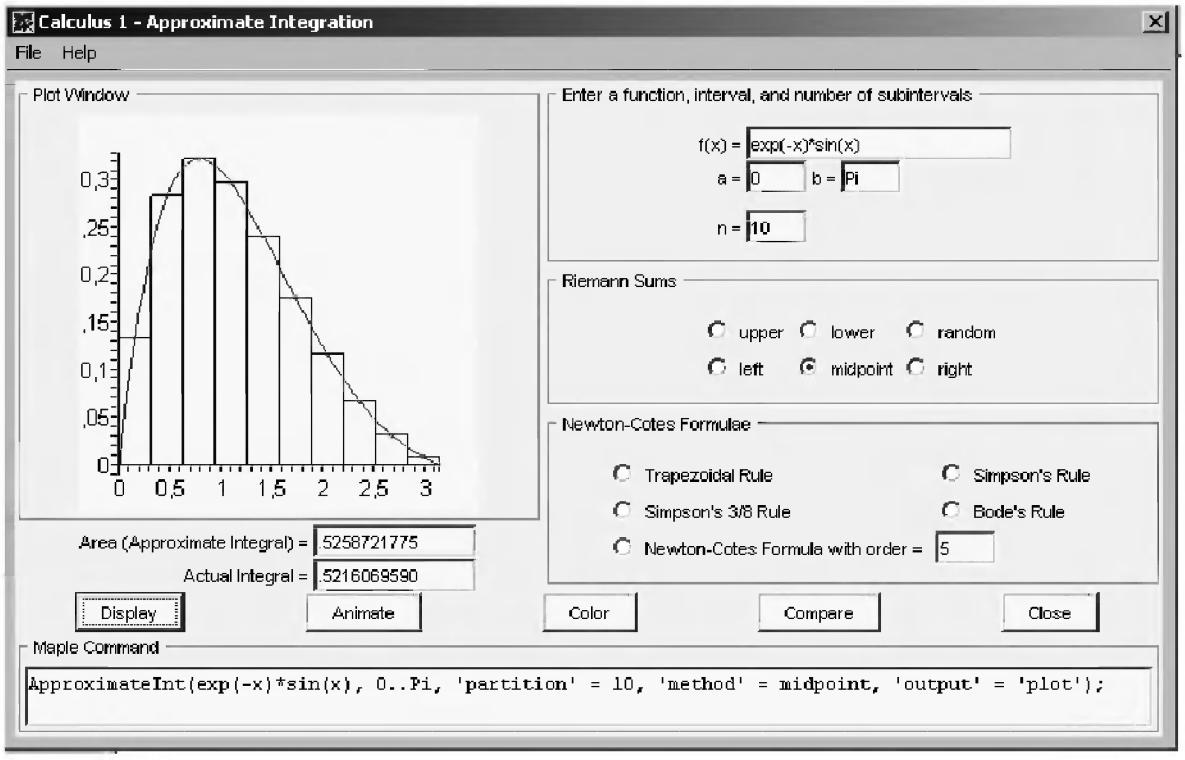


Рис. 4.20. Пример приближения интеграла суммой Римана (10 прямоугольников с центральным расположением)

В правой части окна размещены панели:

- ввода функции f(x), пределов a и b и числа интервалов разбиения n;
- задания расположения прямоугольников, которые образуют сумму Римана;
- методов Ньютона-Котеса.

Относительно каждой ординаты прямоугольник может быть ориентирован сверху или снизу, справа или слева, посередине или даже случайным образом. При реализации формул приближения Ньютона-Котеса возможно применение метода трапеций, двух вариантов метода Симпсона (квадратичное приближение), метода Боде и известных формул Ньютона-Котеса заданного порядка (по умолчанию 5). В функциях численного интегрирования Maple тот или иной вид приближения можно задать явно, но по умолчанию метод выбирается автоматически. После выбора метода можно получить его графическую иллюстрацию (рис. 4.20), нажав мышью кнопку `Display`.

Данный инструмент позволяет наблюдать в анимации повышение точности вычислений по мере увеличения числа прямоугольников – см. рис. 4.21. Для пуска анимации достаточно нажать мышью кнопку `Animate`. На рис. 4.21 показан промежуточный кадр анимации. В конце анимации закраска области интегрирования становится сплошной, после чего анимация циклически повторяется.

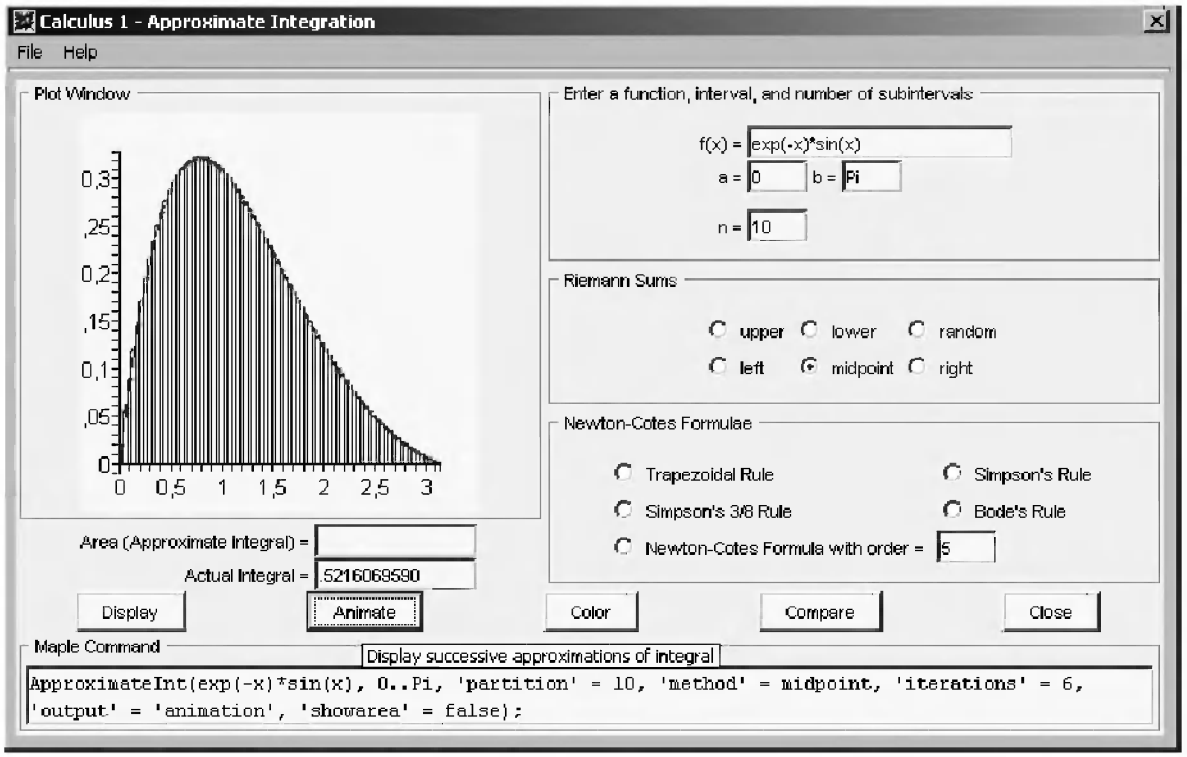


Рис. 4.21. Промежуточный кадр анимации, демонстрирующей приближение интеграла суммами Римана

Приближение суммами Римана относится к довольно медленным методам интегрирования. Значительно повысить скорость интегрирования при заданной погрешности позволяют методы интегрирования повышенного порядка на основе формул Ньютона-Котесса. На рис. 4.22 показан пример приближения опредлен-

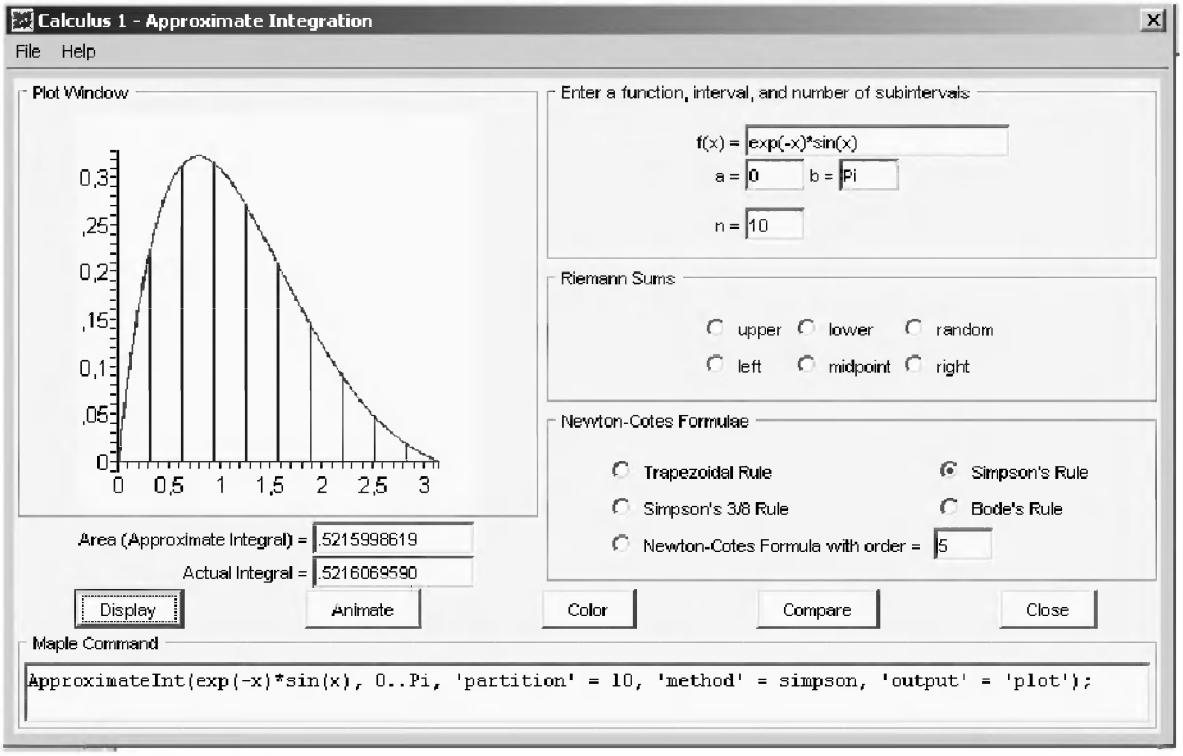


Рис. 4.22. Пример приближения интеграла методом Симпсона

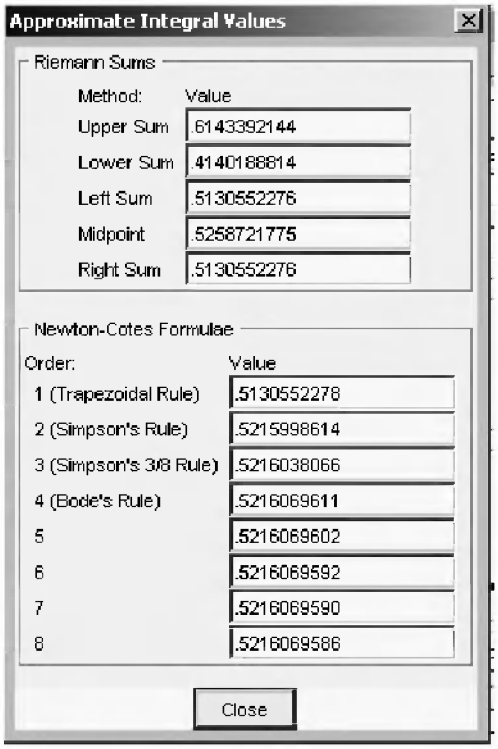


Рис. 4.23. Окно с результатами сравнения интегрирования различными методами

ного интеграла на основе формулы Симсона (параболического приближения подынтегральной функции). Из рисунка хорошо видно, что в этом случае (в отличие от рис. 4.20 при интегрировании методом прямоугольников) исходная подынтегральная функция и ее приближение отрезками парабол практически совпадают и на глаз их отличия выявить трудно.

Кнопка `Compare` позволяет вывести таблицу с данными сравнения результатов интегрирования различными методами. Окно с этой таблицей представлено на рис. 4.23. Хорошо видно, что по мере повышения порядка метода интегрирования погрешность интегрирования уменьшается.

4.7.2. Вычисление длины дуги

Если $f(x)$ непрерывная на отрезке от a до b функция, то *длина дуги* этой функции (длина спрямленного отрезка) определяется известным выражением

$$l = \int_a^b \sqrt{1 + f'^2(x)}\,dx.$$

Для демонстрации вычисления длины дуги заданной аналитической функции имеется Maplet-инструмент ArcLench. Для вызова его окна (рис. 4.24) нужно исполнить команду (в стандартном варианте интерфейса): **Tools** ⇒ **Tutors** ⇒ **Calculus-Single Variables** ⇒ **ArcLench ...**.

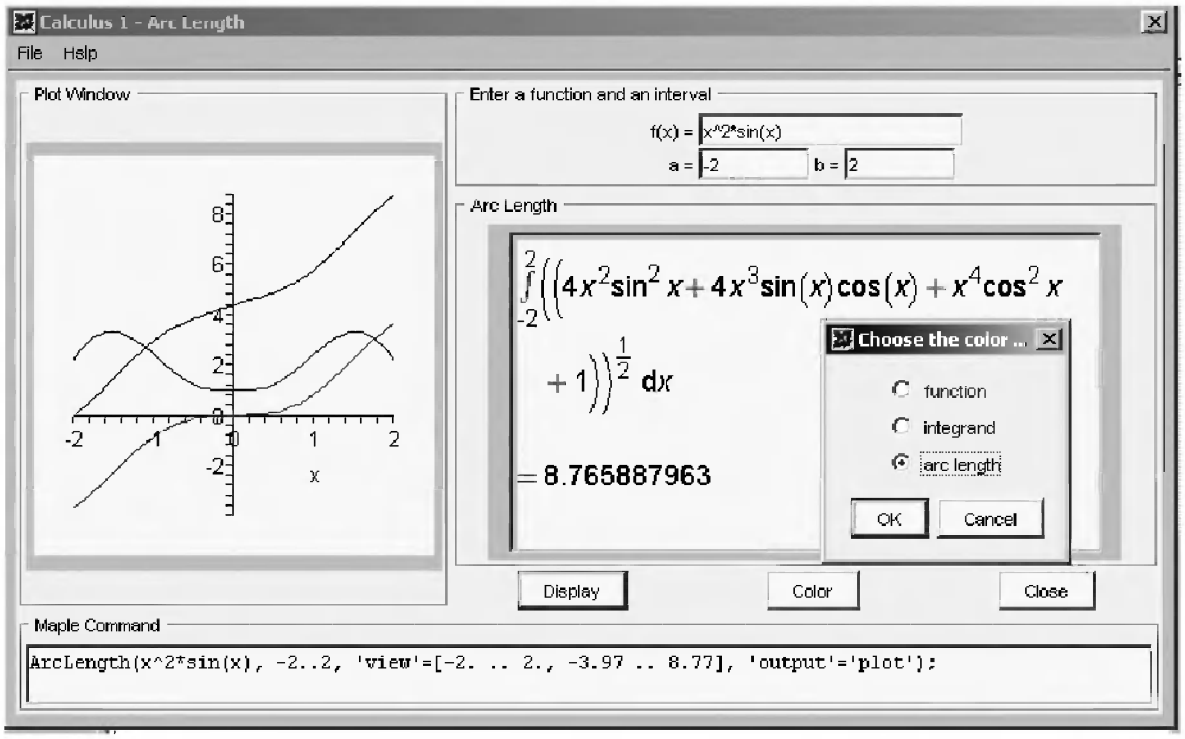


Рис. 4.24. Окно Maplet-инструмента для вычисления длины дуги

Кнопка `Color` открывает окно выбора цвета из списка, который представлен окном Choose the color..., показанным внутри окна инструмента (см. рис. 4.24). Выбрав цвет нужной кривой, нажатием кнопки `OK` можно вызвать панель выбора цветов `Select a color`, показанную на рис. 4.25. По завершении выбора цвета нужная кривая будет отображена в новом цвете.

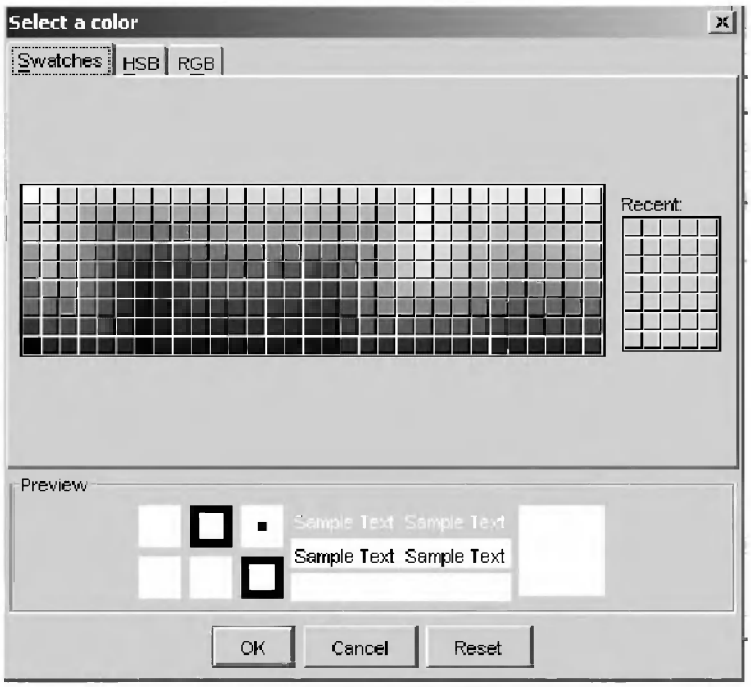


Рис. 4.25. Панель выбора цвета

4.7.3. Иллюстрация теоремы о среднем

Первая *теорема о среднем* гласит, что если $f(x)$ – интегрируемая функция, непрерывная на отрезке $[a,b]$, то существует по крайней мере одно значение $x = \xi$ в интервале $[a,b]$, при котором

$$\int_a^b f(x)dx = f(\xi)(b-a).$$

Иными словами, площадь, определяемая интегралом, может быть вычислена как площадь прямоугольника с основанием – отрезком ab и высотой $f(o)$.

Для иллюстрации этого положения служит Maplet-инструмент Mean Value Theorem. Его окно (рис. 4.26) открывается исполнением команды **Tools ⇒ Tutors ⇒ Calculus-Single Variables ⇒ Mean Value Theorem...**. Работа с окном вполне очевидна. На графике строятся кривая функции, отрезок, проходящий через ее концевые точки, точка со значением $x = c = \xi$ и касательная к ней. Главный результат – значение $c = \xi$.

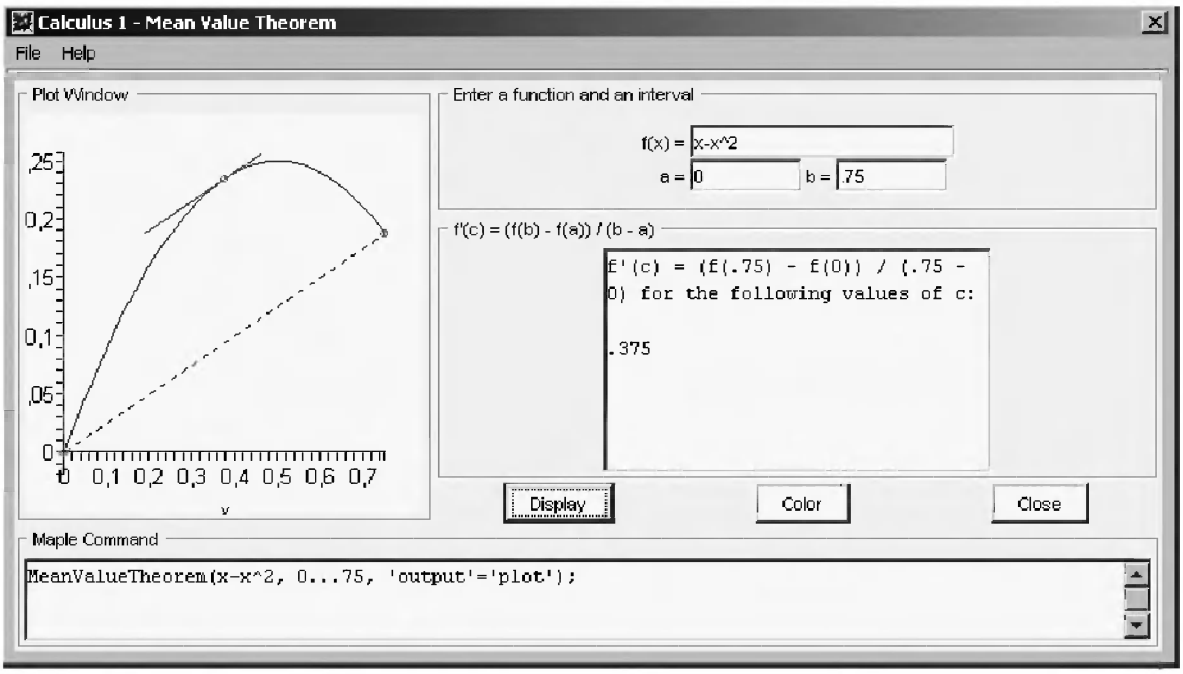


Рис. 4.26. Окно Maplet-инструмента для иллюстрации первой теоремы о среднем

4.7.4. Построение касательной к заданной точке кривой

Для построения *касательной* к заданной точке на кривой $f(x)$ служит Maplet-инструмент Tangent. Его окно (рис. 4.27) открывается исполнением команды **Tools ⇒ Tutors ⇒ Calculus-Single Variables ⇒ Tangent...**. Работа с окном вполне очевидна. На графике строятся кривая функции и касательная к заданной точке x.

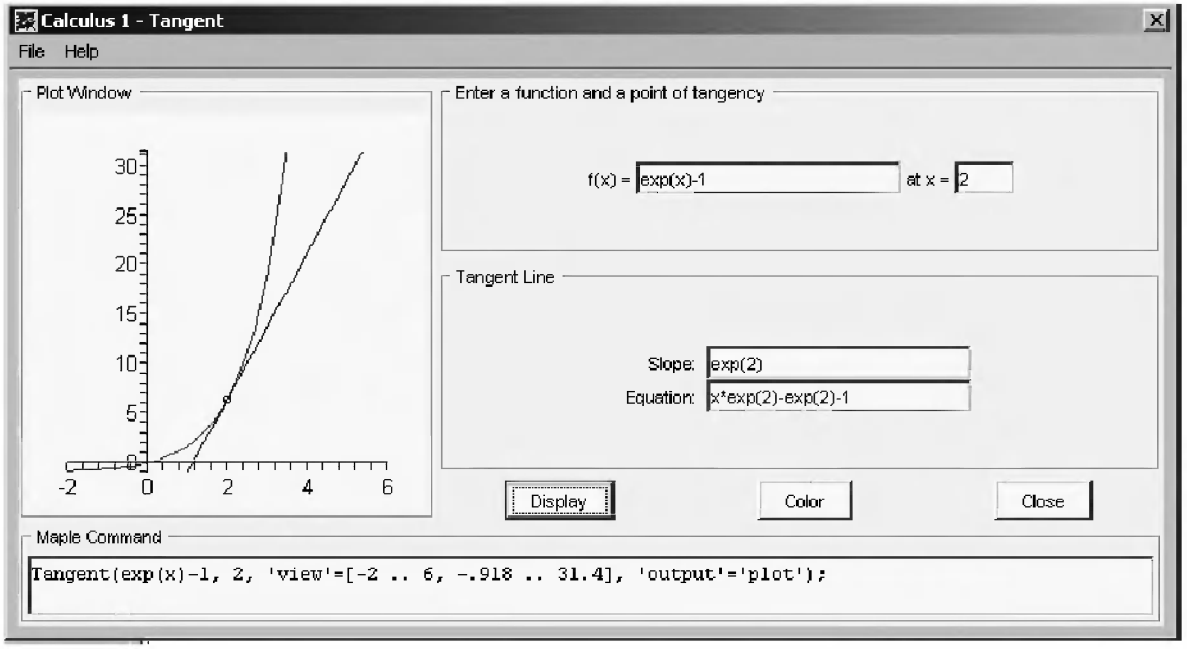


Рис. 4.27. Окно Maplet-инструмента для иллюстрации построения касательной к заданной точке

Наклон касательной определяется значением первой производной $f'(x)$, значение которой Slope и уравнение касательной вычисляются.

4.7.5. Построение касательной к заданной точке кривой и секущих линий

В некоторых случаях, например при реализации метода Ньютона решения нелинейных уравнений, помимо построения касательной к заданной точке кривой f(x), нужно строить *секущие линии* и определять их точки пересечения с f(x).

Для этого служит Maplet-инструмент Tangent and Secant. Его окно (рис. 4.28) открывается исполнением команды **Tools** ⇒ **Tutors** ⇒ **Calculus-Single Variables** ⇒ **Tangent and Secant...**. Работа с окном вполне очевидна. На графике строятся кривая функции и касательная к заданной точке x. Дополнительно строится ряд секущих. Возможно построение с применением анимации.

4.7.6. Вычисление поверхности вращения кривой

Пусть отрезок кривой $f(x)$, при x в интервале $[a,b]$, вращается вокруг оси $0x$. Тогда *площадь фигуры вращения* равна:

$$P = \int_a^b 2\pi f(x)\sqrt{1+f'^2(x)}dx.$$

Для вычисления этой площади служит Maplet-инструмент Surface of Revolution. Его окно (рис. 4.29) открывается исполнением команды **Tools** ⇒ **Tutors** ⇒

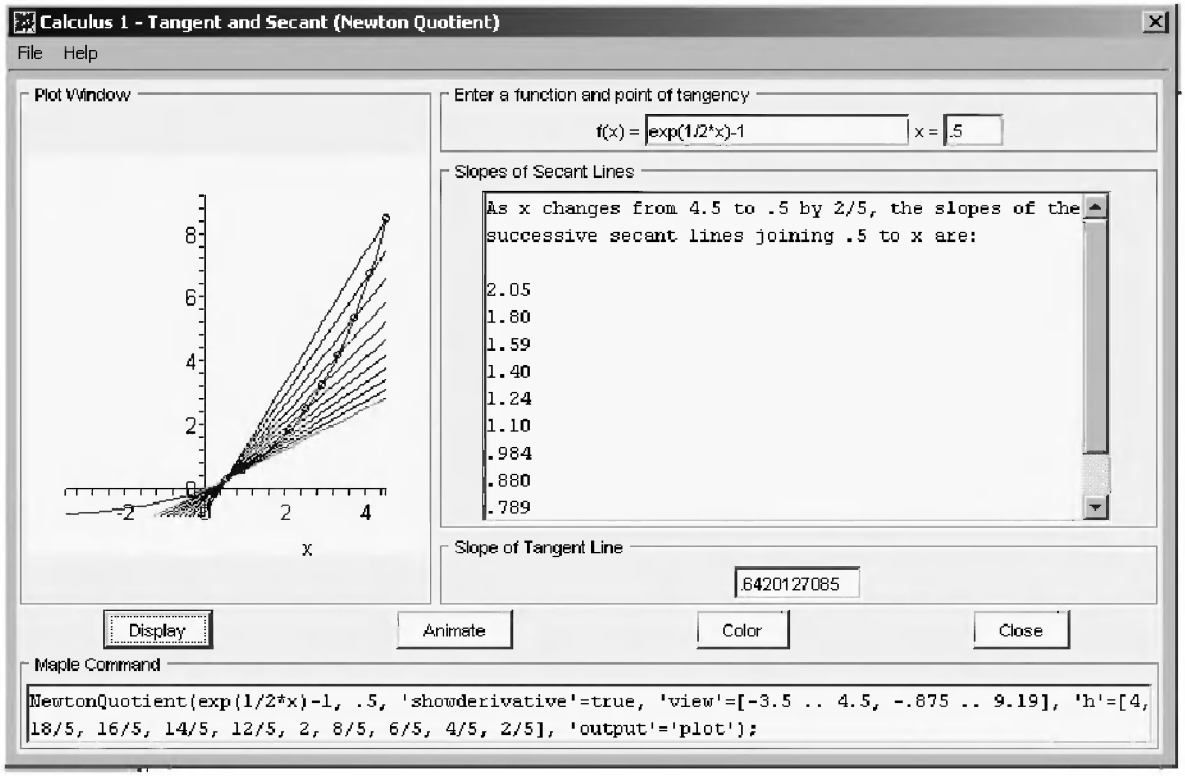


Рис. 4.28. Окно Maplet-инструмента для иллюстрации построения касательной к заданной точке и секущих линий

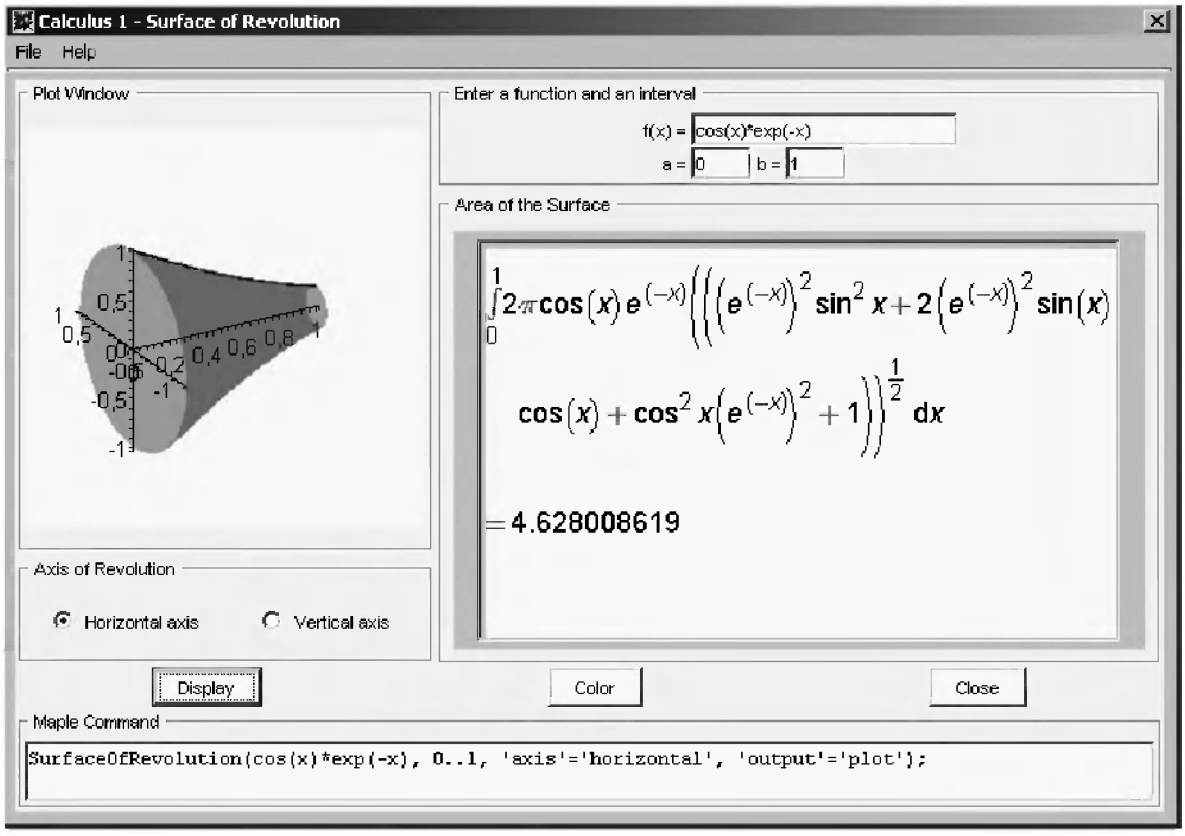


Рис. 4.29. Окно Maplet-инструмента для иллюстрации вычисления площади фигуры, полученной вращением отрезка кривой

Calculus-Single Variables ⇒ Surface of Revolution.... Работа с окном вполне очевидна. На графике строятся кривая функции и поверхность вращения этой кривой в 3D-прямоугольной системе координат. Вычисляется значение площади. Вычисления возможны и при вращении отрезка кривой вокруг оси $0y$.

4.7.7. Вычисление объема фигуры, полученной вращением отрезка кривой

Пусть отрезок кривой $f(x)$, при x в интервале $[a,b]$, вращается вокруг оси $0x$. Тогда *объем* полученной фигуры вращения равен:

$$V = \int_a^b \pi f^2(x)dx.$$

Для вычисления этого объема служит Maplet-инструмент Volume of Revolution. Его окно (рис. 4.30) открывается исполнением команды **Tools ⇒ Tutors ⇒ Calculus-Single Variables ⇒ Volume of Revolution...**. Работа с окном вполне очевидна. На графике строятся кривая функции и поверхность вращаения этой кривой в 3D-прямоугольной системе координат. Вычисляется значение объема полученной фигуры. Вычисления возможны и при вращении отрезка кривой вокруг оси $0y$.

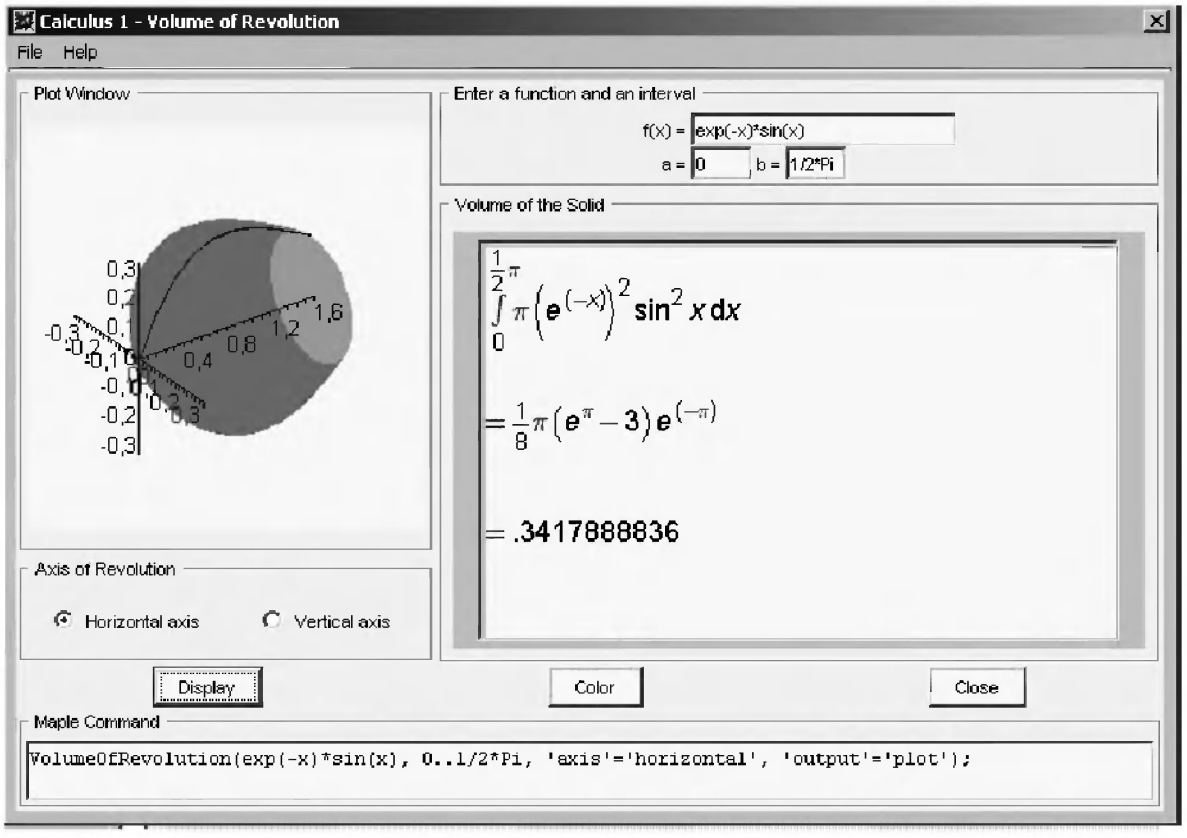


Рис. 4.30. Окно Maplet-инструмента для иллюстрации вычисления объема фигуры, полученной вращением отрезка кривой

4.7.8. Визуализация математических понятий в системе Mathcad

В популярной системе Mathcad нет специальных средств для визуализации ряда математических понятий и приложений математического анализа. Но это не мешает пользователю создавать их на базе имеющихся средств аналитических вычислений и графики. Рассмотрим тройку примеров такого рода.

На рис. 4.31 представлено вычисление площади A, ограниченной двумя кривыми $f1(x)$ и $f2(x)$ и значением x в интервале $[a,b]$, длины дуги кривой $f1(x)$ – L, объема фигуры, полученной вращением кривой f1(x) – V и площади поверхности вращения кривой $f2(x)$ – S. Этот документ является наглядным примером задания и вычисления интегралов.

$$f1(x) := x^3 \qquad f2(x) := x^2 \qquad a := 2 \qquad b := 4$$

$$A := \int_a^b (f1(x) - f2(x))\,dx \qquad A = 41.333$$

$$L := \int_a^b \sqrt{\left[1 + \left(\frac{d}{dx} f1(x)\right)^2\right]}\,dx \qquad L = 56.042$$

$$V := \int_a^b 2 \cdot \pi \cdot f2(x)^2\,dx \qquad V = 1.247 \times 10^3$$

$$S := \int_a^b 2 \cdot \pi \cdot f2(x) \cdot \sqrt{1 + \left(\frac{d}{dx} f2(x)\right)^2}\,dx \qquad S = 763.34$$

Рис. 4.31. Вычисление интегральных параметров кривых в системе Mathcad

Следующий пример, изображенный на рис. 4.32, показывает построение касательной и перпендикуляра к заданной точке кривой $f(x)$. В этом документе заданы уравнения касательной и перпендикуляра, выраженные через производные в заданной точке функции.

Наконец, на рис. 4.33 показана визуализация вычисления интегралов, в частности дано геометрическое представление определенного интеграла через площадь. На этом рисунке показано также контекстное меню правой клавиши мыши, которое демонстрирует возможность выбора метода интегрирования: Ромберга, адаптивного метода интегрирования с заданной погрешностью TOL, специально-

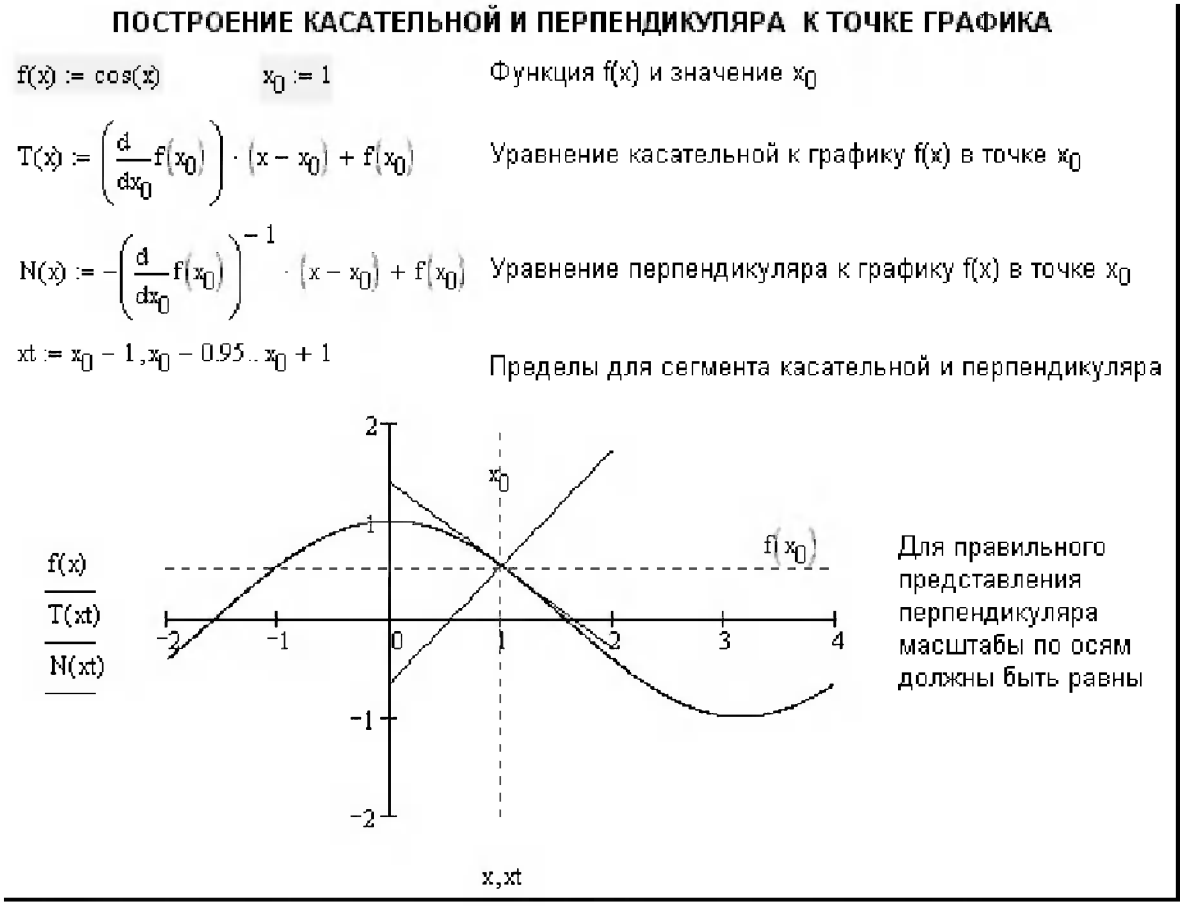


Рис. 4.32. Построение касательной и перпендикуляра в системе Mathcad

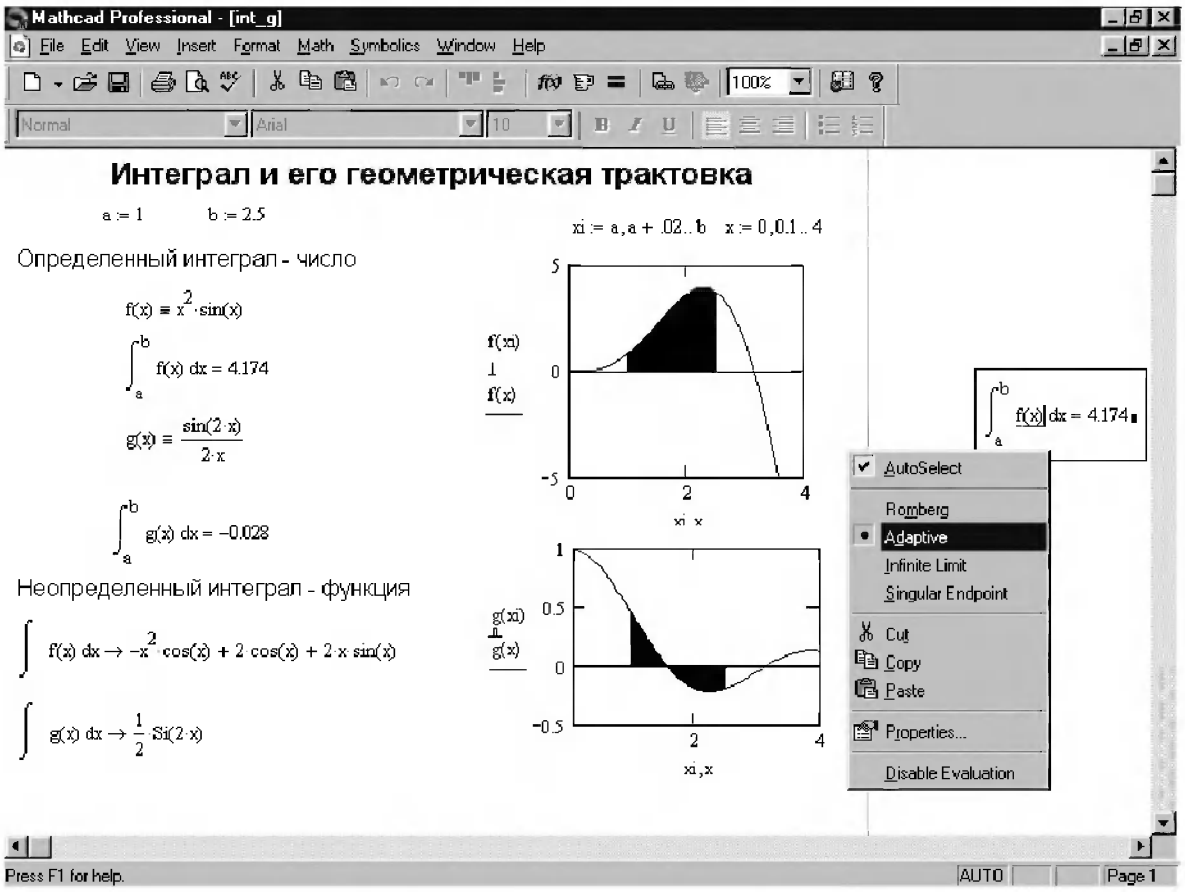


Рис. 4.33. Иллюстрация к вычислению интегралов в системе Mathcad

го метода вычисления интегралов с бесконечными пределами (или хотя бы с одним бесконечным пределом) и метода, учитывающего сингулярности (особенности) подынтегральной функции.

4.8. Решение уравнений и неравенств

4.8.1. Определение систем нелинейных уравнений и неравенств

Одиночное *нелинейное уравнение*, например трансцендентное, можно задать в одной из двух форм:

$F(x) = 0$ или $f(x) = expr$,

где *expr* – выражение. Второе уравнение всегда можно представить в виде $F(x) = f(x) - expr = 0$, то есть в форме первого уравнения.

Далеко не всегда нелинейные уравнения имеют аналитическое решение. Имеется множество методов *численного решения* нелинейных уравнений [71–74]. Некоторые из них представлены приведенными ниже итерационными выражениями:

Простых итераций	**Метод Ньютона**	**Метод Рыбакова**	**Модифицированный метод Ньютона**
$x_{n+1} = F(x_n)$	$x_{n+1} = x_n - F(x_n)/F'(x_n)$	$x_{n+1} = x_n - F(x_n)/M$	$x_{n+1} = x_n - \Delta x \cdot F(x_n)/(F(x_n + \Delta x) - F(x_n))$

Метод хорд	**Комбинированный метод секущих-хорд**
$x_{n+1} = x_n - (x_n - a)\cdot F(x_n)/(F(x_n) - F(a))$ при $a = \text{const}$ $x_{n+1} = x_n - (x_n - b)\cdot F(x_n)/(F(b) - F(x_n))$ при $b = \text{const}$	$x_{n+1} = x_n - (x_n - x_{n-1})\cdot F(x_n)/(F(x_n) - F(x_{n-1}))$

Все эти методы решают уравнение вида $F(x) = 0$, кроме метода *простых итераций*, требующего довольно тривиального приведения уравнения к виду $x = F(x)$. Метод хорд ищет решение в интервале от a до b, причем одна из границ считается неподвижной – в зависимости от того, какая обеспечивает переход к той или иной итерационной формуле. В других методах приходится задавать начальное приближение x_0. Итерации обычно ведутся до тех пор, пока соблюдается условие $(x_{n-1} - x_n) > \varepsilon$, где ε – заданная погрешность вычислений. Существует ряд более сложных методов (Эйткена-Стеффенсона, квадратичной интерполяции $F(x)$ и др.), но широкого распространения они не получили из-за сложности учета сходимости вычислений.

К сожалению, единообразия в обозначениях функций для решения систем нелинейных уравнений и неравенств нет. Чаще всего используются функции с именами **Solve** (Решение) и **Root** (Корень), но конкретное их назначение различно в разных системах. Вместо того чтобы постоянно оговаривать эти различия, мы просто рассмотрим конкретное назначение подобных функций в каждой из систем.

4.8.2. Основная функция solve в Maple

Решение линейных и нелинейных *уравнений* и *неравенств* – еще одна важная область математического анализа. Maple имеет мощные средства для такого решения. Так, для решения линейных и нелинейных уравнений в аналитическом виде используется достаточно универсальная и гибкая функция

`solve(eqn, var)` или **`solve({eqn1,eqn2,...},{var1,var2,...})`**,

где `eqn` – уравнение, содержащее функцию ряда переменных, `var` – переменная, по которой ищется решение. Если при записи `eqn` не используются знак равенства или знаки отношения, считается, что `solve` ищет корни уравнения `eqn=0`. Если `eqn` – полином, то `solve` вычисляет все корни полинома – как действительные, так и комплексные.

Характер решений можно изменить с помощью глобальных системных переменных:

- `_EnvExplicit` – при значении `true` выдает решение без применения конструкции `RootOf`;
- `_EnvAllSolutions` – при значении `true` задает выдачу всех решений;
- `_SolutionsMayBeLost` – при значении `true` дает решение, которое при обычном применении функции `solve` возвращает значения `NULL`;
- `_MaxSols` – задает максимальное число решений;
- `_EnvTryHard` – при значении `true` может дать компактное решение, но это может потребовать увеличения времени вычислений.

В решениях могут встречаться следующие обозначения:

- `_NN` – указывает на неотрицательные решения;
- `_B` – указывает на решения в бинарной форме;
- `_Z` – указывает на то, что решение содержит целые числа;
- `%N` – при текстовом формате вывода задает общие члены решения и обеспечивает более компактную форму его представления.

В форме `solve[subtopic]` возможны параметры `subtopic` функции `solve` следующих типов:

`floats` **`functions`** **`identity`** **`ineq`** **`linear`**
`radical` **`scalar`** **`series`** **`system`**

При решении систем уравнений они и список переменных задаются как множества, то есть в фигурных скобках. При этом и результат решения получается в виде множества. Чтобы преобразовать его к обычному решению, нужно использовать функцию `assign`, которая обеспечивает присваивание переменным значений, взятых из множества.

Функция `solve` старается дать решение в аналитическом виде. Это не означает, что ее нельзя использовать для получения корней уравнений в численном виде. Просто для этого придется использовать функции `evalf` или `convert`. Если результат решения представлен через функцию `RootOf`, то зачастую можно получить все корни с помощью функции `allvalues`.

4.8.3. Решение одиночных нелинейных уравнений

Решение одиночных нелинейных уравнений вида $f(x) = 0$ легко обеспечивается функцией `solve(f(x),x)`. Это демонстрируют следующие примеры:

```
> solve(x^3-2*x+1,x);
```

$$1, \frac{\sqrt{5}}{2} - \frac{1}{2}, -\frac{\sqrt{5}}{2} - \frac{1}{2}$$

```
> solve(x^(3/2)=3,x);
```

$$3^{(2/3)}$$

```
> evalf(%);
```

$$2.080083823$$

```
> solve(sqrt(ln(x))=2,x);
```

$$\mathbf{e}^4$$

```
> evalf(%);
```

$$54.59815003$$

Если уравнение записывается без правой части, то это означает, что она равна нулю. Часто бывает удобно представлять уравнение и его решение в виде отдельных объектов, отождествленных с определенной переменной:

```
> eq:=(2*x^2+x+3=0);
```

$$eq := 2x^2 + x + 3 = 0$$

```
> s:=[solve(eq,x)];
```

$$s := \left[-\frac{1}{4} + \frac{1}{4} I\sqrt{23}, -\frac{1}{4} - \frac{1}{4} I\sqrt{23} \right]$$

В частности, это позволяет легко проверить решение (даже если оно не одно, как в приведенном примере) подстановкой (`subs`):

```
> subs(x=s[1],eq);
```

$$2\left(-\frac{1}{4} + \frac{1}{4} I\sqrt{23}\right)^2 + \frac{11}{4} + \frac{1}{4} I\sqrt{23} = 0$$

```
> subs(x=s[2],eq);
```

$$2\left(-\frac{1}{4} - \frac{1}{4} I\sqrt{23}\right)^2 + \frac{11}{4} - \frac{1}{4} I\sqrt{23} = 0$$

```
> evalf(%);
```

$$0. + 0.\,I = 0$$

Сводящиеся к одному уравнению равенства вида $f_1(x) = f_2(x)$ также решаются функцией `solve(f1(x)=f2(x),x)`:

```
> solve(x^4=-x-1,x);
```

$$\text{RootOf}(\_Z^4 + \_Z + 1, index = 1), \text{RootOf}(\_Z^4 + \_Z + 1, index = 2),$$
$$\text{RootOf}(\_Z^4 + \_Z + 1, index = 3), \text{RootOf}(\_Z^4 + \_Z + 1, index = 4)$$

```
> evalf(%);
```

.7271360845 + .9340992895 *I*, –.7271360845 + .4300142883 *I*,
–.7271360845 – .4300142883 *I*, .7271360845 – .9340992895 *I*

```
> solve({exp(x)=sin(x)},x);
```

$$\{x = \text{RootOf}(\_Z - \ln(\sin(\_Z)))\}$$

```
> evalf(%);
```

$$\{x = .3627020561 - 1.133745919\, I\}$$

```
> solve(x^4=2*x,x);
```

$$0, 2^{(1/3)}, -\frac{1}{2}2^{(1/3)} + \frac{1}{2}I\sqrt{3}\,2^{(1/3)}, -\frac{1}{2}2^{(1/3)} - \frac{1}{2}I\sqrt{3}\,2^{(1/3)}$$

```
> evalf(%);
```

0., 1.259921050, –.6299605250 + 1.091123636 *I*, –.6299605250 – 1.091123636 *I*

Обратите внимание в этих примерах на эффективность применения функции `evalf`, позволяющей получить решения, выраженные через функцию `RootOf`, в явном виде. Возможны решения через специальные математические функции.

4.8.4. Решение тригонометрических уравнений

Функция `solve` может использоваться для решения *тригонометрических уравнений*:

```
> solve(sin(x)=.2,x);
```

.2013579208

```
> solve(sin(x)-1/2,x);
```

$$\frac{\pi}{6}$$

```
> solve(cos(x)=.5,x);
```

1.047197551

Однако из приведенных примеров видно, что при этом найдено только одно (главное) решение. Оно ищется в интервале $[-\pi, \pi]$. Периодичность тригонометрических функций и связанная с этим множественность решений оказались проигнорированы. Однако можно попытаться найти все периодические решения, выполнив следующую команду:

```
> _EnvAllSolutions:=true;
```

$$\_EnvAllSolutions := true$$

Указанная в ней системная переменная отвечает за поиск всех периодических решений, когда ее значение равно `true`, и дает поиск только главных решений при значении `false`, принятом по умолчанию. Так что теперь можно получить следующее:

```
> solve(sin(x)=1/2,x);
```

$$\frac{1}{6}\pi + \frac{2}{3}\pi\_B1\sim + 2\pi\_Z1\sim$$

Здесь вспомогательные переменные *_B1~* и *_Z1~* могут иметь только целочисленные значения (знак ~ означает, что на них наложено ограничение – в нашем случае в виде целочисленности возможных значений).

На рис. 4.34 показан более сложный случай решения нелинейного уравнения вида $f_1(x) = f_2(x)$, где $f_1(x) = \sin(x)$ и $f_2(x) = \cos(x) - 1$. Решение дано в графическом виде и в аналитическом для двух случаев – нахождения главных значений корней и нахождения всех корней. Обратите внимание на команду `_EnvAllSolutions:=true`, задающую поиск всех корней.

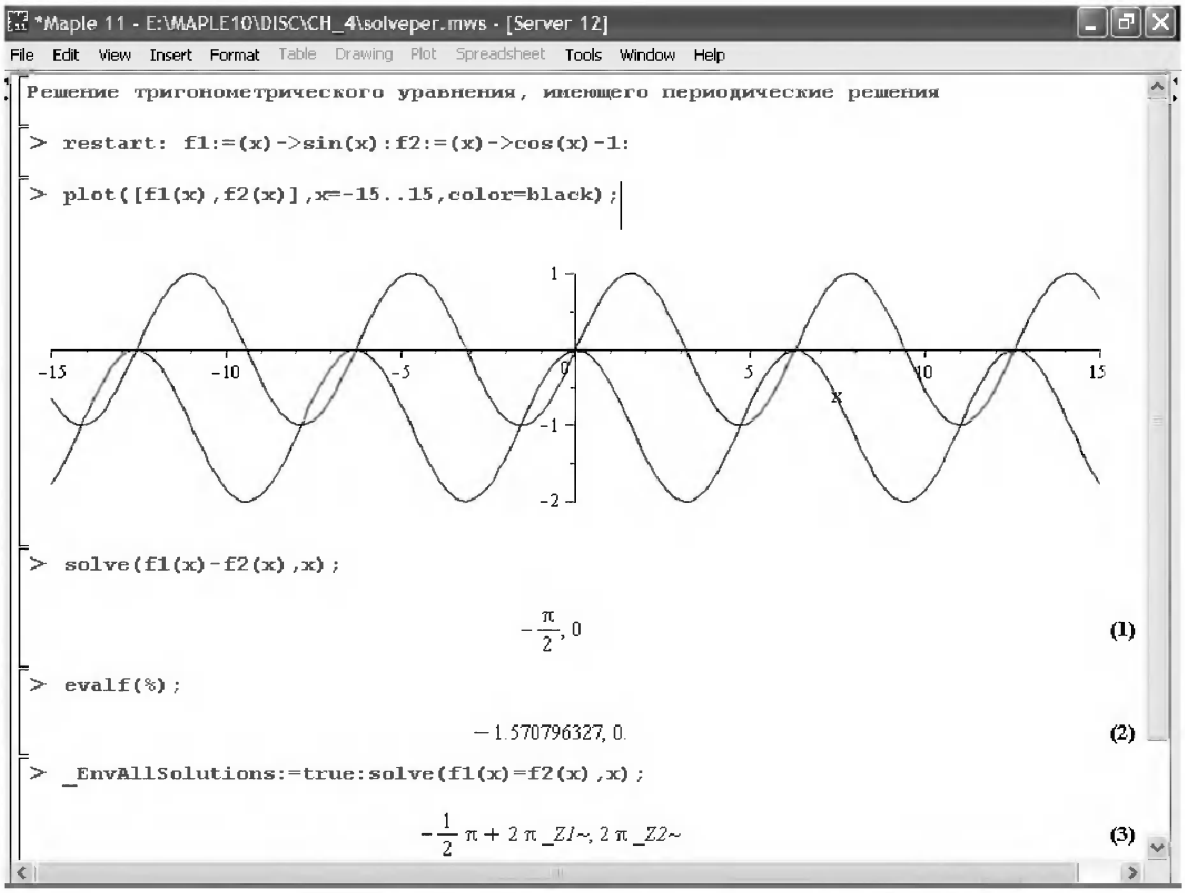


Рис. 4.34. Пример решения уравнения, имеющего периодические решения

В подобных решениях встречаются переменные *_B1~* и *_Z1~*, означающие ряд натуральных чисел. Благодаря этому через них можно представить периодически повторяющиеся решения.

Примеры решения уравнений с обратными тригонометрическими функциями показаны ниже:

```
> eqns := 2*arcsin(x) - arccos(5*x);
```

$$eqns := 2\arcsin(x) - \arccos(5x)$$

```
> solve( eqns, {x} );
```

$$\left\{x=\frac{1}{4}\frac{-5+\sqrt{33}}{\sqrt{\left(-\frac{5}{4}+\frac{1}{4}\sqrt{33}\right)^2-\frac{21}{8}+\frac{5}{8}\sqrt{33}}}\right\}$$

```
> eqns := arccos(x) - arctan(x/2);
```

$$eqns := \arccos(x)-\arctan\left(\frac{1}{2}x\right)$$

```
> solve( eqns, {x} );
```

$$\left\{x=\frac{\sqrt{-2+2\sqrt{2}}}{\sqrt{(\sqrt{2}-1)^2-2+2\sqrt{2}}}\right\}$$

4.8.5. Решение систем линейных уравнений

Для решения *систем линейных уравнений* созданы мощные матричные методы, которые будут описаны отдельно в главе 6. Однако функция `solve` также может успешно решать системы линейных уравнений, причем в символьном (аналитическом) виде. Такое решение в силу простоты записи функции может быть предпочтительным. Для решения система уравнений и перечень неизвестных задаются в виде множеств (см. приведенный ниже пример):

```
> eq1:=a*x+b*y=e; eq2:=c*x+d*y=f;
```

$$eq1 := a\,x + b\,y = e$$
$$eq2 := c\,x + d\,y = f$$

```
> solve({eq1,eq2},{x,y});
```

$$\left\{y=\frac{a\,f-c\,e}{a\,d-c\,b}, x=-\frac{b\,f-e\,d}{a\,d-c\,b}\right\}$$

В данном случае решение системы из двух линейных уравнений представлено в символьном виде.

Рисунок 4.35 дает еще два примера решения систем из двух линейных уравнений, на этот раз в численном виде. В первом примере функция `solve` возвращает решение в виде значений неизвестных x и y, а во втором – отказывается это делать. Причины этого из рис. 4.35 очевидны.

Решение систем из трех линейных уравнений также имеет наглядную геометрическую интерпретацию – в виде точки, в которой пересекаются три плоскости, каждая из которых описывается функцией двух переменных. Для наглядности желательно представить и линии пересечения плоскостей. Это позволяет сделать функция имплицитной трехмерной графики `implicitplot3d`, что и показано на рис. 4.36. Для объединения графиков площадей использована функция `display`.

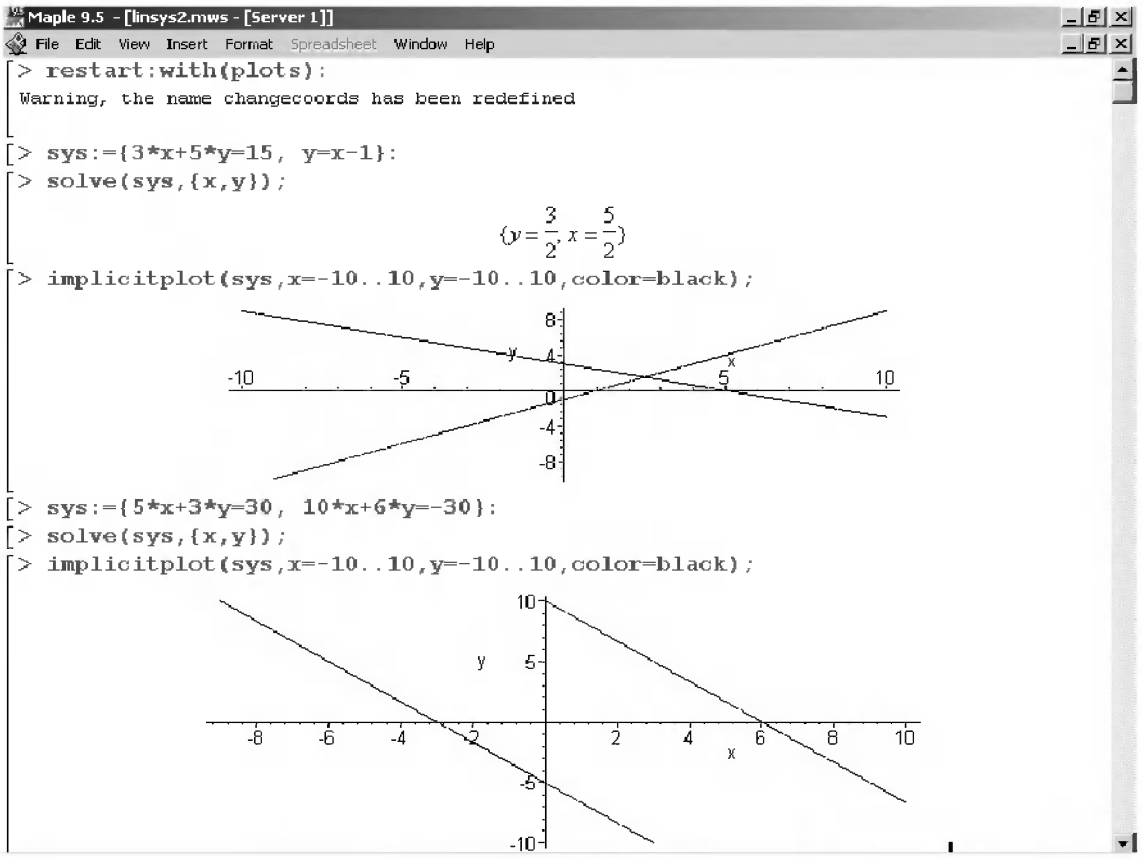


Рис. 4.35. Примеры решения системы из двух линейных уравнений с графической иллюстрацией

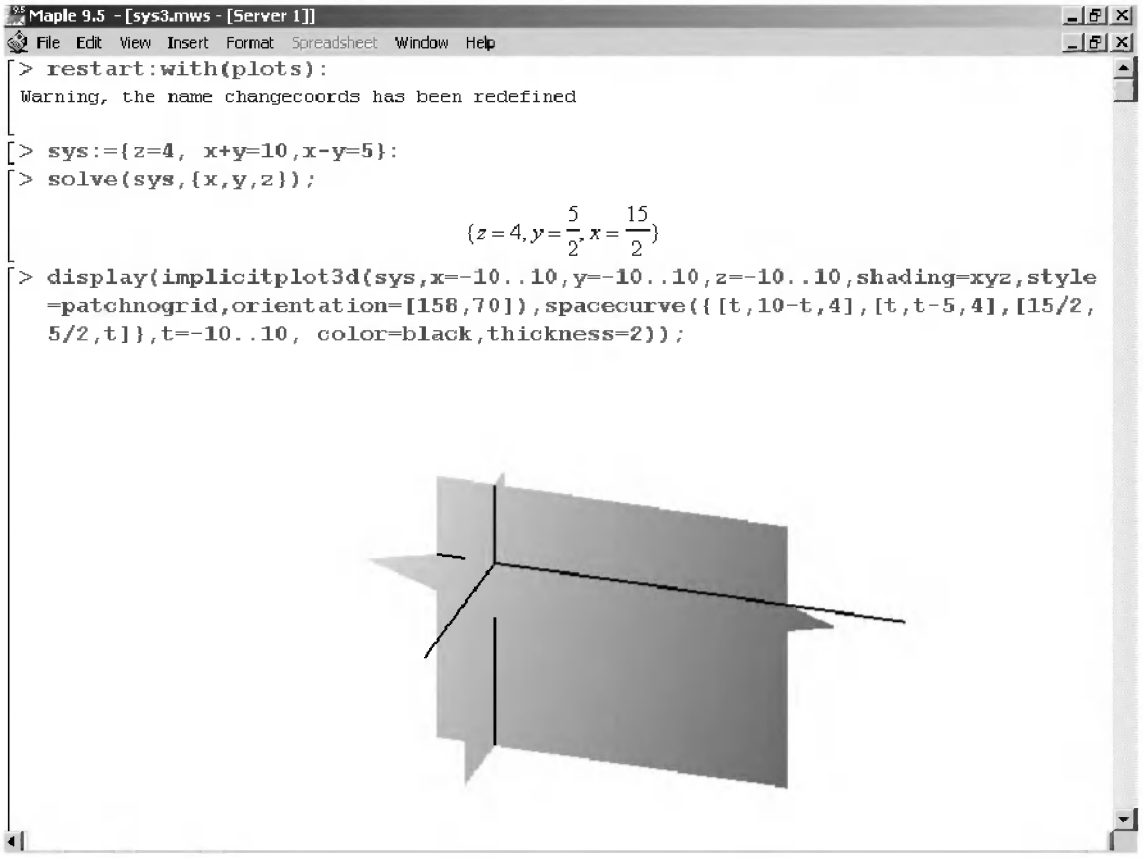


Рис. 4.36. Пример решения системы из трех линейных уравнений с графической иллюстрацией решения

Некоторые проблемы с решением систем из трех линейных уравнений иллюстрируют примеры, приведенные на рис. 4.37. В первом примере решения вообще нет. График показывает, в чем дело: линии пересечения плоскостей идут параллельно и нигде не пересекаются. Во втором примере все три плоскости пересекаются по одной линии.

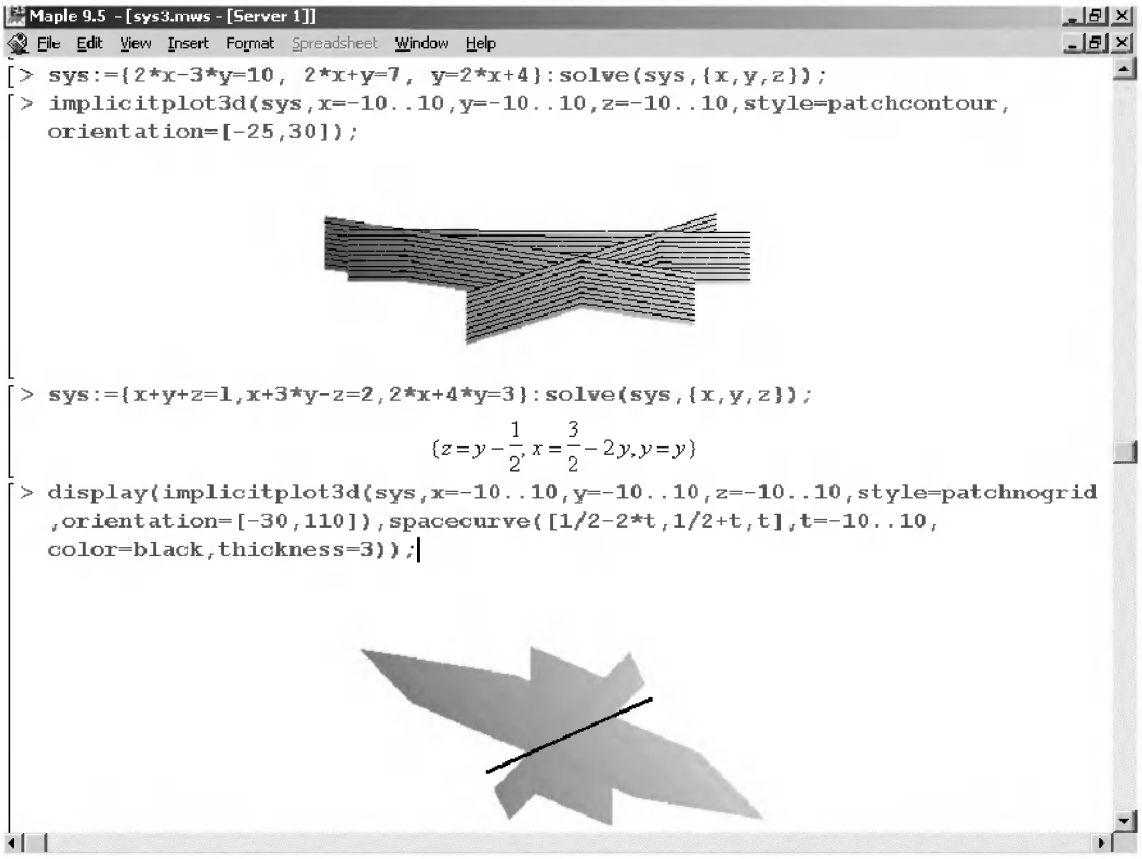


Рис. 4.37. Графическая иллюстрация особых случаев решения системы из трех линейных уравнений

Следующий пример показывает решение системы из четырех линейных уравнений:

```
> sys := { 4*x1 + 7*x2 – x3 + 3*x4 = 11,
         -2*x1 + 2*x2 – 6*x3 + x4 = 4,
         x1 – 3*x2 + 4*x3 – x4 = -3,
         3*x1 – 5*x2 – 7*x3 + 5*x4 = 8 }:
>  solve( sys, {x1, x2, x3, x4 } );
```

$$\{x2=\frac{8}{19}, x3=\frac{-81}{19}, x1=\frac{135}{19}, x4=\frac{-156}{19}\}$$

Эта система имеет решение, но его простая графическая иллюстрация уже невозможна.

Случай решения неполной системы уравнений (уравнений – 3, а неизвестных – 4) иллюстрирует следующий пример:

```
> sys := { x1 + 2*x2 + 3*x3 + 4*x4 = 51,
         x1 - 3*x2 + 4*x3 +  x4  = 32,
         x1 + 2*x2 - 6*x3 +  x4  = -23   }:
> solve( sys, {x1, x2, x3, x4 } );
```

$$\{x4 = \frac{74}{3} - 3x3, x2 = -11 + 2x3, x3 = x3, x1 = -\frac{77}{3} + 5x3\}$$

Как видно из приведенных примеров, функция `solve` неплохо справляется с решением систем линейных уравнений.

4.8.6. Решение систем нелинейных и трансцендентных уравнений

Функция `solve` может использоваться для решения *систем нелинейных и трансцендентных уравнений*. Для этого система уравнений и перечень неизвестных задаются в виде множеств. Ниже приведены примеры решения уравнений:

```
> restart;
> solve({x*y=a,x+y=b},{x,y});
```

$$\{y = \text{RootOf}(\_Z^2 - \_Zb + a), x = -\text{RootOf}(\_Z^2 - \_Zb + a) + b\}$$

```
> allvalues(%);
```

$$\{y = \frac{1}{2}b + \frac{1}{2}\sqrt{b^2 - 4a}, x = \frac{1}{2}b - \frac{1}{2}\sqrt{b^2 - 4a}\},$$

$$\{y = \frac{1}{2}b - \frac{1}{2}\sqrt{b^2 - 4a}, x = \frac{1}{2}b + \frac{1}{2}\sqrt{b^2 - 4a}\}$$

```
> s:=solve({x*y=2,x+y=3},{x,y});
```

$$s := \{y = 1, x = 2\}, \{y = 2, x = 1\}$$

```
> assign(s);x;y;
```

$$1$$

$$2$$

```
> unassign('x');y:='y';
```

$$y := y$$

```
> [x,y];
```

$$[x, y]$$

В этих примерах хорошо видна техника работы с функциями `solve` и `assign`. В конце примеров показаны восстановление неопределенного статуса переменных `x` и `y` с помощью функции `unassign` и снятие определения переменных с помощью заключения их в прямые апострофы.

Приведем еще один пример решения системы нелинейных уравнений с проверкой правильности решения с помощью функции `eval`:

```
> eqs:={2*x+4*y=6,y+1/x=1};
```

$$eqs := \{y + \frac{1}{x} = 1, 2x + 4y = 6\}$$

```
> r:=solve(eqs,{x,y});
```

$$r := \{y = 2, x = -1\}, \{y = \frac{1}{2}, x = 2\}$$

```
> eval(eqs,r[1]);
```

$$\{1 = 1, 6 = 6\}$$

```
> eval(eqs,r[2]);
```

$$\{1 = 1, 6 = 6\}$$

Для проверки всех решений можно использовать также функции `map` и `subs`:

```
> map(subs,[r],eqs);
```

$$[\{1 = 1, 6 = 6\}, \{1 = 1, 6 = 6\}]$$

Maple имеет и еще ряд возможностей для проверки решений, но представленных обычно вполне достаточно для такой проверки. Ее следует принять за правило при выполнении решений уравнений.

4.8.7. Функция RootOf

В решениях уравнений нередко появляется функция `RootOf`, означающая, что корни нельзя выразить в радикалах. Эта функция применяется и самостоятельно в виде `RootOf(expr)` или `RootOf(expr, x)`, где `expr` – алгебраическое выражение или равенство, `x` – имя переменной, относительно которой ищется решение. Если переменная `x` не указана, ищется универсальное решение по переменной `_Z`. Когда `expr` задано не в виде равенства, решается уравнение `expr=0`. Для получения решений вида `RootOf` в явном виде может использоваться функция `allvalues`.

Примеры применения функции `RootOf`:

```
> RootOf(a*b^2+a/b,b);
```

$$\mathrm{RootOf}(\_Z^3 + 1)$$

```
> allvalues(%);
```

$$-1, \frac{1}{2} + \frac{1}{2} I\sqrt{3}, \frac{1}{2} - \frac{1}{2} I\sqrt{3}$$

```
> RootOf(x^3-1,x)mod 7;
```

$$\mathrm{RootOf}(\_Z^3 + 6)$$

```
> allvalues(%);
```

$$-6^{(1/3)}, \frac{1}{2}6^{(1/3)} - \frac{1}{2} I\sqrt{3}\, 6^{(1/3)}, \frac{1}{2}6^{(1/3)} + \frac{1}{2} I\sqrt{3}\, 6^{(1/3)}$$

```
> evalf(%);
```

$$-1.817120593, .9085602965 - 1.573672596\, I, .9085602965 + 1.57372596\, I$$

```
> RootOf(x^2-2*x+1,x)mod 5;
```

$$1$$

Итак, функция `RootOf` является эффективным способом представления решения в компактном виде. Как уже отмечалось, наряду с самостоятельным применением она часто встречается в составе результатов решения нелинейных уравнений.

4.8.8. Решение уравнений со специальными функциями

К важным достоинствам Maple относится возможность решения уравнений, содержащих специальные функции как в записи исходных выражений, так и в результатах решения. Приведем несколько примеров такого рода:

```
> restart:eqn := Psi(3*x-99) - Psi(3*x-100) + 3/x^2=0;
```

$$eqn := \Psi(3x-99) - \Psi(3x-100) + \frac{3}{x^2} = 0$$

```
> r:=solve( eqn, {x} );
```

$$r := \{x = -\frac{9}{2} + \frac{\sqrt{1281}}{2}\}, \{x = -\frac{9}{2} - \frac{\sqrt{1281}}{2}\}$$

```
> eqn := max(x,3*x-12)=min(10*x+8,22-x);
```

$$eqn := \max(x, -12 + 3x) = \min(10x + 8, 22 - x)$$

```
> r:=solve( eqn, {x} );
```

$$r := \{x = \frac{-8}{9}\}, \{x = \frac{17}{2}\}$$

```
> map(subs,[r],eqn);
```

$$\left[\max\left(\frac{-8}{9}, \frac{-44}{3}\right) = \min\left(\frac{-8}{9}, \frac{206}{9}\right), \max\left(\frac{17}{2}, \frac{27}{2}\right) = \min\left(93, \frac{27}{2}\right)\right]$$

```
> eqn := LambertW(3*x)=ln(x);
```

$$eqn := \mathrm{LambertW}(3x) = \ln(x)$$

```
> r:=solve( eqn, {x} );
```

$$r := \{x = \mathbf{e}^3\}$$

```
> map(subs,[r],eqn);
```

$$\{\mathrm{LambertW}(3\mathbf{e}^3) = \ln(\mathbf{e}^3)]$$

```
> evalf(map(subs,[r],eqn));
```

$$[3.000000000 = 3.000000000]$$

Полезно обратить внимание на не вполне обычную проверку правильности решений. Иногда при этом выводятся значения левой и правой частей уравнения, требующие осмысления полученных результатов.

4.8.9. Решение неравенств

Неравенства в математике встречаются почти столь же часто, как и равенства. Они вводятся знаками отношений, например > (больше), < (меньше) и т. д. *Решение неравенств* существенно расширяет возможности функции `solve`. При этом неравенства задаются так же, как и равенства. Приведенные на рис. 4.38 примеры поясняют технику решения неравенств.

Из приведенных примеров очевидна форма решений – представлены критические значения аргумента вплоть до невключаемых значений области действия неравенства (они указываются словом `Open`). Всегда разумным является построе-

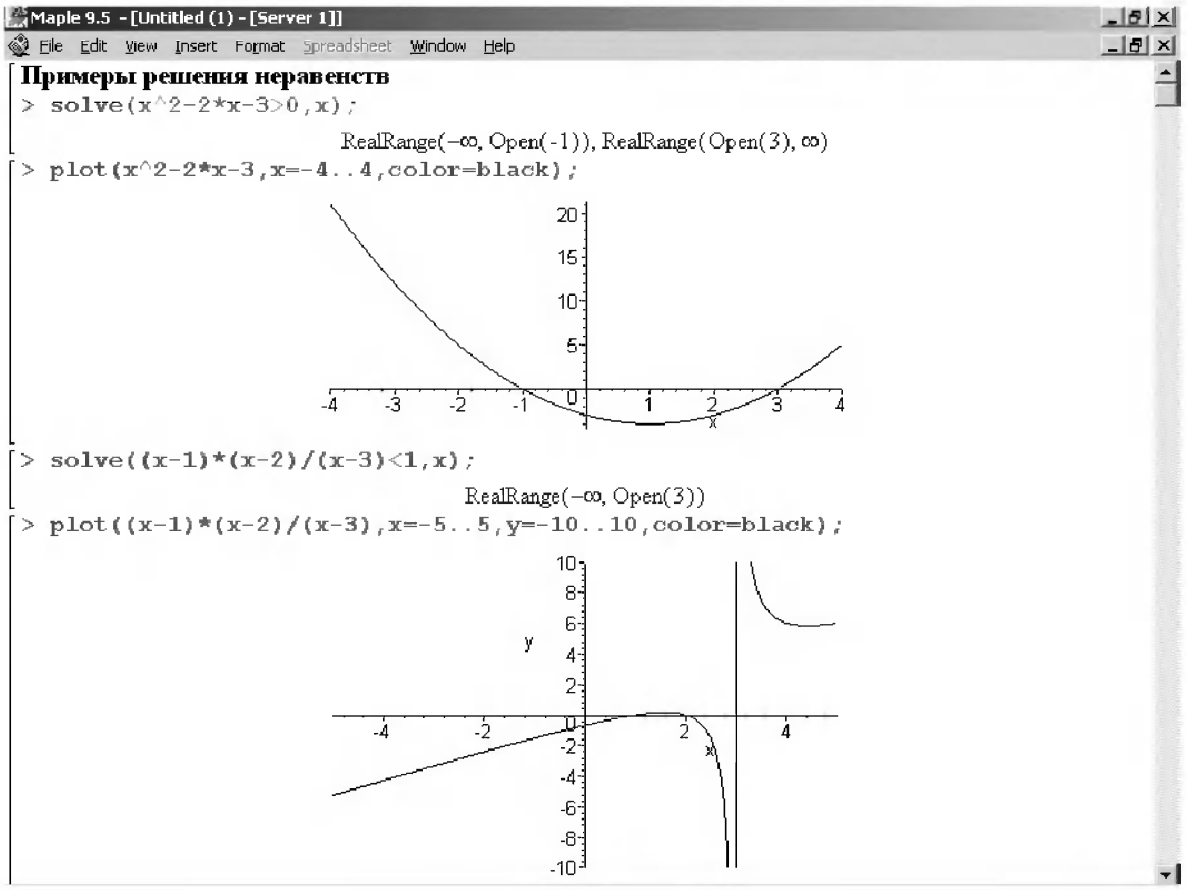


Рис. 4.38. Примеры, иллюстрирующие решение неравенств

ние графика выражения, которое задает неравенство, – это позволяет наглядно убедиться в правильности решения.

Приведем еще несколько примеров решения неравенств в аналитической форме:

```
> solve(5*x>10,x);
```

$$\text{RealRange}(\text{Open}(2), \infty)$$

```
> solve(5*x>=10,x);
```

$$\text{RealRange}(2, \infty)$$

```
> solve(ln(x)>2,x);
```

$$\text{RealRange}(\text{Open}(\mathbf{e}^2), \infty)$$

```
> solve(a*x>b,{x});
```

$$\{-\text{signum}(a)x < -\frac{\text{signum}(a)b}{a}\}$$

```
> eqns := abs( (z+abs(z+2))^2-1 )^2 = 9;
```

$$eqns := |(z + |z + 2|)^2 - 1|^2 = 9$$

```
> solve( eqns, {z} );
```

$$\{z = 0\}, \{z \le -2\}$$

```
> evalf(%);
```

$$\{-2.617866616 \le x, x \le -1.487962064\}, \{.5398352768 \le x\}$$

```
> solve({x*y*z>0,x>-1,y+z>10},{x,y,z});
```

$$\{z = 0, -1 < x, 10 < y\}, \{y = 0, -1 < x, 10 < z\}$$

В последнем примере показано решение системы неравенств. При этом выдаются области определения нескольких переменных.

4.8.10. Решение функциональных уравнений

Решение функционального уравнения, содержащего в составе равенства некоторую функцию $f(x)$, заключается в нахождении этой функции. Для этого можно использовать функцию `solve`, что демонстрируют приведенные ниже примеры:

```
> A:=solve(f(x)^2-x+1,f);
```

$$A := \mathbf{proc}(x)\mathrm{RootOf}(\_Z\text{^}2 - x + 1, label = \_L7)\ \mathbf{end\ proc}$$

```
> convert(A(x),radical);
```

$$\sqrt{x-1}$$

```
> allvalues(%);
```

$$\sqrt{x-1}$$

```
> B:=solve(f(x)*x=ln(x^2),f);
```

$$B := \mathbf{proc}(x)\ln(x\text{^}2)/x\ \mathbf{end\ proc}$$

```
> convert(B(x),radical);
```

$$\frac{\ln(x^2)}{x}$$

```
> C:=solve(f(x)*x^2=a*x^2+b*x+c,f);
```

$$C := \mathbf{proc}(x)(a\times x\text{^}2 + b\times x + c)/x\text{^}2\ \mathbf{end\ proc}$$

```
> convert(C(x),radical);
```

$$\frac{ax^2+bx+c}{x^2}$$

4.8.11. Решение уравнений с линейными операторами

Maple позволяет решать *уравнения с линейными операторами*, например с операторами суммирования рядов и дифференцирования. Ограничимся одним примером такого рода:

```
> S := sum( (a+b*exp(x[i])-y[i])^2, i=0..n );
```

$$S := (n+1)a^2 + \left(\sum_{i=0}^{n}\left(2b\mathbf{e}^{x_i}a - 2ay_i + b^2(\mathbf{e}^{x_i})^2 - 2b\mathbf{e}^{x_i}y_i + y_i^2\right)\right)$$

```
> eqns := { diff(S, a), diff(S,b)};
```

$$eqns := \left\{\sum_{i=0}^{n}\left(2\mathbf{e}^{x_i}a + 2b(\mathbf{e}^{x_i})^2 - 2\mathbf{e}^{x_i}y_i\right), 2(n+1)a + \sum_{i=0}^{n}\left(2\mathbf{e}^{x_i} - 2y_i\right)\right\}$$

```
> solve( eqns, {a, b} );
```

$$\begin{cases} b = -\dfrac{n\left(\sum_{i=0}^{n} \mathbf{e}^{x_i} y_i\right) + \left(\sum_{i=0}^{n} \mathbf{e}^{x_i} y_i\right) - \left(\sum_{i=0}^{n} y_i\right)\left(\sum_{i=0}^{n} \mathbf{e}^{x_i}\right)}{\left(\sum_{i=0}^{n} \mathbf{e}^{x_i}\right)^2 - \left(\sum_{i=0}^{n} (\mathbf{e}^{x_i})^2\right) n - \left(\sum_{i=0}^{n} (\mathbf{e}^{x_i})^2\right)}, \\ a = \dfrac{-\left(\sum_{i=0}^{n} (\mathbf{e}^{x_i})^2\right)\left(\sum_{i=0}^{n} y_i\right) + \left(\sum_{i=0}^{n} \mathbf{e}^{x_i}\right)\left(\sum_{i=0}^{n} \mathbf{e}^{x_i} y_i\right)}{\left(\sum_{i=0}^{n} \mathbf{e}^{x_i}\right)^2 - \left(\sum_{i=0}^{n} (\mathbf{e}^{x_i})^2\right) n - \left(\sum_{i=0}^{n} (\mathbf{e}^{x_i})^2\right)} \end{cases}$$

4.8.12. Решение в численном виде – функция fsolve

Для получения *численного решения* нелинейного уравнения или системы нелинейных уравнений в форме вещественных чисел удобно использовать функцию

```
fsolve( eqns, vars, options )
```

Эта функция может быть использована со следующими параметрами:

- `complex` – находит один или все корни полинома в комплексной форме;
- `fulldigits` – задает вычисления для полного числа цифр, заданного функцией `Digits`;
- `maxsols=n` – задает нахождение только `n` корней;
- `interval` – задается в виде `a..b`, или `x=a..b`, или `{x=a..b, y=c..d, ...}` и обеспечивает поиск корней в указанном интервале.

Функция `fsolve` дает решения сразу в форме вещественных или комплексных чисел, что и показывают следующие примеры:

```
> fsolve(sin(x)=Pi/4,x);
```

.9033391108

```
> fsolve(sin(x)=1/2,x=4..8);
```

6.806784083

```
> fsolve(2*x^2+x-1=10,x);
```

-2.608495283, 2.108495283

```
> fsolve(x^5-x,x,complex);
```

-1.000000000, -1.000000000 *I*, 1.00000000 *I*, 1.000000000

```
> eqns := abs(x)*x+exp(x) > 0;
```

$$eqns := 0 < |x|x + \mathbf{e}^x$$

```
> solve( eqns, {x} );
```

$$\{-2\,\mathrm{LambertW}\left(\frac{1}{2}\right) < x\}$$

```
> f := sin(x+y) - exp(x)*y = 0: g := x^2 - y = 2:
```

$$\text{fsolve}(\{f,g\},\{x,y\},\{x=-1..1,y=-2..0\});$$

$$\{x = -.6687012050,\ y = -1.552838698\}$$

Заметим, что *локализация поиска корней* в заданном интервале позволяет отыскивать такие решения, которые не удается получить с помощью функций `solve` и `fsolve` в обычном применении. В последнем из приведенных примеров дается решение системы нелинейных уравнений, представленных уравнениями `f` и `g`.

Чтобы еще раз показать различие между функциями `solve` и `fsolve`, рассмотрим пример решения с их помощью одного и того же уравнения $\text{erf}(x) = 1/2$:

```
> solve(erf(x)=1/2,x);
```

$$\text{RootOf}(2\,\text{erf}(\_Z) - 1)$$

```
> fsolve(erf(x)=1/2);
```

$$.4769362762$$

Функция `solve` в этом случае находит нетривиальное решение в комплексной форме через функцию `RootOf`, тогда как функция `fsolve` находит обычное приближенное решение.

4.8.13. Решение рекуррентных уравнений – rsolve

Функция `solve` имеет ряд родственных функций. Одну из таких функций – `fsolve` – мы рассмотрели выше. В справке системы Maple 9 можно найти ряд и других функций, например `rsolve` для решения *рекуррентных уравнений*, `isolve` для решения *целочисленных уравнений*, `msolve` для решения по модулю m и т. д. Функция `rsolve` имеет формы:

```
rsolve(eqns, fcns)          rsolve(eqns, fcns, 'genfunc'(z))
rsolve(eqns, fcns, 'makeproc')
```

Здесь `eqns` – одиночное уравнение или система уравнений, `fcns` – функция, имя функции или множество имен функций, `z` – имя, генерирующее функциональную переменную. Ниже представлены примеры применения функции `rsolve`:

```
> restart;
> rsolve({f(n)=-3*f(n-1)-2*f(n-2),f(1..2)=1},{f});
```

$$\{f(n) = -3(-1)^n + (-2)^n\}$$

```
> rsolve({y(n)=n*y(n-1),y(0)=1},y);
```

$$\Gamma(n+1)$$

```
> rsolve({y(n)*y(n-1)+y(n)-y(n-1)=0,y(0)=a},y);
```

$$\frac{a}{1+n\,a}$$

```
> rsolve({F(n)=F(n-1)+F(n-2),F(1..2)=1},F,'genfunc'(x));
```

$$-\frac{x}{-1+x+x^2}$$

А теперь приведем результат вычисления функцией `rsolve` *n*-го *числа Фибоначчи*. Оно задается следующим выражением:

```
> eq1 := {f(n+2) = f(n+1) + f(n) , f(0) = 1 , f(1) = 1};
```

$$eq1 := \{f(n+2) = f(n+1) + f(n), f(0) = 1, f(1) = 1\}$$

```
> a1:=rsolve(eq1,f);
```

$$a1 := \left(-\frac{\sqrt{5}}{10}+\frac{1}{2}\right)\left(\frac{1}{2}-\frac{\sqrt{5}}{2}\right)^{n}+\left(\frac{\sqrt{5}}{10}+\frac{1}{2}\right)\left(\frac{1}{2}+\frac{\sqrt{5}}{2}\right)^{n}$$

В нем задана рекуррентная формула для числа Фибоначчи – каждое новое число равно сумме двух предыдущих чисел, причем нулевое и первое числа равны 1.

4.8.14. Решение уравнений в целочисленном виде – isolve

Иногда бывает нужен результат в форме только целых чисел. Для этого используется функция `isolve(eqns, vars)`, дающая решение в виде целых чисел. Приведем примеры *решения уравнений в целочисленном виде*:

```
> isolve({2*x-5=3*y});
```

$$\{x = 4 + 3\_Z1, y = 1 + 2\_Z1\}$$

```
> isolve(y^4-z^2*y^2-3*x*z*y^2-x^3*z);
```

```
                4                 2       2       2                   3
      _Z3 _Z2          _Z3 _Z1  (-_Z2  + _Z1 )          _Z3 _Z1  _Z2
{z = --------, x = ----------------------, y = - ------------}
        %1                    %1                      %1

                2       2       2        3          4
%1 := igcd(_Z1  (-_Z2  + _Z1 ), -_Z1  _Z2, _Z2 )
```

$$\{z = \frac{\_Z3\ \_Z2^4}{\mathrm{igcd}(-\_Z1^2(\_Z2^2-\_Z1^2), -\_Z1^3\ \_Z2, \_Z2^4)},$$

$$x = -\frac{\_Z3\ \_Z1^2(\_Z2^2-\_Z1^2)}{\mathrm{igcd}(-\_Z1^2(\_Z2^2-\_Z1^2), -\_Z1^3\ \_Z2, \_Z2^4)},$$

$$y = -\frac{\_Z3\ \_Z1^3\ \_Z2}{\mathrm{igcd}(-\_Z1^2(\_Z2^2-\_Z1^2), -\_Z1^3\ \_Z2, \_Z2^4)}\}$$

Здесь вывод представлен с помощью вспомогательных переменных *_ZI*.

4.8.15. Функция msolve

Функция `msolve(eqns,vars,m)` или `msolve(eqns,m)` обеспечивает решение вида $Z \bmod m$ (то есть при подстановке решения левая часть при делении на m дает остаток, равный правой части уравнения). При отсутствии решения возвращается объект `NULL` (*пустой список*).

Ниже даны примеры использования функции `msolve`:

```
> msolve({3*x-4*y=1,7*x+y=2},12);
```

$$\{y = 5, x = 3\}$$

```
> msolve(2^i=3,19);
```

$$\{i = 13 + 18\ \_Z1\sim\}$$

```
> msolve(8^j=2,x,17);
```

$$\{j = 3 + 8x\}$$

4.8.16. Новый пакет расширений RootFinding и работа с ним

В Maple 9.5 был введен небольшой новый пакет RootFinding, предназначенный для поиска корней выражений, уравнений и систем с ними. Пакет примечателен тем, что позволяет задавать пределы для поиска в виде не только действительных, но и комплексных чисел. Кроме того, он обеспечивает вычисление многих корней, если таковые существуют. Пакет загружается командой:

```
> with(RootFinding)
```

Он содержит всего четыре новые функции:

Analytic(f, z, a+c*I..b+d*I, ...) или **Analytic(f, z, re=a..b, im=c..d, ...)**
AnalyticZerosFound()
BivariatePolynomial(fg, varlist, options)
Homotopy(eqns, options)

Назначение этих функций вполне очевидно из их названий. В справке по пакету имеется множество примеров применения пакета. В связи с этим ограничимся приведением нескольких примеров на применение представленных выше функций:

```
> f := cos(sin(x))-1;
```

$$f := \cos(\sin(x)) - 1$$

```
> Analytic( f, x, -I/2..1+I );
```

$$2.68676267894010^{-9},\ -1.12009531101910^{-8}$$

```
> Analytic( 23*x^5 + 105*x^4 - 10*x^2 + 17*x, x, re=-5..1, im=-1..1 );
```

$$-0.6371813185311,\ 0. + 0.\ I,\ 0.3040664542849 + 0.4040619057518\ I,\ -4.536168981343,\ 0.3040664542849 - 0.4040619057518\ I$$

```
> Analytic(sin(x)-x^2, x, re=-6..6, im=-6..6, digits=10);
```

$$0. + 0.\ I,\ 0.8767262154,\ -2.85569323 - 3.81699466\ I,\ -2.85569323 + 3.81699466\ I$$

```
> BivariatePolynomial( [5*u*v-2, u^2 + v^2 - 1], [u,v], exact=true );
```

$$[1/5*5\text{^}(1/2),\ 2/5*5\text{^}(1/2)],\ [-1/5*5\text{^}(1/2),\ -2/5*5\text{^}(1/2)],\ [2/5*5\text{^}(1/2),\ 1/5*5\text{^}(1/2)],\ [-2/5*5\text{^}(1/2),\ -1/5*5\text{^}(1/2)]$$

```
> Homotopy([2*x*y-2, x^2 + y^2 - 1]);
```

$$[[x = -0.8660254038 - 0.5000000000\ I,\ y = -0.8660254038 + 0.5000000000\ I],$$
$$[x = -0.8660254042 + 0.4999999999\ I,\ y = -0.8660254034 - 0.4999999999\ I],$$
$$[x = 0.8660254042 - 0.4999999999\ I,\ y = 0.8660254034 + 0.4999999999\ I],$$
$$[x = 0.8660254038 + 0.5000000000\ I,\ y = 0.8660254038 - 0.5000000000\ I]]$$

На этом мы завершаем рассмотрение функций системы Maple для решения уравнений, неравенств и систем с ними.

4.9. Решение уравнений и неравенств в других СКМ

4.9.1. Решение уравнений в Mathematica

Для решения уравнений в символьном виде Mathematica также имеет функцию **Solve**:

- **Solve[eqns, vars]** – решает уравнение или систему уравнений **eqns** относительно переменных **vars**.
- **Solve[eqns, vars, elims]** – решает уравнение по переменным **vars**, исключая переменные **elims**.

Применение функции **Solve** иллюстрируют следующие примеры:

```
Solve[a*x^2+b*x+c==0,x]
```

$$\left\{\left\{x\to \frac{-b-\sqrt{b^2-4ac}}{2a}\right\},\left\{x\to \frac{-b+\sqrt{b^2-4ac}}{2a}\right\}\right\}$$

```
Solve[2*x^2-5*x-10==0,x]
```

$$\left\{\left\{x\to \frac{1}{4}\left(5-\sqrt{105}\right)\right\},\left\{x\to \frac{1}{4}\left(5+\sqrt{105}\right)\right\}\right\}$$

```
Solve[{y==x^2,x==(a+b)},{x,y}]
```

$$\{\{y\to a^2+2ab+b^2,\ x\to a+b\}\}$$

Первый пример дает аналитическое решение для полного квадратного уравнения с буквенными коэффициентами, а второй – для такого же уравнения, но с численными коэффициентами. Третий пример иллюстрирует применение функции **Solve** для реализации подстановки символьного значения переменной *x* в уравнение *y=x^2*.

Следующий пример показывает решение с помощью функции **Solve** системы из двух нелинейных уравнений:

```
Solve[{x^2+y==a,y^2==a+b},{x,y}]
```

$$\left\{\left\{x\to -\sqrt{a-\sqrt{a+b}},\ y\to \sqrt{a+b}\right\},\right.$$
$$\left\{x\to \sqrt{a-\sqrt{a+b}},\ y\to \sqrt{a+b}\right\},$$
$$\left\{x\to -\sqrt{a+\sqrt{a+b}},\ y\to -\sqrt{a+b}\right\},$$
$$\left.\left\{x\to \sqrt{a+\sqrt{a+b}},\ y\to -\sqrt{a+b}\right\}\right\}$$

В большинстве случаев эти функции решают нужные уравнения. Однако можно уточнить путь решения с помощью ряда опций и установок:

- **InverseFunctions** – опция, указывающая, следует ли использовать обратные функции (по умолчанию Automatic).
- **MakeRules** – опция, указывающая, должен ли результат быть представлен как объект типа AlgebraicRulesData (по умолчанию False).
- **Method** – опция, устанавливающая применяемый при решении алгоритм (возможны алгоритмы 1, 2 и 3 – по умолчанию 3).
- **Mode** – опция, определяющая характер решения:
  - **Generic** – основная установка (по умолчанию).
  - **Racional** – устанавливает решение в рациональном виде.
  - **Modular** – устанавливает, что уравнения должны разрешаться только по модулю целого числа.
- **VerifySolutions** – устанавливает, следует ли проводить проверку полученных решений и удаление посторонних решений.

Многие нелинейные уравнения и системы нелинейных уравнений в принципе не имеют аналитических решений. Однако их решение вполне возможно численными методами. Для численного решения систем нелинейных уравнений используется следующая функция:

- **NSolve[eqns, vars]** – пытается решать численно одно уравнение или систему уравнений относительно переменных **vars**.
- **NSolve[eqns, vars, elims]** – пытается решать численно уравнения относительно **vars**, исключая переменные **elims**.

Приведем примеры использования функции **NSolve** для численного решения уравнений:

```
f[x]:=x^3-6*x^2+21*x
NSolve[f[x]==52,x]
 {{x→1. -3.4641 д},{x→1. +3.4641 д},{x→4.}}
NSolve[{y*x^2==9,x*y^2==3},{x,y}]
 {{x→3.,y→1.},{x→-1.5-2.59808 i,y→-0.5-0.866025 i},{x→-
 1.5+2.59808 i,y→-0.5+0.866025 i}}
```

Для вычисления корней уравнений, например многочленов, используется функция **Roots**:

Roots[lhs==rhs, var] – дает дизъюнкцию уравнений, которая представляет корни полиномиального уравнения. С этой функцией применимы следующие опции:

- **Cubics** – задает поиск явных решений для неприводимых кубических уравнений.
- **EquatedTo** – определяет выражение для замещения переменной в решении.
- **Multiplicity** – устанавливает кратность каждого из корней в конечном результате.
- **Quartics** – указывает на выполнение решения в явном виде для неприводимых квадратных уравнений.
- **Trig** – задает трактовку тригонометрических функций как рациональных.
- **Using** – задает дополнительные условия, которые следует использовать.

- **VerifyConvergence -> True** – устанавливает проверку предела членов последовательности и предела отношения членов последовательности в случае бесконечных сумм и произведений. При назначении **VerifyConvergence -> False** – проверка не проводится.

Приведенные ниже примеры иллюстрируют применение функции **Roots**:

```
Roots[x^5+8*x^4+31*x^3+80*x^2+94*x+20==0,x]
  x == -2 - √3 || x == -2 + √3 || x == -1 - 3 i || x == -1 + 3 i || x == -2
Roots[x^2+2*x+15==0,x]
  x == -1 - i √14 || x == -1 + i √14
```

Имеется также ряд дополнительных функций, которые могут использоваться для решения нелинейных уравнений или используются описанными ранее функциями:

- **Auxiliary[v]** – применяется модулем **Solve** для указания того, что переменная v должна использоваться функцией **Roots** для результирующих решений, но соответствующие значения v не должны быть включены в окончательный ответ.
- **Eliminate[eqns, vars]** – исключает переменные **vars** из системы совместных уравнений **eqns**.
- **FindRoot[lhs == rhs, {x, x0}]** – ищет численное решение уравнения **lhs == rhs**, начиная с x == x0.
- **MainSolve[eqns]** – основная функция для преобразования системы уравнений. **Solve** и **Eliminate** вызывают ее. Уравнения должны быть представлены в форме **lhs == rhs**. Они могут объединяться с помощью && и ||. **MainSolve** возвращает False, если не существует решения уравнений, и возвращает True, если все значения переменных являются решениями. **MainSolve** перестраивает уравнения, применяя определенные директивы.
- **MainSolve[eqns, vars, elim, rest]** – пытается перестраивать уравнения **eqns** так, чтобы найти решения для переменных **vars** и исключить переменные **elim**. Список **rest** может включаться для указания порядка исключения для любых остальных переменных.
- **NRoots[lhs==rhs, var]** – возвращает список численных приближений корней полиномиального уравнения.
- **Residue[expr, {x, x0}]** – ищет вычет expr в точке $x = x0$.
- **SolveAlways[eqns, vars]** – возвращает значения параметров, которые превращают уравнения eqns в тождества для всех значений переменных vars.

Внимание! *Система Mathematica обладает обширными средствами для решения уравнений и их систем. Однако средства этих систем ориентированы на достаточно опытных в решении математических задач пользователей. В частности, грамотное использование многочисленных опций требует хорошего понимания методов и путей решения уравнений. Это ценно для профессионалов.*

4.9.2. Решение уравнений в Mathcad

В системе Mathcad для решения уравнений в символьном виде используется функция solve. Поскольку в этом случае Mathcad использует ядро системы Maple, то возможности данной функции аналогичны описанным для системы Maple. Решение возможно в командном режиме – рис. 4.39.

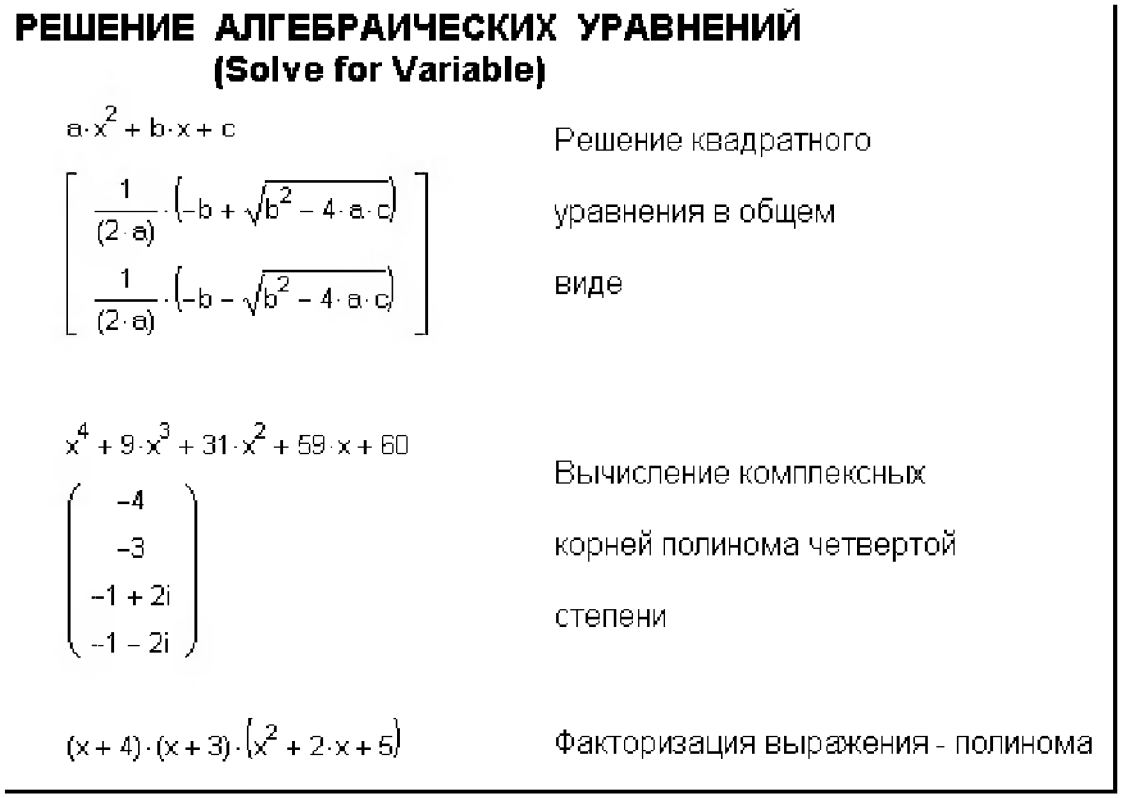


Рис. 4.39. Решение уравнений в командном режиме Mathcad

Функция solve может также использоваться с оператором символьного вида →. При этом для задания систем уравнений можно использовать векторное их задание – см. рис. 4.40.

Для простейших уравнений вида $F(x) = 0$ численное решение находится с помощью функции

`root(Выражение, Имя_переменной).`

Функция реализует вычисления методом секущих, переходя в отсутствии решения к специальному методу Мюллера. Перед решением можно задать начальное значение переменной, которая вычисляется. Решение ищется с погрешностью, заданной системной переменной **TOL**. Пример применения функции root дан на рис. 4.41.

Корни полинома вычисляет функция **polyroots(V)**. Она возвращает вектор всех корней многочлена (полинома) степени n, коэффициенты которого находятся в векторе **V**, имеющем длину, равную $n + 1$. Пример применения этой функции дан на рис. 4.41 снизу.

РЕШЕНИЕ СИСТЕМ НЕЛИНЕЙНЫХ УРАВНЕНИЙ ПРИ ИХ ВЕКТОРНОМ ЗАДАНИИ

$$\begin{pmatrix} x^2+y^2=7 \\ 2\cdot x-y=0 \end{pmatrix} \text{solve},x,y \rightarrow \begin{pmatrix} \frac{1}{5}\cdot\sqrt{35} & \frac{2}{5}\cdot\sqrt{35} \\ \frac{-1}{5}\cdot\sqrt{35} & \frac{-2}{5}\cdot\sqrt{35} \end{pmatrix} \qquad \begin{pmatrix} x^2+y^2+z^2=3 \\ x+y-z=1 \\ x\cdot y\cdot z=1 \end{pmatrix} \text{solve},x,y,z \rightarrow \begin{pmatrix} 1 & -1 & -1 \\ 1 & 1 & 1 \\ -1 & 1 & -1 \end{pmatrix}$$

$$\begin{pmatrix} x^2+y^2+z^2=3 \\ x+y-z=1 \\ x\cdot y=1 \end{pmatrix} \text{solve},x,y,z \rightarrow \begin{pmatrix} 1 & 1 & 1 \\ \frac{-1}{2}-\frac{1}{2}\cdot i\cdot\sqrt{3} & \frac{-1}{2}+\frac{1}{2}\cdot i\cdot\sqrt{3} & -2 \\ \frac{-1}{2}+\frac{1}{2}\cdot i\cdot\sqrt{3} & \frac{-1}{2}-\frac{1}{2}\cdot i\cdot\sqrt{3} & -2 \end{pmatrix}$$

$$\begin{pmatrix} x^2+y^2+z^2=5 \\ 2\cdot x-y^3-1=1 \end{pmatrix} \text{solve},x,y,z \rightarrow \begin{bmatrix} \frac{1}{2}\cdot y^3+1 & y & \frac{1}{2}\cdot\left(16-y^6-4\cdot y^3-4\cdot y^2\right)^{\left(\frac{1}{2}\right)} \\ \frac{1}{2}\cdot y^3+1 & y & \frac{-1}{2}\cdot\left(16-y^6-4\cdot y^3-4\cdot y^2\right)^{\left(\frac{1}{2}\right)} \end{bmatrix}$$

Рис. 4.40. Примеры решения нелинейных уравнений при их задании в векторном виде и с применением функции символьного решения solve

ПРИМЕНЕНИЕ ФУНКЦИЙ root И polyroots

$a3 := 2 \qquad a2 := -8 \qquad a1 := 25 \qquad a0 := -64$ Коэффициенты полинома

$F(x) := a3\cdot x^3 + a2\cdot x^2 + a1\cdot x + a0$ Задание полинома

Вычисление действительного корня

$x := 0 \qquad x1 := \text{root}(F(x), x) \qquad x1 = 3.211$

Вычисление двух других (возможно комплексных) корней

$i := \sqrt{-1} \qquad x := 1 + 1\cdot i \qquad x2 := \text{root}\left(\frac{F(x)}{x-x1}, x\right) \qquad x2 = 0.395 + 3.132i$

$x3 := \text{root}\left[\frac{F(x)}{(x-x1)\cdot(x-x2)}, x\right] \qquad x3 = 0.395 - 3.132i$

Как и следовало ожидать, комплексные корни кубического полинома являются взаимно сопряженными!

$V_0 := a0 \qquad V_1 := a1 \qquad V_2 := a2 \qquad V_3 := a3$

Пример применения функции polyroots: $\text{polyroots}(V) = \begin{pmatrix} 0.395 + 3.132i \\ 0.395 - 3.132i \\ 3.211 \end{pmatrix}$

Рис. 4.41. Примеры применения функций root и polyroot

При решении систем нелинейных уравнений и неравенств используется специальный вычислительный блок, открываемый служебным словом – директивой **Given** – и имеющий следующую структуру:

```
Начальные условия
Given
  Уравнения
  Ограничительные условия
  Выражения с функциями Find, Minerr, Maximize и Minimize
```

Начальные условия определяют начальные значения искомых переменных. Они задаются обычным присваиванием переменным заданных значений. Если переменных несколько, то используется векторное представление для начальных условий. Уравнения задаются в виде expr_left = expr_right с применением жирного знака равенства = между левой и правой частями каждого уравнения. Ограничительные условия обычно задаются в виде неравенств или равенств, которые должны удовлетворяться при решении системы уравнений.

Для решения нелинейных уравнений и систем с ними в блоке используется одна из следующих двух функций:

- **Find(v1, v2, ..., vn)** – возвращает значение одной или ряда переменных для точного решения;
- **Minerr(v1, v2, ..., vn)** – возвращает значение одной или ряда переменных для приближенного решения.

Между этими функциями существуют принципиальные различия. Первая функция используется, когда решение реально существует (хотя и не является аналитическим). Вторая функция пытается найти максимальное приближение даже к несуществующему решению путем минимизации среднеквадратичной погрешности решения. Примеры применения этих функций показаны на рис. 4.42.

Функции **Maximize** и **Minimize** будут рассмотрены в дальнейшем, поскольку они предназначены для решения задач поиска экстремумов функций.

4.9.3. Решение уравнений и неравенств в системе Derive

Система Derive для решения уравнений, неравенств и систем с ними имеет отдельную позицию главного меню – **Solve**. В ней сосредоточены команды:

- **Algebraically...Ctrl+Shift+A** – решение в символьном виде;
- **Numerically... Ctrl+Shift+N** – решение в численном виде;
- **System... Ctrl+Shift+S** – решение систем уравнений.

С помощью простых окон можно задать уравнения, переменные и (в случае численного решения) интервалы изоляции корней. Решение Derive осуществляет с помощью следующих функций:

Рис. 4.42. Примеры решения неравенств и систем нелинейных уравнений в системе Mathcad 11/12

- **SOLVE (u,x)** – вычисление x, при котором u=0;
- **SOLVE (u=v,x)** – вычисление x, при котором u=v;
- **SOLVE (u=v,x,a,b)** – вычисление x на отрезке [a,b] при котором u=v;
- **SOLVE (u<v,x)** – вычисление x для неравенства u<v;
- **SOLVE ([u1=v1,u2=v2, ...], [x1,x2,...])** – вычисление корней x1, x2, ... для системы линейных уравнений.

Ниже представлен пример, показывающий возможности функции **SOLVE** для решения системы из трех линейных уравнений:

```
1: "Решение системы линейных уравнений"
2: SOLVE ([3 a+2 b+c=4, a+b-c=1, a-2 b+c= 3], [a,~
   ~ b, c])
3: [ a = 1.7  b = -0.6  c = 0.1 ]
```

Решена следующая система:

```
3 * a + 2 * b + c = 4
    a +     b + c = 1
    a – 2 * b + c = 3
```

с неизвестными *a*, *b* и *c*. Систему линейных уравнений можно решать и с применением матричных операций, но для простых систем описанный выше способ наиболее простой.

4.9.4. Решение уравнений в системе MuPAD

Система MuPAD также имеет функцию solve для *решения нелинейных уравнений* и систем нелинейных уравнений:

- **solve(eqn,indet)** – решение одиночного уравнения по переменной indet. Если решения нет, возвращается пустой список {}, если все x удовлетворяют решению – выдается список {x}.
- **solve(sys,unk[,MaxDegree=n][,BackSubstitution=b])** – решение системы уравнений **sys** в виде полинома степени n (по умолчанию 2). Опция BackSubstitution может иметь значения True и False.
- **solve(object)** – специальная форма функции – см. примеры ниже.

Следующие примеры иллюстрируют применение этой функции.

```
solve(x^2=9,x);
                                {-3, 3}
solve({x^2-y^2=8,x^2+y^2=10},{x,y});
{[y = -1, x = 3], [y = -1, x = -3], [y = 1, x = 3], [y = 1, x = -3]}
solve(sin(x)/x=0,x);
                        {(PI, -PI, 2 PI, ...)}
solve(tan(x)=5,x);
          {(atan(5), PI + atan(5), - PI + atan(5) , ...)}
float(hold(solve)(2^x-x^2,x=-1..5));
               {3.999999999, 1.999999999, -0.766664696}
float(hold(solve)(2^x-x^2));
                             {-0.7666646959}
```

Можно подметить, что функция **solve** системы MuPAD неплохо справляется с некоторыми весьма специфическими задачами, например с решением тригонометрических уравнений, имеющих множественные и периодические решения.

> ***Внимание!*** *MuPAD имеет довольно эффективную функцию solve для решения нелинейных уравнений и систем нелинейных уравнений. В ряде случаев она позволяет находить все решения и даже периодические решения.*

4.10. Применение пакета расширения Maple student

4.10.1. Функции пакета student

Пакет student – это, несомненно, один из пакетов, наиболее привлекательных для студентов и аспирантов. В нем собраны наиболее распространенные и нужные функции, которые студенты университетов и иных вузов обычно используют на практических занятиях, при подготовке курсовых и дипломных проектов. Пакет вызывается командой

```
> with(student):
```

и содержит следующие функции:

- `D` – функция дифференциального оператора;
- `Diff` – инертная форма функции вычисления производной;
- `Doubleint` – инертная форма функции вычисления двойного интеграла;
- `Int` – инертная форма функции интегрирования `int`;
- `Limit` – инертная форма функции вычисления предела `limit`;
- `Lineint` – инертная форма функции вычисления линейного интеграла `lineint`;
- `Point` – тестирование объекта на соответствие типу точки (point);
- `Product` – инертная форма функции вычисления произведения членов последовательности;
- `Sum` – инертная форма функции вычисления суммы членов последовательности;
- `Tripleint` – инертная форма функции вычисления тройного интеграла;
- `changevar` – замена переменной;
- `combine` – объединение подобных членов;
- `completesquare` – вычисление полного квадрата (многочлена);
- `distance` – вычисление расстояния между точками;
- `equate` – создание системы уравнений из списков, таблицы, массивов;
- `extrema` – вычисление экстремума выражения;
- `integrand` – вывод подынтегрального выражения из-под знака инертного интеграла;
- `intercept` – нахождение точки пересечения двух кривых;
- `intparts` – интегрирование по частям;
- `isolate` – выделение подвыражения;
- `leftbox` – графическая иллюстрация интегрирования методом левых прямоугольников;
- `leftsum` – числовое приближение к интегралу левыми прямоугольниками;
- `makeproc` – преобразование выражения в процедуру Maple;
- `maximize` – вычисление максимума функции;
- `middlebox` – графическая иллюстрация интегрирования методом центральных прямоугольников;
- `middlesum` – числовое приближение к интегралу центральными прямоугольниками;
- `midpoint` – вычисление средней точки сегмента линии;
- `minimize` – вычисление минимума функции;
- `powsubs` – подстановка для множителей выражения;
- `rightbox` – графическая иллюстрация интегрирования методом правых прямоугольников;
- `rightsum` – числовое приближение к интегралу правыми прямоугольниками;
- `showtangent` – график функции и касательной линии;
- `simpson` – числовое приближение к интегралу по методу Симпсона;

- `slope` – вычисление и построение касательной к заданной точке функции;
- `trapezoid` – числовое приближение к интегралу методом трапеций;
- `value` – вычисляет инертные функции.

К сожалению, набор функций этого пакета несколько различается в разных версиях Maple.

4.10.2. Функции интегрирования пакета student

В ядре и в пакетах расширения Maple можно найти множество специальных функций для вычисления интегралов различного типа. Например, в пакете student имеются следующие функции:

- `Int(expr,x)` – инертная форма вычисления неопределенного интеграла;
- `Doubleint(expr,x,y,Domain)` – вычисление двойного интеграла по переменным `x` и `y` по области `Domain`;
- `Tripleint(expr,x,y,z)` – вычисление тройного интеграла;
- `intparts(f,u)` – интегрирование по частям.

Ниже дан пример применения функции `Tripleint` пакета student:

```
> Tripleint(f(x,y,z),x,y,z);
```

$$\int\int\int \mathrm{f}(x, y, z)\, dx\, dy\, dz$$

```
> Tripleint(x*y*z^2,x=0..2,y=0..3,z=0..5);
```

$$\int_0^5 \int_0^3 \int_0^2 x\, y\, z^2\, dx\, dy\, dz$$

```
> evalf(%);
```

$$375.0000000$$

```
> int(int(int(x*y*z^2,x=0..2),y=0..3),z=0..5);
```

$$375$$

4.10.3. Иллюстративная графика пакета student

Пакет student имеет три графические функции для иллюстрации интегрирования методом прямоугольников:

- `leftbox(f(x), x=a..b, o)` или `leftbox(f(x), x=a..b, n, 'shading'=<color>, o);`
- `rightbox(f(x), x=a..b, o)` или `rightbox(f(x), x=a..b, n, o);`
- `middlebox(f(x), x=a..b, o)` или `middlebox(f(x), x=a..b, n, o).`

Здесь `f(x)` – функция переменной `x`, `x` – переменная интегрирования, `a` – левая граница области интегрирования, `b` – правая граница области интегрирования, `n` – число показанных прямоугольников, `color` – цвет прямоугольников, `o` – параметры (см. `?plot,options`).

В этих функциях прямоугольники строятся соответственно слева, справа и посередине относительно узловых точек функции $f(x)$, график которой также

строится. Кроме того, имеется функция для построения касательной к заданной точке $x = a$ для линии, представляющей $f(x)$:

```
showtangent (f(x), x = a)
```

Рисунок 4.44 показывает все эти возможности пакета student. Четыре отмеченных вида графиков здесь построены в отдельных окнах.

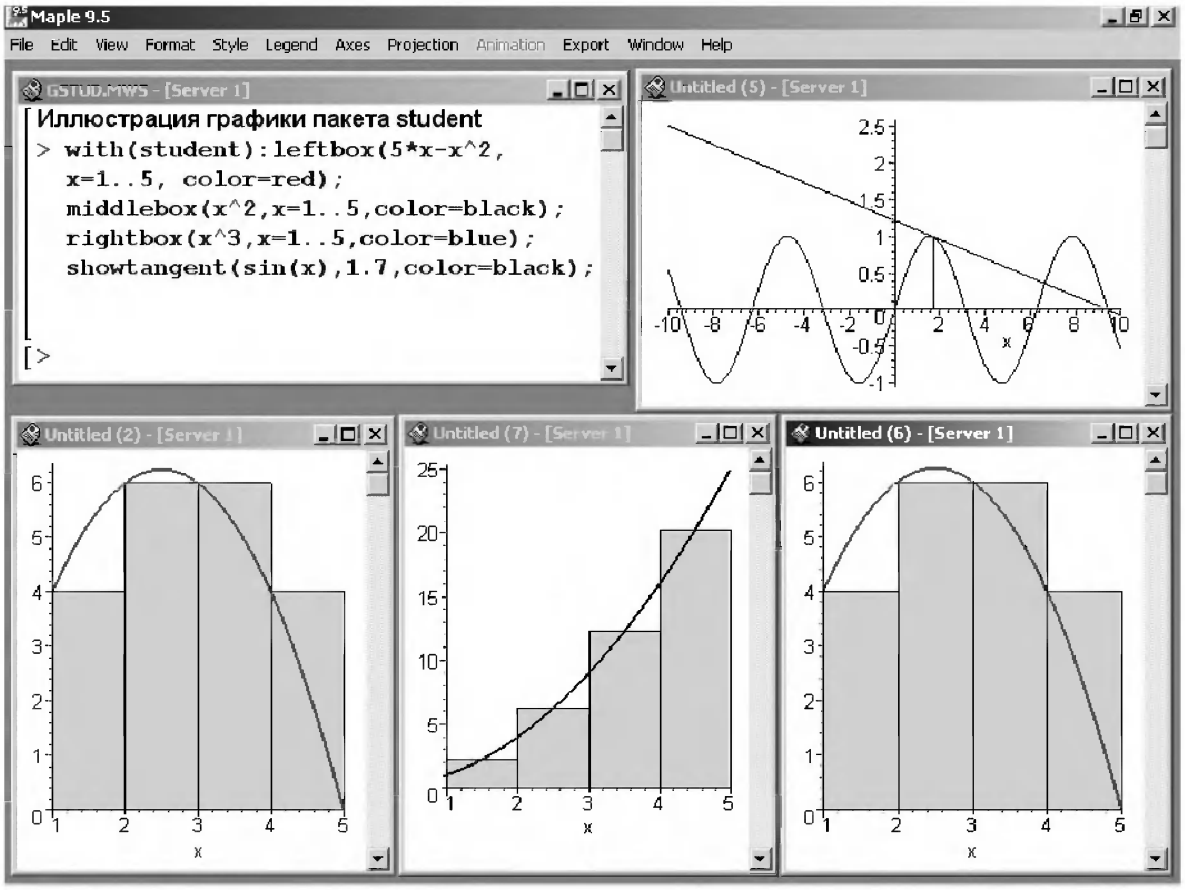


Рис. 4.44. Примеры иллюстративной графики пакета student

В главе 9 мы продолжим рассмотрение возможностей этого пакета в части графической визуализации вычислений.

4.11. Работа с алгебраическими кривыми в Maple

4.11.1. Пакет для работы с алгебраическими кривыми algcurves

Для работы с *алгебраическими кривыми* служит пакет расширения algcurves. Он вызывается командой

```
> restart;with(algcurves);
```

Отметим назначение наиболее важных функций этого пакета:

- `Weierstrassform(f,x,y,x0,y0,opt)` – вычисление нормальной формы для эллиптических или гиперболических алгебраических кривых;
- `differentials(f, x, y, opt)` – голоморфные дифференциалы алгебраических кривых;
- `genus(f,x,y,opt)` – подлинность алгебраической кривой;
- `homogeneous(f,x,y,z)` – создание полинома двух переменных, гомогенного в трех переменных;
- `homology(f, x, y)` – находит канонический гомологический базис по алгоритму Треткоффа;
- `integral_basis(f, x, y, S)` – интегральный базис алгебраического поля функции;
- `is_hyperelliptic(f, x, y)` – тестирует кривую на ее принадлежность к гиперболической;
- `j_invariant(f,x,y)` – j-инвариант алгебраической кривой;
- `monodromy(f, x, y, opt)` – вычисляет монодромию алгебраической кривой;
- `parametrization(f,x,y,t)` – находит параметризацию для кривой с родом (даваемым функцией `genuc`), равным 0;
- `periodmatrix(f, x, y, opt)` – вычисляет периодическую матрицу кривой;
- `plot_knot(f,x,y,opt)` – строит узел – несамопересекающуюся замкнутую кривую в трехмерном евклидовом пространстве;
- `puiseux(f,x=p,y,n,T)` – определяет Пуизе-расширение алгебраической функции (может иметь и более простые формы записи);
- `singularities(f,x,y)` – анализирует кривую на сингулярность.

4.11.2. Примеры работы с алгебраическими кривыми

Приведем примеры применения нескольких функций пакета Algcurves:

```
> Weierstrassform( (y^2-1)^2+x*(x^2+1)^2 ,x,y,x0,y0);
```

$$\left[y0^2 - 1 - x0 - x0^5, \frac{y^2-1}{x^2+1}, -y, -x0^2, -y0 \right]$$

```
> f:=y^3+x^3*y^3+x^4;
```

$$f := y^3 + y^3x^3 + x^4$$

```
> differentials(f,x,y);
```

$$\left[\frac{x^2 dx}{y^2(1+x^3)}, \frac{x\,dx}{y(1+x^3)}, \frac{x^2 dx}{y(1+x^3)} \right]$$

```
> differentials(f,x,y,skip_dx);
```

$$[x^2, yx, yx^2]$$

```
> nops(%);
```

```
> genus(f,x,y);
```

$$3$$

```
> homogeneous(f,x,y,z);
```

$$x^4z^2 + y^3z^3 + y^3x^3$$

```
> g := y^3-x*y^2+2*2^(1/2)*y^2+x^2-2*2^(1/2)*x+2+y^6;;
```

$$g := y^3 - xy^2 + 2\sqrt{2}y^2 + x^2 - 2\sqrt{2}x + 2 + y^6$$

```
> integral_basis(g,x,y);
```

$$\left[1, y, y^2, y^3, y^4, \frac{y^2 + y^5 + \sqrt{2}y}{-\sqrt{2} + x}\right]$$

```
> is_hyperelliptic(f,x,y);
```

$$false$$

```
> f1:=y^2+x^5+1:is_hyperelliptic(f1,x,y);
```

$$true$$

```
> j_invariant(g,x,y);
```

$$\frac{71936606821}{38521803} + \frac{3803393323}{38521803}\sqrt{2}$$

```
> parametrization(x^4+y^4+a*x^2*y^2+b*y^3,x,y,t);
```

$$\left[-\frac{t^3b^3}{b^8 + t^2ab^4 + t^4}, -\frac{t^4b}{b^8 + t^2ab^4 + t^4}\right]$$

```
> Z := periodmatrix(f1,x,y,Riemann);
```

$$Z := \begin{bmatrix} .4999999918 + 1.213922064\,I & -.9999999899 - .5257311260\,I \\ -1.000000004 - .5257311066\,I & -.5000000106 + .6881909548\,I \end{bmatrix}$$

4.11.3. Построение алгебраических кривых класса knot

Функция `plot_knot` позволяет строить одну или несколько алгебраических кривых – узлов. Пример построения целого семейства узлов показан на рис. 4.45.

Для лучшего обзора таких кривых рекомендуется воспользоваться возможностью вращения трехмерных фигур мышью для уточнения угла, под которым рассматривается фигура – в нашем случае семейство алгебраических кривых.

Начиная с версии Maple 7 в пакет расширения Algcurves добавлена новая функция импликативной графики `plot_real_curve`. Она строит алгебраическую кривую для действительной части полиномиального выражения и записывается в виде:

```
plot_real_curve (p, x, y, opt)
```

Функция имеет следующие параметры:

- `p` – полиномиальное выражение переменных `x` и `y`, задающее алгебраическую кривую;
- `opt` – параметр, который задает форму представления графика и его параметры.

Функция `plot_real_curve` вычисляет и строит алгебраическую кривую по точкам. Применение функции `plot_real_curve` показывает рис. 4.46.

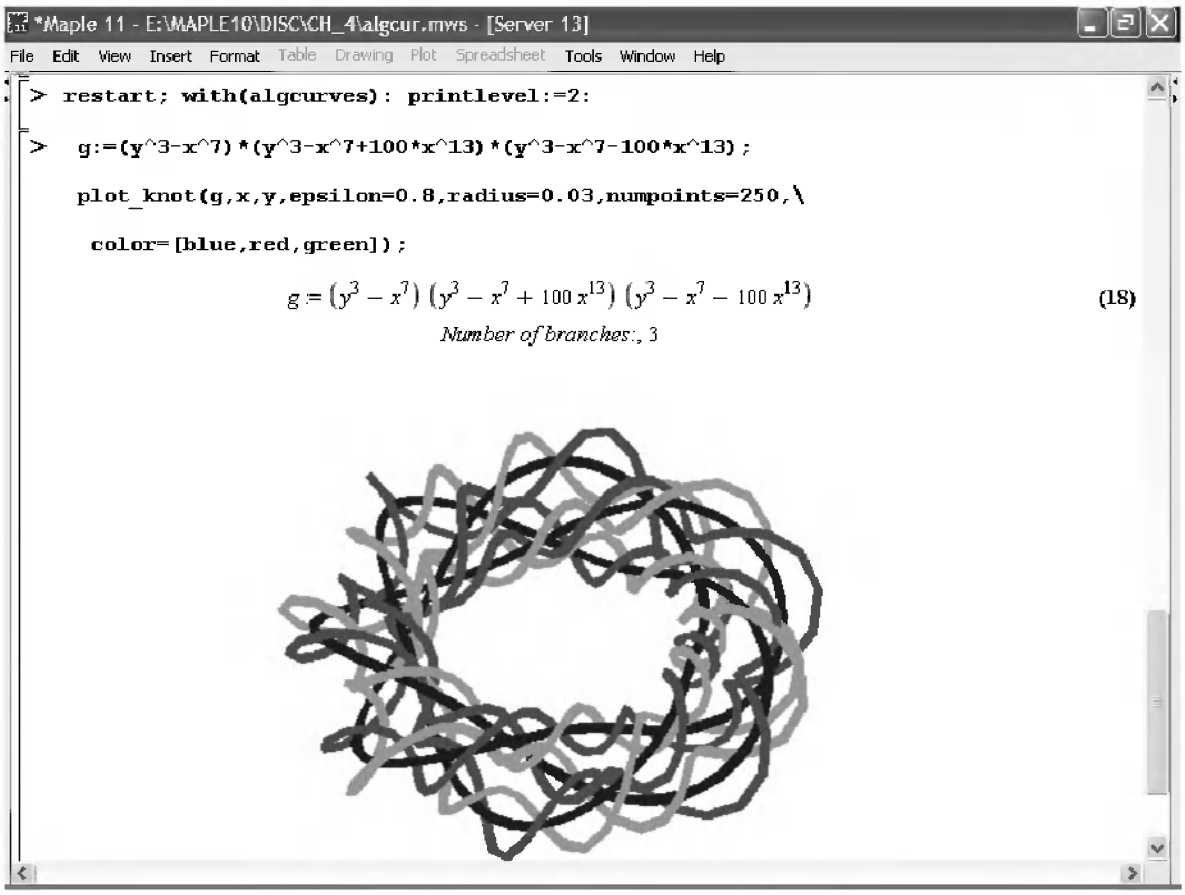


Рис. 4.45. Семейство узлов

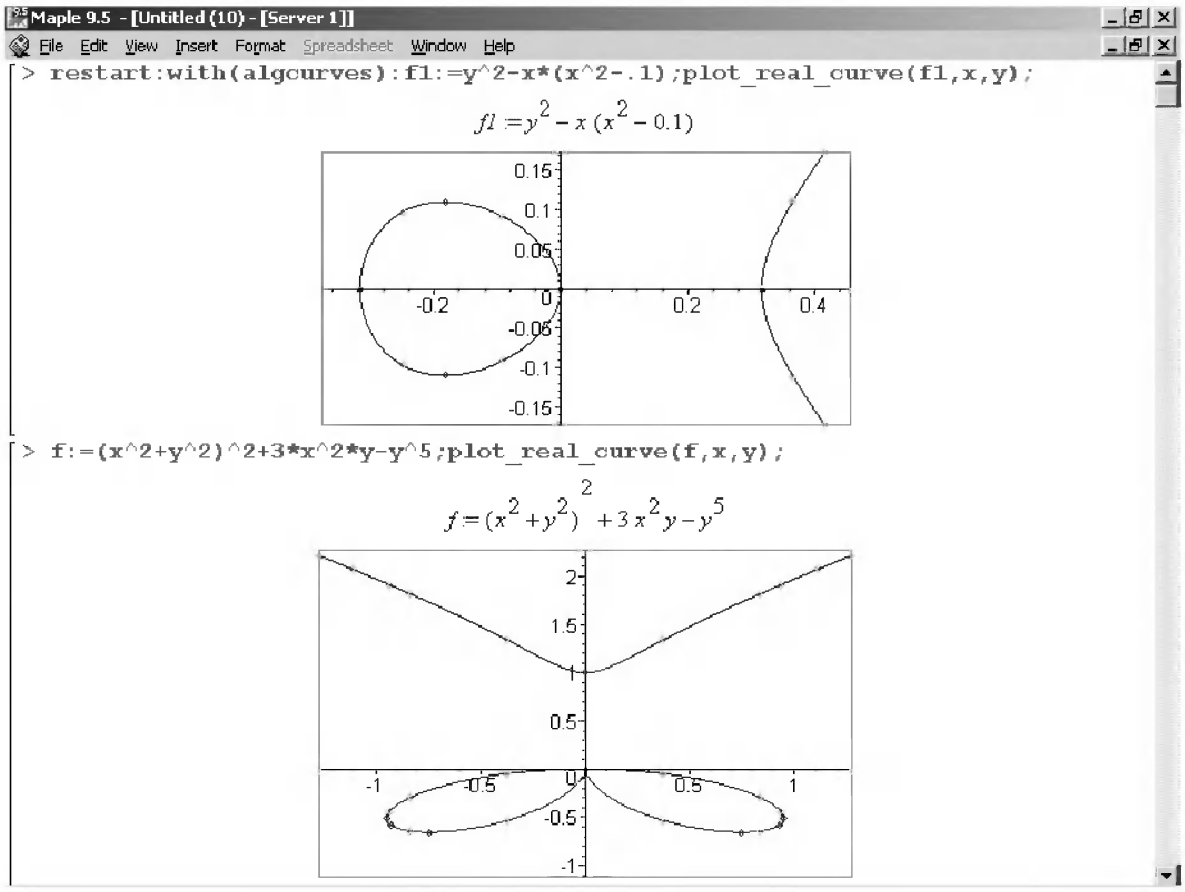


Рис. 4.46. Примеры применения функции plot_real_curve

4.12. Векторные вычисления и функции теории поля

4.12.1. Пакет векторных вычислений VectorCalculus

В Maple 8 были существенно расширены возможности вычислений над векторами (пространственными объектами) и поверхностями. Для этого введен пакет VectorCalculus, который при вызове открывает доступ ко многим командам и функциям векторного анализа, теории поля и приложений дифференциального исчисления [67, 68]. Рекомендуется вызов пакета командами:

```
> restart; with(VectorCalculus); interface(showassumed=0);
```

Данный пакет после загрузки видоизменяет многие операторы, команды и функции, встроенные в ядро системы. При этом меняется их математический и физический смысл. Поэтому пользоваться пакетом надо с известной осторожностью. Для восстановления роли функций можно использовать команду `restart`.

Пакет VectorCalculus оперирует такими понятиями, как поток векторного поля, градиент, торсион, векторный потенциал и др. Приведенный ниже материал поясняет применение большинства функций этого пакета. Полезно просмотреть и файл VectorCalculus.mws, содержащий примеры его применения. В Интернете можно найти целую серию уроков по векторному анализу и теории поля в виде пакета Calculus IV или V (разработчик проффессор J. Wagner).

4.12.2. Объекты векторных вычислений

Вектор в геометрическом представлении в данном пакете по умолчанию задается в прямоугольной системе координат:

```
> v := Vector( [x,y,z] );
```

$$v := x\mathbf{e_x} + y\mathbf{e_y} + z\mathbf{e_z}$$

Здесь $\mathbf{e}_x$, $\mathbf{e}_y$ и $\mathbf{e}_z$ – проекции единичного вектора **e** на оси координат x, y и z. Тип координатной системы (по умолчанию – прямоугольная) можно определить следующим образом:

```
> attributes(v);
```

$$coords = cartesian$$

Для создания векторного поля служит функция

```
VectorField(v, c)
```

где **v** – вектор и `c` – опционально заданный параметр в форме `name[name, name, ...]`, задающий тип координатной системы.

Можно изменить систему координат, например задав (с помощью функции установки координат `SetCoordinates`) полярную систему координат:

```
> SetCoordinates( polar );
```

$$polar$$

```
> w := <r,theta>;
```

$$w := r\mathbf{e_r} + \theta\mathbf{e_\theta}$$

```
> attributes(w);
```

$$coords = polar$$

Аналогично можно задать вектор в сферической системе координат:

```
> SetCoordinates( spherical[r,phi,theta] );
```

$$spherical_{r,\phi,\theta}$$

```
> F := VectorField( <r,0,0> );
```

$$F := r\bar{\mathbf{e}}_\mathbf{r}$$

```
> attributes(F);
```

$$vectorfield, coords = spherical_{r,\phi,\theta}$$

Можно также сменить формат представления вектора и выполнить с ним некоторые простейшие векторные операции:

```
> BasisFormat(false);
```

$$true$$

```
> v := <a,b,c>;
```

$$v := \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

```
> BasisFormat(true);
```

$$false$$

```
> v;
```

$$a\mathbf{e_r} + b\mathbf{e_\sigma} + c\mathbf{e_\theta}$$

```
> SetCoordinates( polar );
```

$$polar$$

```
> MapToBasis( <r,theta>, 'cartesian' );
```

$$r\cos(\theta)\mathbf{e_x} + r\sin(\theta)\mathbf{e_y}$$

Пакет VectorCalculus предусматривает возможность задания новой системы координат с помощью команды

AddCoordinates(newsys, eqns, owrite),

где `newsys` – спецификация новой системы координат в виде `symbol[name, name, ...]`; `eqns` – соотношения между координатами новой системы и прямоугольной системы координат, представленные в виде `list(algebraic)`; `owrite` – заданное опционально равенство.

4.12.3. Основные операции с векторами

В данном пакете переопределены некоторые основные *операции над векторами*. Прежде всего это операции сложения (+) и скалярного умножения (*), которые поясняются следующими примерами:

```
> SetCoordinates( cartesian );
```

$$cartesian$$

```
> <x,y,z> + m*<x1,y1,f1>;
```

$$(x + mx1)\mathbf{e_x} + (y + my1)\mathbf{e_y} + (z + mf1)\mathbf{e_z}$$

```
> (<r(a+h),s(a+h),t(a+h)> - <r(a),s(a),t(a)>) / h;
```

$$\frac{\mathrm{r}(a+h)-\mathrm{r}(a)}{h}\mathbf{e_x} + \frac{\mathrm{s}(a+h)-\mathrm{s}(a)}{h}\mathbf{e_y} + \frac{\mathrm{t}(a+h)-\mathrm{t}(a)}{h}\mathbf{e_z}$$

```
> limit(%,h=0);
```

$$\mathrm{D}(r)(a)\mathbf{e_x} + \mathrm{D}(s)(a)\mathbf{e_y} + \mathrm{D}(t)(a)\mathbf{e_z}$$

Обратите внимание на вычисление предела в конце этих примеров. Далее можно отметить операцию точечного умножения и кросс-умножения. Примеры на них можно найти в справке.

4.12.4. Операции с кривыми

В пакете векторных операций определен ряд типовых *операций с кривыми*. Ниже представлены задание эллиптической кривой и вычисление в аналитической форме нормали и радиуса кривизны:

```
> SetCoordinates( cartesian );
```

$$cartesian$$

```
> assume( t::real );
> ell := <2*cos(t),sin(t)>;
```

$$ell := 2\cos(t)\mathbf{e_x} + \sin(t)\mathbf{e_y}$$

```
> nv := simplify( PrincipalNormal(ell,t) );
```

$$nv := \frac{2\cos(t)}{(3\cos(t)^2-4)\sqrt{-3\cos(t)^2+4}}\mathbf{e_x} + \frac{4\sin(t)}{(3\cos(t)^2-4)\sqrt{-3\cos(t)^2+4}}\mathbf{e_y}$$

```
> len := simplify( LinearAlgebra:-Norm( nv, 2 ) );
```

$$len := -\frac{2}{3\cos(t)^2-4}$$

```
> r := simplify( RadiusOfCurvature(ell) );
```

$$r := -\frac{1}{2}(3\cos(t)^2-4)\sqrt{-3\cos(t)^2+4}$$

Теперь можно представить саму кривую (эллипс) и ее эволюту (рис. 4.47):

```
> ev := simplify( ell + r * nv / len );
```

$$ev := \frac{3}{2}\cos(t)^3\mathbf{e_x} + 3\sin(t)(-1+\cos(t)^2)\mathbf{e_y}$$

```
> plot( [ [ell[1], ell[2], t=0..2*Pi], [ev[1], ev[2], t=0..2*Pi] ] );
```

Нетрудно заметить, что для эллипса эволюта представляет собой удлиненную астроиду.

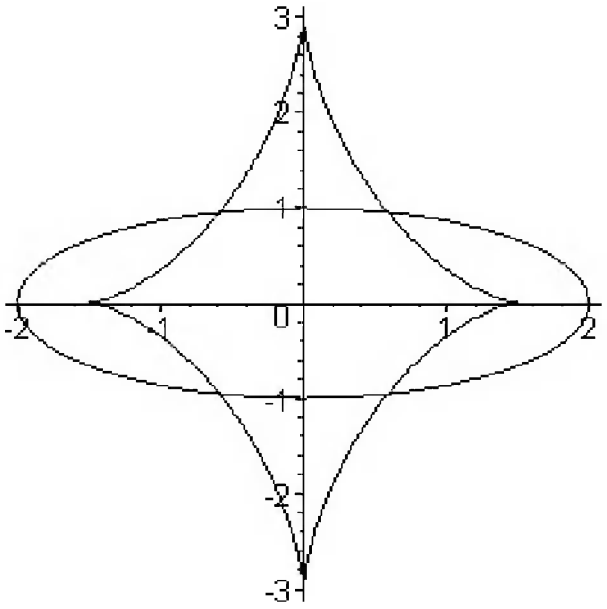


Рис. 4.47. Графики кривой – эллипса и ее эволюты

Для вычисления кривизны кривой C используется функция `Curvature(C, t)`, в которой параметр t может и отсутствовать:

```
> Curvature( <cos(t),t,sin(t)>, t );
```

$$\frac{1}{4}\sqrt{2\cos(t)^2+2\sin(t)^2}\sqrt{2}$$

```
> c := Curvature( t -> <t,t^2,t^4> ):
> simplify( c(t) ) assuming t::real;
```

$$\frac{2\sqrt{64t^6+36t^4+1}}{(1+4t^2+16t^6)^{(3/2)}}$$

4.12.5. Интегрирование в пакете VectorCalculus

В аспекте практических приложений векторного анализа и теории поля особый интерес представляют приложения интегрирования пакета VectorCalculus. Так, видоизмененная функция `int(f, dom)` задает вычисление интеграла от функции f по области dom, например:

```
> int( x^2+y^2, [x,y] = Circle( <0,1>, r ) );
```

$$\frac{\pi r^4}{2}$$

```
> int( sin(x)*cos(y)*tan(z), [x,y,z] = Parallelepiped( 0..Pi,
0..Pi/3, 0..Pi/4 ) );
```

$$\frac{1}{2}\sqrt{3}\ln(2)$$

Функция `PathInt(f, dom)` вычисляет интеграл пути для функции f с $\mathbf{R}^n$ до $\mathbf{R}$:

```
> restart:with(VectorCalculus):
> PathInt( x^2, [x,y] = Line( <0,0>, <1,2> ) );
```

$$\frac{\sqrt{5}}{3}$$

```
> PathInt( x^2+y^2, [x,y] = Circle( <0,0>, 3/2 ) );
```

$$\frac{27\pi}{4}$$

```
> PathInt( 1, [x,y] = Ellipse( x^2+y^2/2-1 ) );
```

$$4\sqrt{2}\,\mathrm{EllipticE}\left(\frac{\sqrt{2}}{2}\right)$$

Другая функция `LineInt(F, dom)`, где `F` – вектор или процедура задания векторного поля, `dom` – параметр, характеризующий направление интегрирования, задает вычисление линейного интеграла в пространстве $\mathbf{R}^n$:

```
> SetCoordinates( cartesian[x,y] );
```

$$cartesian_{x,y}$$

```
> LineInt( VectorField( <x,y> ), Line( <0,1>, <2,-5> ) );
```

$$14$$

```
> LineInt( VectorField( <y,-x> ), Circle( <0,0>, r ) );
```

$$-2r^2\pi$$

Функция `ArcLength(C,dom)` задает вычисление длины дуги C по известному интегральному выражению для нее:

```
> ArcLength( <r*cos(t),r*sin(t)>, t=0..Pi ) assuming r>0;
```

$$\pi r$$

```
> ArcLength( t -> <t,t^2>, 0..2 );
```

$$\sqrt{17}-\frac{1}{4}\ln(-4+\sqrt{17})$$

```
>  evalf(%);
```

$$4.646783762$$

Наконец, функция `SurfaceInt(f, dom)` вычисляет поверхностный интеграл:

```
> SurfaceInt( 1, [x,y,z] = Surface( <r,s,t>, s=0..Pi/2, t=0..Pi,
coords=spherical ) ) assuming r>0;
```

$$\pi r^2$$

```
> SurfaceInt( x+y+z, [x,y,z] = Surface( <s,t,4-2*s-t>, [s,t] =
Triangle(<0,0>,<0,1>,<1,2>) ) );
```

$$\frac{11\sqrt{6}}{6}$$

Рекомендуется просмотреть различные варианты задания области интегрирования `dom` в справке по этому пакету.

4.12.6. Задание матриц специального типа

Пакет VectorCalculus позволяет для заданной функции f задавать несколько матриц специального вида, которые часто используются при решении задач теории поля:

- `Hessian(f, t)` – создание матрицы гессиана;
- `Jacobian(f, v, det)` – создание матрицы якобиана;
- `Wronskian(f, t)` – создание матрицы вронскиана.

Примеры задания таких матриц приведены ниже:

```
> Hessian( exp(x*y), [x,y] );
```

$$\begin{bmatrix} y^2\mathbf{e}^{(x,y)} & \mathbf{e}^{(x,y)} + yx\mathbf{e}^{(x,y)} \\ \mathbf{e}^{(x,y)} + yx\mathbf{e}^{(x,y)} & x^2\mathbf{e}^{(x,y)} \end{bmatrix}$$

```
> Jacobian( [r*cos(t),r*sin(t)], [r,t] );
```

$$\begin{bmatrix} \cos(t) & -r\sin(t) \\ \sin(t) & r\cos(t) \end{bmatrix}$$

```
> Jacobian( [r*cos(t),r*sin(t)], [r,t], 'determinant' );
```

$$\begin{bmatrix} \cos(t) & -r\sin(t) \\ \sin(t) & r\cos(t) \end{bmatrix}, \cos(t)^2 r + r\sin(t)^2$$

```
> Wronskian( [t,t^2,t^3], t );
```

$$\begin{bmatrix} t & t^2 & t^3 \\ 1 & 2t & 3t^2 \\ 0 & 2 & 6t \end{bmatrix}$$

4.12.7. Функции теории поля

К основным *функциям теории поля* относятся:

- `Curl(F)` – вычисляет вихрь векторного поля в $\mathbf{R}^3$;
- `Divergence(F)` – вычисляет дивергенцию векторного поля;
- `Flux(f, dom)` – вычисляет поток векторного поля в $\mathbf{R}^3$;
- `Gradient(f, c)` – вычисляет градиент функции f в пространстве от $\mathbf{R}^n$ до $\mathbf{R}$;
- `Del(f, c)` и `Nabla(f, c)` – векторные дифференциальные операторы;
- `Laplacian(f, c)` или `Laplacian(F)` – вычисляет лапласиан функции f или векторного определения (процедуры) F;
- `ScalarPotential(v)` – вычисляет скалярный потенциал векторного поля;
- `Torsion(C, t)` – вычисляет торсион в $\mathbf{R}^3$;
- `VectorPotential(v)` – вычисляет векторный потенциал в $\mathbf{R}^3$.

Определения этих функций, основанные на использовании криволинейных и поверхностных интегралов, имеются в учебной литературе [66]. Не приводя их,

ограничимся приведенными ниже примерами применения некоторых из указанных выше функций:

```
> restart:with(VectorCalculus):
SetCoordinates( 'cartesian'[x,y,z] );
```

$$cartesian_{x,y,z}$$

```
> F := VectorField( <-y,x,0> );
```

$$F := -y\bar{\mathbf{e}}_{\mathrm{X}} + x\bar{\mathbf{e}}_{\mathrm{Y}}$$

```
> Curl( F );
```

$$2\bar{\mathbf{e}}_{\mathrm{Z}}$$

```
> CrossProduct( Del, F );
```

$$2\bar{\mathbf{e}}_{\mathrm{Z}}$$

```
> F := VectorField( <x^2,y^2,z^2> );
```

$$F := -x^2\bar{\mathbf{e}}_{\mathrm{X}} + y^2\bar{\mathbf{e}}_{\mathrm{Y}} + z^2\bar{\mathbf{e}}_{\mathrm{Z}}$$

```
> Divergence( F );
```

$$2x + 2y + 2z$$

```
> Flux( VectorField( <x,y,z>, cartesian[x,y,z] ), Sphere( <0,0,0>, r ) );
```

$$4r^3\pi$$

```
> Gradient( x^3/3+y^2, [x,y] );
```

$$x^2\bar{\mathbf{e}}_{\mathrm{X}} + 2y\bar{\mathbf{e}}_{\mathrm{Y}}$$

$$0\bar{\mathbf{e}}_{\mathrm{X}}$$

```
> Laplacian( x^2+y^2+z^2, [x,y,z] );
```

$$6$$

```
> Laplacian( f(r,theta,z) );
```

$$\frac{\partial^2}{\partial z^2}\mathrm{f}(r,\theta,z)$$

Обратите внимание на то, что для гарантии правильного выполнения этих команд и отсутствия «зависания» компьютера могут потребоваться команда `restart` и перезагрузка пакета VectorCalculus.

4.12.8. Приближение площади сложной поверхности суммами Римана

Одним из важнейших приложений пакета VectorCalculus является вычисление длин дуг и площадей сложных поверхностей на основе применения линейных и поверхностных интегралов. Иногда это встречает большие трудности и требует специальных подходов. Примером может служить поверхность, заданная в приведенном ниже примере:

```
> restart: with(plots): with(LinearAlgebra): with(VectorCalculus):
> f := <sin(x), sin(y), sin(x+y)>;
```

$$f := \sin(x)\mathbf{e}_{\mathbf{X}} + \sin(y)\mathbf{e}_{\mathbf{Y}} + \sin(x+y)\mathbf{e}_{\mathbf{Z}}$$

```
> plot3d(f, x=-Pi..Pi, y=-Pi..Pi, axes=framed, grid=[70,70],
style=patchnogrid, lightmodel=light2);
```

Эта поверхность с имитацией ее освещения от внешнего источника света показана на рис. 4.48.

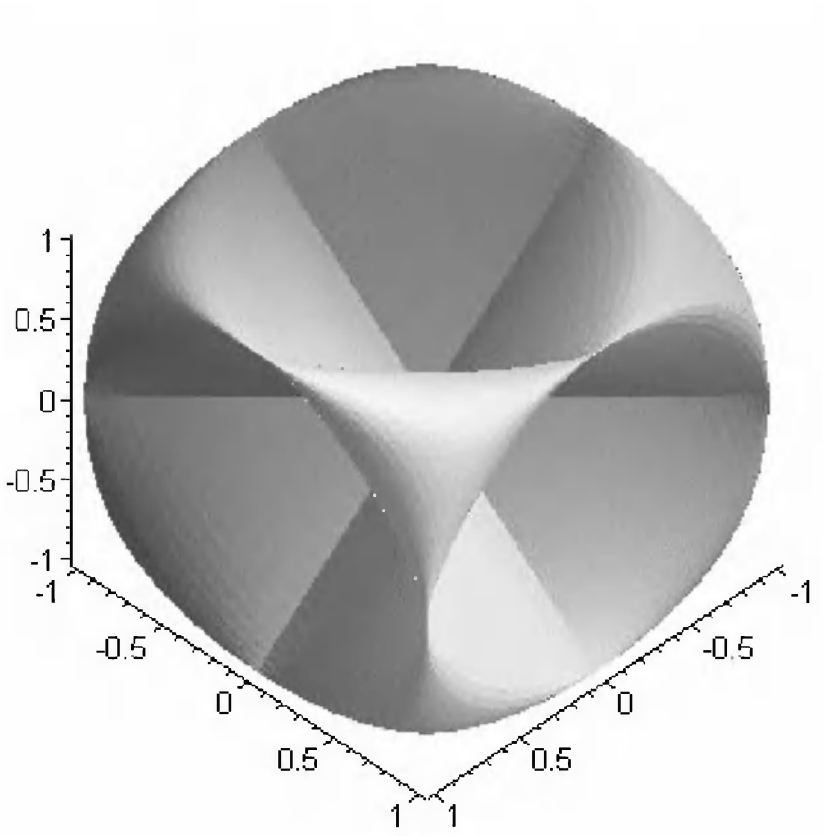


Рис. 4.48. Сложная поверхность с эффектами ее освещения внешним источником света

Применим обычную процедуру вычисления площади поверхности. Для этого вычислим для нее матрицу якобиана и удалим из нее столбец с нулевыми элементами:

```
> J := Jacobian(f, [x, y, z]);
```

$$J := \begin{bmatrix} \cos(x) & 0 & 0 \\ 0 & \cos(y) & 0 \\ \cos(x+y) & \cos(x+y) & 0 \end{bmatrix}$$

```
> J := DeleteColumn(J, [3]);
```

$$J := \begin{bmatrix} \cos(x) & 0 \\ 0 & \cos(y) \\ \cos(x+y) & \cos(x+y) \end{bmatrix}$$

Тогда площадь поверхности вычисляется следующим образом:

```
> dA := sqrt(Determinant(Transpose(J).J));
```

$$dA := \sqrt{\cos(x)^2\cos(y)^2 + \cos(x)^2\cos(x+y)^2 + \cos(x+y)^2\cos(y)^2}$$

```
> Int(Int(dA, x=0..2*Pi), y=0..2*Pi);
```

$$\int_0^{2\pi}\int_0^{2\pi} \sqrt{\cos(x)^2\cos(y)^2 + \cos(x)^2\cos(x+y)^2 + \cos(x+y)^2\cos(y)^2}\, dx\, dy$$

К сожалению, этот двойной интеграл Maple не вычисляет из-за сложности подынтегрального выражения, график которого представлен на рис. 4.49.

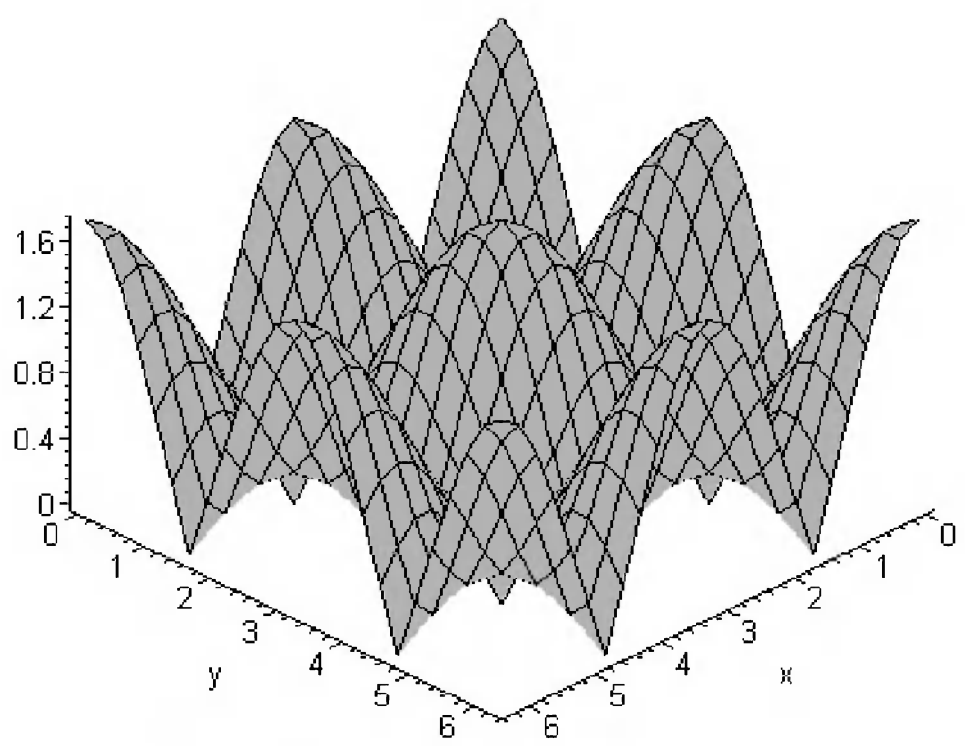


Рис. 4.49. График подынтегрального выражения

Для приближенного вычисления площади можно разбить поверхность на достаточное число сегментов и использовать замену интегралов суммами Римана. Оценка нижней и верхней сумм Римана для четверти поверхности (ее одного квадранта) представлена ниже:

```
> for s from 1 to 8 do
  F := (k, t)->subs({x=k*Pi/(10*s), y=t*Pi/(10*s)}, dA):
  A||s := evalf((Pi/(10*s))^2*sum(sum(F(p, q), p=0..10*s-1),
q=0..10*s-1)):
  print(A||s);
end do:
```

```
                    7.408455387
                    7.429353779
                    7.429810700
                    7.429973244
                    7.430045037
                    7.430081583
                    7.430102033
```

```
> for s from 1 to 8 do
  F := (k, t)->subs({x=k*Pi/(10*s), y=t*Pi/(10*s)}, dA):
  A||s := evalf((Pi/(10*s))^2*sum(sum(F(p, q), p=1..10*s),
q=1..10*s)):
  print(A||s)
end do:
```

7.408455386
7.427471278
7.429353778
7.429810700
7.429973260
7.430045062
7.430081587
7.430102036

Поскольку эти суммы явно сходятся, то можно считать применение сумм Римана приемлемым и принять, что площадь данной поверхности приближенно равна:

```
> Area := 4*7.43;
```

$$Area := 29.72$$

4.12.9. Вычисление поверхностных интегралов

Приведенный выше пример иллюстрирует трудности вычислений поверхностных интегралов. Разумеется, далеко не всегда Maple требует специальных подходов к вычислению подобных интегралов, и многие из них благополучно вычисляются. При этом нередко задача решается в аналитическом (символьном) виде. Поясним это на паре примеров, достаточно характерных для решения задач теории поля.

Первая задача: найти поток $F(x,y,z) = [3/x^2, 3/y^2, 1/z^2]$ через параболическую поверхность $z = x^2 + y^2$ при $1 \le x$ и $y \le 3$. Ниже представлено решение этой задачи в два этапа.

На первом этапе введем основные объекты и построим график потока (в виде стрелок в перспективе) через параболическую поверхность:

```
> restart: with(plots): with(LinearAlgebra): with(VectorCalculus):
> F := <3/x^2, 2/y^2, 1/z^2>;
```

$$F := \frac{3}{x^2}\mathbf{e}_x + \frac{2}{y^2}\mathbf{e}_y + \frac{1}{z^2}\mathbf{e}_z$$

```
> S := <u, v, u^2+v^2>;
```

$$S := u\mathbf{e}_x + v\mathbf{e}_y + (u^2 + v^2)\mathbf{e}_z$$

```
> F := eval(F, {x=S[1], y=S[2], z=S[3]});
```

$$F := \frac{3}{u^2}\mathbf{e}_x + \frac{2}{v^2}\mathbf{e}_y + \frac{1}{(u^2+v^2)^2}\mathbf{e}_z$$

```
> P1 := plot3d(S, u=1..3, v=1..3, color=aquamarine,
style=wireframe, axes=framed):
> P2 := fieldplot3d(F, u=1..3, v=1..3, z=1..18, arrows=THICK,
grid=[5, 5, 5], color=black):
> display(P1, P2);
```

Полученный график (для удобства просмотра он чуть развернут мышью) представлен на рис. 4.50. Он дает прекрасное представление о сути данной задачи.

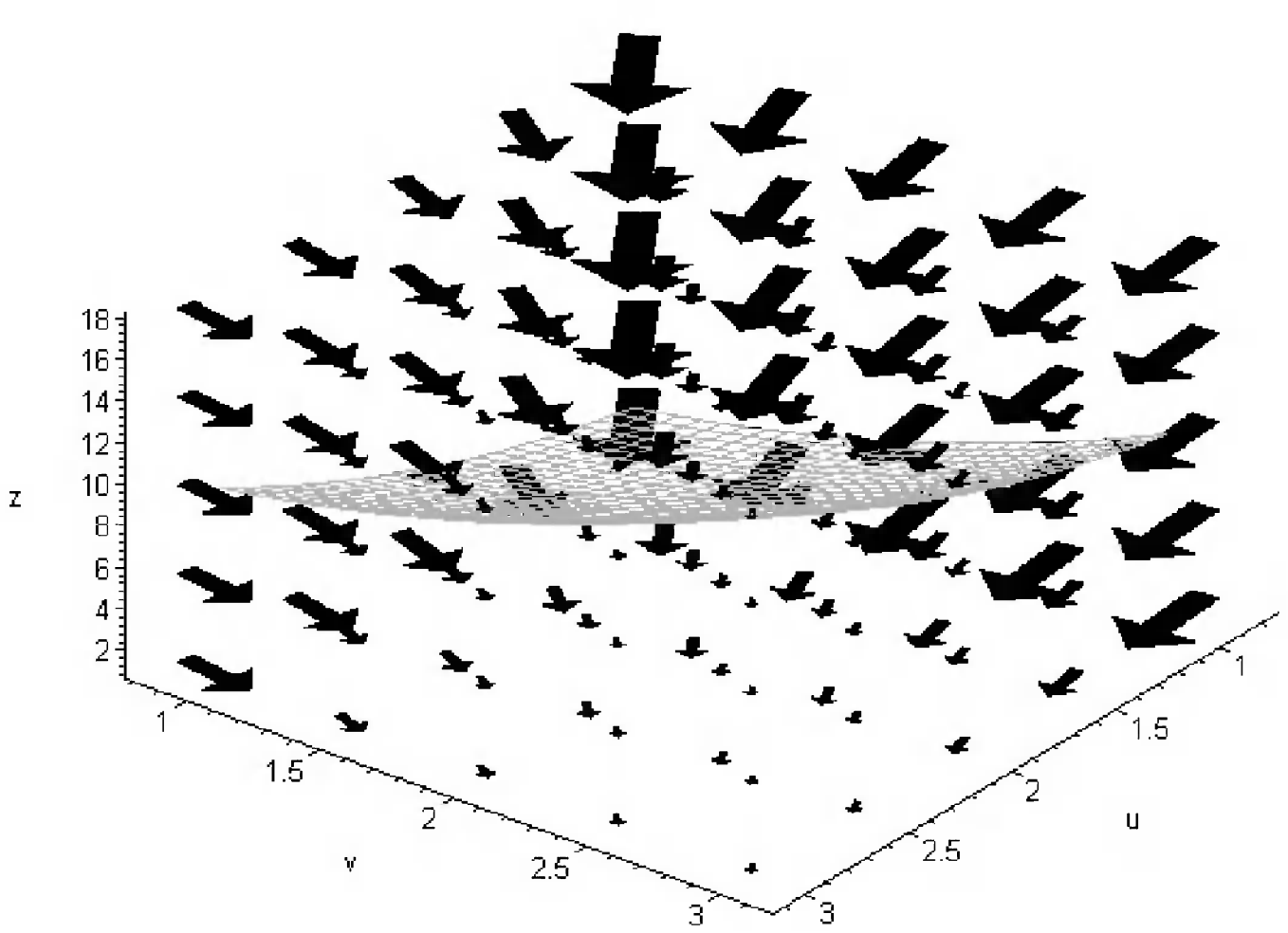


Рис. 4.50. Визуализация задачи из теории поля

Вторая часть задачи завершает заданные вычисления:

```
> dS[u] := diff(S, u);
```

$$dSu := \mathbf{e_x} + 2u\mathbf{e_z}$$

```
> dS[v] := diff(S, v);
```

$$dSv := \mathbf{e_y} + 2v\mathbf{e_z}$$

```
> n := dS[v] &x dS[u];
```

$$n := 2u\mathbf{e_x} + 2v\mathbf{e_y} - \mathbf{e_z}$$

```
>  FdA := F.n;
```

$$FdA := \frac{2}{u} + \frac{4}{v} - \frac{3}{(u^2+v^2)^2}$$

```
> int(int(FdA, u=1..3), v=1..3);
```

$$-\frac{1}{6}\arctan\left(\frac{1}{3}\right) - \frac{3}{2}\arctan(3) + \frac{1}{3} + 12\ln(3) + \frac{5\pi}{12}$$

```
> evalf(%);
```

$$12.98948399$$

Теперь рассмотрим решение второй задачи. Нужно проинтегрировать $F = [x^2, y^2, z^2]$ по поверхности $z = \cos(x*y)$ при $0 \le x, y \le \pi$. Решение этой задачи представлено ниже:

```
> restart: with(LinearAlgebra):
```

```
  with(VectorCalculus):with(linalg):
> F := <x^2, y^2, z^2>;
```

$$F := x^2\mathbf{e_X} + y^2\mathbf{e_Y} + z^2\mathbf{e_Z}$$

```
> S := <u, v, cos(u*v)>;
```

$$S := u\mathbf{e_X} + v\mathbf{e_Y} + \cos(uv)\mathbf{e_Z}$$

```
> F := eval(F, {x=S[1], y=S[2], z=S[3]});
```

$$F := u^2\mathbf{e_X} + v^2\mathbf{e_Y} + \cos(uv)^2\mathbf{e_Z}$$

```
> dS[u] := diff(S, u);
```

$$dSu := \mathbf{e_X} - \sin(uv)v\mathbf{e_Z}$$

```
> dS[v] := diff(S, v);
```

$$dSv := \mathbf{e_Y} - \sin(uv)u\mathbf{e_Z}$$

```
> FdA := <<F>|<dS[u]>|<dS[v]>>;
```

$$FdA := \begin{bmatrix} u^2 & 1 & 0 \\ v^2 & 0 & 1 \\ \cos(uv)^2 & -\sin(uv)v & -\sin(uv)u \end{bmatrix}$$

```
> Int(Int(Determinant(FdA), u=0..Pi), v=0..Pi);
```

$$\int_0^{\pi}\int_0^{\pi} u^2\sin(uv)v + v^2\sin(uv)u + \cos(uv)^2\,du\,dv$$

```
> value(%);
```

$$\frac{1}{4}\frac{-16\cos(\pi^2)+16-8\sin(\pi^2)\pi^2+\mathrm{Si}(2\pi^2)\pi+2\pi^3}{\pi}$$

```
> evalf(%);
```

$$10.44531144$$

4.12.10. Пример вычисления потока через сферу (объемного интеграла)

Заметную роль в решении задач теории поля имеют вычисления (тройных) *объемных интегралов*. Рассмотрим пример на вычисление потока поля F=[x, y, z], проходящего через сферическую поверхность радиуса R. Ограничимся положительным квадрантом.

Зададим основные определения этой задачи и выполним ее графическую визуализацию:

```
> restart: with(plots): with(LinearAlgebra): with(VectorCalculus):
> V := <r*cos(phi)*cos(theta), r*cos(phi)*sin(theta), r*sin(phi)>;
```

$$V := r\cos(\phi)\cos(\theta)\mathbf{e_X} + r\cos(\phi)\sin(\theta)\mathbf{e_Y} + r\sin(\phi)\mathbf{e_Z}$$

```
> F := VectorField(<x, y, z>, 'cartesian'[x, y, z]);
```

$$F := x\bar{\mathbf{e}}_X + y\bar{\mathbf{e}}_Y + z\bar{\mathbf{e}}_Z$$

```
> S := subs(r=2, V);
```

$$S := 2\cos(\phi)\cos(\theta)\mathbf{e_X} + 2\cos(\phi)\sin(\theta)\mathbf{e_Y} + 2\sin(\phi)\mathbf{e_Z}$$

```
> P1 := fieldplot3d(F, x=-2..2, y=0..2, z=0..2, color=red, grid=[4,
4, 4], arrows=SLIM):
```

```
> P2 := plot3d(S, theta=0..Pi, phi=0..Pi/2, color=blue,
style=wireframe):
> display(P1, P2, scaling=constrained, axes=framed);
```

Рисунок 4.51 иллюстрирует суть задачи. На нем изображена часть сферы и стрелками показан поток поля. Фрагмент сферы построен синими линиями, а стрелки – красными, хотя на самом черно-белом рисунке этого не видно.

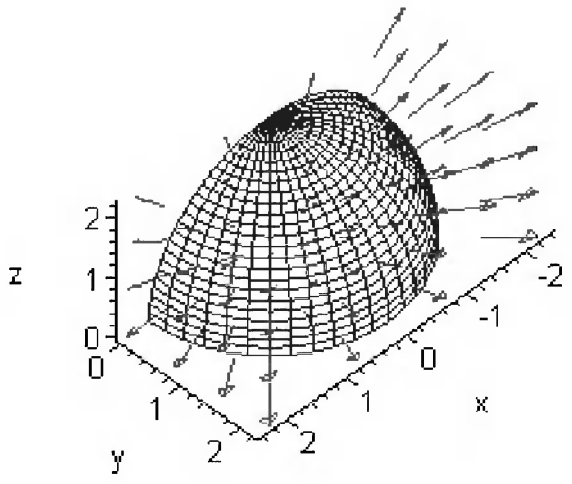


Рис. 4.51. Сферическая поверхность и поток поля через нее

Заключительная часть вычислений представлена ниже:

```
> J := Jacobian(V, [r, phi, theta]);
```

$$J := \begin{bmatrix} \cos(\phi)\cos(\theta) & -r\sin(\phi)\cos(\theta) & -r\cos(\phi)\sin(\theta) \\ \cos(\phi)\sin(\theta) & -r\sin(\phi)\sin(\theta) & r\cos(\phi)\cos(\theta) \\ \sin(\phi) & r\cos(\phi) & 0 \end{bmatrix}$$

```
> dV := simplify(sqrt(Determinant(Transpose(J).J))) assuming real;
```

$$dV := r^2|\cos(\phi)|$$

```
> divF := Divergence(F, [x, y, z]);
```

$$divF := 3$$

```
> Int(Int(Int(divF*dV, r=0..R), phi=0..Pi/2), theta=0..Pi);
```

$$\int_0^{\pi}\int_0^{\frac{\pi}{2}}\int_0^{R} 3r^2\,|\cos(\phi)|\,dr\,d\phi\,d\theta$$

Окончательно имеем для объема следующее выражение:

```
> simplify(4*value(%), symbolic);
```

$$4R^3\pi$$

В пакетах Calculus IV и V, размещенных в Интернете, и в конце главы 8 можно найти множество других примеров применения пакета VectorCalculus в решении задач теории поля и векторного пространства.

4.13. Новые возможности интегрирования в Mathematica 5/6

4.13.1. Интегралы, дающие кусочные функции

В версии СКМ Mathematica 5.1 возможности выполнения операций интегрирования существенно расширены. Так, Mathematica 5.1 легко берет интегралы с кусочной подынтегральной функцией и интегралы, представляемые такими функциями, – рис. 4.52.

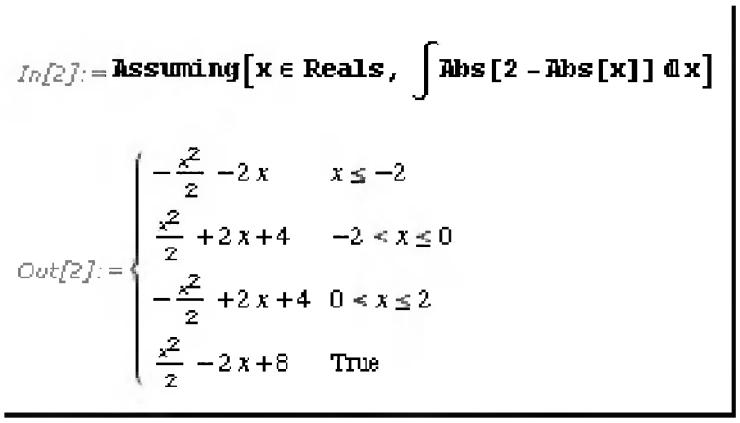


Рис. 4.52. Вычисление интеграла, значение которого представлено кусочной функцией

В этом примере полезно обратить внимание на задание x только вещественных значений и вычисление интегралов нового класса – с подынтегральными функциями, содержащими функции сравнения (в нашем случае это функция вычисления абсолютного значения **Abs**).

4.13.2. Интегралы с импликативными подынтегральными функциями

Еще один вид интегралов, которые поддерживает СКМ Mathematica 5.1/5.2/6, – это интегралы с импликативно заданной подынтегральной функцией. На рис. 4.53 дан пример вычисления такого двойного несобственного интеграла, у которого подынтегральная функция импликативно задает окружность.

Смысл интегрирования представляет рисунок в правой части примера рис. 4.53. В этом примере полезно также обратить внимание на применение булевой функции Boole для задания подынтегральной импликативной функции. Применение этой функции также расширяет класс интегралов, которые способна вычислять СКМ Mathematica.

$$In[3]:= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\left(y^4+x^2\right)\text{Boole}\left[x^2+y^2<1\&\&x<y\right]\,dy\,dx$$

$$Out[3]:= \frac{3\pi}{16}$$

Рис. 4.53. Пример вычисления двойного несобственного интеграла, у которого подынтегральная функция импликативно задает окружность

Еще один пример вычисления подобного интеграла представлен на рис. 4.54. Здесь вычисляется двойной интеграл, импликативная подынтегральная функция которого имеет неизвестный параметр *a*. Решение получено в виде кусочной функции, содержащей этот параметр.

$$In[4]:= \int_{-1}^{1}\int_{-1}^{1}\text{Boole}\left[a\,x^2+y^2<1\right]\,dy\,dx$$

$$Out[4]:= \begin{cases} 4 & a\le 0 \\ \frac{\pi}{\sqrt{a}} & a>1 \\ \frac{2\left(\sin^{-1}\left[\sqrt{a}\right]+\sqrt{1-a}\,\sqrt{a}\right)}{\sqrt{a}} & \text{True} \end{cases}$$

Рис. 4.54. Пример вычисления двойного интеграла с подынтегральной функцией, имеющей дополнительный параметр a

Смысл интегрирования для данного примера поясняет рис. 4.55. В нем представлены область интегрирования и зависимость численного значения интеграла от численного значения параметра *a*.

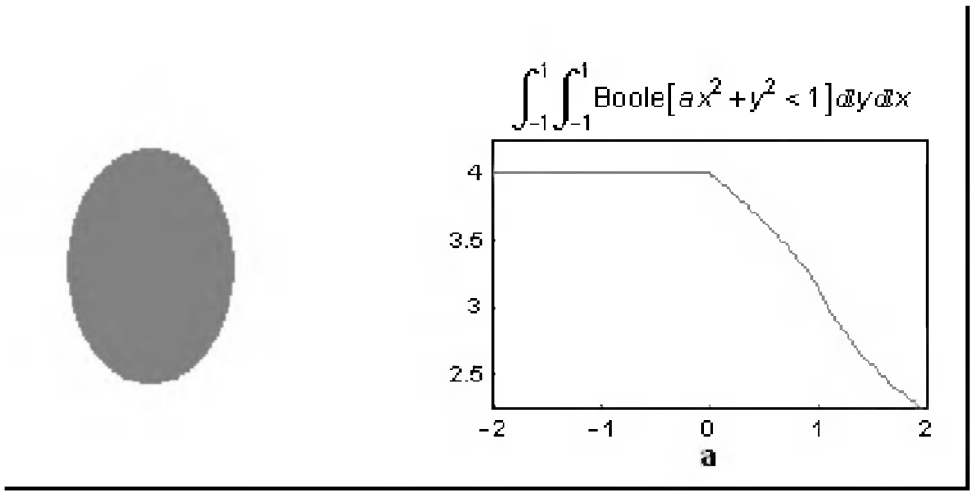


Рис. 4.55. Пояснения к смыслу интегрирования по примеру рис. 4.54

4.13.4. Улучшенная визуализация интегрирования в Mathematica 6

Благодаря возможности создания ноутбуков с *динамическим GUI* система Mathematica 6 имеет улучшенную визуализацию ряда символьных вычислений. Покажем это на простом примере интегрирования выражения Sin[a m!] – рис. 4.56. Интегрирование здесь организовано в модуле Manipulate, который задает с помощью слайдера изменение параметра m от 0 до 10 с шагом 1. Перемещая движок слайдера, можно получить 11 символьных значений этого интеграла. На рис. 4.56 показано значение интеграла при m=1.

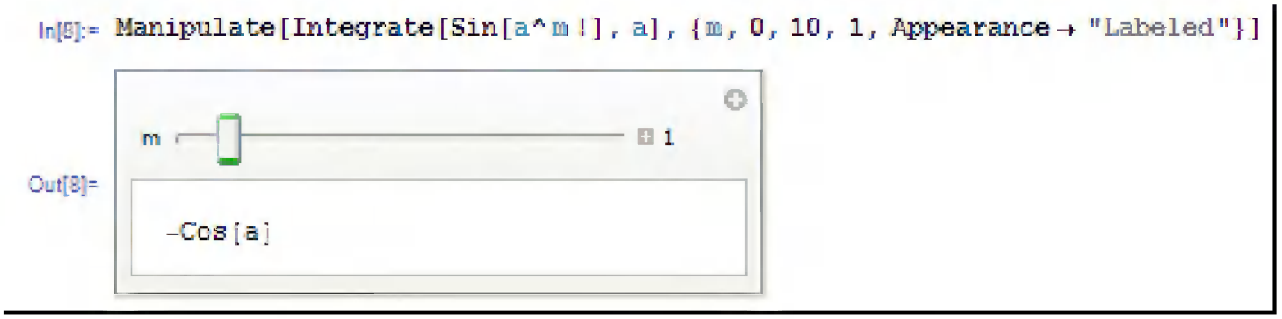


Рис. 4.56. Иллюстрация вычисления интеграла от Sin[a m!] при m=1

На рис. 4.57 показан тот же ноутбук при другом положении движка слайдера, когда m=2. А на рис. 4.58 показан ноутбук, в котором интеграл вычисляется при m=3.

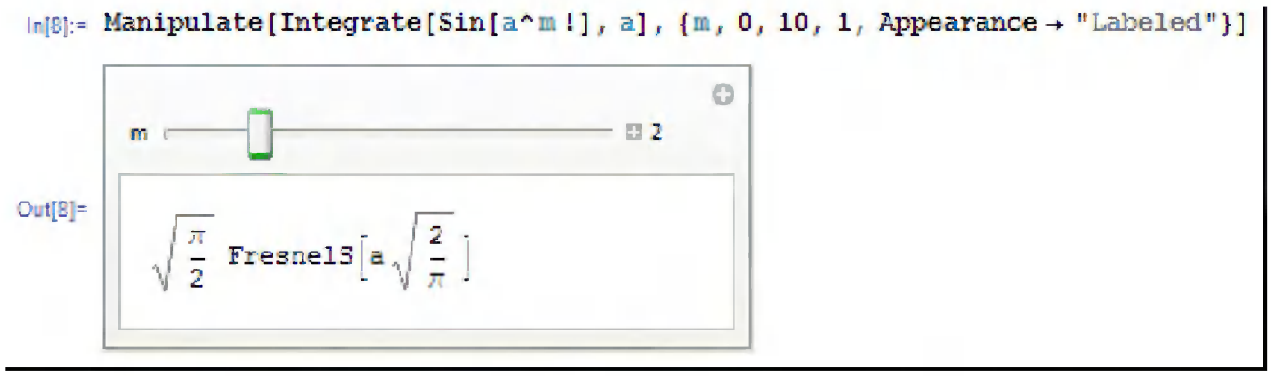


Рис. 4.57. Иллюстрация вычисления интеграла от Sin[a m!] при m=2

Нетрудно заметить, что в данном случае аналитическое выражение для интеграла весьма существенно зависит от установленного слайдером значения m. Работу ноутбука при других положениях движка слайдера читатель может просмотреть самостоятельно.

Естественно, возможности вычисления интегралов новых видов далеко не исчерпываются приведенными примерами. Массу интересных примеров интегрирования можно найти в справке по СКМ Mathematica 5.1/5.2/6.

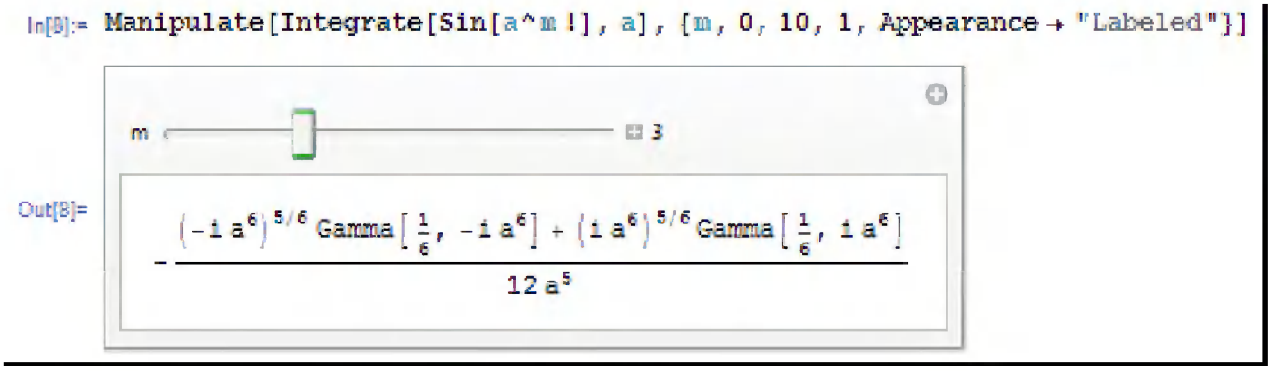


Рис. 4.58. Иллюстрация вычисления интеграла от Sin[a m!] при m=3

Глава 5

Анализ функций и интегральные преобразования

Аналитические функции и *степенные многочлены* (полиномы) широко используются в математике и физике. В этой главе описана работа с функциями и полиномами, включающая в себя традиционный анализ функций, выявляющий их особенности и обеспечивающий различные преобразования функций, вычисление и преобразование полиномов, в том числе ортогональных [123–126]. Описаны таже средства интегральных преобразований, широко используемых в научно-технических расчетах.

5.1. Анализ функциональных зависимостей

5.1.1. Понятие о функциональных зависимостях

Говорят, что $y(x)$ есть *функция*, если известно правило, согласно которому каждому значению аргумента x соответствует некоторое значение y. Возможны и функции двух и более переменных, например функции вида $z(x,y)$ для описания поверхностей или функции Бесселя разного порядка.

Функциональная зависимость или функция $f(x)$ даже от одной переменной может быть достаточно сложной и содержать различные *особенности*: корни (значения x, при которых $f(x) = 0$), полюса (значения x, при которых $f(x) \to \infty$), максимумы и минимумы, разрывы, асимптотические значения, точки перегиба и т. д. К сожалению, пока нет средств, сразу выявляющих все особенности функциональных зависимостей, поскольку даже средства, решающие частные задачи анализа функций, довольно сложны и специфичны. Достаточно отметить проблему поиска экстремумов функций (особенно функций нескольких переменных). Поэтому функции приходится анализировать индивидуально.

5.1.2. Поиск экстремумов функций по нулям первой производной

Для простых функций одной переменной $f(x)$ поиск экстремумом часто сводят к нахождению точек, в которых первая производная $f(x)$ обращается в нуль. Для этого можно использовать функцию `fsolve` (иногда и `solve`, но она дает вывод в более сложной форме). Приведем пару примеров:

```
> y:=expand((x-3)*(x-1)*x*(x+2));
```

$$y := x^4 - 2x^3 - 5x^2 + 6x$$

```
> dy:=simplify(diff(y,x));
```

$$dy := 4x^3 - 6x^2 - 10x + 6$$

```
> plot({y,dy},x=-3..3,-10..10,color=black,thickness=[2,1]):
> extrem:=fsolve(yprime=0,x);
```

$$exrtem := -1.302775638, 0.5000000000, 2.302776538$$

В этом примере создан полином y с корнями 3, 1, 0 и –2 и найдена его производная dy (см. рис. 5.1). Из рисунка видно, что полином p имеет экстремумы в точках, лежащих в промежутках между корневыми точками. Их значения и найдены как значения переменной extrem, для которых вторая производная равна 0. Рекомендуется проверить вид вывода, если `fsolve` заметить на `solve`.

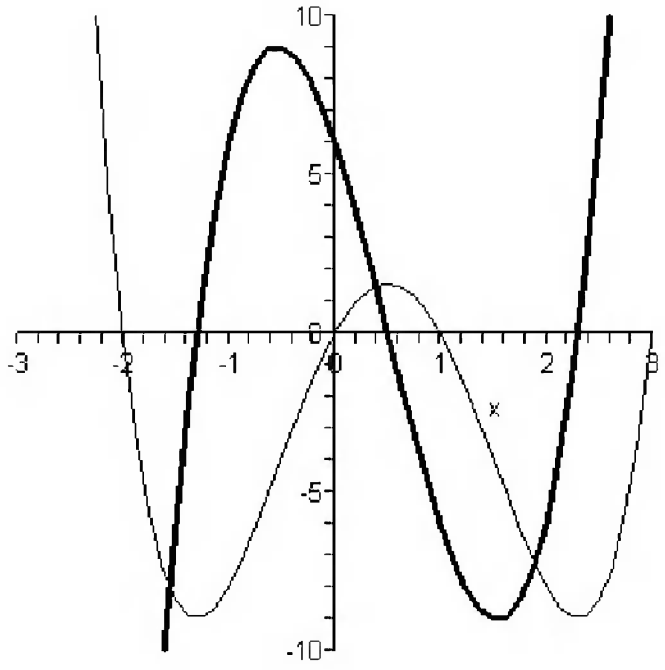


Рис. 5.1. График функциональной зависимости – полинома и ее производной (жирная кривая)

Возьмем еще один пример для поиска экстремумов выражения $\sin(x)/x$. Это выражение имеет бесконечное число экстремумов слева и справа от $x = 0$ (в этой точке расположен главный максимум со значением 1). Ограничимся поиском трех экстремумов в интервале изменения x от 3 до 12:

```
> f:=sin(x)/x:df:=diff(f,x);
```

$$df := \frac{\cos(x)}{x} - \frac{\sin(x)}{x^2}$$

```
>  plot({f,df},x=0..12,color=black,thickness=[2,1]):
> [fsolve(df,x=3..6),fsolve(df,x=7..9),fsolve(df,x=9..12)];
```

[4.493409458, 7.725251837, 10.90412166]

Тут уже приходится искать каждый экстремум поодиночке, задавая поиск в соответствующем интервале изменения x. Для просмотра графика функциональной зависимости и ее проиводной достаточно в конце второй строки ввода заменить знак «:» на «;».

5.1.3. Поиск экстремумов в аналитическом виде

Функция `solve` нередко позволяет найти экстремумы в аналитическом виде как нули первой производной. Приведем примеры этого:

```
> restart:y:=exp(-a*x)-exp(-b*x);dy:=diff(y,x);
```

$$y := e^{(-ax)} - e^{(-bx)}$$

$$dy := -ae^{(-ax)} + be^{(-bx)}$$

```
> solve(dy,x);
```

$$\frac{\ln\left(\frac{a}{b}\right)}{a-b}$$

```
> restart:y:=a*x*exp(-b*x);dy:=diff(y,x);
```

$$y := axe^{(-bx)}$$

$$dy := ae^{(-bx)} - axbe^{(-bx)}$$

```
> solve(dy,x);
```

$$\frac{1}{b}$$

Этот метод иногда можно распространить на случай ряда переменных. Ниже представлен такой пример для функции двух переменных:

```
> restart:
> z:=(x,y)-> a*x^2 + b*x*y + c*y^2 + d*(x-y);
```

$$z := (x, y) \to ax^2 + bxy + cy^2 + d(x - y)$$

```
> xy:=solve({diff(z(x,y),x) = 0, diff(z(x,y),y) = 0},{x,y});
```

$$xy := \{y = \frac{d(b+2a)}{-b^2+4ac}, x = -\frac{d(b+2c)}{-b^2+4ac}\}$$

```
> z(rhs(xy[2]),rhs(xy[1]));
```

$$\frac{ad^2(b+2c)^2}{(-b^2+4ac)^2} - \frac{bd^2(b+2c)(b+2a)}{(-b^2+4ac)^2} + \frac{cd^2(b+2a)^2}{(-b^2+4ac)^2}$$
$$+ d\left(-\frac{d(b+2c)}{-b^2+4ac} - \frac{d(b+2a)}{-b^2+4ac}\right)$$

```
> simplify(%);
```

$$-\frac{d^2(b+c+a)}{-b^2+4ac}$$

Разумеется, подобное решение возможно далеко не всегда, хотя и частные решения данной задачи представляют значительный интерес.

5.1.4. Поиск максимума амплитудно-частотной характеристики

Одной из практически важных задач может служить нахождение пика амплитудно-частотной характеристики слабо демпфированной системы с массой m и частотой собственных колебаний w0. Эту характеристику можно представить следующим известным выражением:

```
> restart;
> A:=A0/sqrt(m^2*(omega0^2-omega^2)^2+gamma^2*omega^2);
```

$$A := \frac{A0}{\sqrt{m^2(\omega 0^2 - \omega^2)^2 + \gamma^2\omega^2}}$$

Найдя ее производную и вычислив корни последней, получим:

```
> dA:=diff(A,omega);
```

$$dA := -\frac{A0(-4m^2(\omega 0^2 - \omega^2)\omega + 2\gamma^2\omega)}{2(m^2(\omega 0^2 - \omega^2)^2 + \gamma^2\omega^2)^{(3/2)}}$$

```
> ss:=solve(dA=0,omega);
```

$$ss := 0, \frac{\sqrt{4m^2\omega 0^2 - 2\gamma^2}}{2m}, -\frac{\sqrt{4m^2\omega 0^2 - 2\gamma^2}}{2m}$$

Из этих трех частот только одна физически реальна – средняя. Остальные могут быть отброшены. А теперь приведем пример с конкретными числовыми данными:

```
> AA:=subs(A0=5,omega0=10,m=1,gamma=1,A);
```

$$AA := \frac{5}{\sqrt{(100 - \omega^2)^2 + \omega^2}}$$

```
> AAprime:=diff(AA,omega);
```

$$AAprime := -\frac{5(-4(100 - \omega^2)\omega + 2\omega)}{2((100 - \omega^2)^2 + \omega^2)^{(3/2)}}$$

```
> ss1:=solve(AAprime=0,omega);
```

$$ss1 := 0, \frac{\sqrt{398}}{2}, -\frac{\sqrt{398}}{2}$$

```
> evalf(ss1);
```

$$0., 9.974968670, -9.974968670$$

Нетрудно подметить, что частота пика амплитудно-частотной характеристики чуть меньше частоты собственных колебаний системы.

5.1.5. Поиск экстремумов с помощью функции extrema

Ряд функций служит специально для вычисления *экстремумов*, максимумов и минимумов функций, а также для определения их непрерывности. Одна из таких функций, `extrema`, позволяет найти экстремумы выражения `expr` (как максимумы, так и минимумы) при ограничениях `constrs` и переменных `vars`, по которым ищется экстремум:

```
extrema(expr, constrs)          extrema(expr, constrs, vars)
extrema(expr, constrs, vars, 's')
```

Ограничения `contrs` и переменные `vars` могут задаваться одиночными объектами или списками ряда ограничений и переменных. Найденные координаты точки экстремума присваиваются переменной `'s'`. При отсутствии ограничений в виде равенств или неравенств вместо них записывается пустой список {}.

Эта функция в старых версиях Maple находилась в стандартной библиотеке и вызывалась командой `readlib(extrema)`. Но, начиная с Maple 7, ее можно использовать без предварительного объявления. В этом убеждают приведенные ниже примеры:

```
> restart:z:=(x,y)-> a*x^2 + b*x*y + c*y^2 + d*(x-y);
```

$$z := (x, y) \to ax^2 + bxy + cy^2 + d(x - y)$$

```
> extrema(z(x,y),{},{x,y},'s');
```

$$\{-\frac{d^2(b+c+a)}{-b^2+4ac}\}$$

```
> s;
```

$$\{\{y=\frac{d(2a+b)}{-b^2+4ca}, x=-\frac{d(b+2c)}{-b^2+4ca}\}\}$$

```
> extrema(a*x^2+b*x+c,{},x,'s');s;
```

$$\{-\frac{1}{4}\frac{b^2-4ca}{a}\}$$

$$\{\{x=-\frac{1}{2}\frac{b}{a}\}\}$$

```
> extrema(x*exp(-x),{},x,'s');s;
```

$$\{e^{(-1)}\}$$

$$\{\{x = 1\}\}$$

```
> extrema(sin(x)^2,{},x,'s');s;
```

$$\{0, 1\}$$

$$\{\{x=0\}, \{x=\frac{1}{2}\pi\}\}$$

```
> extrema(x+y/z,x^2+y^2+z^2=1,{x,y,z},'s');s;
```

$$\{\max(1 - \text{RootOf}(\_Z^4 + 1)^2, -1 + \text{RootOf}(\_Z^4 + 1)^2),$$
$$\min(1 - \text{RootOf}(\_Z^4 + 1)^2, -1 + \text{RootOf}(\_Z^4 + 1)^2)\}$$

$$\{\{z = \text{RootOf}(\_Z^4 + 1), x = -1, y = \text{RootOf}(\_Z^4 + 1)^3\},$$
$$\{x = 1, z = \text{RootOf}(\_Z^4 + 1), y = -\text{RootOf}(\_Z^4 + 1)^3\}\}$$

```
> evalf(%);
```

$$\{\{x = -1., y = -0.7071067812 + 0.7071067812I,$$
$$z = 0.7071067812 + 0.7071067812I\}, \{$$
$$z = 0.7071067812 + 0.7071067812I, x = 1.,$$
$$y = 0.7071067812 - 0.7071067812I\}\}$$

Как видно из приведенных примеров, функция `extrema` возвращает как значения экстремумов, так и значения аргументов, при которых экстремумы наблюдаются. Для проверки оптимизационных алгоритмов существует ряд тестовых функций. Одна из таких функций – функция двух переменных Розенброка. В представленном ниже примере функция `extrema` не справляется с поиском минимума этой функции:

```
> rf:=(x,y)->100*(y-x^2)^2+(1-x)^2;
```

$$rf := (x, y) \to 100(y - x^2)^2 + (1 - x)^2$$

```
> extrema(rf(x,y),{x,y},'s');s;
```

$$s$$

Функция `extrema` дает неплохие результаты при поиске экстремумов простых аналитических функций, не имеющих особенностей. Однако при анализе сложных функций, содержащих функции со сравнением аргумента (например, `abs(x)`, `signum(x)` и др.), функция `extrema` часто отказывается работать и просто повторяет запись обращения к ней.

5.1.6. Поиск минимумов и максимумов аналитических функций

Часто нужно найти минимум или максимум заданной функции. Для поиска минимумов и максимумов выражений (функций) `expr` служат функции стандартной библиотеки:

```
minimize(expr, opt1, opt2, ..., optn)
maximize(expr, opt1, opt2, ..., optn)
```

Эти функции могут разыскивать максимумы и минимумы для функций как одной, так и нескольких переменных. С помощью опций `opt1`, `opt2`, ..., `optn` можно указывать дополнительные данные для поиска. Например, параметр `"infinity"` означает, что поиск минимума или максимума выполняется по всей числовой оси, а параметр `location` (или `location=true`) дает расширенный вывод результатов поиска – выдается не только значение минимума (или максимума), но и значения переменных в этой точке.

Примеры применения функции `minimize` приведены ниже:

```
> minimize(x^2-3*x+y^2+3*y+3);
```

$$\frac{-3}{2}$$

```
> minimize(x^2-3*x+y^2+3*y+3, location);
```

$$\frac{-3}{2}, \left\{ \left[\{y = \frac{-3}{2}, x = \frac{3}{2}\}. \frac{-3}{2} \right] \right\}$$

```
> minimize(x^2-3*x+y^2+3*y+3, x=2..4, y=-4..-2, location);
```

$$-1, \{[\{x = 2, y = -2\}, -1]\}$$

```
> minimize(x^2+y^2,x=-10..10,y=-10..10);
```

$$0$$

```
> minimize(x^2+y^2,x=-10..10,y=-10..10,location);
```

$$0, \{[\{y = 0, x = 0\}, 0]\}$$

```
> minimize(abs(x*exp(-x^2)-1/2), x=-4..4);
```

$$\frac{1}{2} - \frac{1}{2}\sqrt{2}\,\mathbf{e}^{(-1/2)}$$

```
> minimize(abs(x*exp(-x^2)-1/2), x=-4..4,location=true);
```

$$\frac{1}{2} - \frac{1}{2}\sqrt{2}\,\mathbf{e}^{(-1/2)}, \left\{ \left[x = \frac{1}{2}\sqrt{2}, \frac{1}{2} - \frac{1}{2}\sqrt{2}\,\mathbf{e}^{(-1/2)} \right] \right\}$$

Приведем подобные примеры и для функции поиска максимума – `maximize`:

```
> maximize(x*exp(-x));
```

$$\mathbf{e}^{(-1)}$$

```
> maximize(x*exp(-x),location);
```

$$\mathbf{e}^{(-1)}, \{[\{x = 1\}, \mathbf{e}^{(-1)}]\}$$

```
> maximize(sin(x)/x,x=-2..2,location);
```

$$1, \{[\{x = 0\}, 1]\}$$

```
> maximize(exp(-x)*sin(y),x=-10..10,y=-10..10,location);
```

$$\mathbf{e}^{10}, \left\{ \left[\{y = -\frac{3}{2}\pi, x = -10\}, \mathbf{e}^{10} \right], \left[\{x = 10, y = \frac{5}{2}\pi\}, \mathbf{e}^{10} \right], \left[\{y = \frac{1}{2}\pi, x = -10\}, \mathbf{e}^{10} \right] \right\}$$

Применим функцию `minimize` для поиска минимума тестовой функции Розенброка. Рисунок 5.2 показывает, что `minimize` прекрасно справляется с данной задачей. На рис. 5.2 представлено также построение функции Розенброка, хорошо иллюстрирующее ее особенности.

Трудность поиска минимума функции Розенброка связана с ее характерными особенностями. Из рис. 5.2 видно, что эта функция представляет собой поверхность типа «глубокого оврага с почти плоским дном», в котором и расположена точка минимума. Такая особенность этой функции существенно затрудняет поиск минимума. То, что система Maple справляется с данной тестовой функцией, вовсе не означает, что трудности в поиске минимума или максимума других функций остаются позади.

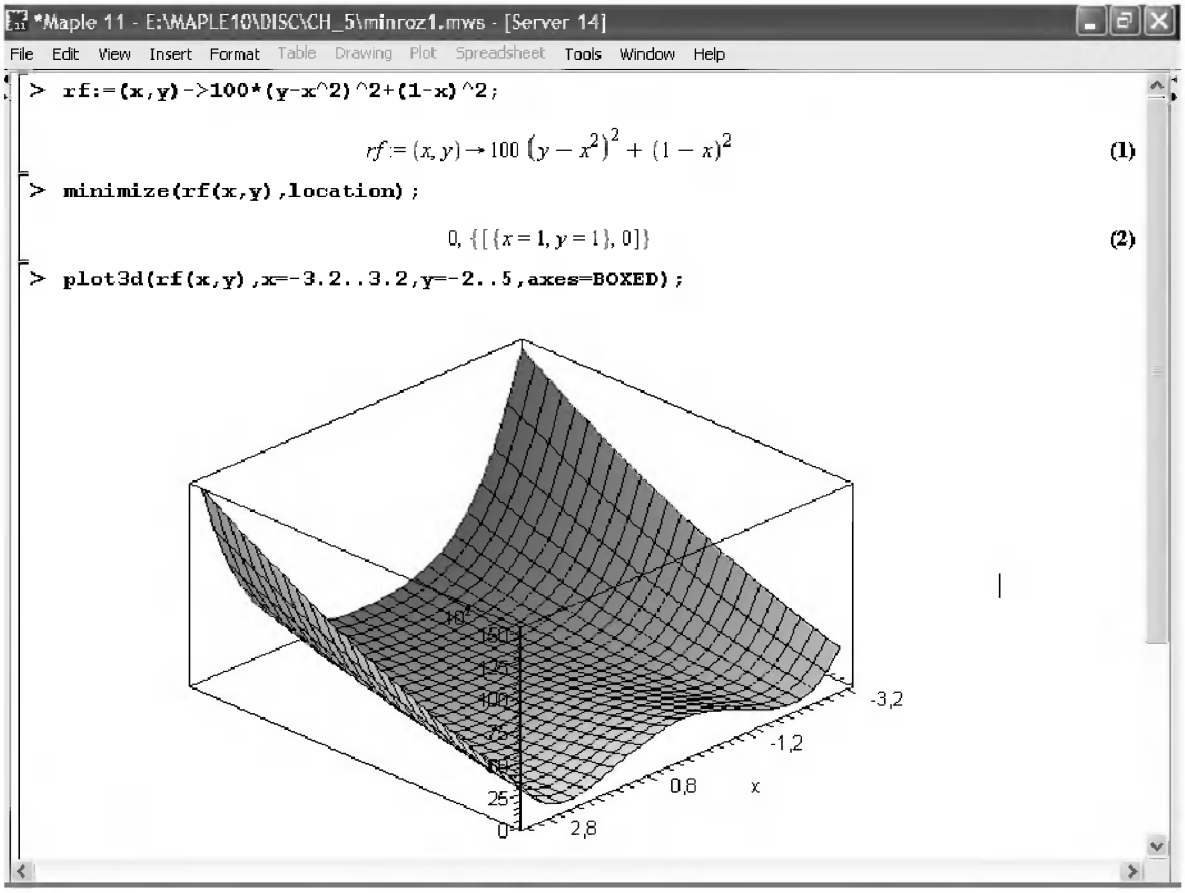


Рис. 5.2. Поиск минимума функции Розенброка и построение ее графика

5.1.7. Поиск минимума функций с ограничениями методом выпуклого программирования

Часто необходимо найти минимум некоторой функции при наличии *ограничений* на значения независимых переменных. Ниже рассматривается нетривиальная задача такого рода, решаемая методом выпуклого программирования (разновидность нелинейного программирования).

Пусть надо найти минимум функции f := x1^2 + (x2 – 1)^2 при следующих ограничениях: 2x1+x2>=7, x1+2*x2>=5, x1>=0 и x2>=0. Составим на основе этого функцию Лагранжа:

```
> F:=x1^2+(x2-1)^2+y1*(7-2*x1-x2)+y2*(5-x1-2*x2);
```

$$F := x1^2 + (x2-1)^2 + y1(7-2x1-x2) + y2(5-x1-2x2)$$

и найдем ее частные производные:

```
> F1:=diff(F,x1);
```

$$F1 := 2x1 - 2y1 - y2$$

```
> F2:=diff(F,x2);
```

$$F2 := 2x2 - 2 - y1 - 2y2$$

```
> F3:=diff(F,y1);
```

$$F3 := 7 - 2x1 - x2$$

```
> F4:=diff(F,y2);
```

$$F4 := 5 - x1 - 2x2$$

Соберем воедино все равенства и неравенства этой задачи:

```
> eq:={F1=u1,F2=u2,x1*F1,x2*F2,F3+v1,F4+v2,y1*F3,y2*F4,
       x1>=0,x2>=0,y1>=0,y2>=0,u1>=0,u2>=0,v1>=0,v2>=0};
```

$$eq := \{2x1 - 2y1 - y2 = u1,\ 2x2 - 2 - y1 - 2y2 = u2,\ 7 - 2x1 - x2 + v1,$$
$$5 - x1 - 2x2 + v2,\ y1(7 - 2x1 - x2),\ y2(5 - x1 - 2x2),$$
$$x1(2x1 - 2y1 - y2),\ x2(2x2 - 2 - y1 - 2y2),\ 0 \le x1,\ 0 \le x2,\ 0 \le y1,$$
$$0 \le y2,\ 0 \le u1,\ 0 \le v1,\ 0 \le v2,\ 0 \le u2\}$$

Первые шесть равенств соответствуют теореме Куна-Такера о том, что в точке минимума существуют целые неотрицательные числа u1, u2, v1 и v2, для которых выполняются эти шесть равенств (обратите внимание на то, что запись только левой части равенства означает, что она приравнивается к 0). Теперь с помощью функции solve можно найти решение данной задачи:

```
> solve(eq,{x1,x2,y1,y2,u1,u2,v1,v2});
```

$$\{u2 = 0, u1 = 0, v1 = 0, v2 = \frac{9}{5}, y2 = 0, x2 = \frac{11}{5}, x1 = \frac{12}{5}, y1 = \frac{12}{5}\}$$

Таким образом, на указанном множестве функция достигает минимума в точке (12/5, 11/5).

На интернет-сайте корпорации Waterloo Maple можно найти много материалов по реализации различных методов оптимизации функций. Так, в файле NLPcode7.mws имеется полная реализация пакета расширения по нелинейному программированию для системы Maple 7/8. Очень много примеров на оптимизацию функций дано в пакете Calculus IV. В частности, там содержатся примеры на оптимизацию с применением множителей Лагранжа и методов, использующих векторный анализ.

5.1.8. Анализ функций на непрерывность

Для исследования функций на *непрерывность* (отсутствие разрывов) Maple имеет функцию `iscont`, записываемую в ряде форм:

```
iscont(expr, x = a .. b) iscont(expr, x = a .. b, 'closed')
iscont(expr, x = a .. b, 'open')
```

Она позволяет исследовать выражение `expr`, заданное в виде зависимости от переменной `x`, на непрерывность. Если выражение непрерывно, возвращается логическое значение `true`, иначе – `false`. Возможен также результат типа `FAIL`. Параметр `'closed'` показывает, что конечные точки должны также про-

веряться, а указанный по умолчанию параметр `'open'` – что они не должны проверяться.

Работу функции `iscont` иллюстрируют следующие примеры:

```
> iscont(1/x^2,x=-1..1);
```

$$false$$

```
> iscont(1/x^2,x=-1..1,'closed');
```

$$false$$

```
> iscont(1/x,x=0..1);
```

$$true$$

```
> iscont(1/x,x=0..1,'closed');
```

$$false$$

```
> iscont(1/(x+a),x=-1..1);
```

$$FAIL$$

Рекомендуется внимательно присмотреться к результатам этих примеров и опробовать свои собственные примеры.

5.1.9. Определение точек нарушения непрерывности

Функции, не имеющие непрерывности, доставляют много хлопот. Поэтому важным представляется анализ функций на непрерывность. Начиная с Maple 7 функция `discont(f,x)` позволяет определить точки, в которых нарушается непрерывность функции $f(x)$. Она вычисляет все точки в пределах изменения x от $-\infty$ до $+\infty$. Результаты вычислений могут содержать особые *экстрапеременные* с именами вида `_Zn~` и `_NNn~`. В частности, они позволяют оценить периодические нарушения непрерывности функций. Примеры применения функции `discont` приведены ниже:

```
> discont(1/(x-2),x);
```

$$\{2\}$$

```
> discont(1/((x-1)*(x-2)*(x-3)),x);
```

$$\{1, 2, 3\}$$

```
> discont(GAMMA(x/2),x);
```

$$\{-2\ \_NN1\sim\}$$

Весьма рекомендуется, наряду с применением функции `discont`, просмотреть график анализируемой функции.

Еще раз полезно обратить внимание на то, что в ряде примеров в выводе используются специальные переменные вида *_NameN~*, где *Name* – имя переменной и N – ее текущий номер. После выполнения команды restart отсчет N начинается с 1. Если вывод с такими переменными уже применялся, то их текущие номера могут казаться произвольными. Специальные переменные часто используются для упрощения выводимых выражений.

5.1.10. Нахождение сингулярных точек функции

Многие операции, такие как интегрирование и дифференцирование, чувствительны к *особенностям* функций, в частности к их разрывам и особым точкам. Функция `singular(expr, vars)` позволяет найти особые (сингулярные) точки выражения `expr`, в которых она испытывает разрывы. Дополнительно в числе параметров может указываться необязательный список переменных. Примеры применения этой функции приведены ниже:

```
> singular(ln(x)/(x^2-a));
```

$$\{a = a, x = 0\}, \{a = x^2, x = x\}$$

```
> singular(tan(x));
```

$$\{x = \_Z22 \sim \pi + \frac{1}{2}\pi\}$$

```
> singular(1/sin(x));
```

$$\{x = \pi \_Z23\sim\}$$

```
> singular(Psi(x*y),{x,y});
```

$$\{y = y, x = -\frac{\_N1\sim -1}{y}\}$$

```
> singular(x+y+1/x,{x,y});
```

$$\{y = y, x = 0\}, \{y = y, x = -\infty\}, \{y = \infty, x = x\}, \{y = -\infty, x = x\}, \{x = \infty, y = y\}$$

5.1.11. Вычисление асимптотических и иных разложений

Важным достоинством системы Maple является наличие в ней ряда функций, позволяющих выполнять детальный анализ функций. К такому анализу относится вычисление *асимптотических разложений* функций, которые представляются в виде рядов (не обязательно с целыми показателями степени). Для этого используется следующая функция:

```
asympt(f,x)        asympt(f,x,n)
```

Здесь `f` – функция переменной `x` или алгебраическое выражение; `x` – имя переменной, по которой производится разложение; `n` – положительное целое число (порядок разложения, по умолчанию равный 6). Ниже представлены примеры применения этой функции:

```
> asympt(x/(1-x^2),x);
```

$$-\frac{1}{x} - \frac{1}{x^3} - \frac{1}{x^5} + O\left(\frac{1}{x^7}\right)$$

```
> asympt(n!,n,3);
```

$$\frac{\frac{\sqrt{2}\sqrt{\pi}}{\sqrt{\frac{1}{n}}}+\frac{1}{12}\sqrt{2}\sqrt{\pi}\sqrt{\frac{1}{n}}+\frac{1}{288}\sqrt{2}\sqrt{\pi}\left(\frac{1}{n}\right)^{(3/2)}+\mathrm{O}\left(\left(\frac{1}{n}\right)^{(5/2)}\right)}{\left(\frac{1}{n}\right)^{n}\mathbf{e}^{n}}$$

```
> asympt(exp(x^2)*(1-exp(x)),x);
```

$$-\mathbf{e}^{(x^2)}\mathbf{e}^{x}+\mathbf{e}^{(x^2)}$$

```
> asympt(sqrt(Pi/2)*BesselJ(0,x),x,3);
```

$$\sin\left(x+\frac{\pi}{4}\right)\sqrt{\frac{1}{x}}-\frac{1}{8}\cos\left(x+\frac{\pi}{4}\right)\left(\frac{1}{x}\right)^{(3/2)}-\frac{9}{128}\sin\left(x+\frac{\pi}{4}\right)\left(\frac{1}{x}\right)^{(5/2)}+\mathrm{O}\left(\left(\frac{1}{x}\right)^{(7/2)}\right)$$

5.1.12. Пример анализа сложной функции

Ниже мы рассмотрим типичный анализ достаточно «сложной» функции, имеющей в интересующем нас интервале изменения аргумента x от –4 до 4, нули, максимумы и минимумы. Определение функции $f(x)$, ее графики и график производной $\mathrm{d}F(x)/\mathrm{d}x$ даны на рис. 5.3. Этот рисунок является началом полного документа, описываемого далее.

Функция $F(x)$, на первый взгляд, имеет не совсем обычное поведение вблизи начала координат (точки с $x = y = 0$). Для выяснения такого поведения разумно построить график функции при малых x и y. Он также представлен на рис. 5.3 (нижний график) и наглядно показывает, что экстремум вблизи точки (0,0) является обычным минимумом, немного смещенным вниз и влево от начала координат.

Теперь перейдем к анализу функции $F(x)$. Для поиска нулей функции (точек пересечения оси x) удобно использовать функцию `fsolve`, поскольку она позволяет задавать область изменения x, внутри которой находится корень. Эти области несложно оценить из графика функции. Как видно из приведенных ниже примеров, анализ корней $F(x)$ не вызвал никаких трудностей, и все корни были уточнены сразу:

```
Поиск нулей функции
> fsolve(F(x),x,-2...-1);
```

$$-1.462069476$$

```
> fsolve(F(x),x,-.01..0.01);
```

$$0.$$

```
> fsolve(F(x),x,-.05..0);
```

$$-.02566109292$$

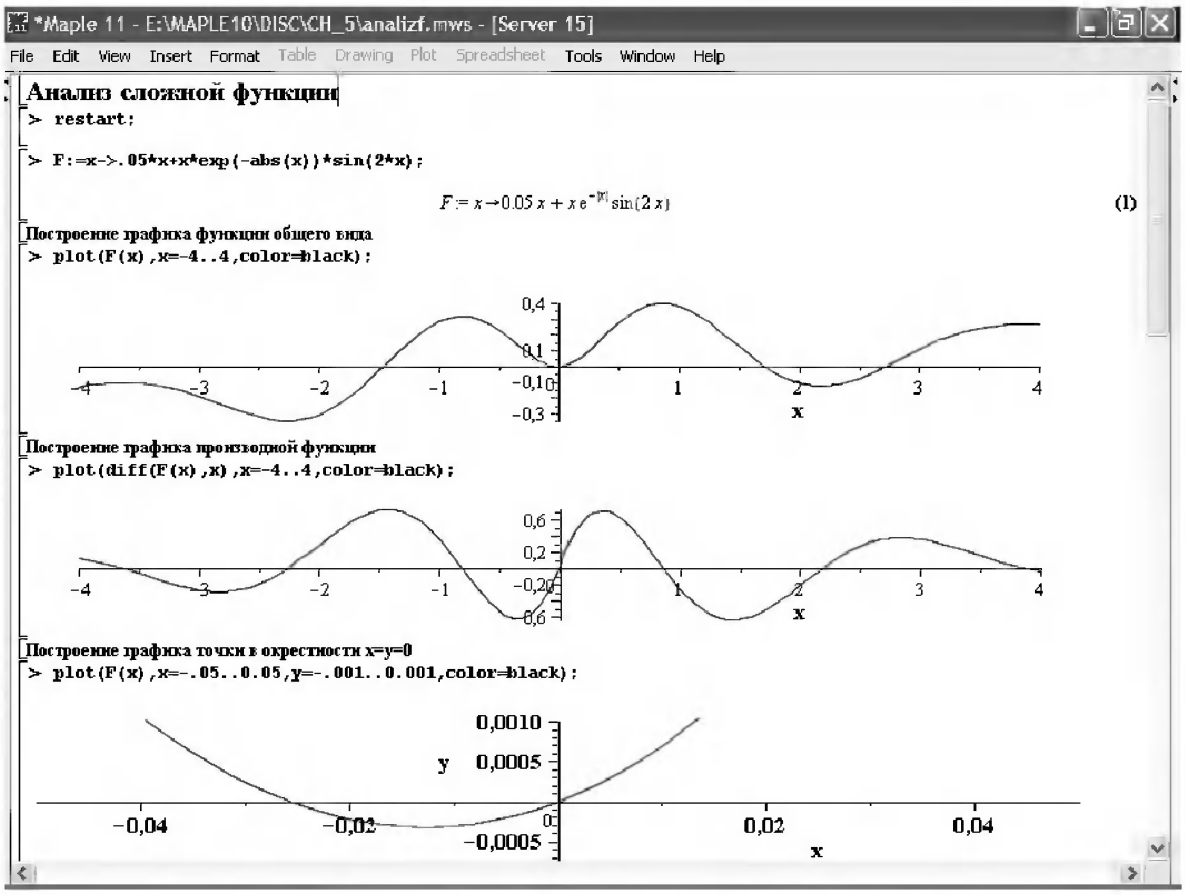


Рис. 5.3. Задание функции F(x) и построение графиков функции и ее производной

```
> fsolve(F(x),x,1..2);
```

$$1.710986355$$

```
> fsolve(F(x),x,2.5..3);
```

$$2.714104921$$

Нетрудно заметить, что функция имеет два очень близких (но различных) корня при x, близких к нулю.

Анализ функции на непрерывность, наличие ее нарушений и сингулярных точек реализуются следующим образом:

```
> iscont(F(x),x=-4..4);
```

$$true$$

```
> discont(F(x),x);
```

$$\{\ \}$$

```
> singular(F(x));
```

$$\{x = \infty\}, \{x = -\infty\}$$

Этот анализ не выявляет у заданной функции каких-либо особенностей. Однако это не является поводом для благодушия – попытка найти экстремумы $F(x)$ с помощью функции `extrema` и минимумы с помощью функции `minimize` завершается полным крахом:

```
> extrema(F(x),{},x,'s');s;
```

$$s$$

```
> minimize(F(x),x=-.1...1);
```

$$\text{minimize}(.05x + x\mathbf{e}^{(-|x|)}\sin(2x), x = \text{-}.1 \,..\, 1)$$

```
> minimize(F(x),x=-2.5..-2);
```

$$\text{minimize}(.05x + x\mathbf{e}^{(-|x|)}\sin(2x), x = \text{-}2.5 \,..\, \text{-}2)$$

Приходится признать, что в данном случае система Maple ведет себя далеко не самым лучшим способом. Чтобы довести анализ $F(x)$ до конца, придется вспомнить, что у функции без особенностей максимумы и минимумы наблюдаются в точках, где производная меняет знак и проходит через нулевое значение. Таким образом, мы можем найти минимумы и максимумы по критерию равенства производной нулю:

```
Поиск минимумов по критерию равенства нулю производной
> fsolve(diff(F(x),x)=0,x,-.5...5);
```

$$\text{-}.01274428224$$

```
> xm:=%;
```

$$xm := \text{-}.0003165288799$$

```
> [F(xm),F(xm+.001),F(xm-.001)];
```

$$[\text{-}.00001562612637,\ .00003510718293,\ \text{-}.00006236451216]$$

```
> fsolve(diff(F(x),x)=0,x,-2.5..-2);
```

$$\text{-}2.271212360$$

```
> fsolve(diff(F(x),x)=0,x,2..2.5);
```

$$2.175344371$$

```
Неудачный поиск максимума
> maximize(F(x),x=-1..-.5);
```

$$\text{maximize}(.05x + x\mathbf{e}^{(-|x|)}\sin(2x), x = \text{-}1 \,..\, \text{-}.5)$$

```
Поиск максимумов по критерию равенства нулю производной
> fsolve(diff(F(x),x),x,-1..-.5);
```

$$\text{-}.8094838517$$

```
> fsolve(diff(F(x),x),x,.5..2);
```

$$.8602002115$$

```
> fsolve(diff(F(x),x),x,-4..-3);
```

$$\text{-}3.629879137$$

```
> fsolve(diff(F(x),x),x,3..4);
```

$$3.899664536$$

Итак, все основные особые точки данной функции (нули, минимумы и максимумы) найдены, хотя и не без трудностей и не всегда с применением специально предназначенных для такого поиска функций.

5.1.13. Maplet-инструмент по анализу функциональных зависимостей

Для анализа функциональных зависимостей Maple 9.5/10 имеет специальный Maplet-инструмент. Он вызывается командой **Tools** ⇒ **Tutors** ⇒ **Calculus-Single Variable** ⇒ **Curve Analysis...**. Она открывает окно инстумента, показанное на рис. 5.4.

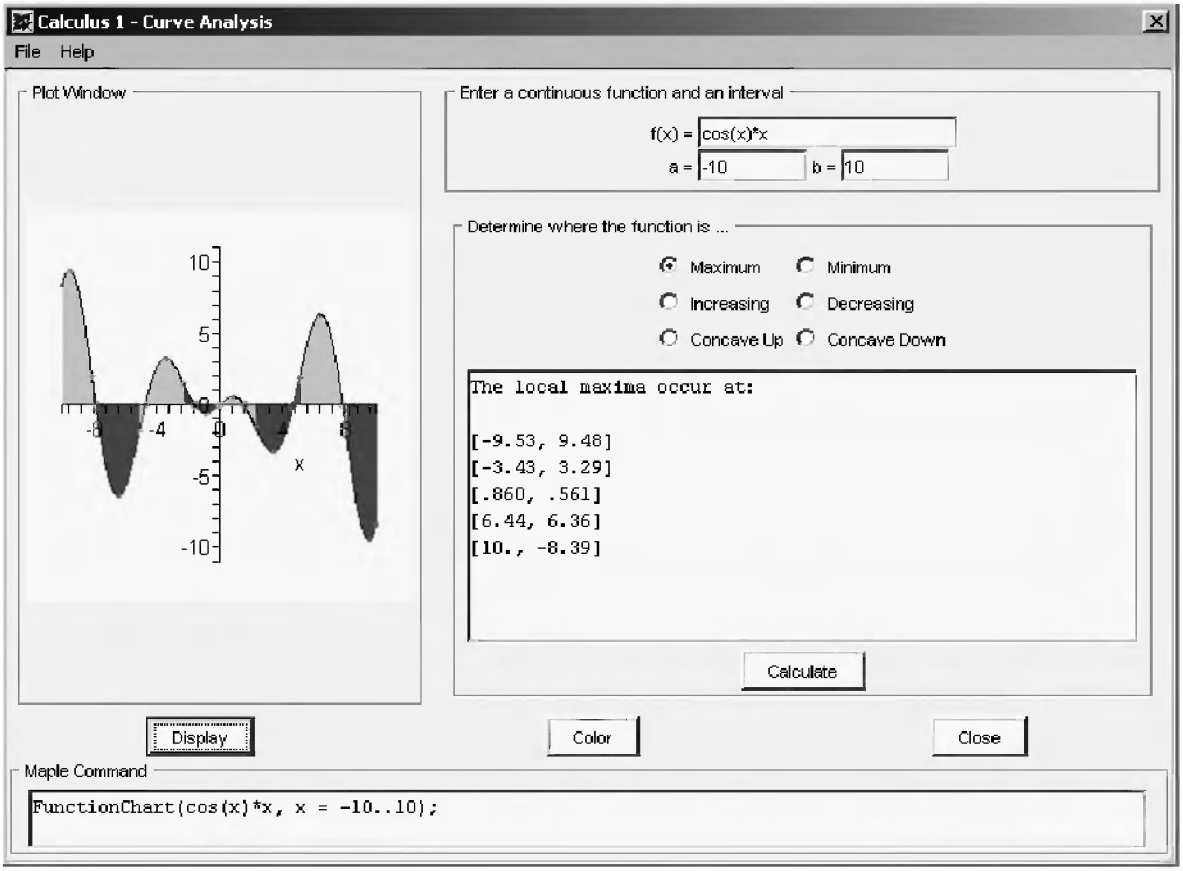


Рис. 5.4. Окно Maplet-инструмента анализа функциональных зависимостей

В верхней правой части окна имеются панели для ввода функциональной зависимости $f(x)$ и границ a и b изменения аргумента x. Под ними имеется набор опций для задания того или иного параметра кривой, например ее максимумов Maximum, минимумов Minimum и др. После нажатия клавиши Calculate вычисляются координаты характерных точек или области определения тех или иных особенностей кривой.

График анализируемой кривой появляется в левой части окна. В нем строятся точки корней, перегибов и экстремумов зависимости. Цветом выделяются участки, на которых зависимость нарастает или падает. Кнопка **Display** порождает запись команды, которая строит полученный рисунок.

5.2. Поиск экстремумов и анализ функций в других СКМ

5.2.1. Средства поиска экстремумов в СКМ Mathematica

Возможности вычисления экстремумов в аналитическом виде у других СКМ весьма ограничены. Так, система Mathematica подобных средств вообще не имеет. Она ориентирована на поиск эктремумов только в численном виде. Вначале отметим функции поиска максимального и минимального значений ряда чисел:

- **Max[x1, x2, ...]** – возвращает наибольшее из *xi*.
- **Max[{x1, x2, ...}, {y1, ...}, ...]** – возвращает наибольший элемент любого из списков.
- **Min[x1, x2, ...]** – возвращает наименьшее из *xi*.
- **Min[{x1, x2,...}, {y1, ...}, ...]** – возвращает наименьший элемент любого из данных списков.

Следующие примеры показывают действие этих простых функций, реализующих алгоритм перебора чисел, входящих в список, система Mathematica предоставляет следующие средства:

Ввод (*In*)	**Вывод (*Out*)**
`Max[1,5,2,6.5,3,4]`	`6.5`
`Max[{1,3,2},{4,5,6},{9,8,7}]`	`9`
`Min[1,5,2,6.5,-3,4]`	`-3`
`Min[{1,3,2},{4 5 6},{9,8,7}]`	`1`

Для поиска локального минимума заданной аналитической функции используется функция:

- **FindMinimum[f, {x, x0}]** – выполняет поиск локального минимума f, начиная со значения x=x0, и возвращает его значение.

Приведем примеры применения функции **FindMinimum**:

```
FindMinimum[-x*Exp[-2*x],{x,1}]
  {-0.18394, {x -> 0.5}}
FindMinimum[-x*Exp[-2*x],{x,0.2,0,1}]
  {-0.18394, {x -> 0.5}}
FindMinimum[-5*x*Exp[-x/2]*(2+Sin[3*x]),{x,1}]
  {-7.17833, {x -> 0.783139}}
FindMinimum[-5*x*Exp[-x/2]*(2+Sin[3*x]),{x,3}]
  {-10.6299, {x -> 2.5805}}
FindMinimum[-5*x*Exp[-x/2]*(2+Sin[3*x]),{x,4}]
  {-6.79134, {x -> 4.6179}}
FindMinimum[100*(y-x^2)^2+(1-x)^2,{x,0},{y,0},
AccuracyGoal->Automatic]
                -13
  {9.98756 10   , {x -> 1., y -> 0.999999}}
```

Эти примеры показывают, что выбором начального значения x можно найти ряд минимумов функции $f(x)$, разумеется, если таковые имеют место. Если необходимо разыскивать локальные максимумы, достаточно перед функцией поставить знак минус или умножить ее на –1. Последний пример показывает, насколько успешно функция **FindMinimum** справляется с поиском минимума тестовой функции Розенброка.

Опытные пользователи могут настроить работу функции **FindMinimum** с помощью шести возможных для нее опций. Из них наиболее важные – это **WorkingPrecision** (рабочая точность – по умолчанию 16 цифр результата) и **Gradient** (по умолчанию **Gradient->{2x, Sign[y]}**). Задав, к примеру, **Gradient->Automatic**, можно перейти к вычислению символьного значения минимума для функции, у которой градиент может вычисляться аналитически.

Mathematica позволяет искать глобальные максимумы и минимумы методом линейного программирования. Но эту возможность мы опишем позже в главе 8.

5.2.2. Поиск экстремумов в СКМ Mathcad 12/13

Прежде чем изучать функции поиска экстремумов в главных версиях Mathcad, рассмотрим применение «старой» функции `Minerr` для решения типовой оптимизационной задачи – поиска минимума функции Розенброка градиентным методом. Фрагмент документа Mathcad, который иллюстрирует процедуру поиска минимума функции Розенброка с помощью функции Minerr, показан на рис. 5.5.

ПОИСК МИНИМУМА ФУНКЦИИ РОЗЕНБРОКА

$x := 1.2 \qquad y := .6$ Инициализация x и y

$f(x,y) := 100\cdot\left(y - x^2\right)^2 + (1-x)^2$ Функция Розенброка

Given Начало вычислительного блока

$\frac{d}{dx}f(x,y) = 0 \qquad \frac{d}{dy}f(x,y) = 0$ Условия минимума

$\begin{pmatrix} x \\ y \end{pmatrix} := \text{Minerr}(x,y)$ Поиск минимума с помощью функции minerr

$x = 1 \qquad y = 1 \qquad f(x,y) = 0$ Проверка решения

Построение графика функции Розенброка

$i := 0..100 \qquad j := 0\ \ 100 \qquad a_i := 0.01\cdot i - 0.5 \qquad b_j := 0.01\cdot j - 0.5 \qquad M_{i,j} := f(a_i, b_j)$

Рис. 5.5. Решение задачи на поиск минимума функции Розенброка средствами системы Marthcad

Нетрудно заметить, что решение данной задачи указанным методом требует вычисления производных заданной функции по переменным `x` и `y`. В точке минимума эти производные равны нулю. На рис. 5.6 показан фрагмент документа Mathcad с графиком функции Розенброка, который весьма наглядно иллюстрирует сложность поиска минимума данной функции.

К чести разработчиков системы Mathcad, надо отметить, что описанный алгоритм блестяще справляется с непростой задачей поиска минимума функции Розенброка градиентным методом – минимум функции найден в точке (1,1) с максимально верным (в пределах точности отображения результатов вычислений) значением – 0.

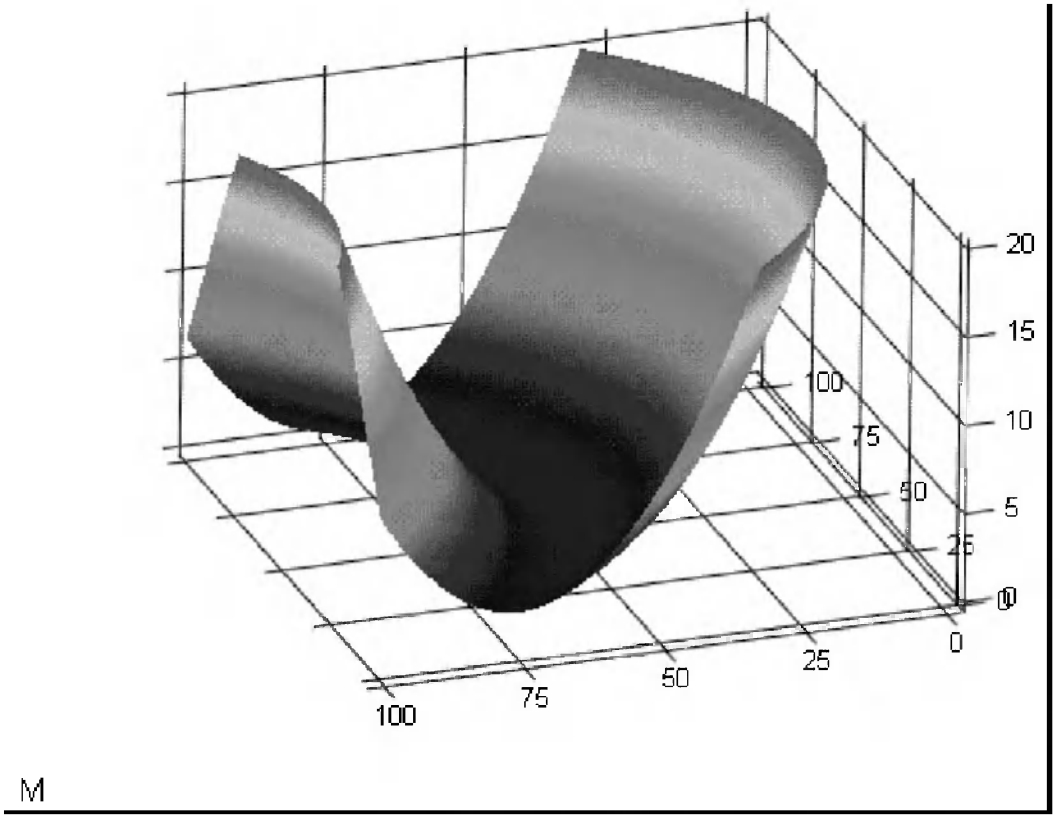


Рис. 5.6. График функции Розенброка для примера рис. 5.5

5.2.3. Поиск экстремумов с помощью функций Maximize и Minimize СКМ Mathcad

Для поиска значений переменных `x1`, `x2`, ..., `xn`, при которых некоторая функция `f(x1,x2,…,xn)` имеет максимальное или минимальное значение, используются функции `Maximize(f,x1,x2,…,xn)` и `Minimize(f,x1,x2,…,xn)`. Обе эти функции реализованы достаточно универсальными алгоритмами оптимизации, которые не требуют вычисления производных функции `f(x1,x2,…,xn)`, что не только упрощает запись алгоритмов, но и позволяет решать задачи, в которых вычисление производных по тем или иным причинам невозможно.

Эти функции должны использоваться в составе блока `Given`. Они возвращают вектор неизвестных, при котором заданная функция имеет максимальное или минимальное значение соответственно. Внутри блока могут быть различные ограничительные условия в виде равенств или неравенств. Число условий ограничено только памятью ПК.

Перед блоком решения надо задать начальные значения искомых переменных. Чем они ближе к верному решению, тем быстрее будет получен правильный результат.

Первый фрагмент документа Mathcad, реализующего решение данной задачи, представлен на рис. 5.7. Здесь показаны задание самого уравнения, построение графика функции и геометрическое представление ограничительных условий. Полезно особо отметить, что решение задачи оптимизации с ограничениями в принципе намного сложнее, чем без таковых.

Рис. 5.7. Начало документа с решением задачи поиска минимума и максимума функции Розенброка с учетом ограничений

Второй фрагмент документа Mathcad показан на рис. 5.8. Обратите внимание, что функция `Minimize` (или `Maximize`) может вычислять минимум (или максимум) как при отсутствии ограничений, так и при их наличии.

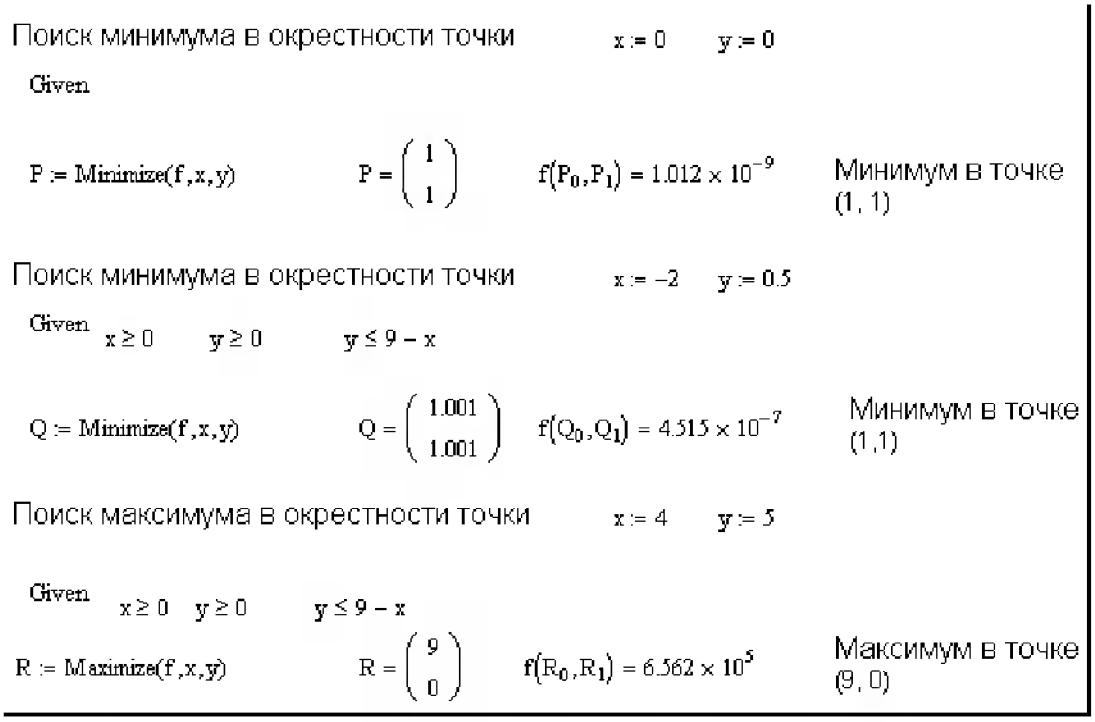


Рис. 5.8. Окончание документа с решением задачи поиска минимума и максимума функции Розенброка с учетом ограничений

Объективности ради надо заметить, что результаты решения сильно зависят от выбора начальных значений переменных и далеко не всегда имеют устраивающую пользователя погрешность.

5.2.4. Поиск глобального максимума средствами СКМ Mathcad

Иногда необходимо отыскать *глобальный максимум* (или минимум) многоэкстремальной функции. Ограничимся рассмотрением решения этой задачи для функции одной переменной. Рисунок 5.9 показывает примеры такого решения. Вначале на нем дан пример того, как функция Minerr не справляется с этой задачей и отыскивает локальный минимум вместо глобального максимума.

Рис. 5.9. Примеры поиска глобального максимума многоэкстремальной функции в системе Mathcad 12

Одним из способов поиска глобального максимума является применение *метода Монте-Карло* – генерация множества случайных значений x и выбор из них просто максимального значения. Этот первый способ, представленный на рис. 5.9, имеет слишком много недостатков, чтобы его можно было всерьез рекомендовать

на практике. Во-первых, надо генерировать слишком много случайных чисел, чтобы найти глобальный максимум с достаточной точностью. Во-вторых, он ведет к большим затратам памяти для хранения массива значений функции. В-третьих, он не позволяет явно найти значение x, при котором $y(x)$ максимальна (впрочем, сделать это достаточно легко – см. прием, описанный ниже).

Более рациональным является просто просчет некоторого множества значений $y(x)$ при определенном числе равноотстоящих значений x. Найдя значение x для приближенного максимума, можно уточнить x, решая уравнение $y(x) = ym$ или $y(x) - ym = 0$, и затем уточненное значение x использовать в качестве начального приближения для поиска максимума.

Наконец, третий способ (самый простой) заключается в свойстве функции `Minerr` отыскивать ближайшее решение. Применение функции `Minerr` для поиска экстремумов – недокументируемый, а потому довольно оригинальный прием. Дело в том, что эта функция в блоке решения `Given` (он был описан выше) старается дать приближение к решению задачи с наименьшей среднеквадратической погрешностью. Другими словами, блок решения с этой функцией может решать *некорректные задачи*. Если вблизи некоторого значения x_0 есть максимум функции $f(x\max) = \max$, то его можно найти, попытавшись решить некорректное уравнение $f(x\max) = c > \max$. Аналогично для нахождения минимума можно найти $f(x\min) = c < \min$.

Задав уравнение $y(x) = y0$, где заведомо $y0 > ym$, можно найти глобальный максимум, ибо только при нем реализуется минимальная среднеквадратическая ошибка.

5.2.5. Анализ сложной функции в Mathcad

График сложной функции нередко обнаруживает различные особенности ее применения, например экстремумы (максимумы и минимумы), а также нули функции, на графике отображаемые как точки пересечения графика функции $F(x)$ с осью абсцисс. Рисунок 5.10 показывает начало документа с анализом нулей функции и нахождением экстремумов этой функции. Нетрудно заметить, что представленная на нем функция имеет 5 нулей и 4 экстремума. Попытаемся найти их точные значения.

Для поиска нулей функции достаточно последовательно применить функцию root, указав приблизительные значения x в точках, где $F(x) = 0$. Конец документа (рис. 5.11) показывает поиск экстремумов с помощью функции `Minerr`. Здесь также надо аккуратно подсказать системе Mathcad условия применения этой функции. Особенно это касается пологих максимумов и минимумов, где алгоритм работы функции `Minerr` может давать сбои.

Разумеется, в новых версиях Mathcad можно использовать встроенные функции для поиска минимумов и максимумов функций вида $f(x)$ – `maximize` и `minimize`. Такое применение, в принципе, не дает ничего нового.

Анализ сложной функции

$F(x) := 2 + 0.05 \cdot x + e^{-\frac{|x|}{10}} - 2 \cdot e^{x^2} \cdot e^{-|x|}$ Функция и ее график

$x := -10, -9.75 .. 10$

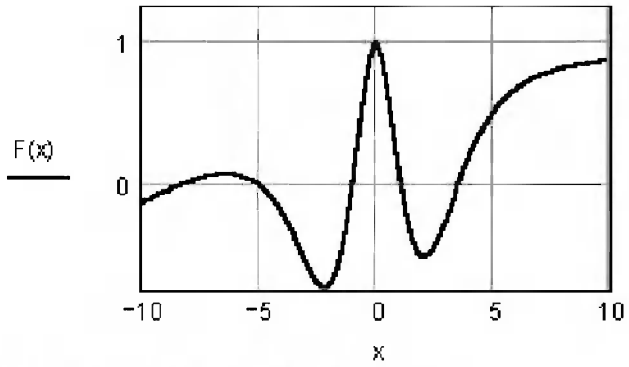


График показывает, что F(x) имеет 5 корней при x, примерно равных -8, -5, -1, 1, 4, два максимума и два минимума.

Поиск корней (нулей функции)

Инициализация	Поиск корня	Значение корня	Проверка
x := –8	x1 := root(F(x), x)	x1 = –8.095	F(x1) = 0
x := –5	x2 := root(F(x), x)	x2 = –5.081	F(x2) = 0
x := –1	x3 := root(F(x), x)	x3 = –0.971	F(x3) = 0
x := 1	x4 := root(F(x), x)	x4 = 1.061	F(x4) = 0
x := 4	x5 := root(F(x), x)	x5 = 3.536	F(x5) = 0

Рис. 5.10. Начало документа Mathcad с анализом сложной функции F(x)

Поиск экстремумов (максимумов и минимумов функции)

Вычисление главного максимума F(x)

			Given	
F(x) = 2	x = 0	x6 := Minerr(x)	x6 = 0	F(x6) = 1
Контроль	F(x6 + 0.1) = 0.977		F(x6 – 0.1) = 0.967	

Вычисление другого максимума F(x)

			Given	
F(x) = 1	x < –5.5	x7 := Minerr(x)	x7 = –6.359	F(x7) = 0.066
Контроль	F(x7 + 0.2) = 0.065		F(x7 – 0.2) = 0.065	

Вычисление первого минимума F(x)

			Given	
F(x) = –0.8	x < –1	x8 := Minerr(x)	x8 = –2.152	F(x8) = –0.728
Контроль:	F(x8 + 0.1) = –0.724		F(x8 – 0.1) = –0.724	

Вычисление второго минимума F(x)

			Given	
F(x) = –0.6	x > 1	x9 := Minerr(x)	x9 = 2.035	F(x9) = –0.518
Контроль:	F(x9 + 0.1) = –0.514		F(x9 – 0.1) = –0.514	

Рис. 5.11. Конец документа Mathcad с анализом сложной функции F(x)

5.2.6. Расчет и построение асимптот функции в Mathcad

Многие функции при устремлении x в бесконечность (положительную или отрицательную) обнаруживают свойство постепенного приближения к некоторой прямой $k \cdot x + b$, называемой асимптотой. Рисунок 5.12 показывает определение основной наклонной асимптоты функции $f(x)$ и ее построение на графике функции. Для определения k и b приходится вычислять пределы функции.

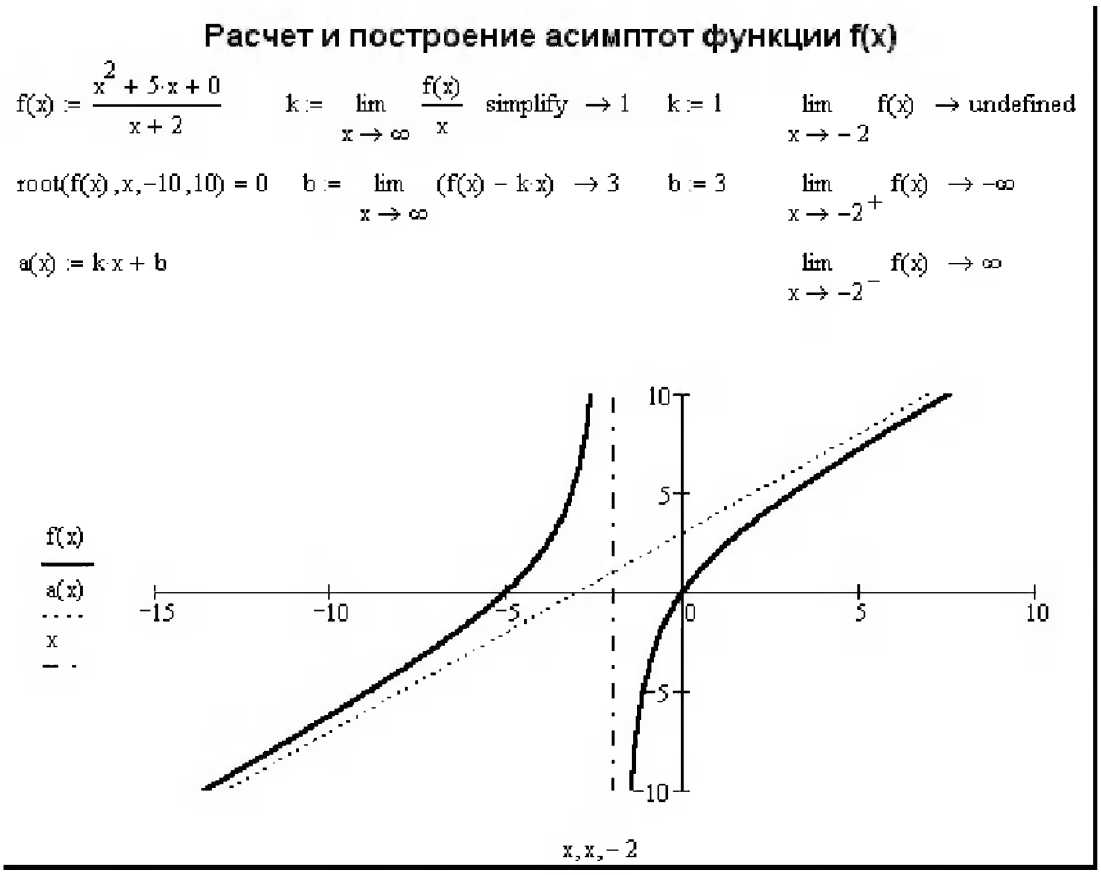


Рис. 5.12. Построение графика функции с ее асимптотами

В местах разрыва функции часто существуют вертикальные асимптоты. Условия их существования также представлены на рис. 5.12. Для данной функции вертикальная асимптота соответствует значению $x = -2$. Обратите внимание на способ построения этой асимптоты – по оси ординат указывается x, а по оси абсцисс –2. График рис. 5.12 дает наглядное представление об асимптотах функции f(x).

5.2.7. Поиск экстремумов в СКА Derive

СКА Derive вообще не содержит встроенных функций минимизации функции одной или многих переменных. Пожалуй, это серьезный недостаток системы, учитывая ту огромную роль, которую задачи минимизации (или, точнее, оптимизации) играют в инженерном проектировании. Разумеется, простые задачи оптимизации непрерывной функции одной переменной легко могут быть сведены к решению уравнения на поиск нулевого значения первой производной функции:

```
1:   "Вычисление максимума функции:"
2:   F (x) := x EXP (-x)
                  /d          \
3:   xmax := SOLVE |- F (x), x|
                  \dx         /
4:   [x = 1]
5:   F (0.99)
6:   0.367860
7:   F (1)
8:   0.367879
9:   F (1.01)
10:  0.367861
```

В строках 5–10 показана простейшая проверка, показывающая, что значение $x = 1$ в данном случае действительно есть значение, при котором функция $F(x)$ имеет максимум, а не минимум, поскольку по обе стороны от значения $x = 1$ величина $F(x)$ уменьшается.

Учитывая возможности задания функций с параметрами – произвольными функциями, можно создать более совершенную функцию для поиска экстремумов (как максимумов, так и минимумов) в заданном отрезке изменения аргумента $[a,b]$:

```
1:   ЭПоиск экстремумов функции одной переменнойЭ
                                  /d            \
2:   X_EXTR (y, x, a, b) := SOLVE |- y, x, a, b|
                                  \dx           /
3:   X_EXTR (x EXP (-x), x, 0.5, 1.5)
4:   [x = 1]
            / SIN (x)           \
5:   X_EXTR |----, x, -1, 1|
            \    x              /
6:   [x = 0]
7:   X_EXTR (SIN (x), x, -2, -1)
8:   [x = -1.57080]
```

В этом примере показано вычисление максимума для функций $x^*\exp(-x)$ и $\sin(x)/x$, а также минимума функции $\sin(x)$ вблизи точки $x = -\pi/2$. Задание отрезка $[a,b]$ поиска экстремума облегчает анализ многоэкстремальных функций.

5.3. Работа с функциями из отдельных кусков

5.3.1. Создание функций из отдельных кусков

Для создания функций, составленных из отдельных кусков – *кусочных функций*, Maple располагает интересной и по-своему уникальной функцией:

```
piecewise(cond_1,f_1, cond_2,f_2, …, cond_n,f_n, f_otherwise)
```

где `f_i` – выражение, `cond_i` – логическое выражение, `f_otherwise` – необязательное дополнительное выражение. В зависимости от того или иного условия эта функция позволяет формировать соответствующую аналитическую зависимость. К кусочным функциям (подчас в скрытой форме) приводят функции с элементами сравнения аргумента, например `abs`, `signum`, `max` и др. Поэтому в Maple 8 введен достаточно мощный аппарат обработки и преобразований таких функций по частям.

5.3.2. Простые примеры применения функции piecewise

Рисунок 5.13 показывает задание функции $f(x)$, содержащей три характерных участка. По определенной через функцию пользователя зависимости $f(x)$ можно, как обычно, построить ее график.

Важно отметить, что созданная с помощью функции `piecewise` зависимость может участвовать в различных преобразованиях. Например, на рис. 5.13 показано, что она легко дифференцируется и интегрируется, так что можно построить графики производной этой функции и ее интегрального значения. При этом каждая часть функции обрабатывается отдельно.

5.3.3. Работа с функциями piecewise

С функциями типа `piecewise` можно работать как с обычными функциями. При этом необходимые операции и преобразования осуществляются для каждой из частей функции и возвращаются в наглядной форме.

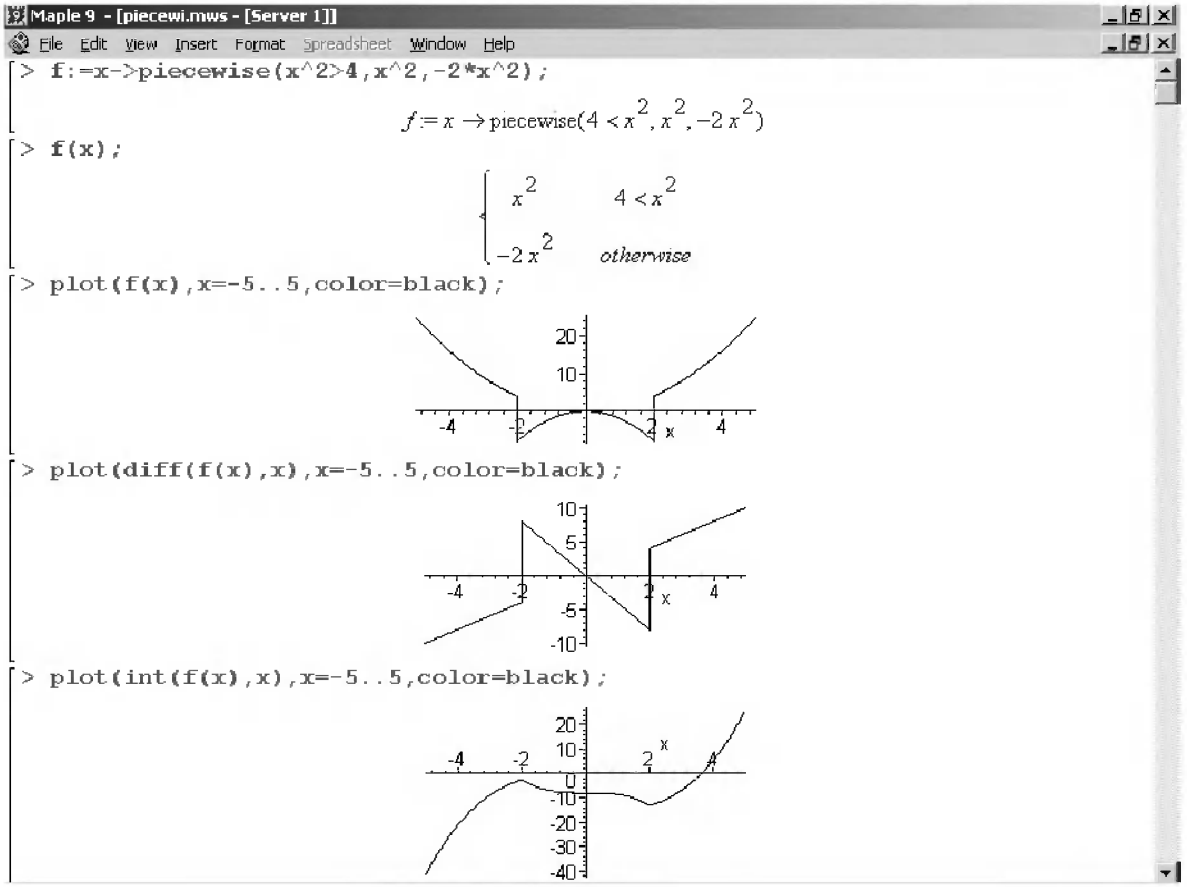


Рис. 5.13. Пример задания и применения функции, составленной из отдельных кусков

Ниже приведен пример задания функции `f` в аналитической форме:

```
> restart;
> f := max(x^2 - 2, x-1);
```

$$f := \max(x^2 - 2, x - 1)$$

Для выявления характера функции воспользуемся функцией `convert` и создадим объект `g` в виде кусочной функции:

```
> g := convert(f, piecewise);
```

$$g := \begin{cases} x^2 - 2 & x < -\dfrac{\sqrt{5}}{2} + \dfrac{1}{2} \\ x - 1 & x \le \dfrac{\sqrt{5}}{2} + \dfrac{1}{2} \\ x^2 - 2 & \dfrac{\sqrt{5}}{2} + \dfrac{1}{2} < x \end{cases}$$

Выполним дифференцирование и интегрирование функции:

```
> fprime := diff(f, x);
```

$$fprime := \begin{Bmatrix} 2x & x < -\frac{\sqrt{5}}{2} + \frac{1}{2} \\ undefined & x = -\frac{\sqrt{5}}{2} + \frac{1}{2} \\ 1 & x < \frac{\sqrt{5}}{2} + \frac{1}{2} \\ undefined & x = \frac{\sqrt{5}}{2} + \frac{1}{2} \\ 2x & \frac{\sqrt{5}}{2} + \frac{1}{2} < x \end{Bmatrix}$$

```
> Int(g,x)=int(g,x);
```

$$\int \begin{Bmatrix} x^2 - 2 & x < -\frac{\sqrt{5}}{2} + \frac{1}{2} \\ x - 1 & x \le \frac{\sqrt{5}}{2} + \frac{1}{2} \\ x^2 - 2 & \frac{\sqrt{5}}{2} + \frac{1}{2} < x \end{Bmatrix} dx = \begin{Bmatrix} \begin{Bmatrix} \frac{1}{3}x^3 - 2x & x \le -\frac{\sqrt{5}}{2} + \frac{1}{2} \\ \frac{1}{2}x^2 - x + \frac{5}{12}\sqrt{5} - \frac{7}{12} & x \le \frac{\sqrt{5}}{2} + \frac{1}{2} \\ \frac{1}{3}x^3 - 2x + \frac{5}{6}\sqrt{5} & \frac{\sqrt{5}}{2} + \frac{1}{2} < x \end{Bmatrix} \end{Bmatrix}$$

Как нетрудно заметить, результаты получены также в виде кусочных функций. Можно продолжить работу с функцией `f` и выполнить ее разложение в степенной ряд:

```
> series( f, x );
```

$$-1 + x + O(x^6)$$

Чтобы убрать член с остаточной погрешностью, можно выполнить эту операцию следующим образом:

```
> series(g, x);
```

$$-1 + x$$

Обратите внимание на то, что поскольку разложение в ряд ищется (по умолчанию) в окрестности точки $x = 0$, то при этом используется только тот кусок функции, в котором расположена эта точка. Читатель может продолжить работу с кусочными функциями и далее. Хорошо развитый аппарат кусочных функций есть в системе Mathematica, особенно Mathematica 6.

5.4. Операции с полиномами в СКМ Maple

5.4.1. Определение полиномов

К числу наиболее известных и изученных аналитических функций относятся *степенные многочлены – полиномы*. Графики полиномов описывают огромное разнообразие кривых на плоскости. Кроме того, возможны рациональные полиномиальные выражения в виде отношения полиномов. Таким образом, круг объектов, которые могут быть представлены полиномами, достаточно обширен, и полиномиальные преобразования широко используются на практике, в частности для приближенного представления других функций.

Под полиномом в СКМ подразумевается сумма выражений с целыми степенями. Многочлен для ряда переменных – *многомерный* полином. К одномерным полиномам относится степенной многочлен

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots a_1 x + a_0,$$

а также отдельная переменная x и константа. Полиномы дают единообразное представление многих зависимостей и для своего вычисления требуют только арифметических операций. Алгоритмы их преобразований и вычислений были первыми в компьютерной алгебре [6, 7].

5.4.2. Выделение коэффициентов полиномов в Maple

Для выделения коэффициентов полиномов в Maple служат следующие функции:

- `coeff(p, x)` – возвращает коэффициент при `x` полинома `p`;
- `coeff(p,x,n)` – возвращает коэффициент для члена со степенью `n` полинома `p`;
- `coeff(p,x^n)` – возвращает коэффициенты при `x^n` полинома `p`;
- `coeffs(p, x, 't')` – возвращает коэффициенты полинома нескольких переменных, относящиеся к переменной `x` (или списку переменных) с опцией `'t'`, задающей имя переменной;
- `collect(p,x)` – возвращает полином, объединяя коэффициенты при степенях переменной `x`.

Ниже даны примеры применения этих функций:

```
> p:=a4*x^4+a3*x^3+a2*x^2+a1*x+a0;
```

$$p := a4x^4 + a3x^3 + a2x^2 + a1x + a0$$

```
> coeff(p,x);
```

$$a1$$

```
> coeff(p,x^3);
```

$$a3$$

```
> coeff(p,x,4);
```

$$a4$$

```
> coeffs(p,x);
```

$$a0, a4, a1, a3, a2$$

```
> q:=x^2+2*y^2+3*x+4*y+5;
```

$$q := x^2 + 2y^2 + 3x + 4y + 5$$

```
> coeffs(q);
```

$$5, 2, 3, 4, 1$$

```
> coeffs(q,y);
```

$$x^2 + 3x + 5, 2, 4$$

```
> coeffs(q,x,y);
```

$$5 + 2y^2 + 4y, 3, 1$$

```
> collect(q,x);
```

$$x^2 + 2(1, x^2, x)^2 + 3x + (4, 4x^2, 4x) + 5$$

```
> collect(q,x,y);
```

$$\mathrm{y}(1)x^2 + \mathrm{y}(3)x + \mathrm{y}(5 + 2y^2 + 4y)$$

Следует обратить внимание на то, что при выполнении операции `collect` в старых версиях Maple (до Maple 7) довольно часто возникала фатальная ошибка. В новых версиях Maple такой ошибки уже не возникает.

5.4.3. Оценка коэффициентов полинома по степеням

Полином может быть неполным, то есть не содержать членов со степенями ниже некоторой. Функция `lcoeff` возвращает старший, а функция `tcoeff` – младший коэффициент полинома нескольких переменных. Эти функции задаются в виде:

```
lcoeff(p)              tcoeff(p)
lcoeff(p, x)           tcoeff(p, x)
lcoeff(p, x, 't')      tcoeff(p, x, 't')
```

Функции `lcoeff` и `tcoeff` возвращают старший (младший) коэффициент полинома `p` относительно переменной `x` или ряда переменных при многомерном полиноме. Если `x` не определено, `lcoeff (tcoeff)` вычисляет старший (младший) коэффициент относительно всех переменных полинома `p`. Если третий аргумент `t` определен, то это имя назначено старшему (младшему) члену `p`. Если `x` – единственное неизвестное и `d` – степень `p` по `x`, то `lcoeff(p, x)` эквивалентно `coeff(p, x, d)`. Если `x` – список или множество неизвестных, `lcoeff (tcoeff)` вычисляет старший (младший) коэффициент `p`, причем `p` рассматривается как полином многих переменных. Имейте в виду, что `p` должен быть разложен по степеням неизвестного `x` до вызова функций `lcoeff` или `tcoeff`.

Приведем примеры применения функций `lcoeff`, `tcoeff` и `coeffs`:

```
> q:=1/x^2+2/x+3+4*x+5*x^2;
```

$$q := \frac{1}{x^2} + \frac{2}{x} + 3 + 4x + 5x^2$$

```
> lcoeff(q,x);
```

$$5$$

```
> lcoeff(q,x,'t');
```

$$5$$

```
> t;
```

$$x^2$$

```
> coeffs(q,x,'t');
```

$$3, 1, 4, 2, 5$$

```
> t;
```

$$1, \frac{1}{x^2}, x, \frac{1}{x}, x^2$$

5.4.4. Оценка степеней полинома

Функция `degree` возвращает высшую степень полинома, а `ldegree` – низшую степень. Эти функции задаются следующим образом:

```
degree(a,x)          ldegree(a,x)
```

Функции `degree` и `ldegree` используются, чтобы определить высшую и низшую степени полинома от неизвестного (неизвестных) `x`, которое чаще всего является единственным, но может быть списком или множеством неизвестных. Полином может иметь отрицательные целые показатели степеней при `x`. Если выражение не является полиномом от `x` с данным параметром, то возвращается `FAIL`.

Чтобы `degree` и `ldegree` возвратили точный результат, полином обязательно должен быть сгруппирован по степеням `x`. Например, для выражения $(x + 1)(x + 2) - x^2$ функция `degree` не обнаружит аннулирование старшего члена и неправильно возвратит результат 2. Во избежание этой проблемы перед вызовом `degree` следует применять к полиному функции `collect` или `expand`. Если `x` – множество неизвестных, `degree/ldegree` вычисляет полную степень. Если `x` – список неизвестных, `degree/ldegree` вычисляет векторную степень. Векторная степень определяется следующим образом:

```
degree(p,[]) = 0     degree(p,[x1,x2,…]) =degree(p,x1)
degree(lcoeff(p,x1),[x2,…])
```

Полная степень тогда определяется следующим образом:

```
degree(p,{x1,…,xn}) = maximum degree(p,{x1,…xn})
```

или

```
degree(p,{x1,…,xn}) = degree(p,[x1,…,xn])
```

Обращаем внимание на то, что векторная степень зависит от порядка перечисления неизвестных, а полная степень не зависит.

Примеры применения функций `degree` и `ldegree`:

```
> restart;
> p:=a4*x^4+a3*x^3+a2*x^2;
```

$$p := a4x^4 + a3x^3 + a2x^2$$

```
> degree(p,x);
```

$$4$$

```
> ldegree(p,x);
```

$$2$$

```
> q:=1/x^2+2/x+3+4*x+5*x^2;
```

$$q := \frac{1}{x^2} + \frac{2}{x} + 3 + 4x + 5x^2$$

```
> degree(q,x);
```

$$2$$

```
> ldegree(q,x);
```

$$-2$$

```
> degree(x*sin(x),x);
```

$$FAIL$$

```
> zero := y*(x/(x+1)+1/(x+1)-1);
```

$$zero := y\left(\frac{x}{x+1} + \frac{1}{x+1} - 1\right)$$

```
> degree(zero,x);degree(zero,y);
```

$$FAIL$$

$$1$$

```
> degree(collect(zero,x,normal),x);degree(collect(zero,y,normal),y);
```

$$-\infty$$

$$-\infty$$

5.4.5. Контроль полинома на наличие несокращаемых множителей

Для контроля того, имеет ли полином несокращаемые множители, могут использоваться функция `irreduc(p)` и ее вариант в инертной форме `Ireduc(p,K)`, где `K` – RootOf-выражение. Функции возвращают логические значения true или false.

5.4.6. Разложение полинома по степеням

Для разложения полинома `p` по степеням служат инертные функции `AFactor(p)` и `AFactors(p)`. Полином может быть представлен в виде зависимости от одной или нескольких переменных.

Функция `Afactor(p)` выполняет полную факторизацию (разложение) полинома `p` от нескольких переменных с коэффициентами в виде алгебраических чи-

сел над полем комплексных чисел. При этом справедливо отношение `evala(AFactor(p))= factor(p,complex)`. Таким образом, эта функция является, по существу, избыточной.

В случае одномерного полинома полное разложение на множители является разложением на линейные множители. Функция `AFactors` аналогична функции `Afactor`, но создает структуру данных формы `[u,[[f[1],e[1]],…,[f[n],e[n]]]]` так, что `p=u*f[1]^e[1]*…*f[n]^e[n]`, где каждый `f[i]` – неприводимый полином.

Ниже даны примеры применения функции `Afactor`:

```
> evala(AFactor(2*x^2+4*x-6));
```

$$2(x+3)(x-1)$$

```
> evala(AFactor(x^2+2*y^2));
```

$$(x-\mathrm{RootOf}(\_Z^2+2)y)(x+\mathrm{RootOf}(\_Z^2+2)y)$$

```
> expand((x-1)*(x-2)*(x-3)*(x-4));
```

$$x^4-10x^3+35x^2-50x+24$$

```
> AFactor(%);
```

$$\mathrm{AFactor}(x^4-10x^3+35x^2-50x+24)$$

```
> evala(%);
```

$$(x-1)(x-2)(x-3)(x-4)$$

Нетрудно заметить, что разложение полинома на множители позволяет оценить наличие у него корней. Однако для этого удобнее воспользоваться специальными функциями, рассмотренными ниже.

5.4.7. Вычисление корней полинома

Для вычисления действительных и комплексных корней полиномов служит уже известная нам функция `solve(p,x)`, возвращающая список корней полинома `p` одной переменной. Кроме того, имеются следующие функции для вычисления корней полиномов:

`roots(p)` **`roots(p, K)`**
`roots(p, x)` **`roots(p, x, K)`**

Эти функции вычисляют точные корни в рациональной или алгебраической области чисел. Корни возвращаются в виде `[[r1,m1], …, [rn,mn]]`, где `ri` – это корень полинома, а `mi` – порядковый номер полинома. С действиями этих функций можно разобраться с помощью приведенных ниже примеров:

```
> p:=x^4+9*x^3+31*x^2+59*x+60;
```

$$p := x^4+9x^3+31x^2+59x+60$$

```
> solve(p,x);
```

$$-3, -4, -1+2I, -1-2I$$

```
> roots(p,x);
```

$$[[-4, 1], [-3, 1]]$$

```
> roots(x^2-4,x);
```

$$[[2, 1], [-2, 1]]$$

```
> expand((x-1)*(x-2)*(x-3)*(x-4));
```

$$x^4 + 10x^3 + 35x^2 - 50x + 24$$

```
> roots(%,x);
```

$$[[1, 1], [2, 1], [3, 1], [4, 1]]$$

5.4.8. Основные операции с полиномами

С полиномами могут выполняться различные операции [6, 7]. Прежде всего отметим некоторые функции, которые относятся к одному полиному:

- `psqrt(p)` – возвращает квадрат полинома;
- `proot(p,n)` – возвращает n-ую степень полинома;
- `realroot(p)` – возвращает интервал, в котором находятся действительные корни полинома;
- `randpoly(vars, eqns)` – возвращает случайный полином по переменным `vars` (список) с максимальной степенью `eqns`;
- `discrim(p,var)` – вычисление дискриминанта полинома по переменной `var`;
- `Primitive(a) mod p` – проверка полинома на примитивность (возвращает `true`, если полином примитивен).

Действие этих функций достаточно очевидно, поэтому ограничимся приведением примеров их использования:

```
> readlib(psqrt):
> readlib(proot):
> psqrt(x^2+2*x*y+y^2);
```

$$y + x$$

```
> proot(x^3+3*x^2+3*x+1, 3);
```

$$x + 1$$

```
> psqrt(x+y);
```

$$\_NOSQRT$$

```
> proot(x+y, 2);
```

$$\_NOROOT$$

```
> p:=x^3-3*x^2+5*x-10;
```

$$p := x^4 - 3x^2 + 5x - 10$$

```
> discrim(p,x);
```

$$-1355$$

```
> readlib(realroot):
> realroot(p);
```

$$[[0, 4]]$$

```
> randpoly([x],degree=10);
```

$$63x^{10} + 57x^8 - 59x^5 + 45x^4 - 8x^3 - 93$$

```
> randpoly([x],degree=10);
```

$$-5x^9 + 99x^8 - 61x^6 - 50x^5 - 12x^3 - 18x$$

```
> randpoly([x],degree=10);
```

$$41x^9 - 58x^8 - 90x^7 + 53x^6 - x^4 + 94x$$

```
> Primitive( x^4+x+1 ) mod 2;
```

$$true$$

Обратите внимание на то, что для использования некоторых из приведенных функций необходим вызов их из стандартной библиотеки. Для функции `randpoly` приведенные результаты случайны, так что их повторение невозможно. С полиномами можно выполнять обычные операции, используя для этого соответствующие операторы:

```
> readlib(psqrt):
> readlib(proot):
> Primitive( x^4+x+1 ) mod 2;
```

$$true$$

```
> p1:=a1*x^3+b1*x^2+c1*x+d1: p2:=a2*x^2+b2*x+c2:
> p1+p2;
```

$$a1x^3 + b1x^2 + c1x + d1 + a2x^2 + b2x + c2$$

```
> p1*p2;
```

$$(a1x^3 + b1x^2 + c1x + d1)(a2x^2 + b2x + c2)$$

```
> collect(%,x);
```

$$a1a2x5 + (b1a2 + a1b2)x^4 + (c1a2 + b1b2 + a1c2)x^3 + (d1a2 + c1b2 + b1c2)x^2 + (d1b2 + c1c2)x + d1c2$$

```
> p1/p2;
```

$$\frac{a1x^3 + b1x^2 + c1x + d1}{a2x^2 + b2x + c2}$$

```
> expand(%,x);
```

$$\frac{a1x^3}{a2x^2 + b2x + c2} + \frac{b1x^2}{a2x^2 + b2x + c2} + \frac{c1x}{a2x^2 + b2x + c2} + \frac{d1}{a2x^2 + b2x + c2}$$

В целом надо отметить, что аппарат действий с полиномами в Maple хорошо развит и позволяет выполнять с ними практически любые математические операции. В частности, можно вычислять производные от полиномов и интегралы, у которых полиномы являются подынтегральными функциями:

```
> diff(p1,x);
```

$$3a1x^2 + 2b1x + c1$$

```
> diff(p1,x$2);
```

$$6a1x + 2b1$$

```
> Int(p1,x)=int(p1,x);
```

$$\int a1x^3 + b1x^2 + c1x + d1dx = \frac{a1x^4}{4} + \frac{b1x^3}{3} + \frac{c1x^2}{2} + d1x$$

```
> Int(p1,x=0..1)=int(p1,x=0..1);
```

$$\int_0^1 a1x^3 + b1x^2 + c1x + d1dx = \frac{a1}{4} + \frac{b1}{3} + \frac{c1}{2} + d1$$

5.4.9. Операции над степенными многочленами с отрицательными степенями

Maple не накладывает особых ограничений и на многочлены с отрицательными степенями. Например, можно задать такой степенной многочлен:

```
> pp:=a*x^(-2)+b*x^(-1)+c*x+d+e*x^2+f*x^3;
```

$$pp := \frac{a}{x^2} + \frac{b}{x} + cx + d + ex^2 + fx^3$$

Нетрудно показать, что с ним можно выполнять различные операции:

```
> pp+pp;
```

$$2\frac{a}{x^2} + \frac{2b}{x} + 2cx + 2d + 2ex^2 + 2fx^3$$

```
> pp-pp;
```

$$0$$

```
> pp^2;
```

$$\left(\frac{a}{x^2} + \frac{b}{x} + cx + d + ex^2 + fx^3\right)^2$$

```
> simplify(%);
```

$$\frac{(a + bx + cx^3 + dx^2 + ex^4 + fx^5)^2}{x^4}$$

```
> Diff(pp,x)=diff(pp,x);
```

$$\frac{\partial}{\partial x}\left(\frac{a}{x^2} + \frac{b}{x} + cx + d + ex^2 + fx^3\right) = -2\frac{a}{x^2} - \frac{b}{x} + c + 2ex + 3fx^2$$

```
> Int(pp,x);
```

$$\int \frac{a}{x^2} + \frac{b}{x} + cx + d + ex^2 + fx^3 dx$$

```
> int(pp,x);
```

$$-\frac{a}{x} + b\ln(x) + \frac{1}{2}cx^2 + dx + \frac{1}{3}ex^3 + \frac{1}{4}fx^4$$

5.5. Операции с полиномами в Mathematica

Другой системой, в которой имеется развитый аппарат для работы с полиномами, является система Mathematica. Ниже рассмотрены особенности этого аппарата.

5.5.1. Основные операции над полиномами

В системе Mathematica над полиномами можно выполнять обычные арифметические операции: сложение, вычитание, умножение и деление. Это иллюстрируют следующие примеры (p1 и p2 – полиномы от одной переменной x):

```
p1:=x^3+2*x^2+3*x+4
p2:=x^2-1
p1+p2
  3 + 3x + 3x^2 + x^3
p1-p2
```

```
 5 + 3x + x^2 + x^3
Expand[p1*p2]
 -4 - 3x + 2x^2 + 2x^3 + 2x^4 + x^5
p1/p2
 4 + 3x + 2x^2 + x^3
-------------------
      -1 + x^2
Simplify[(x^5+2*x^4+2*x^3+2*x^2-3*x-4)/(x^2-1)]
 4 + 3x + 2x^2 + x^3
```

Обратите внимание на применение функций **Expand** и **Simplify**. Нередко именно они позволяют получить результат в стандартном виде. Функция, обеспечивающая деление полиномов и вычисляющая остаток от деления, описана ниже.

5.5.2. Разложение полиномов – функции класса Factor

Разложение чисел, математических выражений и особенно полиномов на простые множители является столь же распространенной операцией, что и функции Simplify, Collect и Expand. Имеется целый ряд функций, в названии которых есть слово Factor и которые решают указанные задачи:

- **Factor[poly]** – выполняет разложение полинома над целыми числами.
- **Factor[poly, Modulus->p]** – выполняет разложение полинома по модулю простого p.
- **FactorInteger[n]** – возвращает список простых множителей целого числа n вместе с их показателями степеней. Опция **FactorComplete** позволяет указать, следует ли выполнять полное разложение.
- **FactorList[poly]** – возвращает список множителей полинома с их показателями степени. Опция **Modulus->p** позволяет представить множители полинома по модулю простого p.
- **FactorSquareFree[poly]** – записывает полином в виде произведения множителей, свободных от квадратов. Опцтия **Modulus->p** позволяет представить разложение полинома по модулю простого p.
- **FactorSquareFreeList[poly]** – возвращает список множителей полинома, свободных от квадратов, вместе с показателями степени. Может использоваться опция **Modulus->p**.
- **FactorTerms[poly]** – извлекает полный (общий) числовой множитель в poly.
- **FactorTermsList[poly]** – возвращает лист всех общих числовых множителей полинома poly.

Ниже представлен ряд примеров на применение этих функций:

```
Factor[x^3-6*x^2+11*x-6]
 (-3 + x)(-2 + x)(-1 + x)
Factor[x^3-6*x^2+21*x-52]
 (-4 + x)(13 - 2x + x^2)
```

```
Factor[x^5+8*x^4+31*x^3+80*x^2+94*x+20,Modulus->3]
 (1 + x)^2 (2 + x)^3
FactorList[x^4-1,Modulus->2]
  {{1,1},{1+x,4}}
FactorSquareFree[(x^2+1)*(x^4-1)]
  (-1 + x^2) (1 + x^2)^2
FactorSquareFree[(x^2+1)*(x^4-1),Modulus->2]
  (1 + x)^6
FactorSquareFreeList[(x^2+1)*(x^4-1),Modulus->2]
   {{1,1},{1+x,6}}
FactorTerms[2*x^2+4*x+6]
   2(3 + 2x + x^2)
FactorTermsList[2*x^2+4*x+6]
  {2, 3 + 2x + x^2}
FactorInteger[123456789]
   {{3,2},{3607,1},{3803,1}}
FactorList[x^4-1]
  {{1, 1}, {-1 + x, 1}, {1 + x, 1}, {1 + x^2, 1}}
FactorSquareFreeList[(x^2+1)*(x^4-1)]
  {{1, 1}, {-1 + x^2, 1}, {1 + x^2, 2}}
```

Обычно функция **Factor** выявляет внутреннюю суть полинома, раскладывая его множители, содержащие корни полинома. Однако в ряде случаев корни полинома удобнее получать в явном виде с помощью уже рассмотренной функции **Roots**.

Функция **Factor** может работать и с тригонометрическими выражениями, поскольку многие из них подчиняются правилам преобразований, присущим полиномам. При этом тригонометрический путь решения задается опцией **Trig->True**. Это иллюстрируют следующие примеры:

```
Factor[Csc[x]+Sec[x],Trig->True]
  Csc[x] Sec[x] (Cos[x]+Sin[x])
Factor[Sin[3*x],Trig->True]
  (1+2 Cos[2 x]) Sin[x]
```

5.5.3. Функции для работы с полиномами

В Mathematica 4/5 имеется множество и других функций для работы с полиномами:

- **Decompose[poly, x]** – выполняет разложение полинома, если это возможно, на более простые полиномиальные множители.
- **GroebnerBasis[{poly1, poly2, ...}, {x1, x2, ...}]** – возвращает список полиномов, которые образуют базис Гробнера для идеала, порожденного полиномами polyi.
- **PolynomialDivision[p, q, x]** – возвращает список частного и остатка, полученных делением полиномов p и q от x.
- **PolynomialGCD[poly1, poly2, ...]** – возвращает наибольший общий делитель ряда полиномов poly1, poly2, ... С опцией **Modulus->p]** функция возвращает GCD по модулю простого p.

- **PolynomialLCM[poly1, poly2, ...]** – возвращает наименьшее общее кратное полиномов poly1, poly2, ... С опцией **Modulus->p]** функция возвращает LCM по модулю простого p.
- **PolynomialMod[poly, m]** – возвращает полином poly, приведенный по модулю m.
- **PolynomialMod[poly, {m1, m2, ...}]** – выполняет приведение по модулю всех mi.
- **PolynomialQ[expr, var]** – выдает значение True, если expr является полиномом от var, иначе дает False.
- **PolynomialQ[expr, {var1, ...}]** – проверяет, является ли expr полиномом от vari.
- **PolynomialQuotient[p, q, x]** – возвращает частное от деления p и q как полиномов от x, игнорируя какой-либо остаток.
- **PolynomialRemainder[p, q, x]** – возвращает остаток от деления p на q как полиномов от x.
- **Resultant[poly1, poly2, var]** – вычисляет результат полиномов poly1 и poly2 по переменной var. С опцией **Modulus->p]** функция вычисляет результат по модулю простого p.

5.5.4. Примеры работы с полиномами

Итак, работа с этими функциями по существу сводит операции с таким сложным видом символьных данных, как многочлены, к типовым алгебраическим операциям над обычными символьными переменными. Следующие примеры поясняют работу с полиномами:

```
P[x]:=a*x^3+b*x^2+c*x+d
Q[x]:=e*x^2-f*x-1
Collect[P[x]+Q[x],x]
 -1 + d + (c - f)x + (b + e)x^2 + ax^3
Collect[P[x]*Q[x],x]
 -d + (-c - df)x + (-b + de - cf)x^2 + (-a + ce - bf)x^3 + (be - af)x^4 + aex^5
{PolynomialQ[P[x]],PolynomialQ[Q[x]]}
 {True,True}
PolynomialQ[Sin[x],x]
 False
PolynomialQ[P[x]+Q[x]]
 True
Decompose[P[x],x]
 {d + cx + bx^2 + ax^3}
PolynomialQuotient[P[x],Q[x],x]
 b    af    ax
 - + --- + --
 e    2     e
      e
PolynomialRemainder[Q[x],P[x],x]
 -1 - fx + ex^2
CoefficientList[P[x],x]
 {d,c,b,a}
```

```
Decompose[x^6+x+1-x^3+2*x^5,x]
 {1 + x - x^3 + 2x^5 + x^6}
PolynomialGCD[P[x],Q[x]]
 1
PolynomialLCM[P[x],Q[x]]
 (-1 - fx + ex^2)(d + cx + bx^2 + ax^3)
PolynomialQuotient[3*x^3-2*x^2+x,x^2-x+1,x]
 1+3 x
PolynomialRemainder[3*x^3-2*x^2+x,x^2-x+1,x]
 -1-x
Reduce[a*x^2+b*x+c==0,x]
```

$$a \neq 0 \;\&\&\; \left(x = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \;||\; x = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \right) ||$$

$$a = 0 \;\&\&\; b \neq 0 \;\&\&\; x = -\frac{c}{b} \;||\; c = 0 \;\&\&\; b = 0 \;\&\&\; a = 0$$

Любопытно отметить, что версии Mathematica 4/5 в некоторых случаях дают разные по форме, но тождественные результаты при использовании функций для работы с полиномами. Это свидетельствует о постоянной работе разработчика над ядром символьной математики в этих системах.

5.6. Работа с ортогональными полиномами

5.6.1. Состав пакета orthopoly системы Maple

Ортогональные многочлены (полиномы) находят самое широкое применение в различных математических расчетах [123–125]. В частности, они широко используются в алгоритмах интерполяции, экстраполяции и аппроксимации различных функциональных зависимостей, где свойство ортогональности обеспечивает оценку погрешности приближения и сведение ее к минимуму – вплоть до нуля.

В пакете `orthopoly` Maple 9.5 заданы 6 функций:

```
> with(orthopoly);
```

$$[G, H, L, P, T, U]$$

Однобуквенные имена этих функций отождествляются с первой буквой в наименовании ортогональных полиномов. Вопреки принятым в Maple правилам, большие буквы в названиях этих полиномов не указывают на инертность данных функций – все они являются немедленно вычисляемыми. В данном разделе функции этого пакета будут полностью описаны.

Отметим определения указанных функций:

- `G(n,a,x)` – полином Гегенбауэра (из семейства ультрасферических полиномов);
- `H(n,x)` – полином Эрмита;
- `L(n,x)` – полином Лагерра;
- `L(n,a,x)` – обобщенный полином Лагерра;
- `P(n,x)` – полином Лежандра;
- `P(n,a,b,x)` – полином Якоби;
- `T(n,x)` – обобщенный полином Чебышева первого рода;
- `U(n,x)` – обобщенный полином Чебышева второго рода.

Свойства ортогональных многочленов хорошо известны. Все они характеризуются целочисленным порядком `n`, аргументом `x` и иногда дополнительными параметрами `a` и `b`. Существуют простые рекуррентные формулы, позволяющие найти полином n-го порядка по значению полинома $(n-1)$-го порядка. Эти формулы и используются для вычисления полиномов высшего порядка.

5.6.2. Вычисление ортогональных полиномов

Ниже представлены примеры вычисления ортогональных полиномов:

```
> G(0,1,x);
```

$$1$$

```
> G(1,1,x);
```

$$2x$$

```
> G(1,1,5);
```

$$10$$

```
> H(3,x);
```

$$8x^3 - 12x$$

```
> L(3,x);
```

$$1 - 3x + \frac{3}{2}x^2 - \frac{1}{6}x^3$$

```
> L(2,a,x);
```

$$1 + \frac{3}{2}a - 2x + \frac{1}{2}a^2 - ax + \frac{1}{2}x^2$$

```
> P(2,x);
```

$$\frac{3}{2}x^2 - \frac{1}{2}$$

```
> P(2,1,1,x);
```

$$\frac{15}{4}x^2 - \frac{3}{4}$$

```
> T(5,x);
```

$$16x^5 - 20x^3 + 5x$$

```
> U(5,x);
```

$$32x^5 - 32x^3 + 6x$$

В отличие от ряда элементарных функций, ортогональные многочлены определены только для действительного аргумента x. При комплексном аргументе ранее результат просто повторял исходное выражение с многочленом:

```
> evalf(U(2,2+3*I));
```

$$\mathrm{P}(2,\ 2 + 3I)$$

Но уже в Maple 9 ортогональные полиномы с комплексными аргументами могут вычисляться:

```
> evalf(U(2,2+3*I));
```

$$-21. + 48.I$$

Ортогональные многочлены не определены и для дробного показателя n. Впрочем, надо отметить, что такие многочлены на практике используются крайне редко.

5.6.3. Построение графиков ортогональных полиномов

Наглядное представление о поведении ортогональных многочленов дает их графическая визуализация. На рис. 5.14 построены графики ряда ортогональных многочленов Чебышева $T(n, x)$ и $U(n, x)$.

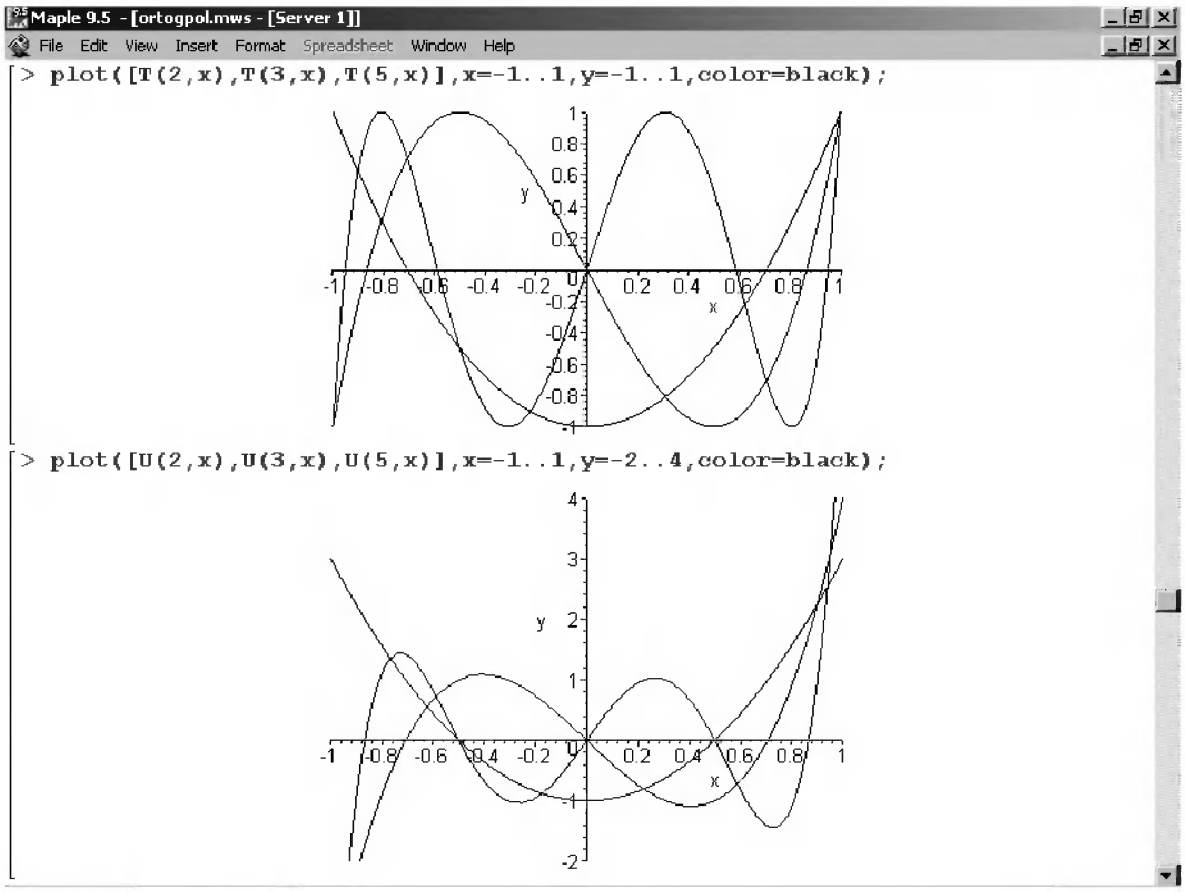


Рис. 5.14. Графики ортогональных многочленов Чебышева

Видно, что многочлены Чебышева имеют минимальное отклонение от оси абсцисс в заданном интервале изменения x. Это их свойство объясняет полезное применение таких многочленов при решении задач аппроксимации функций. Можно порекомендовать читателю по их образцу и подобию построить графики других ортогональных многочленов.

5.6.4. Работа с рядами ортогональных многочленов

Для работы с рядами ортогональных многочленов имеется пакет OrthogonalSeries. Он имеет довольно представительный набор функций, список которых возвращает команда:

```
> with(OrthogonalSeries);
```

Поскольку этот пакет представляет интерес в основном опытным математикам, мы не будем рассматривать его функции (в целом достаточно простые) подробно и ограничимся несколькими примерами. В следующем примере с помощью функции `Create` создается бесконечный ряд с ортогональным многочленом Эрмита в составе базового выражения ряда:

```
> OrthogonalSeries[Create](u(n),HermiteH(n,x)) ;
```

$$\sum_{n=0}^{\infty} \mathrm{u}(n)\,\mathrm{HermiteH}(n,x)$$

В другом примере показано представление полиномиального выражения в новом базисе с ортогональными многочленами Чебышева с помощью функции `ChangeBasis`:

```
> OrthogonalSeries[ChangeBasis](1+3*y*x^2+y^3*x,
ChebyshevT(n,x),ChebyshevU(m,y)) ;
```

$$1+\frac{3}{4}\,\mathrm{ChebyshevT}(2,x)\,\mathrm{ChebyshevU}(1,y)+\frac{3}{4}\,\mathrm{ChebyshevU}(1,y)$$
$$+\frac{1}{2}\,\mathrm{ChebyshevT}(1,x)\,\mathrm{ChebyshevU}(1,y)$$

```
> OrthogonalSeries[Evaluate](%) ;
```

$$3x^2y + yx + 1$$

Обратите внимание на то, что новое выражение после исполнения команды `Evaluate` приняло вид исходного выражения.

Следующий пример демонстрирует создание ряда на основе ортогональных многочленов Чебышева и его копирование с помощью функции `Copy`:

```
> S:=Create((-1)^n/n!,ChebyshevT(n,x));
```

$$S := \sum_{n=0}^{\infty} \frac{(-1)^n\,\mathrm{ChebyshevT}(n,x)}{\mathrm{n!}}$$

```
> T:=Copy(S);
```

$$T := \sum_{n=0}^{\infty} \frac{(-1)^n\,\mathrm{ChebyshevT}(n,x)}{\mathrm{n!}}$$

Вычисление производной от ряда с ортогональными многочленами представлено ниже:

```
> S := Create(u(n),ChebyshevT(n,x));
```

$$S := \sum_{n=0}^{\infty} \mathrm{u}(n)\mathrm{ChebyshevT}(n,x)$$

```
> Derivate(S,x) ;
```

$$\sum_{n=0}^{\infty} (n+1)\mathrm{u}(n+1)\mathrm{ChebyshevU}(n,x)$$

Еще один пример демонстрирует операцию скалярного умножения ряда с помощью функции `ScalarMultiply`:

```
> S := Create(n+1,Kravchouk(n,p,q,x));
```

$$S := \sum_{n=0}^{\infty} (n+1)\mathrm{Kravchouk}(n,p,q,x)$$

```
> ScalarMultiply(alpha,S) ;
```

$$\sum_{n=0}^{\infty} \alpha(n+1)\mathrm{Kravchouk}(n,p,q,x)$$

5.6.5. Ортогональные многочлены в Mathematica 4/5

Система Mathematica 4/5 также имеет набор функций, вычисляющих значения ортогональных многочленов:

- **ChebyshevT[n, x]** – Чебышева n-й степени первого рода.
- **CyebyshevU[n, x]** – Чебышева n-й степени второго рода.
- **HermiteH[n, x]** – Эрмита n-ой степени.
- **JacobiP[n, a, b, x]** – Якоби n-ой степени.
- **GegenbauerC[n, m, x]** – Гегенбауэра.
- **LaguerreL[n, x]** – Лагерра n-ой степени.
- **LaguerreL[n, a, x]** – обобщенного Лагерра n-ой степени.
- **LegendreP[n, x]** – n-го порядка Лежандра.
- **LegendreP[n, m, x]** – присоединенного полинома Лежандра.
- **LegendreQ[n, z]** – n-го порядка функция Лежандра второго рода.
- **LegendreQ[n, m, z]** – присоединенная функция Лежандра второго рода.
- **LegendreType** – опция для **LegendreP** и **LegendreQ**; она указывает выборы разрывов кривой для функций Лежандра на комплексной плоскости.

Все ортогональные полиномы имеют простые рекуррентные представления. Поэтому приведенные выше функции вычисляются по ним довольно быстро и точно. Следующие примеры иллюстрируют работу с ортогональными многочленами в системе Mathematica 4/5:

Ввод (*In*)	**Ввод (*Out*)**
`ChebyshevT[8,x]`	$1 - 32\ x^2 + 160\ x^4 - 256\ x^6 + 128\ x^8$
`ChebyshevT[5,0.2]`	`0.84512`

```
ChebyshevU[3,0.15]      -0.573
HermiteH[4,3]           876
JacobiP[3,1,2,0.2]      -0.256
GegenbauerC[3,1,x]      -4 x + 8 x^3
N[LaguerreL[3,x]]       0.166667 (6. - 18. x + 9. x^2 - 1. x^3)
LegendreQ[2,0.2]        -0.389202
```

Важно отметить, что при указании конкретного значения параметра n и символьном значении параметра x функции этой группы возвращают присущие им представления через степенные многочлены с соответствующими коэффициентами.

5.6.6. Вычисление ортогональных полиномов в СКМ Mathcad

Ортогональные полиномы легко вычисляются по рекуррентным соотношениям. В первых версиях Mathcad не было функций для вычисления ортогональных полиномов, но это не мешало вычислять их по рекуррентным соотношениям, что и представлено на рис. 5.15 и 5.16. Они хорошо иллюстрируют технику таких вычислений.

Наряду с вычислениями по рекуррентным формулам в документе рис. 5.17 и 5.18 даны примеры вычислений ортогональных полиномов по встроенным функциям для системы Mathcad 2001i и более поздним версиям:

ВЫЧИСЛЕНИЕ ОРТОГОНАЛЬНЫХ ПОЛИНОМОВ ПО РЕКУРЕНТНЫМ ФОРМУЛАМ

1. Вычисление полинома Чебышева Tn(x) по аналитической формуле

$T(n,x) := \cos(n \cdot \operatorname{acos}(x))$ $T(3,0.1) = -0.296$ $T(10,1) = 1$

2. Вычисление полинома Чебышева Tn(x) по рекуррентной формуле

$x := 0.1$ $n := 3$ $T_0 := 1$ $T_1 := x$

$i := 1 .. n - 1$ $T_{i+1} := 2 \cdot x \cdot T_i - T_{i-1}$ $T = \begin{pmatrix} 1 \\ 0.1 \\ -0.98 \\ -0.296 \end{pmatrix}$

$Tcheb(n,x) = -0.296$

3. Вычисление полинома Чебышева Un(x) по рекуррентной формуле

$x := 0.1$ $n := 3$ $U_0 := 1$ $U_1 := 2 \cdot x$

$i := 1 .. n - 1$ $U_{i+1} := 2 \cdot x \cdot U_i - U_{i-1}$ $U = \begin{pmatrix} 1 \\ 0.2 \\ -0.96 \\ -0.392 \end{pmatrix}$

$Ucheb(n,x) = -0.392$

Рис. 5.15. Начало документа с вычислениями ортогональных полиномов

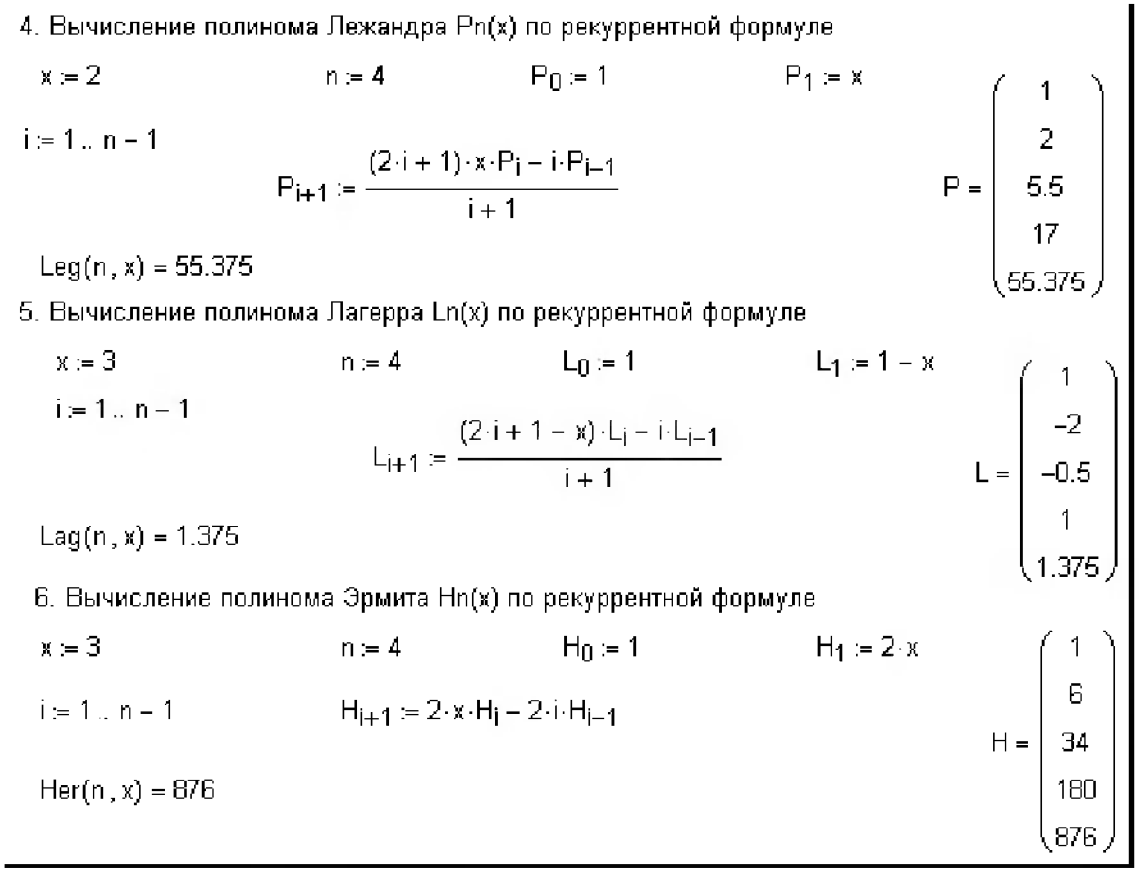


Рис. 5.16. Конец документа с вычислениями ортогональных полиномов

- `Her(n,x)` – полином Эрмита степени `n` с аргументом `x`;
- `Jac(n,a,b,x)` – полином Якоби степени `n` в точке `x` с параметрами `a` и `b`;
- `Lag(n,x)` – полином Лагерра степени `n` в точке `x`;
- `Leg(n,x)` – полином Лежандра степени `n` в точке `x`;
- `Tcheb(n,x)` – полином Чебышева первого рода степени `n` в точке `x`;
- `Ucheb(n,x)` – полином Чебышева второго рода степени `n` в точке `x`.

Средства для вычисления ортогональных полиномов есть и в СКА Derive и MuPAD. Но ничего нового в сравнении с описанными выше средствами они не несут.

5.7. Пакет PolynomialTools системы Maple 9.5/10

5.7.1. Обзор возможностей пакета PolynomialTools

Для выполнения ряда специальных операций с полиномами или создания полиномов с заданными свойствами в Maple 9.5/10 служит пакет PolynomialTools. Этот пакет имеет небольшое число функций:

```
> with(PolynomialTools);
> with(PolynomialTools);
```
[*CoefficientList, CoefficientVector, Hurwitz, IsSelfReciprocal, MinimalPolynomial, PDEToPolynomial, PolynomialToPDE, Shorten, Shorter, Sort, Split, Splits, Translate*]

В пакет входят функции расщепления, сортировки и преобразования полиномов (в том числе в дифференциальные уравнения и наоборот) и др.

5.7.2. Функции для работы с полиномами

Рассмотрим несколько функций пакета PolynomialTools общего характера.

Функция `IsSelfReciprocal(a,x,'p')` – проверяет полином `a(x)` на условие `coeff(a,x,k) = coeff(a,x,d-k)` для всех `k = 0..d`, где `d = degree(a, x)` – порядок полинома. Если это условие выполняется, то возвращается логическое значение `true`, иначе – `false`. Если порядок `d` четный и если задан третий аргумент `p`, то `p` будет представлять полином `P` порядка `d/2`, такой, что `x^(d/2)*P(x+1/x) = a`. При нечетном `d` полином `a` будет взаимообратным, что подразумевает деление на `x+1`. В этом случае если `p` указано, результат вычисляется в форме `a/(x+1)`.

Примеры применения этой функции представлены ниже:

```
> with(PolynomialTools):
IsSelfReciprocal(x^4+x^3+x+1, x, 'p');
```
$$true$$
```
> p;
```
$$-2 + x + x^2$$
```
> IsSelfReciprocal(x^5-3*x^4+x^3+x^2-3*x+1, x, 'p');
```
$$true$$
```
> p;
```
$$3 - 4x + x^2$$
```
> r := evalf( 1+sqrt(2) );
```
$$r := 2.414213562$$

Функция `MinimalPolynomial(r, n, acc)` возвращает полином минимальной степени, не превышающей `n`, имеющий корень `r`. Необязательный аргумент `acc` задает погрешность приближения. Функция `MinimalPolynomial(r, n)` использует решетчатый алгоритм и находит полином степени `n` (или менее) с наименьшими целыми коэффициентами. Корень `r` может быть действительным или комплексным. Результат зависит от значения переменной окружения `Digits`. По умолчанию `acc` задано как `10^(Digits-2)`. Примеры применения данной функции:

```
> MinimalPolynomial( r, 2 );
```
$$-1 - 2\_X + \_X^2$$
```
> r := 1+sqrt(2);
```
$$r := 1 + \sqrt{2}$$

```
> ( r, 2 );
```

$$1+\sqrt{2},2$$

```
> MinimalPolynomial( 1.234, 3 );
```

$$-109 + 61\,\_X - 5\,\_X^2 + 22\,\_X^3$$

```
> fsolve( %, _X );
```

$$1.234000001$$

Функция `Split(a, x, b)` служит для расщепления полинома `a` с независимой переменной `x`. Параметр `b` – необязательный. Функция `Split(a, x)` осуществляет комплексную факторизацию инвариантного полинома `a` по `x`. Если третий аргумент `b` задан, он представляет множество элементов `{t1,...,tm}`, таких что полином `a` расщепляется над `K=Q(t1,...,tm)`, где `Q` означает поле рациональных чисел. Примеры:

```
> Split(x^2+x+1,x);
```

$$(x - \text{RootOf}(\_Z^2 + \_Z + 1))(x + 1 + \text{RootOf}(\_Z^2 + \_Z + 1))$$

```
> Split(x^2+y*x+1+y^2, x, 'b');
```

$$(x - \text{RootOf}(\_Z^2 + y\_Z + 1 + y^2))(x + y + \text{RootOf}(\_Z^2 + y\_Z + 1 + y^2))$$

```
> b;
```

$$\{\text{RootOf}(\_Z^2 + y\_Z + 1 + y^2)\}$$

В пакете определена еще одна подобная функция `Splits`, с которой можно познакомиться по справке на нее.

Функция `Translate(a,x,x0)` преобразует полином `a(x)` с подстановкой `x = x + x0`, где `x0` – константа. Примеры применения этой функции даны ниже:

```
> Translate(x^2,x,1);
```

$$1 + 2x + x^2$$

```
> Translate(x^3,x,2);
```

$$8 + 12x + 6x^2 + x^3$$

```
> Translate((x+1)^3,x,-1);
```

$$x^3$$

5.7.3. Функции сортировки полиномов

Для *сортировки полиномов* предназначены следующие три функции:

`Shorter(f, g, x)` **`Sort(v, x)`** **`Shorten(f, x)`**

Здесь `f` и `g` – полиномы, `v` – список полиномов и `x` – независимая переменная. Функции отличаются характером сортировки.

Функция `Shorter` определяет полином `f` как более короткий, чем `g`, по следующим признакам: меньшая длина, меньшее имя независимой переменной `x`, недробный и меньшая степень других переменных. Функция `Sort` сортирует лист полиномов `x` по признакам, определяемым `Shorter`. Функция `Shorten` использует преобразования Мёбиуса. Многочисленные детали ее применения можно найти в справке по данной функции. Примеры применения функций сортировки:

```
> Shorten(x^2+x+1,x);
```

$$x^2 + 3$$

```
> Shorten(3*x^3+18*x+14,x);
```

$$x^3 - 6$$

```
> Shorten(x^4+32);
```

$$x^4 + 2$$

```
> Shorter(x^3,x+5,x);
```

$$false$$

```
> Sort([x^3,x^2,x+1,x+5]);
Error, (in sort_poly) sort_poly uses a 2nd argument, x,
which is missing
> Sort([x^3,x^2,x+1,x+5],x);
```

$$[1 + x, x + 5, x^2, x^3]$$

5.7.4. Функции преобразования полиномов в PDE и обратно

Функция `PolynomialToPDE(polys, vars, depvars)` преобразует полиномы `polys` по независимым переменным `vars` в дифференциальные уравнения с частными производными (PDE). Другая функция `PDEToPolynomial(pdes, vars, depvars)` осуществляет обратное преобразование. Следующие примеры иллюстрируют применение этих функций:

```
> S:=  PolynomialToPDE([(x^2 - 2*x + 1)*u + x^3*v], [x], [u,v]);
```

$$S := \left[\left(\frac{\partial}{\partial x^2}\mathrm{u}(x)\right) - 2\left(\frac{\partial}{\partial x^2}\mathrm{u}(x)\right) + \mathrm{u}(x) + \left(\frac{\partial^3}{\partial x^3}\mathrm{v}(x)\right)\right]$$

```
> PDEToPolynomial(S, [x], [u,v]);
```

$$[(x^2 - 2x + 1)u + x^3 v]$$

5.8. Интегральные преобразования

5.8.1. Прямое и обратное Z-преобразования

Прямое и обратное Z-преобразования функций широко используются при решении задач автоматического управления и обработке дискретных сигналов [156–160]. Прямое Z-преобразование последовательности $f(n)$ в функцию комплексной переменной z задается выражением

$$f(z) = \sum_{n=-\infty}^{\infty} f(n) \cdot z^{-n}.$$

Обратное Z-преобразование сводится к преобразованию комплексной функции $f(z)$ в функцию $f(z)$.

Эти преобразования задаются следующими функциями:

- `ztrans(f, n, z)` – прямое преобразование функции $f(n)$ в $f(z)$;
- `invztrans(f, z, n)` – обратное преобразование $f(z)$ в $f(n)$.

Заметим, что прямое Z-преобразование базируется на соотношении `ztrans(f(n),n,z) = sum(f(n)/z^n,n=0..infinity)`, записанном на Maple-языке. В первых версиях системы Maple Z-преобразования выполнялись средствами библиотеки и требовали вызова командой `readlib(ztrans)`. Но в Maple 7/8 они уже включены в ядро системы и предварительного вызова не требуют. В этом убеждают следующие примеры:

```
> a:=ztrans(n^2,n,z);
```

$$a := \frac{z(z+1)}{(z-1)^3}$$

```
> invztrans(a,z,n);
```

$$n^2$$

```
> ztrans(cos(Pi/4*t),t,z);
```

$$\frac{2z^2 - z\sqrt{2}}{2z^2 - 2z\sqrt{2} + 2}$$

```
> invztrans(%,z,t);
```

$$\cos\left(\frac{\pi t}{4}\right)$$

Нетрудно заметить, что в этих примерах функции после прямого и обратного преобразований восстанавливают свои значения.

5.8.2. Быстрое преобразование Фурье

Преобразование Фурье широко используется в математике, физике и электрорадиотехнике. Ввиду широких сфер применения этого преобразования в технике часто используется его особая разновидность – *быстрое преобразование Фурье*, или FFT (Fast Fourier Transform).

В Maple на уровне ядра реализованы функции быстрого прямого FFT и обратного iFFT преобразований Фурье для числовых данных:

```
FFT(m, x, y)      evalhf(FFT(m, var(x), var(y)))
iFFT(m, x, y)     evalhf(iFFT(m, var(x), var(y)))
```

Здесь m – целое неотрицательное число, **x** и **y** – массивы с числом элементов, кратным степени 2 (например, 4, 8, 16 и т. д.), представляющие действительные и мнимые части массива комплексных чисел (данных). Функции возвращают число элементов выходных массивов, а результат преобразований помещается в исходные массивы:

```
> x := array([1.,2.,3.,4.]): y := array([5.,6.,7.,8.]):
> FFT(2,x,y);
```

4

```
> print(x);
```

[10., -4., -2., 0.]

```
> print(y);
```

[26., 0., -2., -4.]

```
> iFFT(2,x,y);
```

4

```
> print(x);
```

[1.0000000, 2.0000000, 3.0000000, 4.0000000]

```
> print(y);
```

[5.0000000, 6.0000000, 7.0000000, 8.0000000]

Несмотря на высокую эффективность быстрых преобразований Фурье, их недостатком является применение только к дискретно заданным численным данным, причем с числом отсчетов, кратным двум в целой степени. Если данных меньше, недостающие элементы обычно заменяются нулями.

5.8.3. Общая характеристика пакета `inttrans`

Для расширенной поддержки *интегральных преобразований* служит пакет `inttrans`.

Это один из пакетов, наиболее важных для общематематических и научно-технических приложений. Он вызывается командой

```
> with(inttrans);
```

[*addtable*, *fourier*, *fouriercos*, *fouriersin*, *hankel*, *hilbert*, *invfourier*, *invhilbert*, *invlaplace*, *invmellin*, *laplace*, *mellin*, *savetable*]

и содержит небольшой набор функций. Однако эти функции охватывают такие практические важные области математики, как ряды Фурье, прямые и обратные преобразования Лапласа и Фурье и ряд других интегральных преобразований. Ниже они обсуждены более подробно.

5.8.4. Прямое и обратное преобразования Фурье

Прямое преобразование Фурье преобразует функцию времени $f(t)$ в функцию частот $F(w)$ и заключается в вычислении следующей интегральной функции:

$$F(w) = \int_{-\infty}^{\infty} f(t) e^{-jwt} dt.$$

Оно в аналитическом виде реализуется следующей функцией пакета интегральных преобразований `inttrans`:

```
fourier(expr,t,w)
```

Здесь `expr` – выражение (уравнение или множество), `t` – переменная, от которой зависит `expr`, и `w` – переменная, относительно которой записывается результирующая функция.

Обратное преобразование Фурье задается вычислением интеграла

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(w) e^{jtw} dw.$$

Оно фактически переводит представление сигнала из частотной области во временную. Благодаря этому преобразования Фурье удобны для анализа прохождения воздействий (сигналов) si(t) через устройства (цепи), заданные их частотной характеристикой K(w):

si(t) → fourier → s(w) → s(w)·K(w) → invfourier → so(t).

Здесь si(t) и so(t) – временные зависимости соответственно входного и выходного сигналов.

Определение (визуализация) преобразований Фурье и примеры их осуществления представлены ниже:

```
> restart:with(inttrans): assume(lambda>0,a>0):
> convert(fourier(f(t),t,s),int);
```

$$\int_{-\infty}^{\infty} \mathrm{f}(t)\mathbf{e}^{(-Its)}dt$$

```
> convert(invfourier(f(t),t,s),int);
```

$$\frac{1}{2}\left(\frac{1}{\pi}\int_{-\infty}^{\infty} \mathrm{f}(t)\mathbf{e}^{(tsI)}dt\right)$$

```
> fourier(sin(t),t,w);
```

$$-I\pi\ \mathrm{Dirac}(w-1)+I\pi\ \mathrm{Dirac}(w+1)$$

```
> invfourier(%,w,t);
```

$$\frac{1}{2}I(-\mathbf{e}^{(It)}+\mathbf{e}^{(-It)})$$

```
> simplify(%);
```

$$\sin(t)$$

```
> fourier(1-exp(-a*t),t,w);
```

$$2\pi\ \mathrm{Dirac}(w)-\mathrm{fourier}(\mathbf{e}^{(-at)},t,w)$$

```
> invfourier(%,w,t);
```

$$1-\mathbf{e}^{(-at)}$$

```
> fourier(ln(1/sqrt(1+x^2)),x,y);
```

$$\frac{\pi(\mathbf{e}^{(-y)}\mathrm{Heaviside}(y)-\mathbf{e}^{y}\mathrm{Heaviside}(-y))}{y}$$

```
> fourier(BesselJ(n,x),x,y);
```

$$\frac{2I(-1)^{\left(\frac{n}{2}-1/2\right)}\mathrm{ChebyshevT}(n,y)(\mathrm{Heaviside}(y+1)-\mathrm{Heaviside}(y-1))}{\sqrt{1\quad y^2}}$$

Обратите внимание на то, что даже в простом первом примере применение обратного преобразования Фурье вслед за прямым не привело к буквальному восстановлению исходной функции $\sin(t)$. Потребовалась команда `simplify`, чтобы перевести результат в виде представления синуса через экспоненциальные функции к обычному виду $\sin(t)$.

5.8.5. Вычисление косинусного и синусного интегралов Фурье

Разложение функции $f(t)$ в ряд Фурье требует вычисления интегралов следующего вида:

$$\sqrt{\frac{2}{\pi}}\int_0^\infty f(t)\cos(st)dt,\ \sqrt{\frac{2}{\pi}}\int_0^\infty f(t)\sin(st)dt.$$

Они получили название косинусного и синусного интегралов Фурье и фактически задают вычисление коэффициентов ряда Фурье, в который может быть разложена функция $f(t)$.

Для вычисления этих интегралов в пакете используются следующие функции:

fouriercos(expr,t,s) **fouriersin(expr,t,s).**

Поскольку формат задания этих функций вполне очевиден, ограничимся примерами визуализации сути этих функций и примерами их применения:

```
> convert(fouriercos(f(t),t,s),int);
```

$$\frac{\sqrt{2}}{\sqrt{\pi}}\int_0^\infty \mathrm{f}(t)\cos(ts)dt$$

```
> convert(fouriersin(f(t),t,s),int);
```

$$\frac{\sqrt{2}}{\sqrt{\pi}}\int_0^\infty \mathrm{f}(t)\sin(ts)dt$$

```
> fouriercos(5*t,t,s);
```

$$-5\frac{\sqrt{2}}{\sqrt{\pi}\,s^2}$$

```
> fouriersin(5*t,t,s);
> fouriercos(exp(-t),t,s);
```

$$\frac{\sqrt{2}}{\sqrt{\pi}\,(s^2+1)}$$

```
> fouriercos(arccos(x)*Heaviside(1-x),x,y);
```

$$\frac{1}{2}\frac{\sqrt{2}\sqrt{\pi}\,\mathrm{StruveH}(0,y)}{y}$$

```
> fouriersin(arcsin(x)*Heaviside(1-x),x,y);
```

$$\frac{1}{2}\frac{\sqrt{2}\sqrt{\pi}\,(\mathrm{BesselJ}(0,y)-\cos(y))}{y}$$

Нетрудно заметить, что эти преобразования нередко порождают специальные математические функции. Много примеров на преобразования Фурье содержится в файле демонстрационных примеров fourier.mws.

5.8.6. Прямое и обратное преобразования Лапласа

Преобразования Лапласа – одни из самых часто применяемых интегральных преобразований. Они широко применяются в электрорадиотехнике и часто используются для решения линейных дифференциальных уравнений.

Прямое преобразование Лапсаса заключается в переводе некоторой функции времени $f(t)$ в операторную форму $F(p)$. Это преобразование означает вычисление интеграла

$$F(p)=\int_0^{\infty} f(t)e^{-st}dt.$$

Для осуществления прямого преобразования Лапласа служит функция

```
laplace(expr,t,p)
```

Здесь `expr` – преобразуемое выражение, `t` – переменная, относительно которой записано `expr`, и `p` – переменная, относительно которой записывается результат преобразования.

Обратное преобразование Лапласа означает переход от функции $F(p)$ к функции $f(t)$ с помощью формулы

$$F(t)=\int_{s-I\infty}^{s+I\infty} F(p)e^{-pt}dp.$$

Для вычисления этого интеграла служит функция

```
invlaplace(expr, p, t),
```

где `expr` – выражение относительно переменной `p`, `t` – переменная, относительно которой записывается результирующая зависимость. Оба преобразования широко применяются в практике научно-технических вычислений и отражают суть операторного метода. При этом прямое преобразование создает *изображение*, а обратное – *оригинал* функции. Ниже приведены примеры определения и применения прямого и обратного преобразований Лапласа:

```
> restart:with(inttrans):assume(a>0):
> convert(laplace(f(t),t,s),int);
```

$$\int_0^{\infty} \mathrm{f}(t)\mathbf{e}^{(-ts)}dt$$

```
> laplace(sin(t)+a*cos(t),t,p);
```

$$\frac{1}{p^2+1}+\frac{ap}{p^2+1}$$

```
> invlaplace(%,p,t);
```

$$\sin(t)+a\cos(t)$$

Нетрудно заметить, что в данном случае последовательное применение прямого, а затем обратного преобразования восстанавливает исходную функцию $\sin(t) + a\cos(t)$. Преобразования Лапласа широко используются со специальными функциями и, в свою очередь, порождают специальные функции:

```
> laplace(FresnelC(t),t,p);
```

$$\frac{1}{2}\frac{LommelS2\left(0,\frac{1}{2},\frac{p^2}{2\pi}\right)}{\pi}$$

```
> laplace(Si(t)+Ci(t)+erf(t),t,p);
```

$$\frac{\operatorname{arccot}(p)}{p}-\frac{1}{2}\frac{\ln(p^2+1)}{p}+\frac{e^{\left(\frac{p^2}{4}\right)}\operatorname{erfc}\left(\frac{p}{2}\right)}{p}$$

```
> laplace(BesselJ(0,t),t,p);
```

$$\frac{1}{\sqrt{p^2+1}}$$

```
> invlaplace(1/sqr(p^2+1),t,p);
```

$$\frac{\mathrm{Dirac}(p)}{\mathrm{sqr}(p^2+1)}$$

Преобразования Лапласа широко используются для решения линейных дифференциальных уравнений в аналитическом виде. Ниже дана пара простых примеров, иллюстрирующих технику такого решения для дифференциальных уравнений второго порядка с применением функции `dsolve`:

```
> de1 := diff(y(t),t$2) + 2*diff(y(t),t) + 3*y(t) = 0;
```

$$de1 := \left(\frac{d^2}{dt^2}\mathrm{y}(t)\right)+2\left(\frac{d}{dt}\mathrm{y}(t)\right)+3\mathrm{y}(t)=0$$

```
> dsolve({de1,y(0)=0,D(y)(0)=1},y(t),method=laplace);
```

$$\mathrm{y}(t)=\frac{1}{2}\sqrt{2}\,\mathbf{e}^{(-t)}\sin(\sqrt{2}\,t)$$

```
> de2 := diff(y(x),x$2) - y(x) = x*cos(x);
```

$$de2 := \left(\frac{d^2}{dt^2}\mathrm{y}(x)\right)-\mathrm{y}(x)=x\cos(x)$$

```
> dsolve({de2,y(0)=0,D(y)(0)=0},y(x), method=laplace);
```

$$\mathrm{y}(x)=\frac{1}{2}\sin(x)-\frac{1}{2}x\cos(x)$$

Множество примеров на применение преобразования Лапласа можно найти в файле laplace.mws, имеющемся на интернет-сайте корпорации MapleSoft, а также в книге [188].

5.8.7. Интегральное преобразование Ханкеля

Интегральное *преобразование Ханкеля* задается следующим выражением:

$$F[nu](s) = \int_0^{\infty} f(t) \cdot t \cdot \mathrm{BesselJ}(nu, s \cdot t) dt$$

и выполняется функцией

```
hankel(expr, t, s, nu)
```

Здесь `expr` – выражение, равенство (или множество, или список с выражениями/равенствами), `t` – переменная в `expr`, преобразуемая в параметр преобразования `s`, `nu` – порядок преобразования. Следующий пример демонстрирует вывод и применение функции Ханкеля:

```
> convert(hankel(f(t),t,s,v),int);
```

$$\int_0^{\infty} \mathrm{f}(t)\sqrt{ts}\ \mathrm{BesselJ}(v\sim, ts)dt$$

```
> hankel(sqrt(t)/(alpha+t),t,s,0);
```

$$\frac{1}{2}\frac{\sqrt{s}\ \pi\ \mathrm{BesselI}(0, \alpha s)}{\alpha}$$

```
> hankel(sqrt(t)*Ci(alpha*t^2),t,s,0);
```

$$-\frac{2\left(-1+\cos\left(\frac{s^2}{4\alpha}\right)\right)}{s^{(3/2)}}$$

```
> assume(-1/2<mu,mu<1/2);
  hankel(1/sqrt(t)*BesselY(mu,alpha/t),t,s,mu);
```

$$\frac{-2\mathrm{BesselK}(2\mu\sim, 2\sqrt{\alpha}\sqrt{s}) + \pi\mathrm{BesselY}(2\mu\sim, 2\sqrt{\alpha}\sqrt{s})}{\pi\sqrt{s}}$$

В файле hankel.mws демонстрационных примеров можно найти множество интересных примеров на применение функции преобразования Ханкеля.

5.8.8. Прямое и обратное преобразования Гильберта

Прямое *преобразование Гильберта* задается следующим выражением:

$$F(s) = \frac{1}{\pi}\int_{-\infty}^{\infty} \frac{f(t)}{t-s} dt$$

и превращает функцию $f(t)$ в $F(s)$. Обратное преобразование Гильберта означает нахождение $f(t)$ по заданной $F(s)$. Эти преобразования выполняются функциями:

```
hilbert(expr, t, s)  invhilbert(expr, t, s),
```

где назначение параметров очевидно. Приведенные ниже примеры иллюстрируют выполнение этих преобразований:

```
> restart:with(inttrans):
> assume(-1/2<v,v<3/2,nu>0,a>0,alpha>0,beta>0):
> convert(hilbert(f(t),t,s),int);
```

$$\frac{1}{\pi}\int_{-\infty}^{\infty}\frac{\mathrm{f}(t)}{t-s}dt$$

```
> convert(invhilbert(f(t),t,s),int);
```

$$-\left(\frac{1}{\pi}\int_{-\infty}^{\infty}\frac{\mathrm{f}(t)}{t-s}dt\right)$$

```
> hilbert(exp(1), r, z);
```

$$0$$

```
> hilbert(f(u), u, t);
```

$$\mathrm{hilbert}(\mathrm{f}(u),u,t)$$

```
> hilbert(%, t, s);
```

$$-\mathrm{f}(s)$$

```
> hilbert(t/(t^2+1),t,s);
```

$$\frac{1}{s^2+1}$$

```
> invhilbert(%,s,t);
```

$$\frac{1}{t^2+1}$$

```
> hilbert(Ci(abs(t)),t,s);
```

$$-\mathrm{signum}(s)\mathrm{Ssi}(|s|)$$

```
> hilbert(signum(t)*Ssi(abs(t)),t,s);
```

$$\mathrm{Ci}(|s|)$$

Следует отметить, что обратное преобразование Гильберта, осуществленное над результатом прямого преобразования, не всегда восстанавливает функцию $f(t)$ буквально. Много интересных примеров на это преобразование Гильберта можно найти в файле gilbert.mws.

5.8.9. Интегральное преобразование Меллина

Интегральное *преобразование Меллина* задается выражением

$$M(s)=\int_0^{\infty} m(x)x^{s-1}dx$$

и реализуется функцией

```
mellin(expr, x, s)
```

с очевидными параметрами `expr`, `x` и `s`. Применение преобразования Меллина иллюстрируют следующие примеры:

```
> assume(a>0);
> mellin(x^a,x,s);
```

$$\frac{\Gamma(s+a\sim)\text{Heaviside}(s+a\sim)}{\Gamma(s+a\sim+1)}+\frac{\Gamma(-a\sim-s)\text{Heaviside}(-a\sim-s)}{\Gamma(1-s-a\sim)}$$

```
> mellin(f(a*x),x,s);
```

$$\left(\frac{1}{a\sim}\right)^{s}\text{mellin}(\text{f}(x),x,s)$$

```
> invmellin((gamma+Psi(1+s))/s,s,x,-1..infinity);
```

$$-\text{Heaviside}(1-x)\ln(1-x)$$

Примеры на применение преобразования Меллина можно найти в файле mellin.mws.

5.8.10. Функция addtable

Как видно из приведенных примеров, не всегда интегральные преобразования дают результат в явном виде. Получить его позволяет вспомогательная функция

```
addtable(tname,patt,expr,t,s)
```

где `tname` – наименование преобразования, для которого образец `patt` должен быть добавлен к таблице поиска. Остальные параметры очевидны.

Следующие примеры поясняют применение этой функции:

```
> fouriersin(f(t),t,s);
```

$$\text{fouriersin}(\text{f}(t),t,s)$$

```
> addtable(fouriersin,f(t),F(s),t,s);
> fouriersin(f(x),x,z);
```

$$\text{F}(z)$$

5.9. Работа в Maple с функциями двух переменных

5.9.1. Maplet-инструмент для работы с функциями двух переменных

Для эффектной демонстрации работы с функциями многих переменных в состав пакета Student системы Maple введен новый подпакет MultivariateCalculus. Его примеры можно запускать как с командной строки, так и из позиции **Tools** меню в стандартном варианте интерфейса – **Tutors** ⇒ **Calculus** ⇒ **Calculus-Multi-Variables**.

- **Approximate Integration ...** – открывает Maplet-окно аппроксимации двойных интегралов;
- **Cross Section...** – открывает Maplet-окно демонстрации сечения поверхности;

- **Directional Derivatives...** – открывает Maplet-окно вычисления производных в заданном направлении;
- **Gradient...** – открывает Maplet-окно вычисления градиента;
- **Taylor Series...** – открывает Maplet-окно разложения функций в ряд Тейлора.

Представленные средства носят учебный характер – не случайно они входят в пакет Student. Реально визуализация возможна только для функций двух переменных.

5.9.2. Демонстрация разложения в ряд Тейлора функции двух переменных

Команда **Taylor Series...** открывает Maplet-окно разложения функции двух переменных z(x,y) в ряд Тейлора относительно заданной точки (x0,y0). Это окно представлено на рис. 5.17.

В данном окне дан пример разложения в ряд Тейлора функции sin(*x***y*) в окрестности точки (0,0) в интервале изменения *x* [−2,2], *y* [−2,2] и *z* [−1,1]. Установки

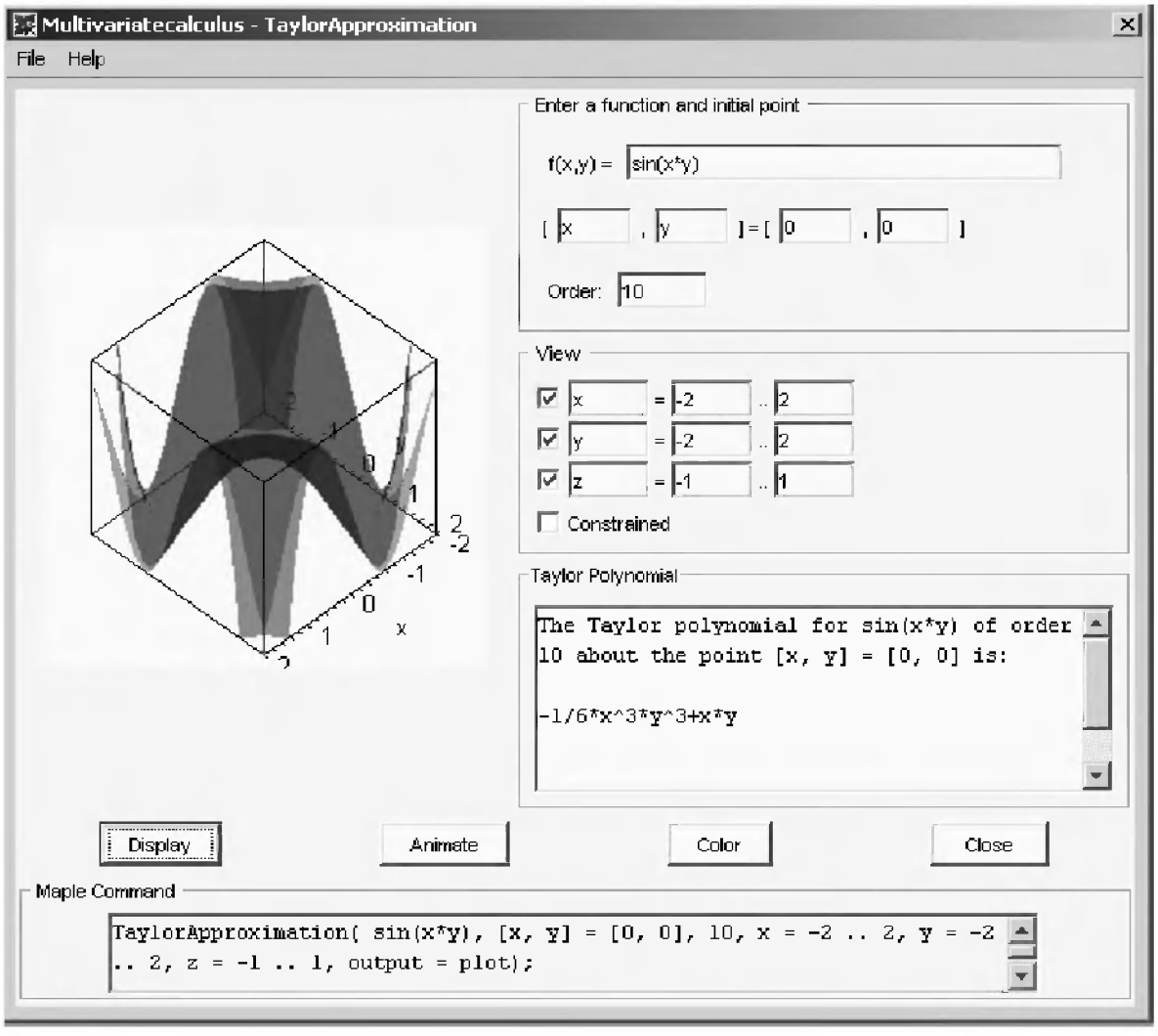


Рис. 5.17. Maplet-окно демонстрации разложения в ряд Тейлора функции двух переменных

в окне совершенно очевидны. Графики в правой части представляют поверхность, описываемую исходной функцией, и поверхность, представленную рядом Тейлора. Кнопка **Display** начинает построение графиков, кнопка **Animation** позволяет наблюдать анимацию разложения, а кнопка **Close** закрывает окно и переносит рисунок в текущий документ системы Maple 9.5.

5.9.3. Демонстрация вычисления градиента функции двух переменных

Команда **Gradient...** открывает Maplet-окно демонстрации вычисления градиента функции двух переменных $z(x,y)$ в ряд Тейлора относительно заданной точки (x0,y0). Это окно представлено на рис. 5.18.

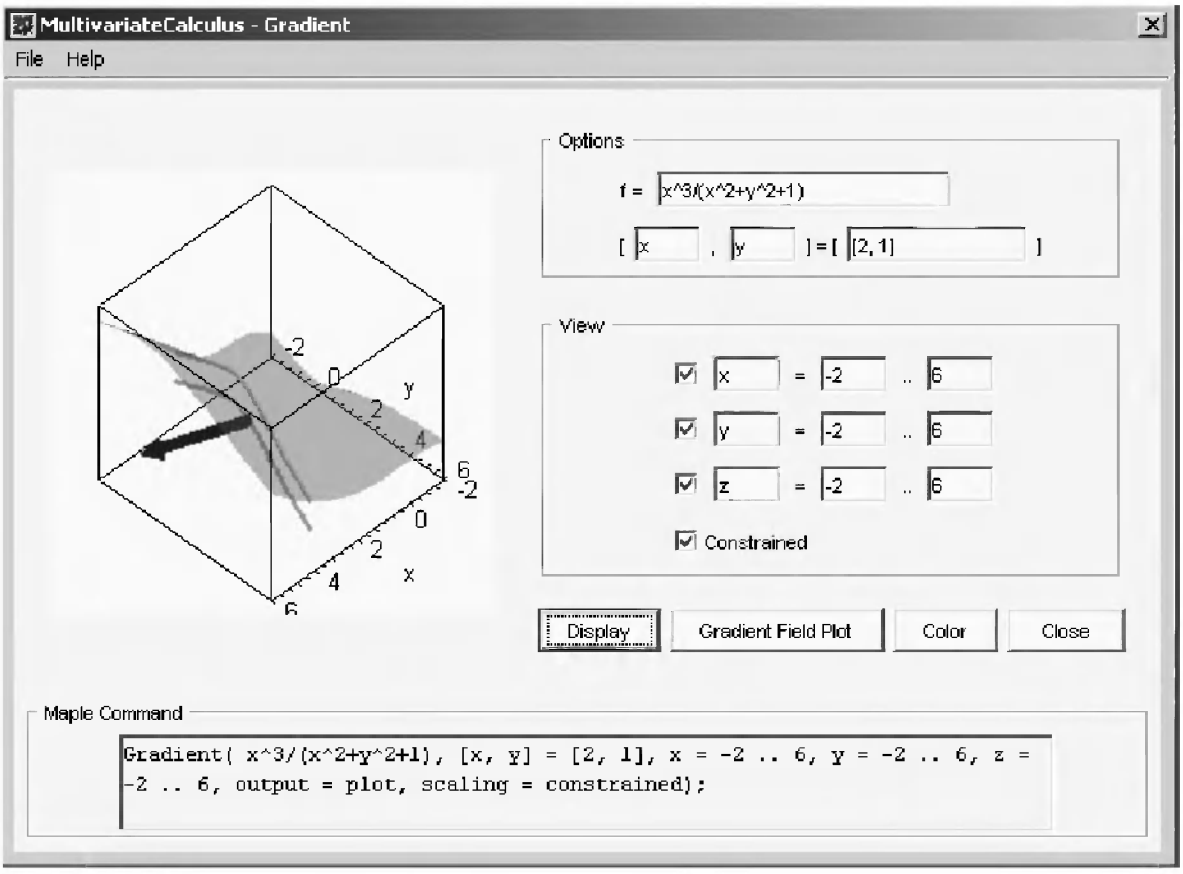


Рис. 5.18. Maplet-окно демонстрации вычисления градиента функции двух переменных

Работа с этим окном практически не отличается от описанной для примера с рядом Тейлора. Единственное исключение – новая кнопка **Gradient Field Plot**. Она позволяет строить график поля градиента с помощью стрелок.

5.9.4. Демонстрация вычисления производной в заданном направлении

Команда **Directional Derivatives...** открывает Maplet-окно демонстрации вычисления производных функции двух переменных $z(x,y)$ в заданном направлении, указанном точкой с координатами (x, y). Это окно представлено на рис. 5.19.

Работа с этим окном практически не отличается от описанной для предшествующих примеров.

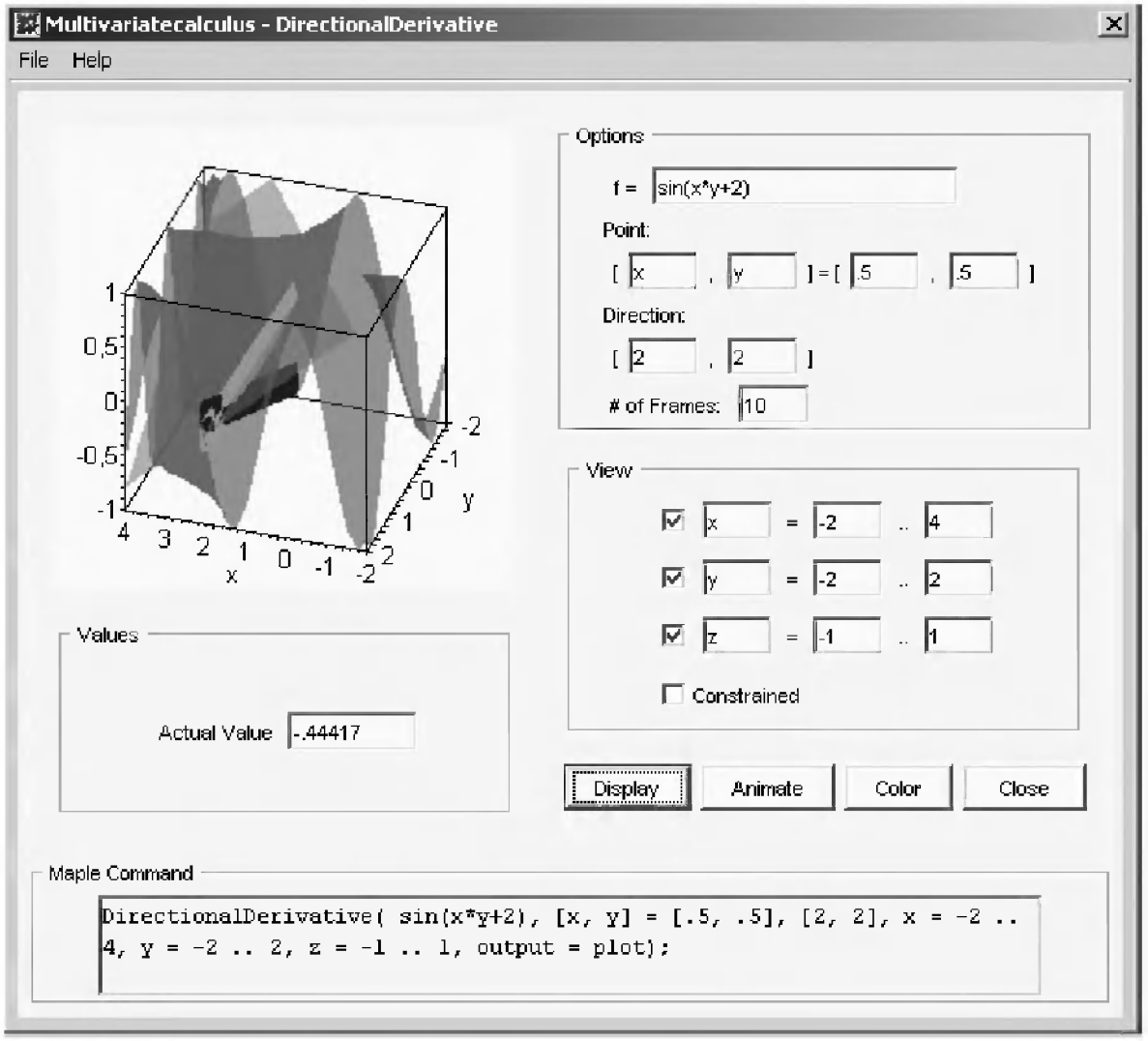


Рис. 5.19. Maplet-окно демонстрации вычисления градиента функции двух переменных

5.9.5. Демонстрация приближенного вычисления интеграла

Команда **Approximate Integration ...** открывает Maplet-окно демонстрации вычисления двойных интегралов с подынтегральной функцией двух переменных z(x,y). Это окно представлено на рис. 5.20.

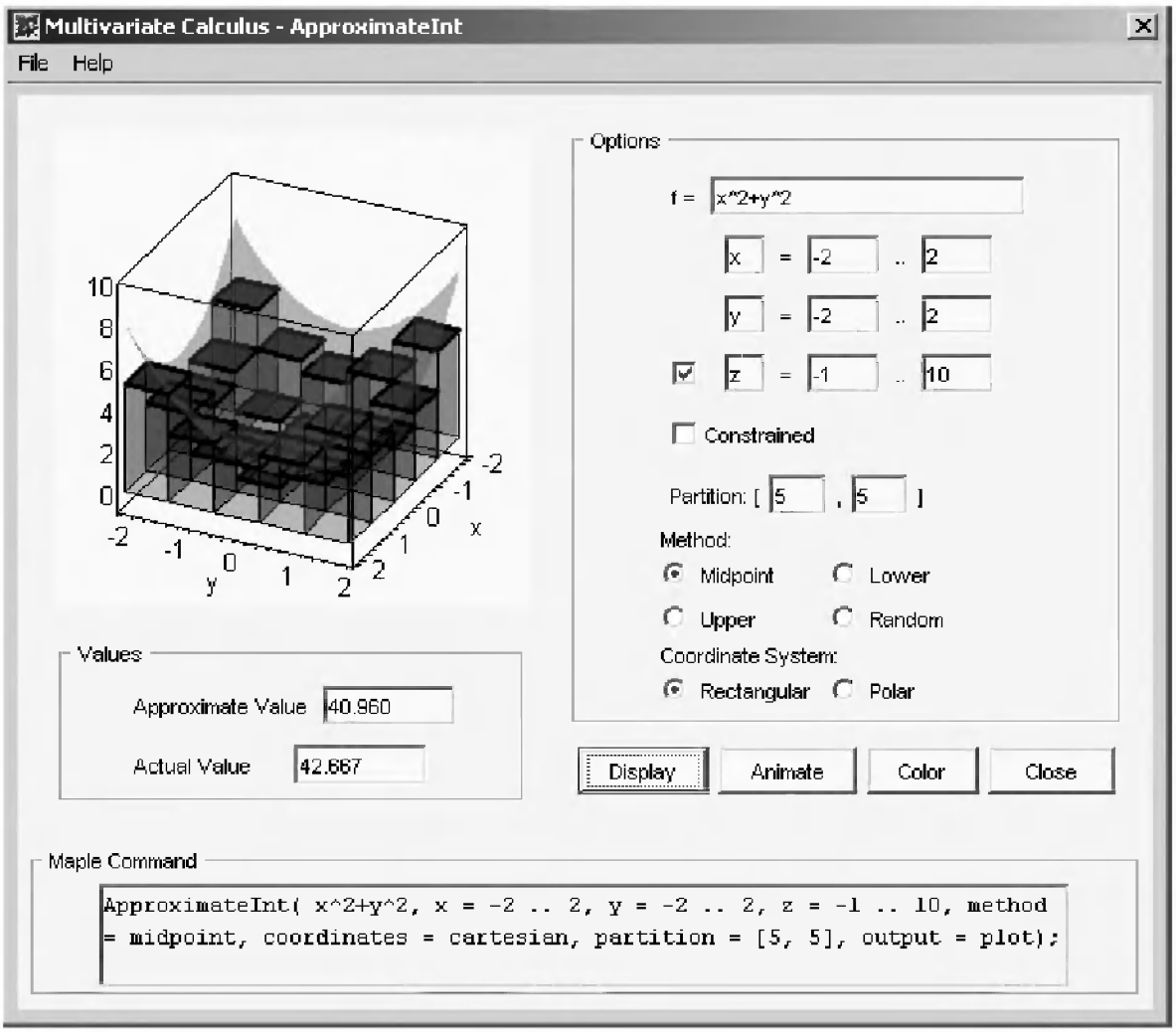


Рис. 5.20. Maplet-окно демонстрации приближенного вычисления двойного интеграла в прямоугольной системе координат

Для вычисления интеграла нужно задать подынтегральную функцию и пределы по переменным x и y. Для построения графика можно также задать пределы по переменной z. Приближенное значение интеграла вычисляется суммированием объмов прямоугольных столбиков, на которые разбивается пространство под поверхностью $z(x, y)$. Число разбиений устанавливается списком Partition. Можно задать один из четырех методов расположения столбиков. В области Value отображаются точное и приближенное (сумма объемов столбиков) значения интеграла. Возможно представление интеграла и в полярной системе координат.

5.9.6. Демонстрация сечения поверхности

Команда **Cross Section...** открывает Maplet-окно демонстрации сечения поверхности плоскостями. Поверхность задается функцией двух переменных z(x,y). Окно этой команды представлено на рис. 5.21.

Работа с этим окном вполне очевидна. На рисунке в левой части окна строятся исходная поверхность, секущие плоскости и линии их пересечения.

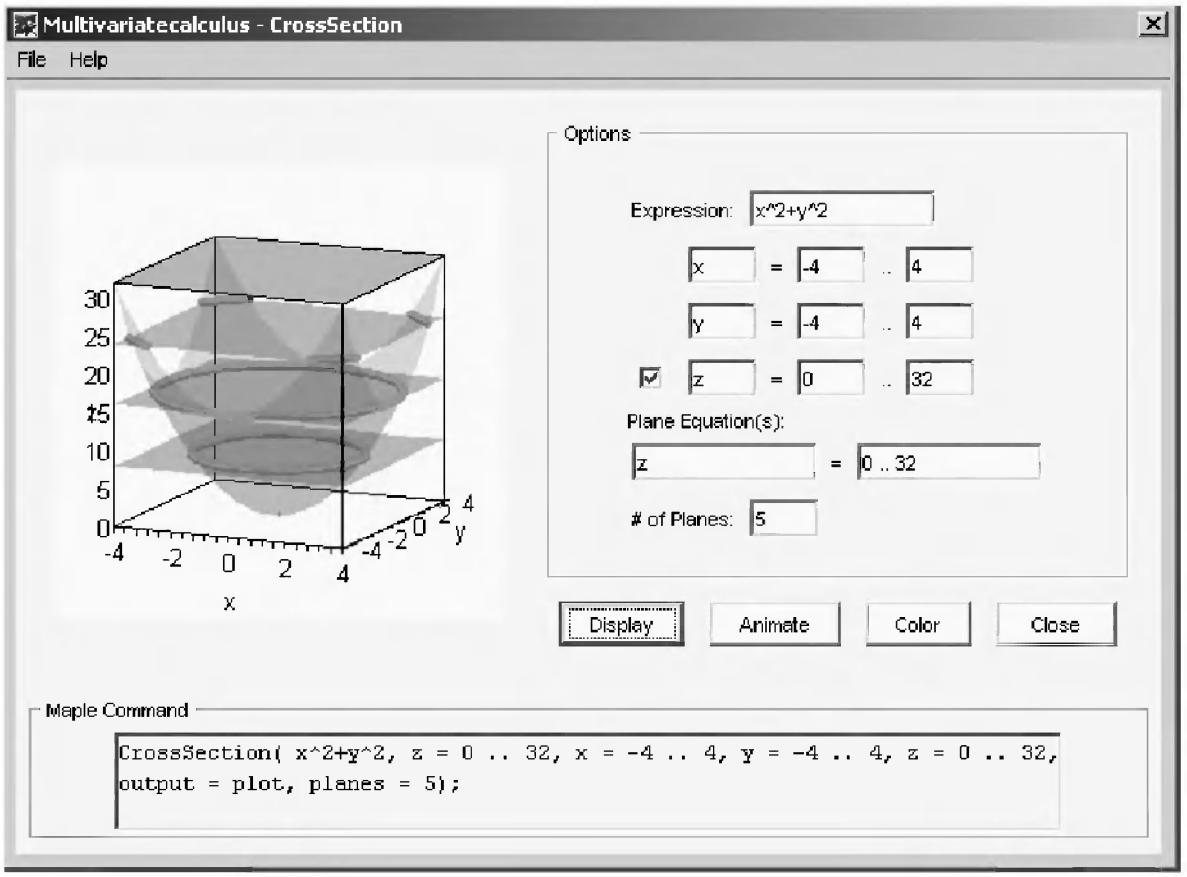


Рис. 5.21. Maplet-окно демонстрации сечения поверхности параллельными плоскостями

5.10. Работа в Mathematica 6 с функциями двух переменных

5.10.1. Интерполяция при построении контурных графиков

При построении контурных графиков с малым числом точек они получаются грубыми. Этого недостатка позволяет избежать интерполяция функций двух переменных. Пример этого представлен на рис. 5.22. Для интерполяции созданных

```
Flatten[Table[{{x, y}, LCM[x, y]}, {x, 5}, {y, 5}], 1];

f = Interpolation[%]

InterpolatingFunction[{{1, 5}, {1, 5}}, <>]

ContourPlot[f[x, y], {x, 1, 5}, {y, 1, 5}]
```

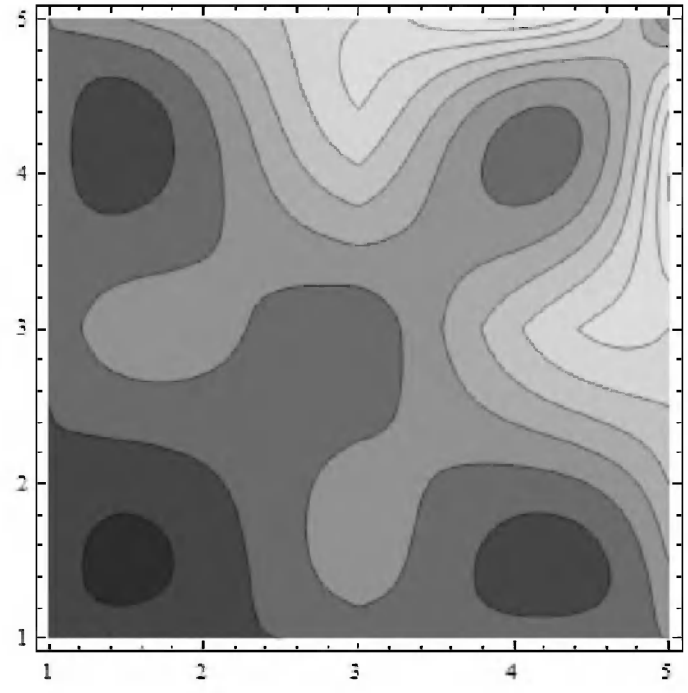


Рис. 5.22. Пример построения контурного графика с интерполяцией

таблицей данных (точек кривых контурного графика) используется функция Interpolation, а для построения самого графика – функция ContourPlot.

5.10.2. Применение однокоординатных слайдеров для построения 3D-графиков

Степень визуализации построения 3D-графиков (графиков функций двух переменных) значительно повышается при изменении значений независимых переменных функции двух переменных с помощью однокоординатных слайдеров. Пример построения и просмотра графика поверхности с применением слайдеров показан на рис. 5.23. Слайдеры выводятся автоматически при применении блока Manipulate. Перемещение движков слайдеров мышью позволяет менять значения переменных функции двух переменных и наблюдать изменение вида поверхности, которая строится функцией Plot3D.

Можно управлять видом и возможностями GUI-интерфейса окна ноутбука с помощью контекстного меню правой клавиши мыши. Это меню показано на

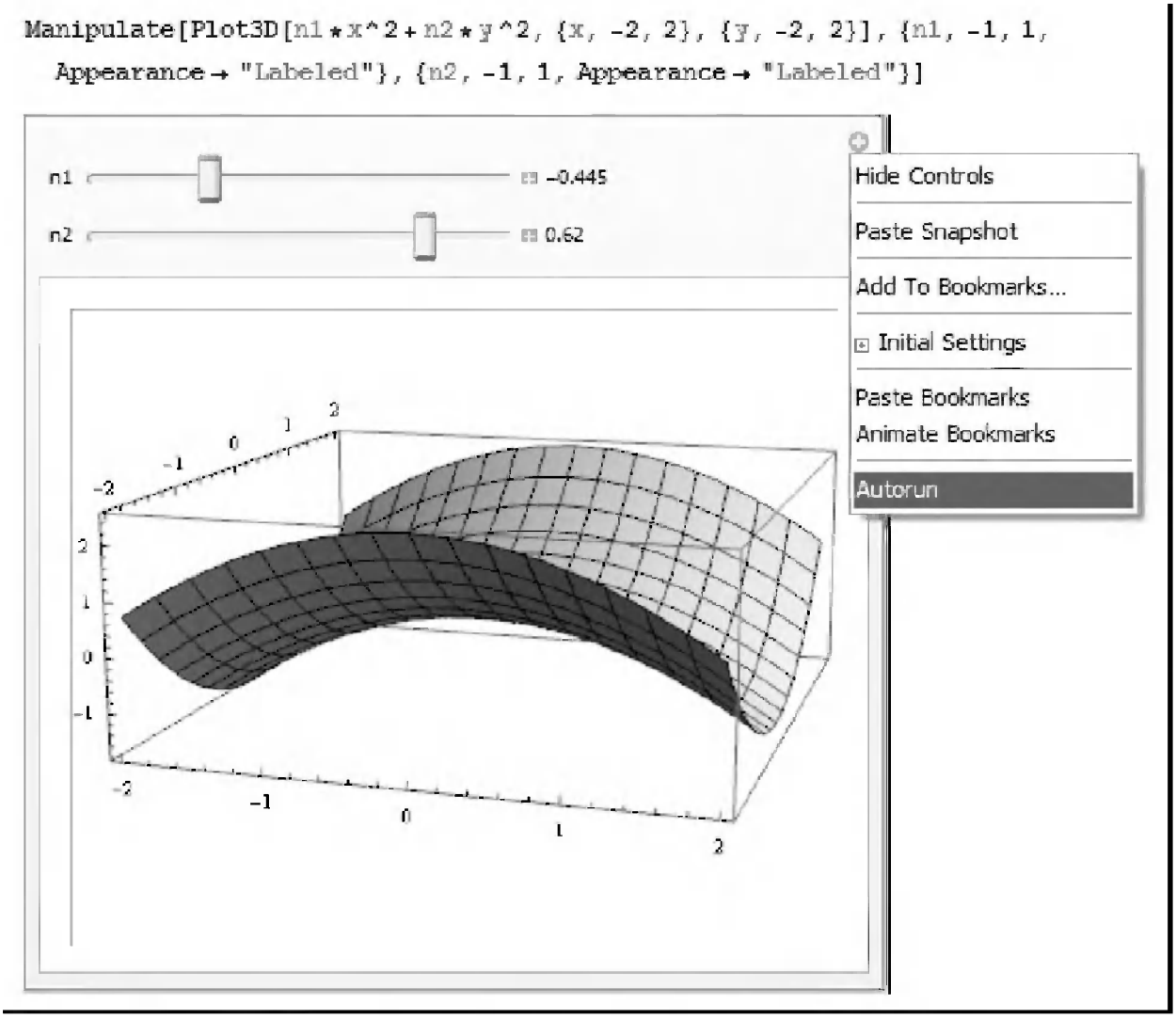


Рис. 5.23. Построение поверхности с применением однокоординатных слайдеров

рис. 5.23. Команда Autorun в этом меню позволяет вывести проигрыватель, обеспечивающий автоматическое изменение значений переменных и изменение положения движка слайдеров, – рис. 5.24.

5.10.3. Применение локаторов при построении контурных графиков

Нередко интерес представляют контурные графики, которые зависят от особых точек – фокусов, перемещение которых меняет характер графиков. К примеру, это графики поверхностей, образованных вершинами гор, которые могут перемещаться. Для задания перемещаемых точек в Mathematica 6 введены особые объекты GUI – локаторы. Рисунок 5.25 демонстрирует построение контурных графиков, линии которых концентрируются вокруг двух точек – локаторов, которые можно перемещать мышью в любое место и тут же наблюдать изменение контурного графика.

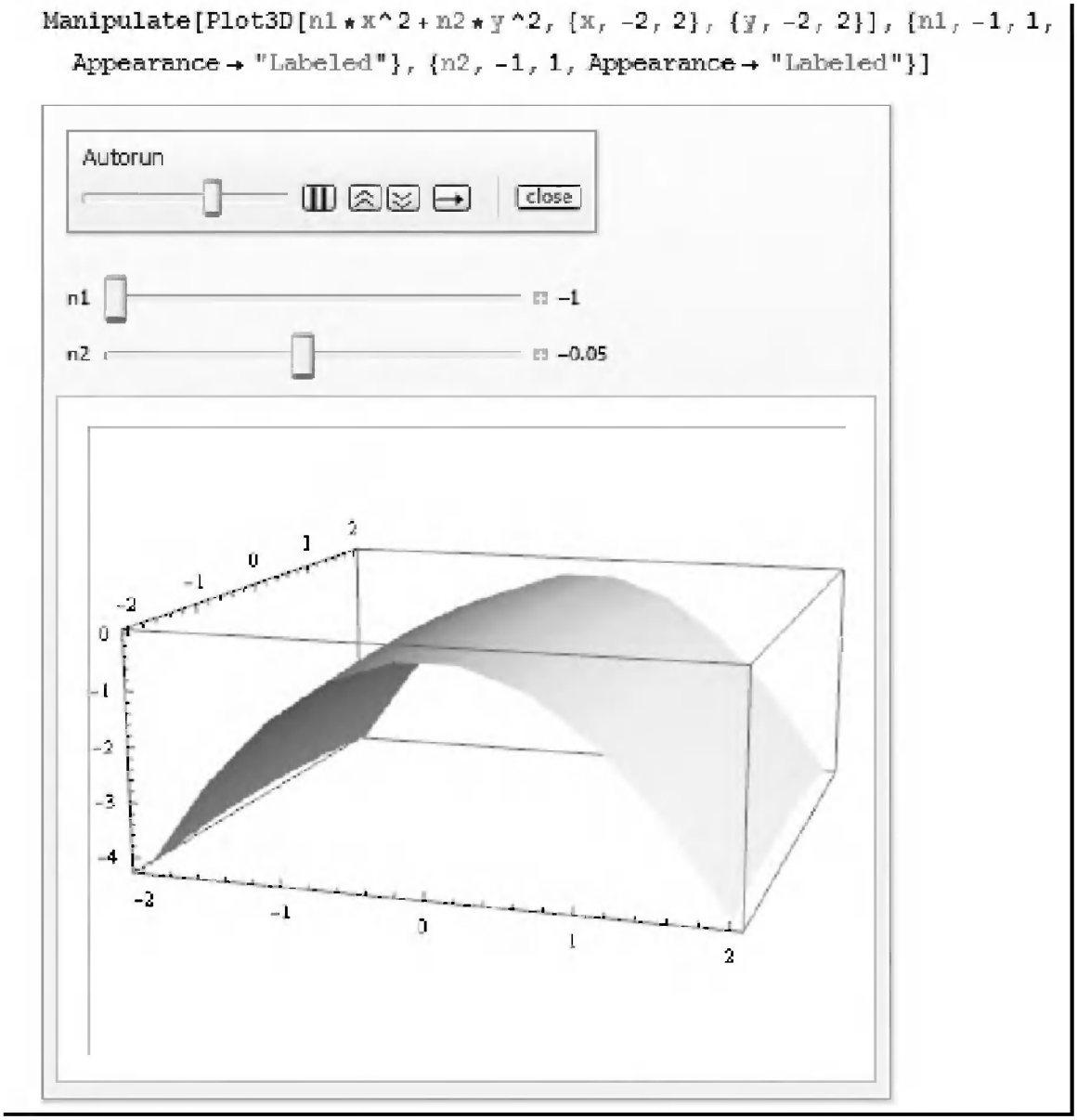


Рис. 5.24. Построение поверхности с автоматическим изменением значений переменных

5.10.4. Двухкоординатный слайдер

Если нужно менять значение одной из двух переменных при сохранении значения второй переменной (и наоборот), то для этого удобно использовать два однокоординатных слайдера. Однако Mathematica 6 имеет специальный GUI-объект – двухкоординатный слайдер (рис. 5.26).

Движок двухкоординатного слайдера имеет вид точки-рукоятки, которую можно перемещать мышью в любом направлении, в том числе диагональном. Слайдер вырабатывает два числа – координаты x и y точки.

```
Manipulate[ContourPlot[q1 / Norm[{x, y} - p[[1]]] + q2 / Norm[{x, y} - p[[2]]],
  {x, -2, 2}, {y, -2, 2}, Contours → 10], {{q1, -1}, -3, 3}, {{q2, 1}, -3, 3},
 {{p, {{-1, 0}, {1, 0}}}, {-1, -1}, {1, 1}, Locator}, Deployed → True]
```

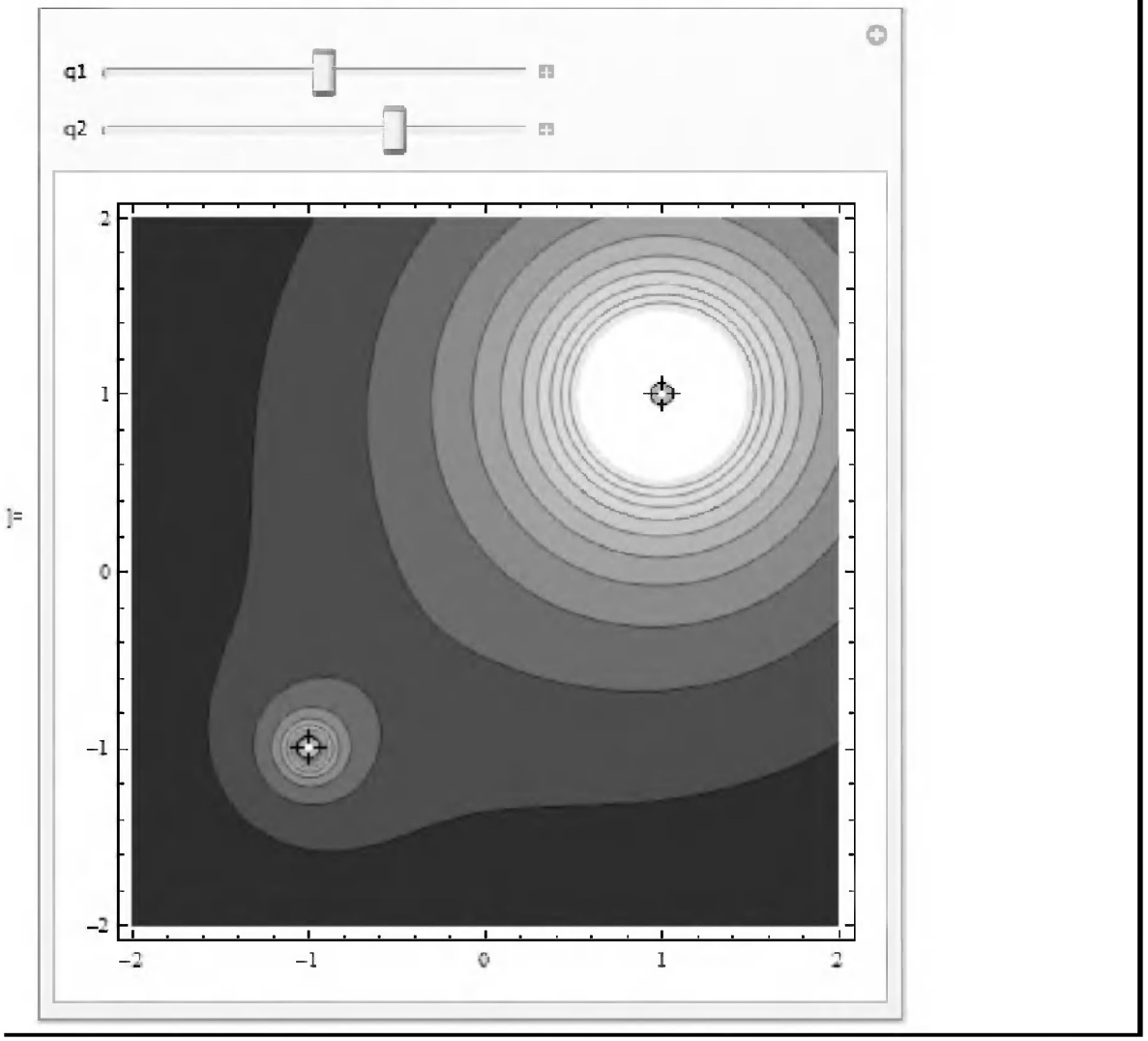


Рис. 5.25. Построение контурного графика, линии которого зависят от положения двух перемещаемых точек – локаторов

```
In[1]:= {Slider2D[Dynamic[r]], Dynamic[r]}

Out[1]= {                , {0.38, 0.36}}
```

Рис. 5.26. Двухкоординатный слайдер и его применение

5.10.5. Комбинированное применение объектов GUI

Модуль Manipulate в Mathematica 6 допускает комбинированное применение объектов GUI в документах. Пример такого применения представлен на рис. 5.27. Здесь моделируется пересечение шаров друг с другом и гранями ограничивающего их куба. Радиусы шаров r1 и r2 устанавливаются однокоординатными слайдерами, а координаты их центров – с помощью двухкоординатного слайдера. Это позволяет эффектно наблюдать за пересечением шаров и их обрезанием гранями ограничивающего куба.

Множество подобных примеров применения средств GUI при построении различных графиков можно найти в справке по системе Mathematica 6.

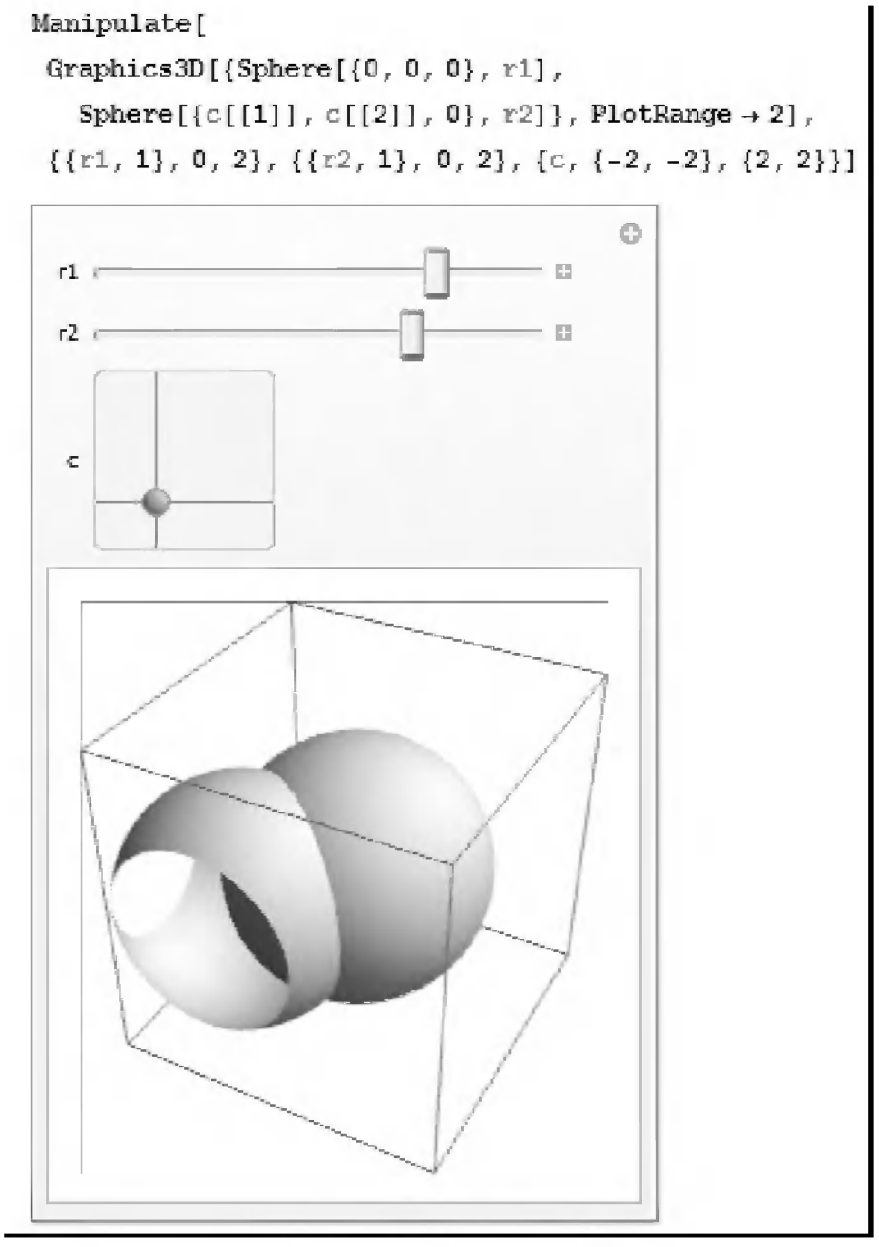


Рис. 5.27. Моделирование пересечения шаров друг с другом и гранями ограничивающего их куба

Глава 6

Приближение функций и прогноз

В этой главе описано приближение различных функций и данных, нередко заданных таблично, некоторыми сравнительно простыми аналитическими зависимостями. Это одна из важных задач математики. Рассматриваются такие виды приближений, как аппроксимация, интерполяция и приближение по методу наименьших квадратов. Описаны также некоторые методы экстраполяции и прогноза.

6.1. Основы теории аппроксимации

6.1.1. Преамбула

Вычисление многих функций, даже элементарных и особенно специальных, требует больших затрат времени. Поэтому не так давно широко применялись таблицы для интерполяции таких функций, например [53]. В наше время для этого повсеместно используются СКМ [1, 189].

Аппроксимацией в системах компьютерной математики обычно называют получение приближенных значений какого-либо выражения. Однако под *аппроксимацией функциональных зависимостей* чаще всего подразумевается получение некоторой конкретной и достаточно легко вычисляемой функции, вычисленные значения которой с заданной погрешностью близки к значениям аппроксимируемой зависимости.

Если некоторая зависимость $y(x)$ представлена рядом табличных отсчетов $y_i(x_i)$, то *интерполяцией* принято называть вычисление значений $y(x)$ при заданном x, расположенном в интервале между отсчетами. За пределами общего интервала определения функции $[a, b]$, то есть при $x < a$ и $x > b$, вычисление $y(x)$ называют *экстраполяцией* (или, иногда, *предсказанием* значений функции либо *прогнозом* [150–153]). В данном случае речь идет об одномерной интерполяции, но возможны двумерная интерполяция функций двух переменных $z(x, y)$ и даже многомерная интерполяция для функций многих переменных.

Здесь мы будем рассматривать вначале такие виды аппроксимации, которые дают точные значения функции $y(x)$ в узловых точках в пределах погрешности вычислений СКМ по умолчанию. Обычно они используются для целей интерполяции и намного реже – для экстраполяции (прогноза). Если аппроксимирующая зависимость выбирается из условия наименьшей среднеквадратической погрешности в узловых точках (метод наименьших квадратов), то мы имеем *регрессию*, или *приближение* функций по методу наименьших квадратов.

6.1.2. Полиномиальная аппроксимация аналитических зависимостей

Пусть приближаемая функция $\varphi(x)$ должна совпадать с исходной функцией $f(x)$ в $(n+1)$-точке, то есть должно выполняться равенство: $\varphi(x_i) = f(x_i) = f_i$, $i = 0,...,n$. В качестве приближающей функции примем алгебраический полином:

$$\varphi(x) \equiv P_n(x) = \sum_{k=0}^{n} a_k x^{n-k}. \qquad (6.1)$$

Задача *полиномиальной аппроксимации* заключается в нахождении коэффициентов a_k полинома, с тем чтобы его значения в заданных точках совпадали со значениями исходной функции. Выбор конкретного значения n во многом определяется свойствами приближающей функции, требуемой точностью, а также выбором узлов интерполяции. В случае аналитической функциональной зависимости выбор степени полинома может быть любым и чаще всего определяется компромиссом между высотой порядка полинома, скоростью его вычисления и погрешностью. В качестве критерия согласия принимается условие совпадения функций f и q в узловых точках:

$f(x_i) = Pn(x_i)$, $(i = 0, 1, ..., n)$.

Полином $P_n(x)$, удовлетворяющий данному условию, будет *интерполяционным полиномом*. Погрешность интерполяции в узловых точках будет равна 0, а степень интерполирующего полинома n на 1 меньше числа узловых точек.

Для задачи интерполяции в интервале $[a,b]$ выбираются значения аргументов $a \le x_0 < x_1 < ... < x_n \le b$, которые соответствуют значениям $f_i = f(x_i)$ $(i = 0,1,...,n)$ функции f. Для этой функции будет существовать, и притом единственный, полином степени не выше n, который принимает в узлах x_i заданные значения f_i.

Для нахождения этого полинома составим *систему алгебраических уравнений*

$$a_0 x_i^n + a_1 x_i^{n-1} + ... + a_n = f_i, (i = 0,1,...,n).$$

Определитель этой системы

$$W = \begin{vmatrix} x_0^n & x_0^{n-1} ... & x_0 & 1 \\ x_1^n & x_1^{n-1} ... & x_1 & 1 \\ ... & ... & ... & .. \\ x_n^n & x_n^{n-1} ... & x_n & 1 \end{vmatrix} \qquad (6.2)$$

является *определителем Вандермонда*. Он отличен от нуля, если выполняется условие попарного различия его элементов, то есть $x_i \ne x_j$ при $i \ne j$. Это условие выполняется, так как $x_0 < x_1 < ... < x_n$. Следовательно, система линейных алгебраических уравнений имеет единственное решение, то есть существует единственный набор коэффициентов a_k. Коэффициенты a_k интерполяционного полинома (6.2) можно определить, решив ее, например, по *формулам Крамера*:

$a_k = \Delta_k / W$,

где Δ_k является определителем, который получен из W путем замены столбца членов, содержащих $(n-k)$-ую степень x_i $(i = 0,1,...,n)$, на столбец f_i свободных членов системы (6.2):

$$\Delta_k = \begin{vmatrix} x_0^n & ... & x_0^{n-k+1} & f_0 & x_0^{n-k-1} & ... & 1 \\ x_1^n & ... & x_1^{n-k+1} & f_1 & x_1^{n-k-1} & ... & 1 \\ ... & ... & ... & ... & ... & ... & ... \\ x_{n-1}^n & ... & x_{n-1}^{n-k+1} & f_{n-1} & x_{n-1}^{n-k-1} & ... & 1 \\ x_n^n & ... & x_n^{n-k+1} & f_n & x_n^{n-k-1} & ... & 1 \end{vmatrix}$$

Подставив полученные значения a_k в равенство (6.1), имеем новую форму представления интерполяционного полинома $P_n(f,x)$:

$$\begin{vmatrix} P_n & 1 & x & \dots & x^n \\ f_0 & 1 & x_0 & \dots & x_0{}^n \\ \dots & \dots & \dots & \dots & \dots \\ f_n & 1 & x_n & \dots & x_n^n \end{vmatrix} = 0.$$

Разложив определитель по элементам 1-го столбца, получим:

$$P_n \begin{vmatrix} 1 & x_0 & .. & x_0^n \\ 1 & x_1 & .. & x_1^n \\ .. & .. & .. & .. \\ 1 & x_n & .. & x_n^n \end{vmatrix} - f_0 \begin{vmatrix} 1 & x & .. & x^n \\ 1 & x_1 & .. & x_1^n \\ .. & .. & .. & .. \\ 1 & x_n & .. & x_n^n \end{vmatrix} + f_1 \begin{vmatrix} 1 & x_0 & .. & x^n \\ 1 & x_0 & .. & x_0^n \\ .. & .. & .. & .. \\ 1 & x_n & .. & x_n^n \end{vmatrix} - \dots - f_n \begin{vmatrix} 1 & x_0 & .. & x_0^n \\ 1 & x_1 & .. & x_1^n \\ .. & .. & .. & .. \\ 1 & x_{n-1} & .. & x_{n-1}^n \end{vmatrix} = 0.$$

Учитывая, что

$$\begin{vmatrix} 1 & x_i & \dots & x_i^n \\ 1 & x_j & \dots & x_j^n \\ \dots & \dots & \dots & \dots \\ 1 & x_k & \dots & x_k^n \end{vmatrix} = (x_j - x_i)(x_k - x_i)\dots(x_k - x_j),$$

получаем обобщенную форму интерполяционного полинома

$$P_n(x) = f_0 \frac{(x-x_1)(x-x_2)\dots(x-x_n)}{(x_0-x_1)(x_0-x_2)\dots(x_0-x_n)} + \dots + f_n \frac{(x-x_0)(x-x_1)\dots(x-x_{n-1})}{(x_n-x_0)(x_n-x_1)\dots(x_n-x_{n-1})}. \quad (6.3)$$

Получив интерполяционный полином (6.3), необходимо выяснить, насколько близко он приближается к исходной функции в других точках отрезка $[a,b]$. Как видно из рис. 6.1, между $f(x)$ и $P_n(x)$ – узловыми точками существуют различия этих функций, то есть имеет место *погрешность интерполяции*. Для ее оценки обычно строятся графики $f(x)$ и $P_n(x)$ и график их разности, то есть абсолютной погрешности.

Рассмотрим вначале случай определения погрешности в некоторой точке x^* отрезка $[a,b]$. В этом случае погрешность вычисляют как абсолютную величину разности между точным и приближенным значениями:

$$\varepsilon(x^*) = \left| f(x^*) - P_n(x^*) \right|.$$

Погрешность интерполяции на рассматриваемом отрезке принимается как максимальное отклонение полинома P_n от функции f:

$$\varepsilon(x) = \max_{[a,b]} \left| f(x) - P_n(x) \right|.$$

Для нахождения погрешности рассмотрим вспомогательную функцию $q(x)$:

$$q(x) = f(x) - P_n(x) - k\prod_{i=0}^{n}(x - x_i), \quad (6.4)$$

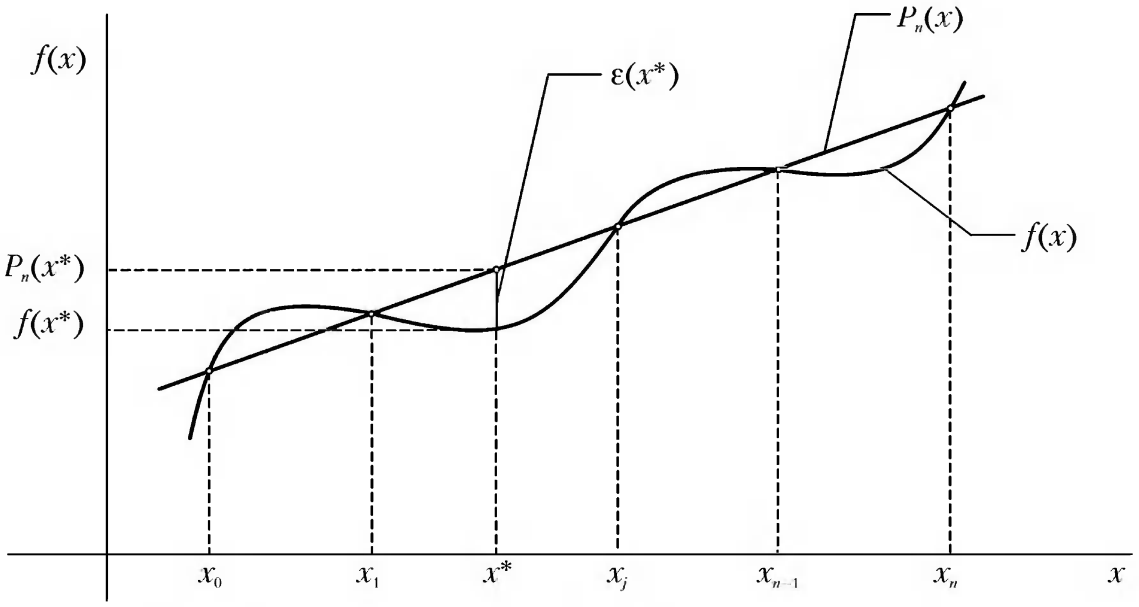


Рис. 6.1. Графическая иллюстрация интерполяции функции

где k – постоянная, значение которой выбрано таким образом, чтобы функция $q(x)$ была равной нулю при $x = x^*$, то есть

$$q(x^*) = 0 = f(x^*) - P_n(x^*) - k\prod_{i=0}^{n}(x^* - x_i).$$

Тогда имеем следующее:

$$k = \frac{f(x^*) - P_n(x^*)}{\prod_{i=0}^{n}(x^* - x)}. \qquad (6.5)$$

При таком выборе k функция q на отрезке $[a,b]$ в точках $x_0, x_1,..., x_n, x^*$ будет равна нулю $(n+2)$ раз. Тогда если использовать теорему Ролля [60], то можно утверждать, что производная q' на интервале a,b равна нулю, по крайней мере $(n+1)$ раз. Производная q'' равна нулю n раз и т. д. до производной $q^{(n+1)}$, которая обратится в нуль, по крайней мере в одной точке, принадлежащей отрезку интерполирования $[a,b]$. Обозначим эту точку через $\xi \in [a,b]$.

Продифференцируем правую и левую части соотношения (6.4) $(n+1)$ раз по x, учитывая при этом, что $x = \xi$, получаем в левой части нуль, так как $q^{(n+1)}(\xi) = 0$. Первое слагаемое правой части дает значение производной в точке ξ, равное $f^{(n+1)}(\xi)$. Второе слагаемое правой части равно нулю как производная $(n+1)$-го порядка от полинома степени не выше n. Третье слагаемое дает произведение постоянной k на полином $(n+1)$-й степени со старшим коэффициентом 1. Производная $(n+1)$-го порядка от этого полинома будет равна $(n+1)!$

Таким образом, получаем следующее выражение:

$$0 = f^{(n+1)}(\xi) - k(n+1)! \qquad (6.6)$$

Подставляя в (6.6) выражение (6.5), имеем:

$$f(x^*)-P_n(x^*)=\frac{f^{(n+1)}(\xi)}{(n+1)!}\prod_{i=0}^{n}(x-x_i). \qquad (6.7)$$

Так как точка ξ неизвестна, то используют мажорантную оценку, заменяя величины, входящие в (6.7), на их максимальные значения. Введем для определенности ограничение:

$$f^{(n+1)}(x)\le\max\left|f^{(n+1)}(\xi)\right|,\xi\in[x_0,x_n],$$

$$\omega_n(x)=\max_{[a,b]}\prod_{i=0}^{n}(x-x_i).$$

Тогда получаем, что погрешность интерполяции на рассматриваемом отрезке можно рассчитать следующим образом:

$$\varepsilon(x)\left|=\right|f(x)-P_n(x)\left|\le\frac{f^{(n+1)}(x)}{(n+1)!}\right|\omega_n(x)\left|. \qquad (6.8)\right.$$

Вопреки существующему мнению о быстрой потери точности полиномиальной аппроксимации при $n > (5–7)$ теоретическая погрешность ее быстро уменьшается при увеличении n. Но это только при условии, что все вычисления выполняются точно!

6.1.3. Полиномиальная аппроксимация табличных зависимостей

Табличный способ задания функций широко распространен в технике, физике, экономике, естествознании и других областях науки и техники. Чаще всего задание функциональных зависимостей в виде таблиц используется в ходе проведения экспериментов. Недостаток табличного способа задания функции состоит в том, что функция представлена дискретными данными, и всегда найдутся такие значения независимой переменной, которых нет в таблице. Поэтому выполняют замену функции $f(x)$ ее аналитическим выражением для нахождения промежуточных значений. Таким образом, применяют интерполяцию или аппроксимацию.

Кроме того, при экспериментальных исследованиях возможны сложности при снятии их значений, а аналитическое выражение зависимости неизвестно.

Задача интерполяции не является новой, и в математической литературе классические методы изложены достаточно подробно [1, 125, 189]. Ниже мы кратко рассмотрим их основы.

6.1.4. Интерполяционный метод Лагранжа

Метод Лагранжа предполагает введение вспомогательного полинома $l_i(x)$ степени n. Полином $l_i(x)$ в точке x_i должен быть равен 1, а в остальных точках отрезка интерполяции должен обращаться в нуль, то есть должны выполняться следующие условия:

$$l_i(x_i)=1 \quad (i=0,1,...,n), \tag{6.9}$$

$$l_i(x_k)=0 \quad (i\neq k; i,k=0,1,...,n). \tag{6.10}$$

В силу условия (6.10) и требования, чтобы полином $l_i(x)$ был степени n, получаем:

$$l_i(x)=c_i(x-x_0)...(x-x_{i-1})(x-x_{i+1})...(x-x_n). \tag{6.11}$$

Используя требование (6.9), найдем значение постоянной c_i с помощью выражения

$$c_i(x_i-x_0)...(x_i-x_{i-1})(x_i-x_{i+1})...(x_i-x_n)=1,$$

или

$$c_i=\frac{1}{(x_i-x_0)...(x_i-x_{i-1})(x_i-x_{i+1})...(x_i-x_n)}. \tag{6.12}$$

Подставляя (6.12) в (6.11), можно записать выражение для $l_i(x)$ в явном виде:

$$l_i(x)=\frac{(x-x_0)...(x-x_{i-1})(x-x_{i+1})...(x-x_n)}{(x_i-x_0)...(x_i-x_{i-1})(x_i-x_{i+1})...(x_i-x_n)}.$$

Затем найдем алгебраическую сумму произведения f и $l_i(x)$:

$$L_n(x)=\sum_{i=0}^{n} f_i l_i(x).$$

Данное выражение является полиномом степени не выше n, в точках x_i принимает значение f_i, так как соответствующее слагаемое суммы $f_i l_i(x_i)$ равно f_i, а остальные слагаемые $f_j l_j(x_i)$ обращаются в нуль.

Интерполяционный полиномом Лагранжа обычно записывается в следующем конечном виде:

$$L_n(x)=\sum_{i=0}^{n} f_i \frac{(x-x_0)...(x-x_{i-1})(x-x_{i+1})...(x-x_n)}{(x_i-x_0)...(x_i-x_{i-1})(x_i-x_{i+1})...(x_i-x_n)}. \tag{6.13}$$

Важным достоинством этой формы записи интерполяционного полинома является то, что число арифметических операций, необходимых для построения полинома Лагранжа, пропорционально n^2 и является наименьшим для всех форм записи. Это минимизирует затраты времени на вычисления. (6.13) применимо как для равноотстоящих, так и для неравномерно отстоящих узлов. Достоинством является и то, что интерполяционный полином Лагранжа удобен, когда значения функций меняются, а узлы интерполяции неизменны, что имеет место во многих экспериментальных исследованиях.

Выражение (6.13) можно записать в более компактной форме. Для этого нужно ввести вспомогательную функцию

$$\omega_n(x)=(x-x_0)(x-x_1)...(x-x_n).$$

Отметим, что эта функция является полиномом степени n. Ее производная в точке x_i имеет следующий вид:

$$\omega'_n(x_i)=(x_i-x_0)...(x_i-x_{i-1})(x_i-x_{i+1})...(x_i-x_n).$$

Можно заметить, что знаменатель функции (6.13) равен $\omega'_n(x_i)$. Тогда, умножив и разделив каждое из слагаемых выражения (6.13) на $(x-x_i)$, заменив выражение в числителе функцией $\omega_n(x)$, получим формулу интерполяционного полинома Лагранжа в компактной форме:

$$L_n(x)=\sum_{i=0}^{n} f_i \frac{\omega_n(x)}{(x-x_i)\omega'_n(x_i)}. \qquad (6.14)$$

Аппроксимация с помощью интерполяционного полинома Лагранжа является достаточно эффективной, когда интерполируются гладкие функции и число n является малым.

6.1.5. Интерполяционный метод Ньютона

На практике для повышения точности интерполяционного полинома незначительно увеличивают количество узлов интерполяции. В этом случае использование метода Лагранжа неудобно, так как добавление дополнительных узлов приводит к необходимости пересчета всего интерполяционного полинома в целом. Эти недостатки устраняются, если записать полином Лагранжа, используя *интерполяционный метод Ньютона*.

Для этого вводится понятие *разделенной разности*. Разделенные разности нулевого порядка совпадают со значениями функции в узлах интерполяции. Разделенные разности первого порядка обозначаются $f(x_i,x_j)$ и определяются через разделенные разности нулевого порядка:

$$f(x_i,x_j)=\frac{f_i-f_j}{x_i-x_j},$$

разделенные разности второго порядка определяются через разделенные разности первого порядка:

$$f(x_i,x_j,x_k)=\frac{f(x_i,x_j)-f(x_j,x_k)}{(x_i-x_k)}.$$

Разделенная разность порядка n определяется соотношением

$$f(x_i,x_j,x_k,...,x_{n-1},x_n)=\frac{f(x_i,x_j,x_k,...,x_{n-1})-f(x_j,x_k,...,x_n)}{x_i-x_n}.$$

Для $(n+1)$-ой точки могут быть построены разделенные разности n-го порядка. Разделенные разности более высоких порядков равны нулю. Между разделенными разностями и производными соответствующих порядков существует соотношение

$$f^n(x)=n!f(x_0,x_1,...,x_n).$$

Разделенная разность $(n+1)$-го порядка от многочлена n-ой степени равна нулю. Для интерполяции с постоянным шагом h вводится понятие конечные разности $\Delta^n f$, связанные с разделенными следующим соотношением:

$$\Delta^n f=h^n n!f(x_0,x_1,...,x_n).$$

Пусть аппроксимируемая функция $f(x)$ – полином n-ой степени. Выбираем точки $x_0, x_1,...,x_n$ в качестве узлов интерполяции и найдем такой интерполяционный полином, значение которого в узлах совпадает со значениями функции $f(x)$.

Запишем, выбирая в качестве узла точку с координатой x, очевидные соотношения:

$$\begin{aligned} f(x)-f(x_0)&=(x-x_0)f(x,x_0),\\ f(x)&=f(x_0)+(x-x_0)f(x,x_0). \end{aligned} \tag{6.15}$$

Также имеем

$$\begin{aligned} f(x,x_0)-f(x_0,x_1)&=(x-x_1)f(x,x_0,x_1),\\ f(x,x_0)&=f(x_0,x_1)+(x-x_1)f(x,x_0,x_1). \end{aligned} \tag{6.16}$$

Подставляя (6.16) в (6.15), получаем:

$$f(x)=f(x_0)+(x-x_0)f(x_1,x_0)+(x-x_0)(x-x_1)f(x,x_0,x_1). \tag{6.17}$$

Аналогичную процедуру продолжаем далее:

$$\begin{aligned} f(x,x_0,x_1)-f(x_0,x_1,x_2)&=(x-x_2)f(x,x_0,x_1,x_2),\\ f(x,x_0,x_1)&=f(x_0,x_1,x_2)+(x-x_2)f(x,x_0,x_1,x_2). \end{aligned} \tag{6.18}$$

Подставляя (6.18) в (6.17), получаем:

$$\begin{aligned} f(x)=f(x_0)+(x-x_0)f(x_1,x_0)+(x-x_0)(x-x_1)f(x_0,x_1,x_2)+\\ +(x-x_0)(x-x_1)(x-x_2)f(x,x_0,x_1,x_2). \end{aligned}$$

В итоге получаем выражение:

$$\begin{aligned} N_n(x)=f(x_0)+(x-x_0)f(x_1,x_0)+(x-x_0)(x-x_1)f(x_0,x_1,x_2)+...+\\ +(x-x_0)(x-x_1)...(x-x_n)f(x,x_0,x_1,...,x_n). \end{aligned} \tag{6.19}$$

Следующий член в этом ряду будет равен нулю, так как он будет содержать разделенную разность $(n+1)$-го порядка. В силу условия единственности интерполяционного полинома $N_n(x)$ является интерполяционным полиномом n-ой степени с узлами $x_0, x_1, ..., x_n$. Выражение (6.19) и является *интерполяционным полиномом Ньютона*. Представление интерполяционного полинома в форме Ньютона является более удобным в практических расчетах. Добавление новых узлов интерполяции приводит лишь к появлению новых слагаемых полинома, без изменения уже существующих, что не требует пересчета всех коэффициентов заново. При добавлении новых узлов интерполяции не важно, в каком порядке они подключаются, но существует одно условие – узлы x_i не должны совпадать.

6.1.6. Итерационно-интерполяционный метод Эйткена

Итерационно-интерполяционный метод Эйткена позволяет свести вычисления коэффициентов интерполяционного полинома Лагранжа (6.14) с учетом его равенства в узлах интерполяции с исходными данными к вычислению функцио-

нальных определителей второго порядка. При этом эффективность метода повышается в тех случаях, когда нет необходимости в получении приближенного аналитического выражения функции $f(x)$, заданной таблично, а требуется лишь определить значение в некоторой точке x^*, отличной от узловых точек. Этот метод заключается в последовательной линейной интерполяции. Процесс вычисления $f(x^*)$ состоит в следующем: необходимо пронумеровать узлы интерполяции, например в порядке убывания их от x^*. Затем для каждой узловой точки интерполяции строятся соотношения:

$$P_0^k = f(x_k);$$

$$P_1^{ij}(x) = fi\frac{x - x_j}{x_i - x_j} + f_j\frac{x - x_i}{x_j - x_i} = \frac{1}{x_j - x_i}\begin{vmatrix} x - x_i & P_0^i \\ x - x_j & P_0^j \end{vmatrix};$$

$$P_2^{ijk}(x) = \frac{1}{x_k - x_i}\begin{vmatrix} x - x_i & P_1^{ij}(x) \\ x - x_k & P_1^{jk}(x) \end{vmatrix},$$

которые являются интерполяционными полиномами, построенными соответственно по узлам x_i, x_j, x_k. Продолжая этот процесс, имеем следующий полином:

$$P_n^{ij...km}(x) = \frac{1}{x_m - x_i}\begin{vmatrix} x - x_i & P_{n-1}^{ij...k}(x) \\ x - x_m & P_{n-1}^{j...km}(x) \end{vmatrix}. \qquad (6.20)$$

Полученный полином является интерполяционным полиномом, построенным по узлам x_i, x_j, ..., x_k, x_m. Это утверждение верное, так как $P_{n-1}{}^{ij..k}(x)$ и $P_{n-1}{}^{j...km}(x)$ являются интерполяционными полиномами.

Полином $P_n{}^{ij...km}(x)$ – полином степени не выше n, что очевидно из соотношения (6.20). Во всех узлах интерполяции полином $P_n{}^{ij...km}(x)$ принимает соответствующие значения:

$$P_n^{ij...km}(x_i) = \frac{-(x_i - x_m)f_i}{x_m - x_i} = f_i, \text{ при } x_p = x_i,$$

$$P_n^{ij...}(x_m) = \frac{(x_m - x_i)f_m}{x_m - x_i} = f_m, \text{ при } x_p = x_m,$$

$$P_n^{ij...km}(x_p) = \frac{1}{x_m - x_i}((x_p - x_i)f_p - (x_p - x_m)f_p) = f_p.$$

Вычисляя по формуле (6.20) значения $P_n{}^{01...n}(x^*)$, принимают их за последовательные приближения $f(x^*)$. Процесс вычисления можно закончить, когда абсолютная величина разности двух последовательных приближений $|P_{n-1}{}^{i..m}(x) - P_n{}^{i..m}(x)| = \varepsilon_n$ становится достаточно малой, то есть при некотором значении n $\varepsilon_n \le E$ требуемой погрешности.

Данный метод часто используется на практике. При его реализации предполагается, что функция гладкая, а также критерием оценки погрешности определяется некоторое значение, определяемое условиями конкретной задачи.

6.1.7. Интерполяция по методу Чебышева

При равномерном расположении узлов интерполяции погрешность ее распределена в заданном интервале изменения x крайне неравномерно. Чаще всего наблюдается выбег погрешности на краях интервала интерполяции. Поэтому важен иной выбор узлов интерполяции, дающий уменьшение ее погрешности.

Метод интерполяции Чебышева был создан для оптимального выбора узлов интерполяции, если это возможно при решении конкретной задачи, и для получения минимально возможной погрешности аппроксимации. Для рассмотрения оценки погрешности интерполяции на рассматриваемом отрезке имеется соотношение (6.8).

Предполагается, что в выборе расположения узлов интерполяции ограничений нет и что узлы выбираются произвольно. Ставится задача о наилучшем выборе узлов. Наилучшими узлами x_i следует признать те, для которых выражение $\max_{[a,b]}|\omega_n(x)|$ минимально для рассматриваемого класса функций (алгебраических полиномов). Определение этих узлов сводится к нахождению корней полинома, наименее уклоняющихся от нуля на $[a,b]$. Такой полином порождается полиномом Чебышева первого рода T_{n+1}, общий вид которого:

$$T_{n=1}(x) = 2^n x^{n+1} + \dots .$$

Предварительно рассматривается полином

$$T_{n+1}(x) = 2^{-n} T_{n+1}(x) = x^{n+1} + \dots$$

на отрезке $[-1,1]$, который называется полиномом, наименее уклоняющимся от нуля. Это определение подтверждается следующим доказательством.

Если $P_{n+1}(x)$ – полином степени $n+1$ со старшим коэффициентом 1, то

$$\max_{[-1,1]}\left|P_{n+1}(x)\right| \geq \max_{[-1,1]}\left|\overline{T}_{n+1}(x)\right| = 2^{-n}.$$

Для доказательства этого предполагают, что неравенство не выполняется. Тогда полином

$$[T_{n+1}(x) - P_{n+1}(x)],$$

имеющий степень n, во всех $(n+2)$ экстремальных точках x_m полинома T_{n+1} совпадал бы с ним по знаку и, следовательно, поочередно принимал бы в этих точках то положительное, то отрицательное значение. Поэтому полином

$$[T_{n+1}(x) - P_{n+1}(x)]$$

должен иметь $(n+1)$ различных корней, что невозможно для полинома степени не выше n.

Все рассуждения проводились для отрезка $[-1,1]$, который линейной заменой переменных

$$x' = {(b+a)}/{2} + {(b-a)x}/{2}$$

можно перевести в заданный отрезок $[a,b]$. Полином $T_{n+1}(x)$ преобразуется в $\overline{T}_{n+1}\left(\frac{2x-(b+a)}{b-a}\right)$ со старшим коэффициентом $\left(\frac{2}{b-a}\right)^{n+1}$. Следовательно,

$$\bar{T}_{n+1}^{[a,b]}(x) = (b-a)^{n+1} 2^{-2n-1} T_{n+1}\left(\frac{2x-(b+a)}{b-a}\right)$$

является полиномом со старшим коэффициентом 1, наименее уклоняющимся от нуля на отрезке $[a,b]$. Это означает, что для любого полинома $P_{n+1}(x)$ степени $(n+1)$ со старшим коэффициентом 1 справедливо неравенство:

$$\max_{[a,b]}\left|P_{n+1}(x)\right| \geq \max_{[a,b]}\left|\bar{T}_{n+1}^{[a,b]}(x)\right| = 2^{-n}\left(\frac{b-a}{2}\right)^{n+1}.$$

Корнями полинома $T_{n+1}^{[a,b]}(x)$ являются:

$$x_k = \frac{b+a}{2} + \frac{b-a}{2}\cos\left(\frac{\pi(2k+1)}{2n+1}\right), \quad k = 0,\ldots,n. \tag{6.21}$$

Для уменьшения погрешности необходимо минимизировать величину $\max_{[a,b]}|\omega_n(x)|$, которая является полиномом $(n+1)$-й степени со старшим коэффициентом 1. В качестве узлов интерполяции следует выбрать x_k, которые определяются соотношением (6.21). При этом

$$\max_{[a,b]}\left|\omega_n(x)\right| = \max_{[a,b]}\left|\bar{T}_{n+1}^{[a,b]}(x))\right| = 2^{-n}\left(\frac{b-a}{2}\right)^{n+1}.$$

Тогда оценка погрешности

$$\varepsilon \leq \frac{f^{n+1}(x)}{(n+1)!} 2^{-n}\left(\frac{b-a}{2}\right)^{n+1}.$$

Эта оценка является наилучшей, так как здесь знак ≤ меняется на знак =, если в качестве $f(x)$ выбрать следующий полином степени $n+1$:

$$f(x) = \frac{f^{n+1}(x)}{(n+1)!} x^{n+1} + a_n x^n + \ldots$$

в качестве узлов – точки x_k, определяющиеся по соотношению (6.21).

6.1.8. Сплайновая интерполяция, экстраполяция и аппроксимация

Использование одной интерполяционной формулы для большого числа узлов нецелесообразно, так как при этом интерполяционный полином имеет высокий порядок и сильно проявляет свои колебательные свойства. Одна из возможностей преодоления этого недостатка заключается в применении *сплайн-интерполяции*. Идея сплайновой интерполяции состоит в определении интерполирующей функции по формулам одного типа для различных подмножеств данных и в стыковке значений функции и ее производных на границе подмножеств. Таким образом, сплайновая интерполяция является многоинтервальной.

Наиболее известным и широко применяемым является случай, когда между двумя точками строится полином n-ой степени

$$S(x)=\sum_{k=0}^{n} a_{ik}x^k, \text{ где } x_{i-1}\le x\le x_i,$$

который в узлах интерполяции принимает значения интерполируемой функции и непрерывен вместе со своими $(n-1)$-ми производными. Такой кусочно-непрерывный интерполяционный полином называется *сплайном*. Его коэффициенты находят из условий в узлах интерполяции – равенства значений сплайна и приближаемой функции, а также равенства $(n-1)$-ой производной соответствующих полиномов. Максимальная по всем частичным отрезкам степень полинома является степенью сплайна. На рис. 6.2 показано приближение таблично заданной функциональной зависимости $f(x)$ сплайнами первого порядка (кусочно-линейное приближение) и второго порядка.

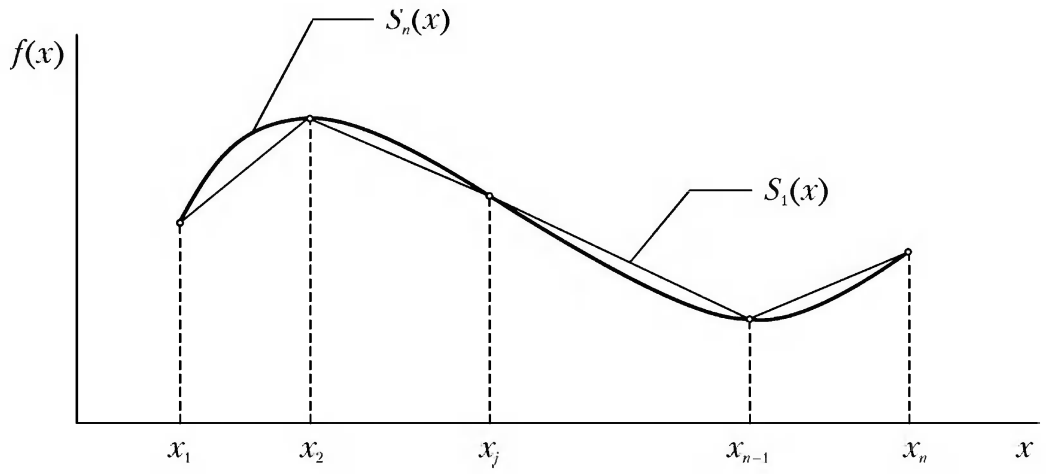


Рис. 6.2. Приближение таблично заданной f(x) сплайнами первого и второго порядков

Одним из наиболее распространенных интерполяционных сплайнов является *кубический сплайн*. Для вывода уравнения кубического интерполяционного сплайна можно воспользоваться его представлением в виде гибкой линейки, изогнутой таким образом, что она проходит через значения функции в узлах, то есть является упругой рейкой в состоянии равновесия. Это его состояние описывается уравнением $S''''(x) = 0$, где $S''''(x)$ – четвертая производная. Из этого следует, что между каждой парой соседних узлов интерполяционная формула записывается в виде полинома третьей степени. Этот полином удобно представить следующим образом:

$$S(x)=a_i+b_i(x-x_{i-1})+c_i(x-x_{i-1})^2+d_i(x-x_{i-1})^3,$$
$$x_{i-1}\le x\le x_i, i=1,2,...,n.$$

Для построения кубического сплайна необходимо построить n полиномов третьей степени, то есть определить $4n$ неизвестных a_i, b_i, c_i, d_i. Эти коэффициенты

ищутся из условия равенства в узлах. В узлах сплайн должен принимать табличные значения функции:

$$S(x_{i-1}) = a_i = f_{i-1}, \tag{6.22}$$

$$S(x_i) = a_i + b_i h_i + c_i h_i^2 + d_i h_i^3 = f_i, \text{ при } h_i = x_i - x_{i-1}. \tag{6.23}$$

Соотношения (6.22) и (6.23) содержат $2n$ уравнений. Дополнительные условия можно получить, если потребовать непрерывности первой и второй производных функции $S(x)$ во внутренних узлах:

$$S'(x) = b_i + 2c_i(x - x_{i-1}) + 3d_i(x - x_{x-1})^2,$$

$$S''(x) = 2c_i + 6d_i(x - x_{i-1}) \text{ при } x_{i-1} \le x \le x_i, i = 1,...,n-1.$$

Получим еще $2(n-1)$ уравнений:

$$b_{i+1} = b_i + 2c_i h_i + 3d_i h_i^2, \tag{6.24}$$

$$c_{i+1} = c_i + 3d_i h_i. \tag{6.25}$$

Недостающие уравнения можно получить из граничных условий, предполагая нулевую кривизну сплайна на концах отрезка. Приравнивая значения второй производной в точках x_0 и x_n к нулю, получим два недостающих уравнения:

$$S''(x_0) = c_i = 0, \tag{6.26}$$

$$S''(x_n) = c_n + 3d_n h_n = 0. \tag{6.27}$$

Заметим, что граничные условия (6.26) и (6.27) дают погрешность $O(h^2)$ вблизи границы. Поэтому вместо них можно рекомендовать условие непрерывности $S'''(x)$ в двух внутренних узлах, которые близки к граничным. Если существуют другие условия поведения функции в точках x_0 и x_n, то уравнения (6.26) и (6.27) будут другими. Таким образом, система (6.22)–(6.27) для отыскания $4n$ неизвестных a_i, b_i, c_i, d_i, $i = 1,2,...,n$, замкнута. Ее удобно решать, проведя предварительно следующие преобразования. Коэффициент a_i сразу получается из решения уравнения (6.22). Из уравнения (6.25) и (6.27) следует:

$$d_i = \frac{(c_{i+1} - c_i)}{3h_i}, \quad \text{при} \quad i = 1,2,...,n, \tag{6.28}$$

$$d_n = \frac{-c_n}{3h_n}. \tag{6.28а}$$

Подставляя выражение (6.28) и (6.28а) в (6.23) и заменяя в нем a_i на f_{i-1}, получаем

$$b_i = \frac{f_i - f_{i-1}}{h_i} - \frac{h_i(c_{i+1} + 2c_i)}{3}, \tag{6.29}$$

$$b_n = \frac{f_n - f_{n-1}}{h_n} - \frac{2h_n c_n}{3}. \tag{6.29а}$$

Из (6.24) и (6.25) имеем

$$b_{i+1} = b_i + h_i(c_i + c_{i+1}), \tag{6.29б}$$

или

$$b_i = b_{i-1} + h_{i-1}(c_{i-1} + c_i) \tag{6.29в}$$

Полагая в (6.29) $i=n$ и сравнивая с (6.29а), имеем $c_{n+1} = 0$.

Подставляя теперь (6.29) и (6.29а) в (6.29в), получаем систему уравнений относительно c_i, имеющую следующий вид:

$$\begin{cases} c_1 = 0 \\ h_{i-1}c_{i-1} + 2(h_{i-1} + h_i) + h_i c_{i+1} = 3\left[\dfrac{f_i - f_{i-1}}{h_i} - \dfrac{f_{i-1} - f_{i-2}}{h_{i-1}}\right] \text{ где } i = 2,...,n. \\ c_n = 0 \end{cases} \quad (6.30)$$

Матрица системы (6.30) является трехдиагональной, ее решение находится методом прогонки. Решив ее, находим затем коэффициенты b_i из уравнений (6.29), (6.29а) и d_i – из уравнений (6.28), (6.28а).

Метод сплайновой интерполяции дает хорошие результаты при интерполяции непрерывных функций с гладкими производными 1-ой и 2-ой степеней. При этом кубическая сплайновая интерполяция, построенная по узлам $f_i = f(x_i)$, $i = 0,1,...,n$, будет иметь минимум кривизны по сравнению с любой интерполяционной функцией, имеющей непрерывные первую и вторую производные. Результат сплайн-интерполяции функций с резким изменением производных имеет, как правило, большие ошибки. Сплайны более высоких порядков, чем третий, используются редко, так как при вычислении большого числа коэффициентов может накапливаться ошибка, приводящая к значительным погрешностям.

6.1.9. Рациональная интерполяция и аппроксимация

Большую точность приближения по сравнению с полиномиальным приближением можно получить, если исходную функцию заменить, используя рациональную интерполяцию, при которой аппроксимирующая функция ищется как отношение двух полиномов. Наиболее важным свойством рациональных функций является то, что ими можно приближать такие функции, которые принимают бесконечные значения для конечных значений аргумента и даже внутри интервала его изменения.

Итак, при задании $f(x_1)$, ..., $f(x_n)$ приближение к $f(x)$ ищется в виде

$$R(x) = \frac{a_0 + a_1 x + ... + a_p x^p}{b_0 + b_1 x + ... + b_q x^q}, \text{ где } p + q + 1 = n. \quad (2.31)$$

Коэффициенты a_i, b_i находятся из совокупности соотношений $R(x_j) = f(x_j)$ ($j = 1,...,n$), которые можно записать в виде

$$\sum_{j=0}^{p} a_j x_i^j - f(x_i)\sum_{j=0}^{q} b_j x_i^j = 0, \quad j = 1,...n.$$

Данное уравнение образует систему n линейных уравнений относительно $(n+1)$ неизвестных. Такая система всегда имеет нетривиальное решение.

Функция $R(x)$ может быть записана в явном виде в случае n нечетного, если $p = q$, и n четного, если $p - q = 1$.

Для записи функции $R(x)$ в явном виде следует вычислять так называемые обратные разделенные разности, определяемые условиями

$$f^{-}(x_l, x_k) = \frac{x_l - x_k}{f(x_l) - f(x_k)}$$

и рекуррентным соотношением

$$f^{-}(x_k, ..., x_l) = \frac{x_l - x_k}{f^{-}(x_{k+1}, ..., x_l) - f^{-}(x_k, ..., x_{l-1})}.$$

Интерполирование функций рациональными выражениями обычно рассматривают на основе аппарата цепных дробей. Тогда интерполирующая рациональная функция записывается в виде цепной дроби

$$f(x) = f(x_1) + \cfrac{x - x_1}{f^{-}(x_1, x_2) + \cfrac{x - x_2}{f^{-}(x_1, x_2, x_3) + ... + \cfrac{x - x_{n-1}}{f^{-}(x_1, ..., x_n)}}}.$$

Использование рациональной интерполяции часто целесообразнее интерполяции полиномами в случае функций с резкими изменениями характера поведения или особенностями производных в точках.

6.1.10. Метод наименьших квадратов (МНК)

При обработке экспериментальных данных, полученных с некоторой погрешностью, интерполяция описанными выше методами становится неразумной. В этом случае целесообразно строить приближающую функцию таким образом, чтобы сгладить влияние погрешности измерения и числа точек эксперимента [29]. Такое сглаживание реализуется при построении приближающей функции по *методу наименьших квадратов*, при котором кривая интерполирующей (аппроксимирующей) функции проходит в облаке исходных точек, отклоняясь от них с минимальной среднеквадратической погрешностью. Этот вид приближения называют также *регрессией*.

Рассмотрим совокупность значений таблично заданной функции f_i в узлах x_i при $i = 0,1,...,n$. Предположим, что приближающаяся функция $F(x)$ в точках $x_1, x_2, ..., x_n$ имеет значения $\overline{f}_1, \overline{f}_2, ..., \overline{f}_n$. Будем рассматривать совокупность значений функции $f(x)$ и функции $F(x)$ как координаты двух точек n-мерного пространства. С учетом этого задача приближения функции может быть определена другим образом: найти такую функцию $F(x)$ заданного вида, чтобы расстояние между точками $M(f_1, f_2, ..., f_n)$ и $\overline{M}(\overline{f}_1, \overline{f}_2, ..., \overline{f}_n)$ было наименьшим. Воспользовавшись метрикой евклидова пространства, приходим к требованию, чтобы величина

$$\sqrt{(f_1 - \overline{f}_1)^2 + (f_2 - \overline{f}_2)^2 + ... + (f_n - \overline{f}_n)^2}$$

была наименьшей, что соответствует следующему:

$$(f_1 - \bar{f}_1)^2 + (f_2 - \bar{f}_2)^2 + \ldots + (f_n - \bar{f}_n)^2, \qquad (6.32)$$

то есть сумма квадратов должна быть наименьшей. Задачу приближения функции $f(x)$ теперь можно формулировать иначе. Для функции $f(x)$, заданной таблично, необходимо найти функцию $F(x)$ определенного вида так, чтобы сумма квадратов (6.32) была наименьшей. Это хорошо иллюстрирует рис. 6.3, полученный с помощью средств пакета расширения Student в подразделе LinearAlgebra. На рисунке у каждой точки построен квадрат со стороной, равной отклонению точки от кривой. Суть метода наименьших квадратов заключается в таком построении приближающей кривой, при котором сумма площадей квадратов сверху и снизу от кривой будет равна 0.

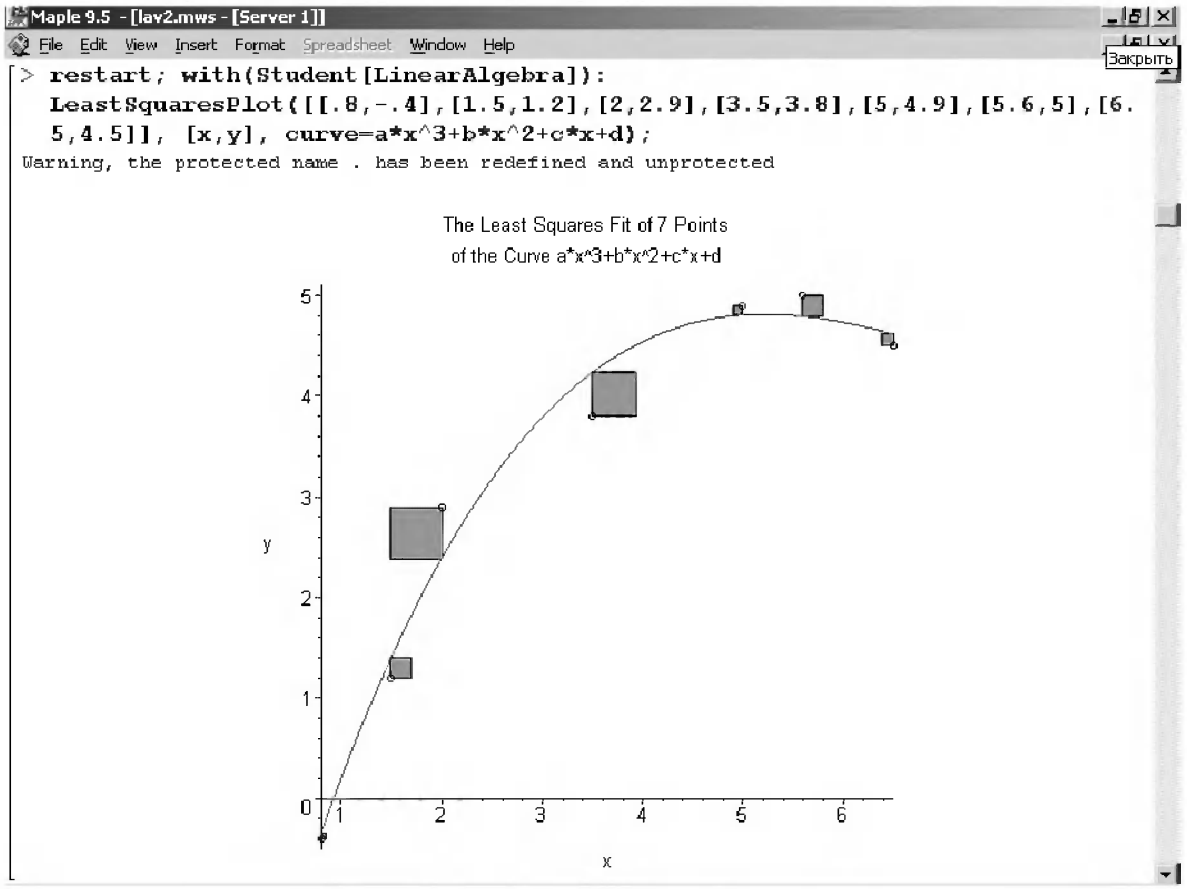


Рис. 6.3. Графическая иллюстрация приближения таблично заданных данных по методу наименьших квадратов

Выбор класса приближающихся функций определяется характером поведения графика функции f. Это могут быть линейная зависимость, любые элементарные функции, выражения с ними и т. д. Практически вид приближающей функции F можно определить, построив точки $f(x)$, а затем построить плавную кривую, по возможности наилучшим образом отражающую характер расположения точек. По полученной кривой выбирают вид приближающей функции.

Когда вид приближающей функции выбран, то последующая задача сводится к отысканию значений параметров функции. Рассмотрим метод нахождения параметров приближающей функции в общем виде на примере приближающейся функции с тремя параметрами $f = F(x,a,b,c)$. Тогда имеем

$$F(x_i,a,b,c) = f_i, i = 1,2,...,n. \tag{6.33}$$

Сумма квадратов разностей соответствующих значений функций f и F будет иметь вид:

$$\sum_{i=1}^{n}[f_i - F(x_i,a,b,c,)]^2 = \phi(a,b,c). \tag{6.34}$$

Сумма (6.34) является функцией $\phi(a,b,c)$ трех переменных a, b, c. Задача сводится к отысканию ее минимума. Для этого используем необходимое условие экстремума:

$$\frac{\partial\phi}{\partial a} = 0, \frac{\partial\phi}{\partial b} = 0, \frac{\partial\phi}{\partial c} = 0,$$

или

$$\begin{aligned}
&\sum_{i=1}^{n}[f_i - F(x_i,a,b,c,)]\cdot F'_a(x_i,a,b,c) = 0,\\
&\sum_{i=1}^{n}[f_i - F(x_i,a,b,c,)]\cdot F'_b(x_i,a,b,c) = 0,\\
&\sum_{i=1}^{n}[f_i - F(x_i,a,b,c,)]\cdot F'_c(x_i,a,b,c) = 0.
\end{aligned} \tag{6.35}$$

Решив эту систему из трех уравнений (6.35) с тремя неизвестными относительно параметров a, b, c, получим конкретный вид искомой функции $F(x,a,b,c)$. Изменение количества параметров не приведет к изменению сущности метода, а отразится только на количестве уравнений в системе (6.35).

Как следует из начальных условий, найденные значения функции $F(x,a,b,c)$ в точках $x_1, x_2, ..., x_n$ будут отличаться от табличных значений $y_1, y_2, ..., y_n$. Значение разностей

$$f_i - F(x_i,a,b,c,) = \varepsilon_i, i = 1,2,...,n$$

будет определять отклонение измеренных значений f от вычисленных по формуле (6.34). Для найденной эмпирической формулы (6.34) в соответствии с исходными табличными данными можно найти сумму квадратов отклонений

$$\sigma = \sum_{i=1}^{n}\varepsilon_i^{\,2}. \tag{6.36}$$

Она, в соответствии с принципом наименьших квадратов для заданного вида приближающей функции и ее найденных параметров (параметры a,b,c), должна быть наименьшей. Из двух разных приближений одной и той же табличной функции, следуя принципу наименьших квадратов, лучшим нужно считать тот, для которого сумма (6.36) имеет меньшее значение.

6.1.11. Тригонометрическая интерполяция рядами Фурье

Функциональный ряд для интерполяции вида

$$\frac{a_0}{2}+\sum_{n=1}^{\infty}(a_n\cos(nx)+b_n\sin(nx)) \tag{6.37}$$

называется *тригонометрическим рядом*. Его коэффициенты a_n и b_n – действительные числа, не зависящие от x. Если этот ряд сходится для любого x из промежутка $[-\pi,\pi]$, тогда он определяет периодическую функцию $f(x)$ с периодом $T=2\pi$. Ряд вида (2.37) называется *рядом Фурье* для интегрируемой на отрезке $[-\pi,\pi]$ функции $f(x)$, если коэффициенты его вычисляются по следующим правилам [73]:

$$a_0=\frac{1}{\pi}\int_{-\pi}^{\pi}f(x)dx, \tag{6.38}$$

$$a_n=\frac{1}{\pi}\int_{-\pi}^{\pi}f(x)\cos(nx)dx,\quad (n=1,2,...,), \tag{6.39}$$

$$b_n=\frac{1}{\pi}\int_{-\pi}^{\pi}f(x)\sin(nx)dx,\quad (n=1,2,...,). \tag{6.40}$$

В практических расчетах, как правило, ограничиваются конечным числом первых членов ряда Фурье. В результате получается приближенное аналитическое выражение для функции $f(x)$ в виде тригонометрического полинома N-го порядка

$$Q_N(x)=\frac{a_0}{2}+\sum_{n=1}^{N}(a_n\cos(nx)+b_n\sin(nx)),\qquad -\pi\le x\le\pi.$$

Но соотношения для вычисления коэффициентов Фурье (6.38)–(6.40) пригодны для случая аналитического задания исходной функции. Если функция задана в виде таблицы, то возникает задача приближенного отыскания коэффициентов Фурье по конечному числу имеющихся значений функции [74].

Таким образом, формулируется следующая задача практического, гармонического анализа: аппроксимировать на интервале $(0,T)$ тригонометрический полином N-го порядка функцию $y = f(x)$, для которой известны m ее значений $y_k = f(x_k)$ при $x_k = kT/m$, где $k = 0,1,2,...,m-1$.

Тригонометрический полином для функции, определенной на интервале $(0,T)$, имеет вид:

$$Q_n(x)=\frac{a_0}{2}+\sum_{n=1}^{N}\left(a_n\cos\left(\frac{2\pi}{T}x\right)+b_n\sin\left(n\frac{2\pi}{T}x\right)\right). \tag{6.41}$$

Коэффициенты a_n и b_n определяются следующими соотношениями:

$$a_n=\frac{2}{T}\int_{0}^{T}f(x)\cos\left(n\frac{2\pi}{T}x\right)dx, \tag{6.42}$$

$$b_n = \frac{2}{T}\int_0^T f(x)\sin\left(n\frac{2\pi}{T}x\right)dx, \quad n = 0,1,2,...,N. \tag{6.43}$$

Применяя в соотношениях (6.42)–(6.43) формулу прямоугольников для вычисления интегралов по значениям подынтегральных выражений в точках $x_k = kT/m$, где $k = 0,1,2,...,m-1$, имеем:

$$a_n = \frac{2}{m}\sum_{k=0}^{m-1} y_k \cos\left(n\frac{2\pi k}{m}\right), \tag{6.44}$$

$$b_n = \frac{2}{m}\sum_{k=0}^{m-1} y_k \sin\left(n\frac{2\pi k}{m}\right), \quad n = 0,1,2,...,N. \tag{6.45}$$

Таким образом, тригонометрический полином (6.41), коэффициенты a_n и b_n находятся по формулам (6.44)–(6.45), служит решением поставленной задачи. При этом коэффициенты (6.44)–(6.45) минимизируют сумму квадратов отклонений

$$\delta_N^2 = \sum_{k=0}^{m-1}[Q_n(x_k) - y_k]^2. \tag{6.46}$$

В случае, когда $m = 2N$, коэффициенты a_n и b_n для $n = 0,1,2,...N$ определяются соотношениями (2.44)–(2.45), а коэффициент a_N – соотношением

$$a_n = \frac{1}{m}\sum_{k=0}^{m-1}(-1)^k y_k. \tag{6.47}$$

Сам же полином $Q_N(x)$ становится интерполяционным полиномом, так как в этом случае при любом b_N выполняется соотношение $Q_N(x_k) = y_k$ для всех $x_k = kT/m$, где $k = 0,1,2,...,m-1$.

6.1.12. Паде-аппроксимация

Паде-аппроксимация использует для приближения *рациональную функцию* в виде отношения двух полиномов. Коэффициенты этих полиномов определяются коэффициентами разложения функции в ряд Тейлора.

Пусть имеется степенной ряд

$$f(x) = \sum_{i=0}^{\infty} c_i x^i, \tag{6.48}$$

который является исходным для использования аппроксимации Паде.

Паде-аппроксимация представляет собой рациональную функцию вида

$$[L/M] = \frac{a_0 + a_1 x + ... + a_L x^L}{b_0 + b_1 x + ... + b_M x^M}, \tag{6.49}$$

разложение которой в ряд Тейлора (с центром в нуле) совпадает с разложением (6.48). Функция (4.2) имеет $(L+1)$ коэффициентов в числителе и $(M+1)$ в знаменателе. Положим, что $b_0 = 1$. Теперь имеем $(L+1)$ свободных коэффициентов в числителе и M в знаменателе в соотношении (6.49). Всего $(L+M+1)$ свободных

параметров. Это означает, что в общем случае коэффициенты разложения Тейлора функции $[L/M]$ при степенях $1, x, x^2, ..., x^{L+M}$ должны совпадать с соответствующими коэффициентами ряда (4.1), другими словами, должно выполняться соотношение

$$\sum_{i=0}^{\infty} c_i x^i = \frac{a_0 + a_1 x + ... + a_L x^L}{b_0 + b_1 x + ... + b_M x^M} + \mathrm{O}(x^{L+M+1}). \tag{6.50}$$

Умножая (6.50) на знаменатель дроби, имеем:

$$(b_0 + b_1 x + ... + b_M x^M)(c_0 + c_1 x + ...) = a_0 + a_1 x + ... + a_L x^L + \mathrm{O}(x^{L+M+1}). \tag{6.51}$$

Сравнивая коэффициенты при $x^{L+1}, x^{L+2}, ..., x^{L+M}$, получим систему равенств:

$$\begin{aligned} &b_M c_{L-M+1} + b_{M-1} c_{L-M+2} + ... + b_0 c_{L+1} = 0, \\ &b_M c_{L-M+2} + b_{M-1} c_{L-M+3} + ... + b_0 c_{L+2} = 0, \\ &b_M c_L + b_{M-1} c_{L+1} + ... + b_0 c_{L+M} = 0. \end{aligned} \tag{6.52}$$

Положим $c_j = 0$ при $j < 0$. С учетом соглашения $b_0 = 1$ равенства (6.52) можно переписать в виде системы M линейных уравнений с M неизвестными коэффициентами знаменателя:

$$\begin{bmatrix} c_{L-M+1} & c_{L-M+2} & c_{L-M+3} & ... & c_L \\ c_{L-M+2} & c_{L-M+3} & c_{L-M+4} & ... & c_{L+1} \\ c_{L-M+3} & c_{L-M+4} & c_{L-M+5} & ... & c_{L+2} \\ ... & ... & ... & ... & ... \\ c_L & c_{L+1} & c_{L+3} & ... & c_{L+M-1} \end{bmatrix} \begin{bmatrix} b_M \\ b_{M-1} \\ b_{M-2} \\ ... \\ b_1 \end{bmatrix} = - \begin{bmatrix} c_{L+1} \\ c_{L+2} \\ c_{L+3} \\ ... \\ c_{L+M} \end{bmatrix}. \tag{6.53}$$

Отсюда могут быть найдены b_i. Коэффициенты числителя $a_0, a_1, ..., a_L$ находят из (4.4) сравнением коэффициентов при $1, x, x^2, ..., x^L$:

$$\begin{aligned} &a_0 = c_0, \\ &a_1 = c_1 + b_1 c_0, \\ &a_2 = c_2 + b_1 c_1 + b_2 c_0, \\ &... \\ &a_L = c_L + \sum_{i=1}^{\min(L,M)} b_i c_{L-1}. \end{aligned} \tag{6.53}$$

Уравнения (6.53) и (6.54) являются уравнениями Паде. С помощью Паде-аппроксимации ряда

$$\sum_{i=0}^{\infty} c_i x^i$$

можно приблизить функцию $f(x)$, для которой этот ряд будет является рядом Тейлора. Способ построения указывают соотношения (6.53) и (6.54).

Главным является численное решение системы линейных уравнений (6.53). Для их вычисления разработаны различные алгоритмы. Например, *алгоритм Кронекера*, который относится к общей задаче рациональной интерполяции, аппроксимация Паде соответствует случаю совпадения всех узлов интерполяции $x_0 = x_1 = ... = x_{L+M} = 0$. Коэффициенты интерполяционного полинома Ньютона совпадают в этом случае с коэффициентами c_0, c_1, ..., c_{L+M} ряда Тейлора. Алгоритм корректно определяется в тех случаях, когда все элементы рассматриваемой последовательности существуют и не вырождены.

Алгоритм Бейкера и Ватсона основан на построении последовательности числителей $\eta_i(x)$, $i = 0, 1, ...$ и знаменателей $\theta_i(x)$, $i = 0, 1, ...$ аппроксимацией Паде, которые образуют таблицу Паде, представляемую лестничными последовательностями следующего вида:

- для алгоритма Бейкера – $\dfrac{\eta_{2j}(x)}{\theta_{2j}(x)} = [L+M-j/j]$, $\dfrac{\eta_{2j+1}(x)}{\theta_{2j+1}(x)} = [L+M-1/j]$,
- для алгоритма Ватсона – $\dfrac{\eta_{2j}(x)}{\theta_{2j}(x)} = [L+j/j]$, $\dfrac{\eta_{2j+1}(x)}{\theta_{2j+1}(x)} = [L+j+1/j]$.

Коэффициенты полиномов $\eta_i(x)$, $\theta_i(x)$ вычисляются по рекуррентным формулам. Они являются быстрыми методами вычисления последовательностей типа лестниц, параллельных главной диагонали.

Обобщением аппроксимации Паде может служить *аппроксимация Паде–Чебышева*, которая часто реализуется в системах компьютерной математики наряду с аппроксимацией Паде. Например, это функция **EconomizedRationalApproximation** в Mathematica [170] или функция `chebpade` в Maple.

6.1.13. Методика минимаксной аппроксимации

Часто возникает задача для функции $f(x)$ найти близкую функцию $\varphi(x)$ таким образом, чтобы максимальное расстояние между ними было минимальным. Подобная задача является аппроксимацией функций по минимаксному критерию, или просто *минимаксной аппроксимацией*. Формальная постановка данной задачи сводится к следующему.

Пусть имеется функция $y = f(x)$, заданная на отрезке $[a,b]$ значениями $y_1, ..., y_N$ в точках $x_1, ..., x_N$, а также имеется некоторая система непрерывных функций $L=\{\varphi_i(x)\}$, которые линейно независимы на $[a,b]$. Пусть приближение для $f(x)$ ищется в виде линейной комбинации

$$\varphi(x) = \sum_{k=0}^{n} c_k \varphi_k(x),$$

где c_k – неизвестные коэффициенты. Значения аппроксимирующей функции $\varphi(x)$ должны равняться значениям y_i функции $f(x)$ или быть как можно более близкими к ним. Поэтому поиск неизвестных коэффициентов аппроксимирующей

функции можно осуществить путем решения системы линейных алгебраических уравнений

$$\sum c_k \varphi_k(x_i) = y_i, i = 1,...,N. \qquad (6.54)$$

При $N > (n+1)$ система (6.54) является несовместной. В этом случае в качестве решения системы можно взять вектор $c = (c_0, c_1, ..., c_n)$, минимизирующий функцию

$$\phi(c_0, c_1, ..., c_n) = \max_{1 \le i \le N} \left| \sum_{k=0}^{n} c_k \varphi_k(x_i) - y_i \right|. \qquad (6.55)$$

Если выбирать коэффициенты аппроксимирующей функции из условия минимума функции (6.55), то критерий оценивания параметров называется минимаксным, а соответствующая ему задача аппроксимации – минимаксной, или задачей чебышевского приближения. Минимаксный критерий впервые был рассмотрен П. Л. Чебышевым во второй половине XIX века.

При решении минимаксных задач часто используется класс *алгоритмов Ремеза и Вале-Пуссена*. В этих алгоритмах используются множества точек, удовлетворяющих условиям чебышевского альтернанса.

Пусть функция $y = f(x)$ задана на отрезке $[a,b]$, а функция $P_n(x)$ – ее аппроксимирующий полином, причем остаток $R_n(x) = f(x) - P_n(x)$.

Совокупность точек $x_0 < x_1 < ... < x_n < x_{n+1}$ отрезка $[a,b]$ удовлетворяет условиям *чебышевского альтернанса*, если значения остатка $R_n(x)$ в этих точках удовлетворяют условиям

$$R_n(x_0) = -R_n(x_1) = R_n(x_2) = ... = (-1)^{n+1} R_n(x_{n+1}) = \pm \max_{a \le x \le b} |R_n(x)|. \qquad (6.56)$$

Для поиска решения задач полиномиальной аппроксимации используются численные процедуры, носящие итерационный характер. В этих методах осуществляется последовательный выбор точек, удовлетворяющих условиям (6.56).

В достаточно общем виде процедуру поиска точек чебышевского альтернанса при решении задачи полиномиальной можно сформулировать следующим образом:

- на отрезке $[a,b]$ выбирается начальное приближение $x_0^{(0)} < x_1^{(0)} < ... < x_n^{(0)} < x_{n+1}^{(0)}$ для точек альтернанса, а параметр цикла $k = 0$;
- пусть на k-ой итерации найдено приближение $x_0^{(k)} < x_1^{(k)} < ... < x_n^{(k)} < x_{n+1}^{(k)}$. По значениям функции в этих точках определяются коэффициенты полинома $P_n^{(k)}(x)$ и параметр μ_k, удовлетворяющие условиям

  $$R_n^{(k)}(x_i^{(k)}) = \left[f(x_i^{(k)} - P_n^{(k)}(x_i^{(k)}) \right] = (-1)^i \mu_k, \quad i = 0,1,...,n+1; \qquad (6.57)$$

- вычисляется величина

  $$\rho_k = \max_{a \le x \le b} |R_n^{(k)}(x)|;$$

- проверяется условие

  $$\rho_k = \mu_k, \qquad (6.58)$$

  которое можно заменить на условие

  $$\rho \le (1+\varepsilon)\mu_k, \qquad (6.59)$$

  где $\varepsilon > 0$ – допустимая погрешность при проверке условия (6.58);

- если условия (6.58) и (6.59) выполняются, то проверяется выполнение условия для приближений последовательных итераций

$$\max_{0\le i\le n+1}\left|x_i^{(k)}-x_i^{(k+1)}\right|\le\delta, \qquad (6.60)$$

где $\delta>0$ – допустимая погрешность при определении точек альтернанса;

- при выполнении условий (6.58)–(6.60) полученные на k-ой итерации точки $x_0<x_1<\ldots<x_n<x_{n+1}$ являются точками приближения чебышевского альтернанса, а соответствующий им полином – оптимальным по минимаксному критерию. Данный алгоритм сходится со скоростью геометрической прогрессии.

Для определения коэффициентов полинома $P_n(x)$ и параметра μ_k необходимо решить систему из $((n+2)$–x) уравнений (6.57), содержащую $(n+2)$ неизвестных. Если функция $y=f(x)$ задана на дискретном множестве из $(n+2)$ точек, то параметр μ_k можно вычислить по формуле

$$\mu_k=\frac{\sum_{i=0}^{n+1}(-1)^i f(x_i^{(k)})Q_i^{(k)}}{\sum_{i=0}^{n+1}Q_i^{(k)}},$$

где

$$Q_i^{(k)}=\prod_{\substack{j=0\\ j\ne i}}^{n}\prod_{\substack{l=j+1\\ l\ne i}}^{n+1}(x_l^{(k)}-x_j^{(k)}),\quad i=0,1,\ldots,n,n+1,$$

причем

$$P_n^{(k)}(x_i^{(k)})=f(x_i^{(k)})+(-1)^i\mu_k,\quad i=0,1,\ldots,n,n+1.$$

Используя интерполяционную формулу Лагранжа, полином $P_n^{(k)}(x)$ можно представить в виде:

$$P_n^{(k)}(x)=\sum P_n(x_i^{(k)})\frac{\prod^{(k)}(x)}{\prod^{(k)'}(x_i^{(k)})(x-x_i^{(k)})}. \qquad (6.61)$$

Полином Лагранжа наиболее точно аппроксимирует функцию $y=f(x)$, если его узлы являются нулями полинома Чебышева n-го порядка. Поэтому в качестве начального приближения для точек альтернанса наиболее целесообразно выбрать точки

$$x_i^{(0)}=\frac{1}{2}\left[(a+b)+(b-a)\cos\frac{\pi(n+1-i)}{(n+1)}\right],\quad i=0,1,\ldots,n+1. \qquad (6.62)$$

Скорость сходимости алгоритма чебышевской аппроксимации существенно зависит от эффективности алгоритма поиска приближения на каждой итерации. Простейшим из класса алгоритмов Ремеза является *алгоритм Вале-Пуссена*, в котором на каждой итерации осуществляется замена только одной точки приближения для точек альтернанса. Такое приближение называется текущим базисным.

Преобразование базисного множества точек в алгоритме Валле-Пуссена осуществляется следующим образом:

- на отрезке $[a,b]$ определяется точка x, в которой остаток $R_n(x) = f(x) - P_n(x)$ имеет максимальное значение, то есть

$$\left|R_n^{(k)}(\bar{x})\right| = \max_{a \le x \le b}\left|R_n^{(k)}(x)\right|;$$

- возможны три случая расположения точки x на отрезке [a,b]:

$$\bar{x} < x_0^{(k)}, \quad x_0^{(k)} < \bar{x} < x_{n+1}^{(k)}, \quad \bar{x} > x_{n+1}^{(k)};$$

Если $x < x_0^{(k)}$ и если $R_n(x)R_n(x_0^{(k)}) > 0$, то есть остатки $R_n(x)$ и $R_n(x_0^{(k)})$ имеют один и тот же знак, то на $(k+1)$-ой итерации принимается $x_0^{(k+1)} = x$, а все остальные точки не изменяются; если же они имеют различные знаки, то осуществляется преобразование всех точек по формулам

$$x_0^{(k+1)} = \bar{x}, \quad x_i^{(k+1)} = x_{i-1}^{(k)}, \quad i = 1,2,...,n+1.$$

Если $x_0^{(k)} < x < x_{n+1}^{(k)}$, то существует отрезок $[x_{p-1}^{(k)}, x_p^{(k)}]$, которому принадлежит точка x. Тогда если $R_n(x)R_n(x_{p-1})^{(k)} > 0$, то $x_{p-1}^{(k+1)} = x$. В противном случае $x_p^{(k+1)} = x$. Все остальные точки базиса остаются без изменений.

Если $x > x_{n+1}^{(k)}$, то осуществляется замена последней точки $x_{n+1}^{(k+1)} = x$ при $R_n(x)R_n(x_{n+1}^{(k)}) > 0$. При невыполнении этого условия осуществляется преобразование всех точек базиса по формулам $x_i^{(k+1)} = x_{i+1}^{(k)}$, при $i = 0,1,...,n$ и $x_{n+1}^{(k+1)} = x$.

Последовательность приближений, получаемых с помощью этого алгоритма, сходится к точкам чебышевского альтернанса независимо от выбора начального базиса.

В отличие от алгоритма Валле-Пуссена, в алгоритме Ремеза на каждой итерации осуществляется замена всех точек текущего базиса, в результате чего возрастает скорость сходимости к точкам чебышевского альтернанса.

Пусть аппроксимируемая функция $y = f(x)$ определена во всех точках $[a,b]$, или заданы ее значения только в N дискретных точках, принадлежащих $[a,b]$. Алгоритм Ремеза замены базиса можно сформулировать следующим образом:

- пусть $x_0^{(0)} < x_1^{(0)} < ... < x_n^{(0)} < x_{n+1}^{(0)}$ – начальный базис на $[a,b]$, ρ_0 – максимальное значение остатка $R_n^{(0)}(x)$;
- пусть в результате выполнения итерационной процедуры найдено k-ое приближение базиса $x_0^{(0)} < x_1^{(0)} < ... < x_n^{(0)} < x_{n+1}^{(0)}$, а также соответствующее значение ρ_k.

Преобразование базиса в алгоритме Ремеза на $(k+1)$-ой итерации состоит из $(n+1)$ шагов.

Первый шаг: на отрезке $[a,x_1^{(k)}]$ определяются две точки $v_0 < v_1$, в одной из которых $R_n^{(k)}(x)$ имеет наименьшее значение, а в другой – наибольшее. В этом случае нулевая и первая точки $(k+1)$-го приближения принимаются равными $x_0^{(k+1)} = v_0$, $x_1^{(k+1)} = v_1$.

Второй шаг: на отрезке $[x_1^{(k+1)}, x_2^{(k)}]$ определяются две точки $v_1 < v_2$, в одной из которых $R_n^{(k)}(x)$ имеет наименьшее значение (на этом отрезке), а в другом – наибольшее. Если $R_n^{(k)}(x_1^{(k+1)})R_n^{(k)}(v_1) > 0$, то вторая точка $(k+1)$-го приближения

$x_2^{(k+1)} = v_2$, причем если $|R_n^{(k)}(x_1^{(k+1)})| < |R_n^{(k)}(v_1)|$, то преобразуется также и первая точка $(k+1)$-го приближения $x_1^{(k+1)} = v_1$. В этом случае запоминается значение параметра $g = 0$ и осуществляется переход к следующему шагу алгоритма. Если же $R_n^{(k)}(x_1^{(k+1)})R_n^{(k)}(v_1) < 0$, то принимается $x_2^{(k+1)} = v_1$ и запоминаются $r = v_2$, $g = R_n^{(k)}(r)$ и т. д. Предположим, что в результате преобразований такого типа найдены базисные точки $x_0^{(k+1)} < x_1^{(k+1)} < ... < x_p^{(k+1)}$, а на предыдущем шаге при $g \neq 0$ получено значение r.

$(p+1)$-ый шаг: на отрезке $[x_p^{(k+1)}, x_{p+1}^{(k)}]$ аналогично предыдущему находим точки v_1 и v_2, в одной из которых $R_n^{(k)}(x)$ имеет минимальное, а в другой – максимальное значение. Если $R_n^{(k)}(x_p^{(k+1)})R_n^{(k)}(v_1) > 0$ и $|R_n^{(k)}(x_p^{(k+1)})| \geq |R_n^{(k)}(v_1)|$, то осуществляется преобразование $(p+1)$-ой точки

$$x_{p+1}^{(k+1)} = \begin{cases} v_1, \text{ при } \left|R_n^{(k)}(v_1)\right| \geq g, \\ r, \text{ при } \left|R_n^{(k)}(v_1)\right| < g. \end{cases}$$

При условии, что $R_n^{(k)}(x_p^{(k+1)})R_n^{(k)}(v_1) > 0$ и $|R_n^{(k)}(x_p^{(k+1)})| < |R_n^{(k)}(v_1)|$, $(p+1)$-ая точка принимается равной $x_{p+1}^{(k+1)} = v_2$, повторно изменяется $x_p^{(k+1)} = v_1$, а при $|g| > R_n^{(k)}(x_{p-1}^{(k+1)})$ имеем $x_{p-1}^{(k+1)} = r$. После этого параметр g принимается равным нулю и осуществляется переход к следующему шагу алгоритма замены базисных точек.

$(n+1)$-ый шаг (последний): на отрезке $[x_n^{(k+1)}, b]$ осуществляется описанная процедура для p-го шага. Если после выполнения рассмотренных шагов $(n+1)$-го шага $g > |R_n^{(k)}(x_0^{(k+1)})|$, то осуществляется преобразование точек базиса

$$x_i^{(k+1)} = x_{i+1}^{(k+1)}, \quad i = 0,1,...,n,$$
$$x_{n+1}^{(k+1)} = r.$$

Следует отметить, что на практике не всегда можно найти алгебраический полином наилучшего приближения, даже с помощью такого эффективного алгоритма, как алгоритм Ремеза. Одной из причин этого является плохая обусловленность полинома $P_n(x)$. Числовой характеристикой обусловленности может выступать величина

$$\varphi_{об} = \frac{\max\limits_{0 \leq k \leq n} |a_k|}{\max\limits_{a \leq x \leq b} |P_n(x)|},$$

где a_k – коэффициент полнома $P_n(x)$. Поэтому на практике при построении полинома наилучшего равномерного приближения часто приходится выбирать различные начальные приближения для базисных точек, чтобы добиться сходимости итерационной процедуры.

6.1.14. Оптимизация временных затрат и схема Горнера

Для оптимизации временных затрат при аппроксимации возможно преобразование аппроксимирующих выражений в форму или *схему Горнера*. Это дает ряд преимуществ перед обычным вычислением полиномов: уменьшается время вычислений, повышается точность вычислений, уменьшается вероятность расхождения численных методов, в которых используются полиномы.

Рассмотрим алгебраический полином

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n, \tag{6.63}$$

где a_0, a_1, ..., a_n – коэффициенты, n – степень полинома.

Вычисление значения этого полинома при фиксированном значении $x = a$ можно выполнить различными способами.

Например, можно сначала с помощью $(n - 1)$ умножений найти степени параметра a, то есть a, a^2, ..., a^n. Затем в соответствии с формулой (6.53), где $x = a$, выполнить еще n умножений и n сложений. В результате будем иметь значение $P_n(a)$. Этот наиболее естественный на первый взгляд способ требует в общем случае при $n \geq 1$ выполнения $(3n - 1)$ арифметических действий.

Более экономичный способ вычисления представляется, если полином (6.63) записать следующим образом:

$$P_n(x) = a_0 + x(a_1 + x(a_2 + ... + x(a_{n-2} + x(a_{n-1} + xa_n))...)). \tag{6.64}$$

Согласно формуле (6.54) вычисление $P_n(x)$ сводится к последовательному нахождению следующих величин:

$$\begin{aligned} b_n &= a_n \\ b_{n-1} &= a_{n-1} + ab_n \\ b_{n-2} &= a_{n-2} + ab_{n-1} \\ &... \\ b_1 &= a_1 + ab_2 \\ b_0 &= a_0 + ab_1 = P_n(a) \end{aligned} \tag{6.65}$$

Способ нахождения значения полинома по формулам (6.55) и называется схемой Горнера, которая реализуется с помощью n умножений и n сложений, то есть за $2n$ арифметических действий. В общем случае не существует способа вычисления алгебраического полинома n-ой степени менее чем за $2n$ арифметических действий.

Схема Горнера удобна для программной реализации благодаря цикличности вычислений и необходимости сохранять, кроме коэффициентов полинома, и значения аргумента только одной промежуточной величины, а именно b_i или ab_i при текущем значении $i = n, n - 1, ..., 0$.

Однако следует отменить, что при вычислении полиномов по схеме Горнера с большими коэффициентами может произойти значительная потеря точности за счет вычитания больших округленных чисел. Избежать потери точности иногда удается применением рекуррентных формул.

6.2. Аппроксимация функций в системе Maple

6.2.1. Аппроксимация аналитически заданных функций в Maple

В большинстве СКМ есть специальные средства для автоматизации задач приближения аналитически заданных функций. В Maple, если функция задана аналитически, то наиболее простым способом нахождения ее аппроксимирующей зависимости является применение функции `convert`, которая позволяет представить функцию в виде иного выражения, чем исходное. Например, при опции `polynom` осуществляется полиномиальная аппроксимация. Это поясняет пример аппроксимации экспоненциальной функции, показанный на рис. 6.4.

На рис. 6.5 представлен пример полиномиальной аппроксимации хорошо известной статистической функции `erfc(x)`. Для полинома задана максимальная степень 12, но ввиду отсутствия в разложении четных степеней максимальная степень результата оказывается равна 11.

Как видно из приведенного рисунка, в интервале изменения x от $-1{,}4$ до $1{,}4$ аппроксимирующее выражение почти повторяет исходную зависимость. Однако затем график аппроксимирующей функции быстро отходит от графика исходной зависимости и погрешность аппроксимации резко возрастает. При этом он ведет себя иначе даже качественно, никоим образом не показывая асимптотического поведения, характерного для исходной зависимости. Это говорит о том, что полиномиальная аппроксимация плохо подходит для экстраполяции (предсказания) зависимостей.

Как уже отмечалось, считается, что полиномиальная аппроксимация дает большую погрешность при степени полинома более 5–6. Однако этот вывод базируется на том, что большинство вычислительных программ работает со всего 5–10 точными знаками в промежуточных и окончательных результатах.

СКМ Maple по умолчанию имеет 10 точных знаков чисел. Это показывает следующий пример:

```
> restart:Digits;
```

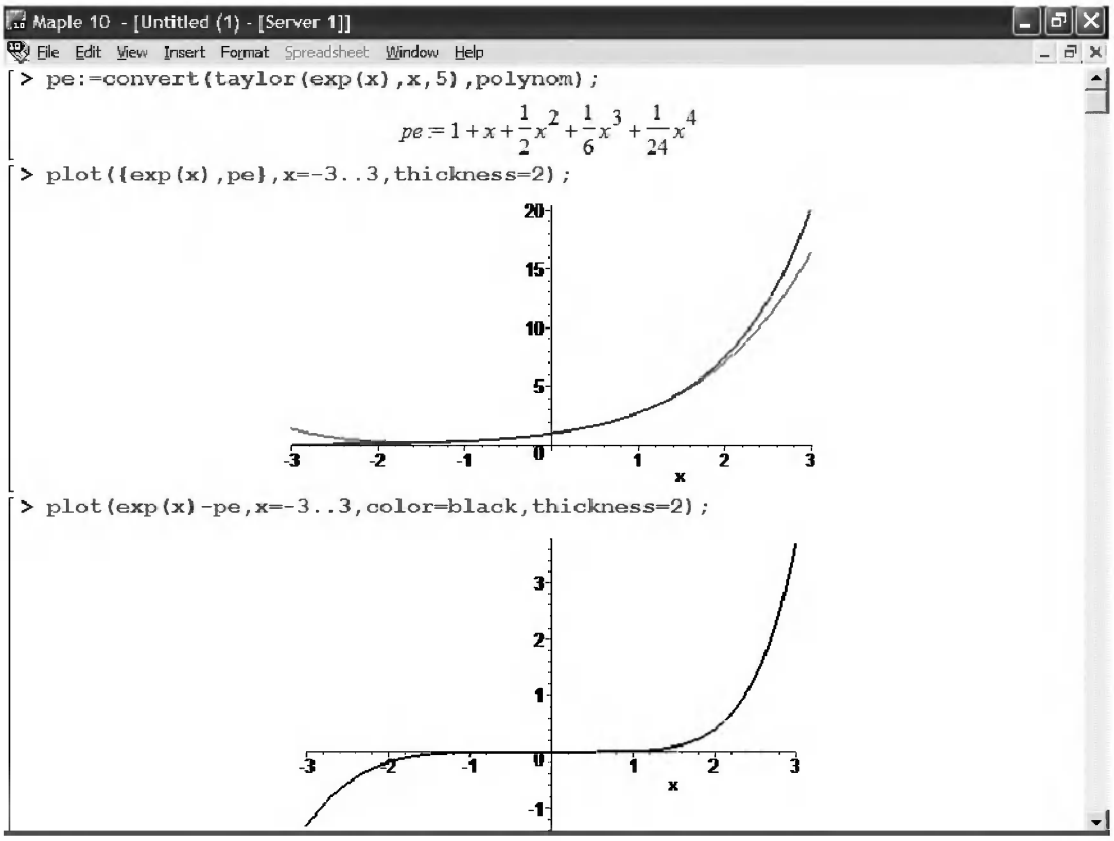


Рис. 6.4. Пример полиномиальной аппроксимации экспоненциальной функции exp(x): сверху построены графики исходной функции и полинома, снизу – график абсолютной погрешности

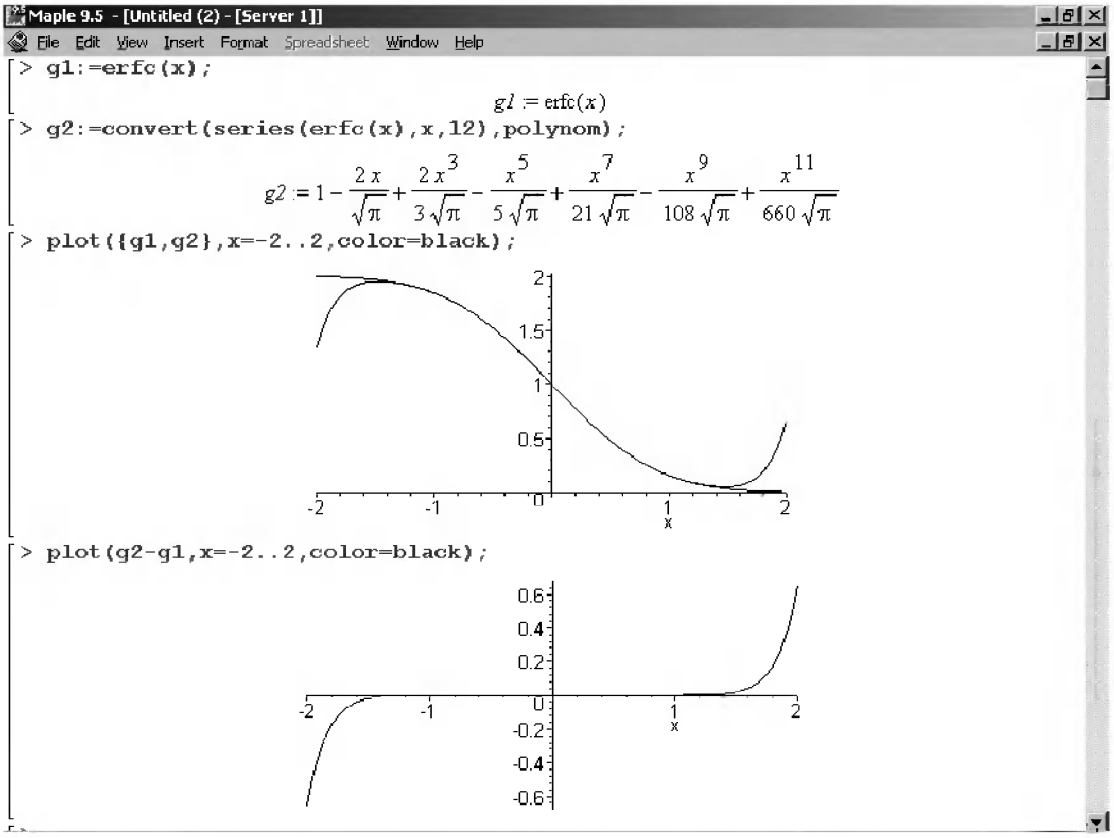


Рис. 6.5. Пример полиномиальной аппроксимации функции `erfc(x)`: сверху построены графики исходной функции и полинома, снизу – график абсолютной погрешности

Таким образом, Maple, как и любая другая программа, может давать большую погрешность при высоких степенях аппроксимирующего полинома.

В этом убеждает рис. 6.6. На нем представлена программа полиномиальной аппроксимации функции синуса с возможностью выбора степени полинома N. Программа автоматически задает (N + 1) отсчетов функции синуса и затем выполняет ее полиномиальную аппроксимацию для N = 10 и Digits = 8. Результат аппроксимации совершенно неудовлетворительный – видно, что вычисления под конец пошли вразнос – так именуются хаотические изменения кривой аппроксимирующей функции.

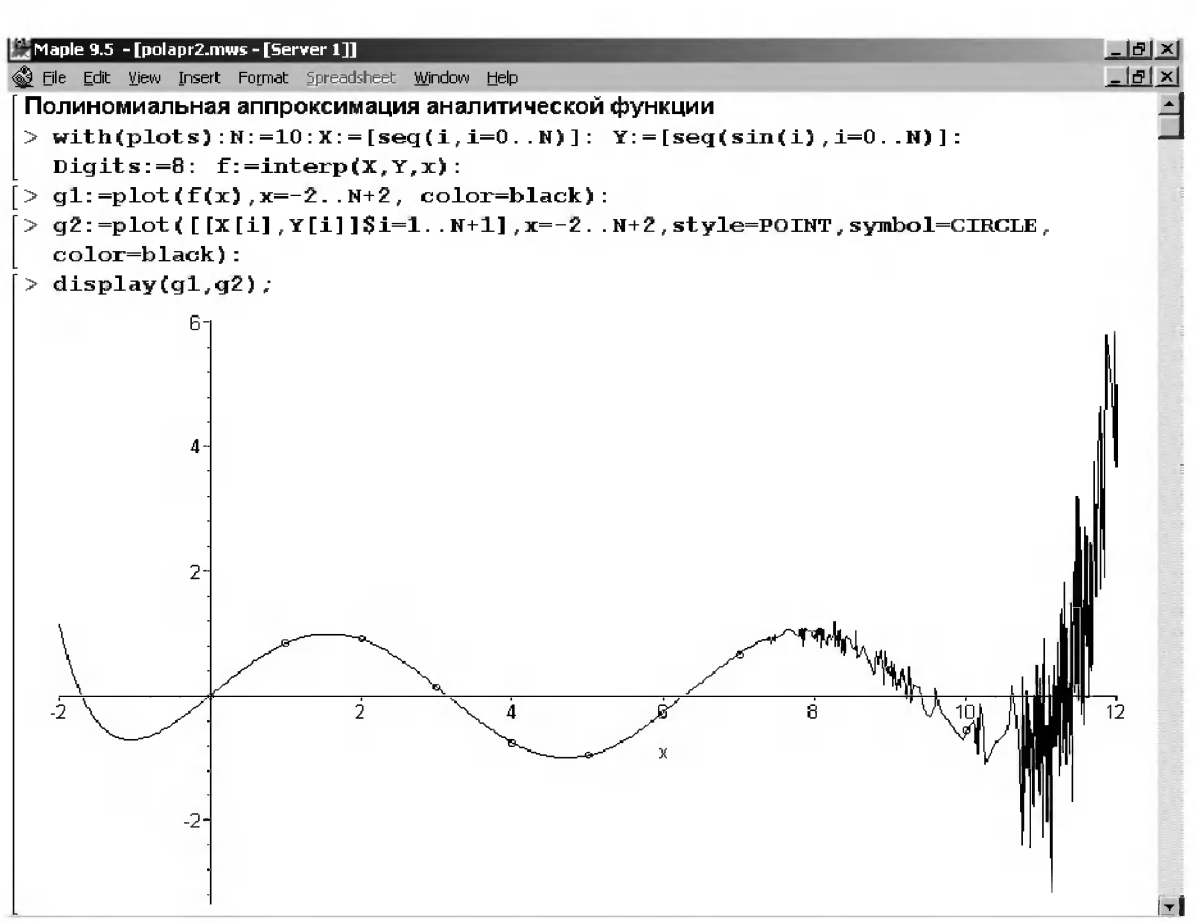


Рис. 6.6. Пример неудачной аппроксимации синуса при N = 10 и Digits = 8

Практическая рекомендация при полиномиальной аппроксимации выглядит следующим образом: число точных цифр в промежуточных результатах Digits должно на несколько цифр превышать значение N. Рисунок 6.7, приведенный для N = 10 и Digits = 15, удовлетворяет этому правилу. При этом все точки точно укладываются на кривую полинома 10-го порядка. Однако за пределами интервала, в котором находятся узловые точки, кривая аппроксимации резко отклоняется от функции синуса – экстраполяция явно неудовлетворительна.

СКМ Maple 9.5 является системой, позволяющей выполнять арифметические вычисления с практически произвольным числом точных цифр. Ограничение на это число накладывается объемом памяти ПК (для современных ПК не актуально) и возрастанием времени вычислений. В качестве примера аппроксимации по-

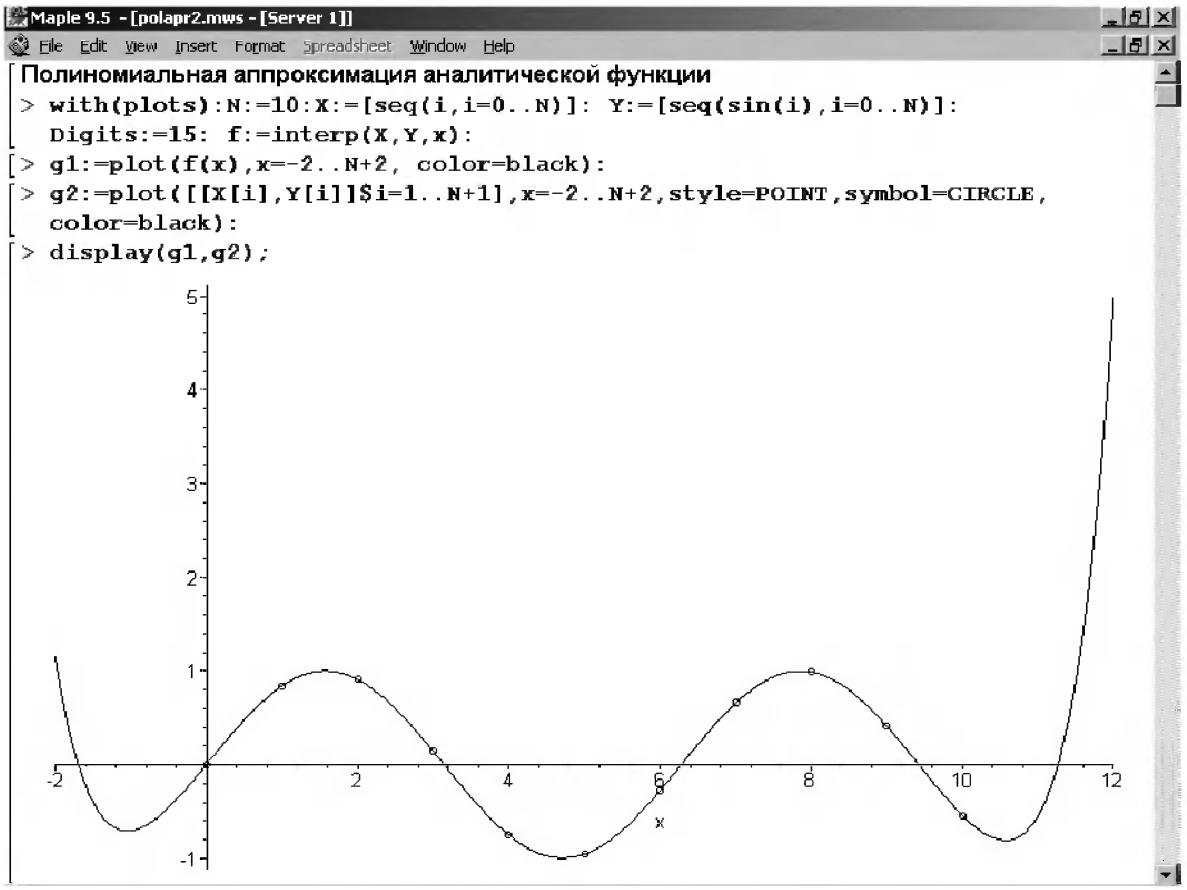


Рис. 6.7. Пример достаточно корректной аппроксимации синуса при N = 10 и Digits = 15

линомом высокой степени на рис. 6.8 приведен документ, осуществляющий аппроксимацию функции синуса для степени полинома $N = 30$ при числе точных цифр Digits = 40. Нетрудно заметить, что все 31 узловые точки прекрасно укладываются на кривую полинома и что за пределами расположения этих точек она резко отклоняется от синусоидальной функции.

Для оценки погрешности аппроксимации можно исполнить команду:

```
> plot(f(x)-sin(x),x=0..30, color=black);
```

Она строит график абсолютной погрешности аппроксимации в интервале изменения x от 0 до 30. Он приведен на рис. 6.9 и показывает заметный рост погрешности аппроксимации только в начале и в конце интервала изменения x.

В целом аппроксимация полиномами высокой степени хотя и возможна, но непрактична, поскольку такие полиномы едва ли можно назвать простыми аппроксимирующими функциями.

6.2.2. Полиномиальная интерполяция табличных данных в Maple 9.5

Полиномиальная интерполяция и аппроксимация табличных данных довольно сложна. Но на самом деле выполнять все расчеты для полиномиальной аппроксимации в Maple и не нужно, поскольку система для этого имеет встроенную функ-

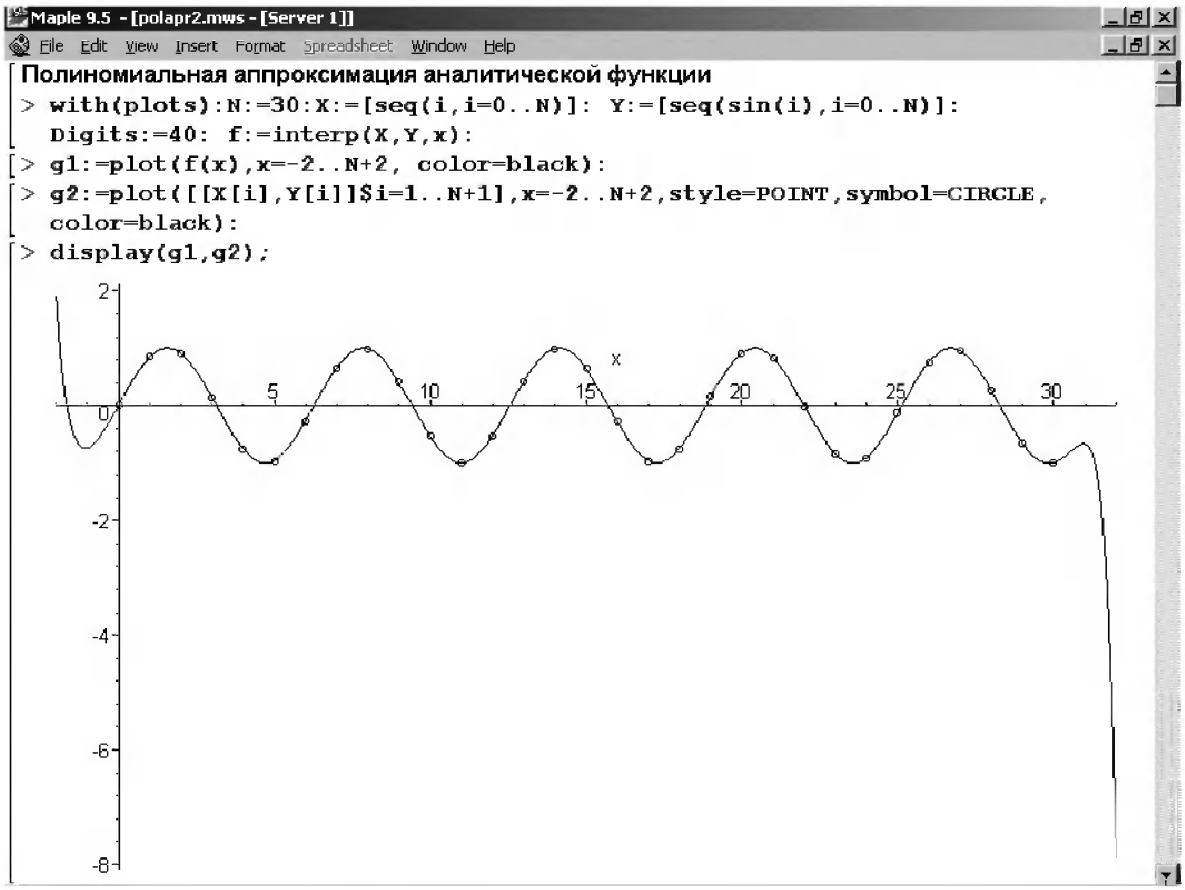


Рис. 6.8. Пример аппроксимации функции синуса полиномом степени N = 30 при Digits = 40

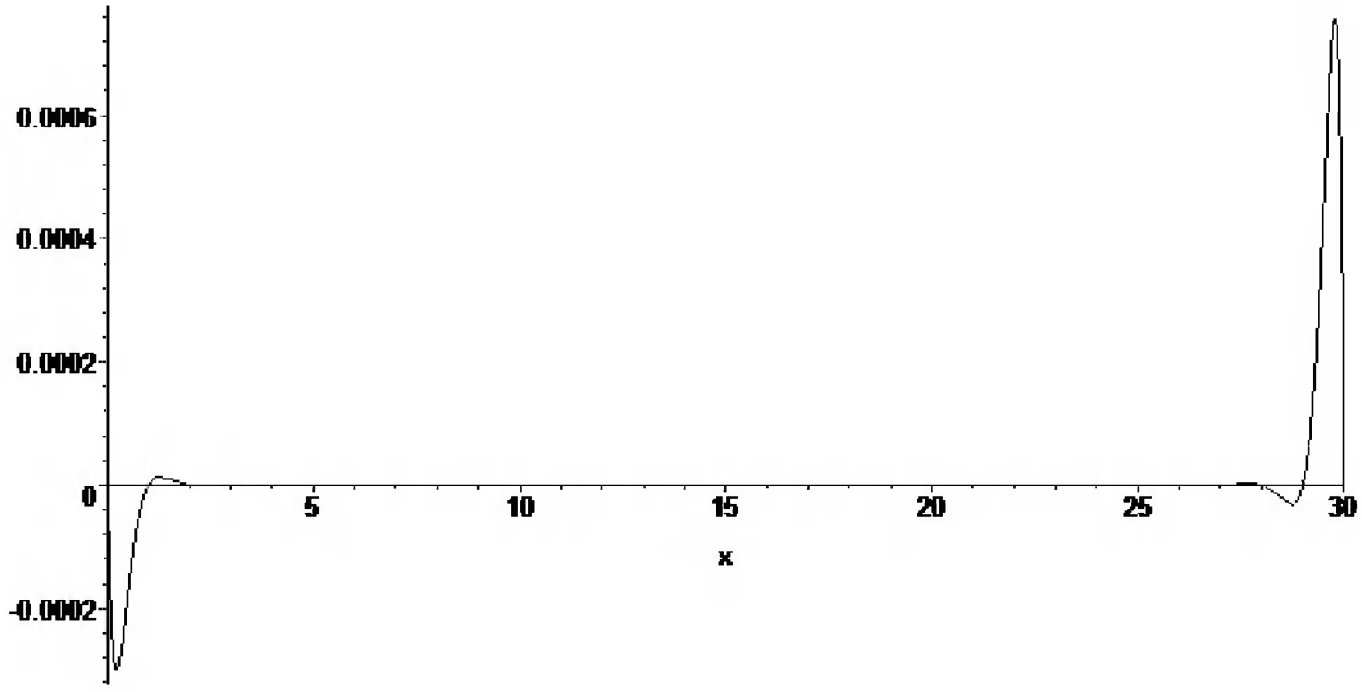


Рис. 6.9. Абсолютная погрешность аппроксимации функции синуса полиномом степени N=30 при Digits=40 в интервале изменения x от 0 до 30

цию `interp(X,Y,v)`, или, в инертной форме, `Interp(X,Y, v)`. Переменная `v` указывает имя переменной интерполяционного полинома. Векторы `X` и `Y` должны содержать $n + 1 = N$ координат точек исходной зависимости, где n – степень интерполирующего полинома.

Рисунок 6.10 показывает технику применения полиномиальной аппроксимации на основе функции `interp` с построением графика исходных точек и аппроксимирующего полинома. Нетрудно заметить, что график полинома проходит точно через исходные точки – они показаны квадратиками.

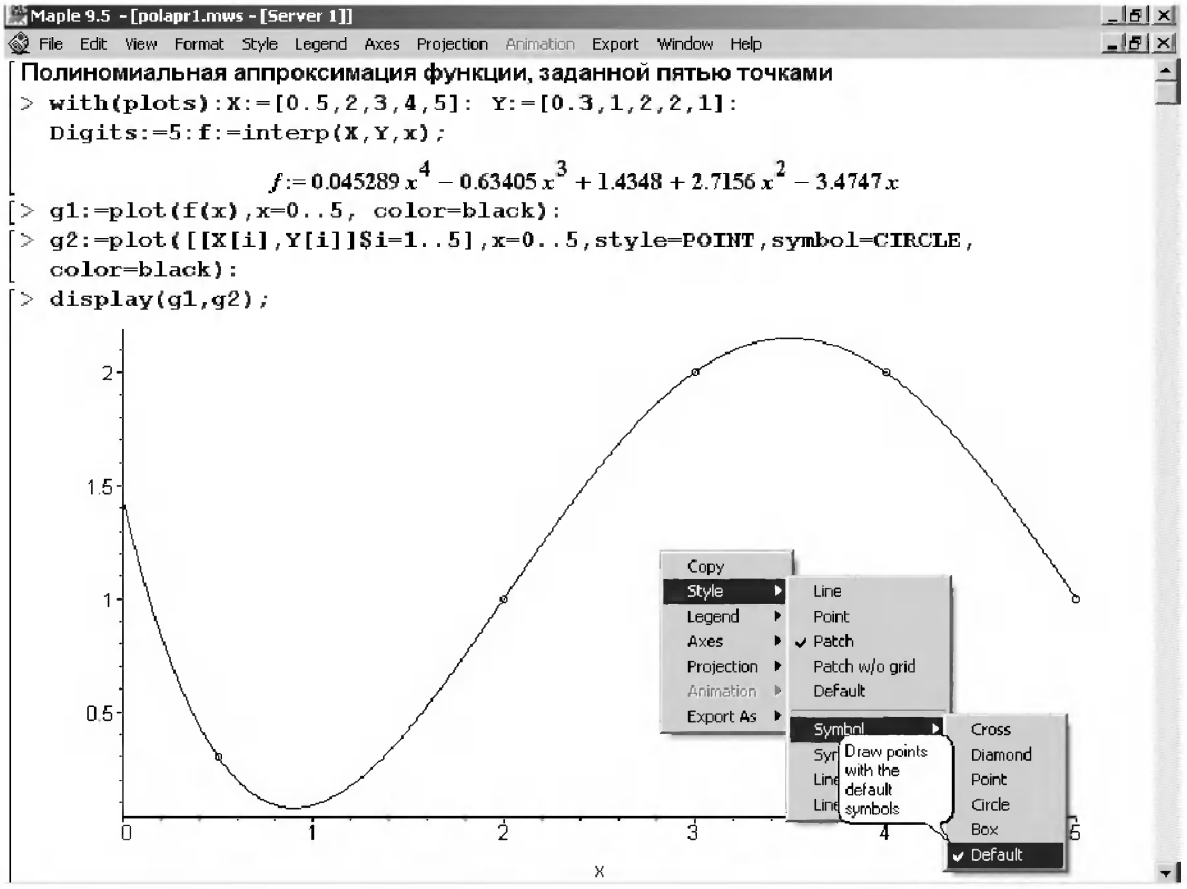


Рис. 6.10. Пример осуществления полиномиальной аппроксимации для таблично заданных данных (точек)

Приведем еще несколько примеров использования функции `Interp`:

```
> Interp([2,5,6], [9,8,3], x) mod 11;
```

$$8x^2 + 6x + 9$$

```
> alias(alpha=RootOf(x^4+x+1));
```

$$\alpha$$

```
> a := Interp([0,1,alpha],[alpha,alpha^2,alpha^3], x) mod 2;
```

$$a := x^2 + (\alpha^2 + \alpha + 1)x + \alpha$$

6.2.3. Сплайновая аппроксимация в Maple

Для получения сплайн-интерполяции в Maple используется Maple-функция `spline(X,Y,var,d)`. Здесь `X` и `Y` – одномерные векторы одинакового размера, несущие значения координат узловых точек исходной функции (причем в произвольном порядке), var – имя переменной, относительно которой вычисляется сплайн-функция, наконец, необязательный параметр `d` задает вид сплайна. Он может иметь цифровые – 1, 2, 3 или 4 – либо символьные значения:

- `linear` – линейная функция, или полином первого порядка;
- `quadratic` – квадратичная функция, или полином второго порядка;
- `cubic` – полином третьего порядка;
- `quartic` – полином четвертого порядка.

Если параметр `d` опущен, то сплайн-функция будет строиться на основе полиномов третьего порядка (кубические сплайны). Важно отметить, что за пределами узловых точек сплайны обеспечивают экстраполяцию, представляя данные в соответствии с первым полином слева и последним справа.

Технику сплайновой аппроксимации наглядно поясняет рис. 6.11. На нем представлено задание векторов узловых точек `X` и `Y` и четырех сплайновых функций, по которым построены их графики. Для одной из функций (кубических сплайнов) показан вид сплайновой функции.

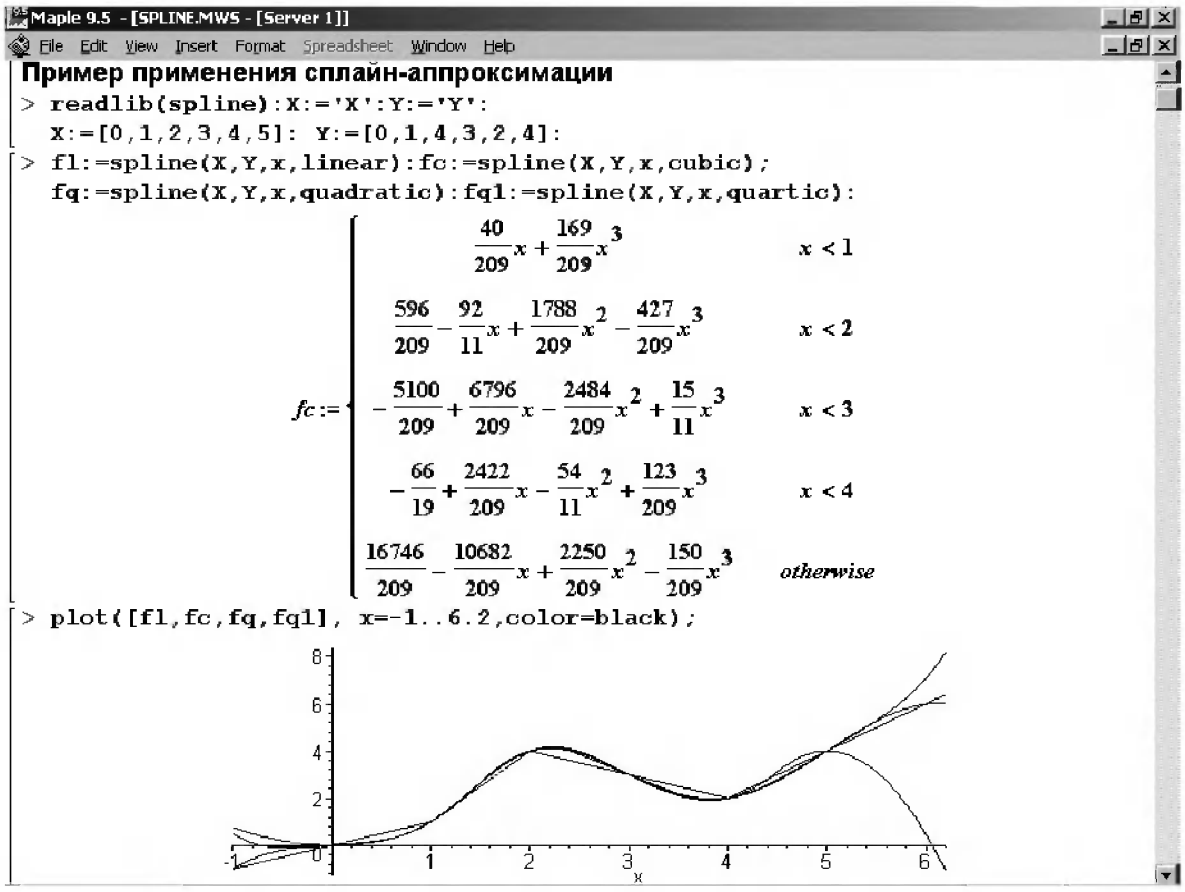


Рис. 6.11. Задание сплайновой аппроксимации и построение графиков полученных функций

Как хорошо видно из рис. 6.11, сплайновая функция действительно представляет собой кусочную функцию, определяемую на каждом отрезке отдельно. При этом на каждом участке такая функция описывается отдельным полиномом соответствующей степени. Функция построения графиков `plot` «понимает» такие функции и позволяет без преобразования типов данных строить их графики. Для работы с кусочными функциями можно использовать функции `convert` и `piecewise`.

Обычно удобно представлять на одном графике узловые точки и кривые интерполяции и экстраполяции. На рис. 6.12 дан пример такого рода. Здесь для одних и тех же данных, представленных векторами datax и datay, заданы все 4 возможных типа сплайновой интерполяции/экстраполяции (заданы числами, указывающими на степень полиномов сплайн-функций).

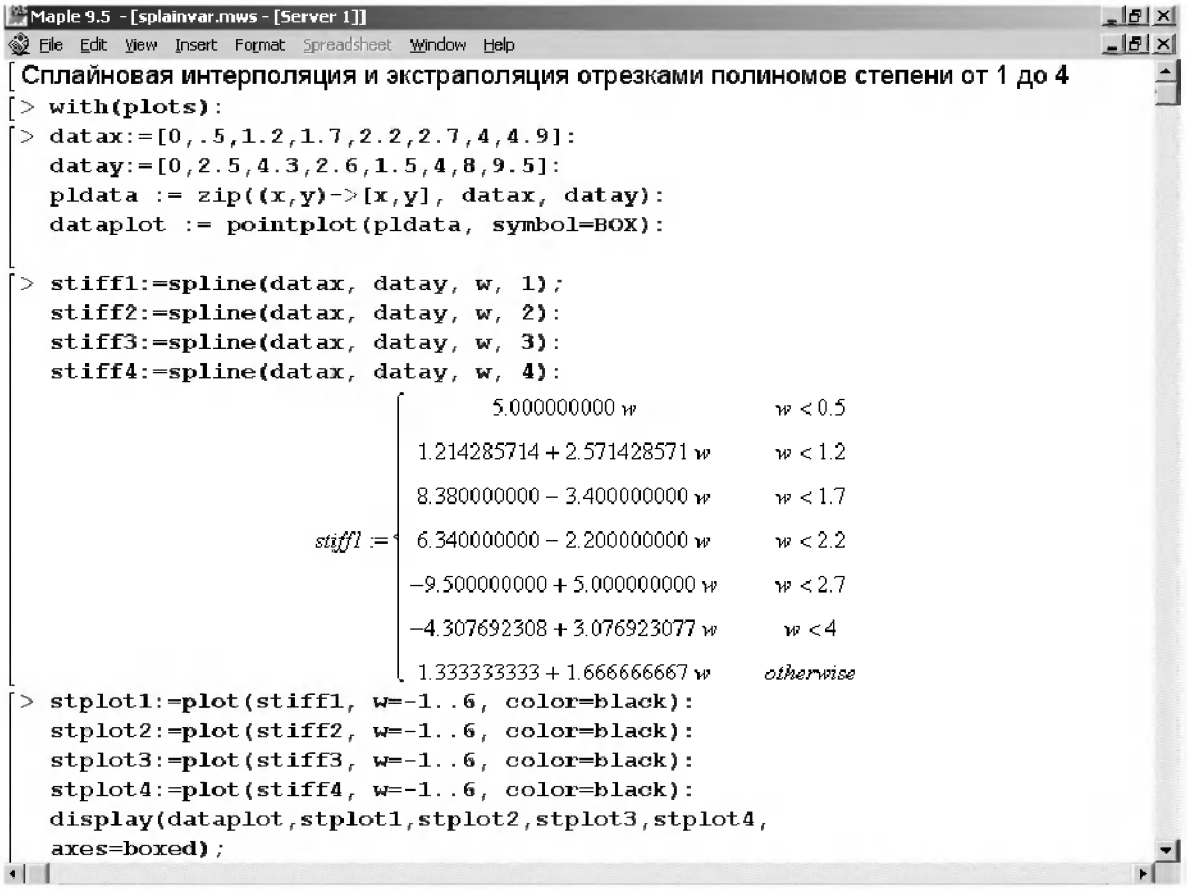


Рис. 6.12. Сплайновая интерполяция/экстраполяция при степени полиномов от 1 до 4

Вывод указан для степени полиномов 1, что соответствует линейной интерполяции/экстраполяции. Для других случаев вывод заблокирован двоеточием, поскольку выглядит очень громоздким. Тем не менее читатель может просмотреть его, заменив двоеточие на точку с запятой. С помощью графической функции display выводятся как все кривые сплайновой интерполяции/экстраполяции, так и узловые точки – рис. 6.13. Полезно обратить внимание на плохую пригодность для экстраполяции сплайнов второго порядка.

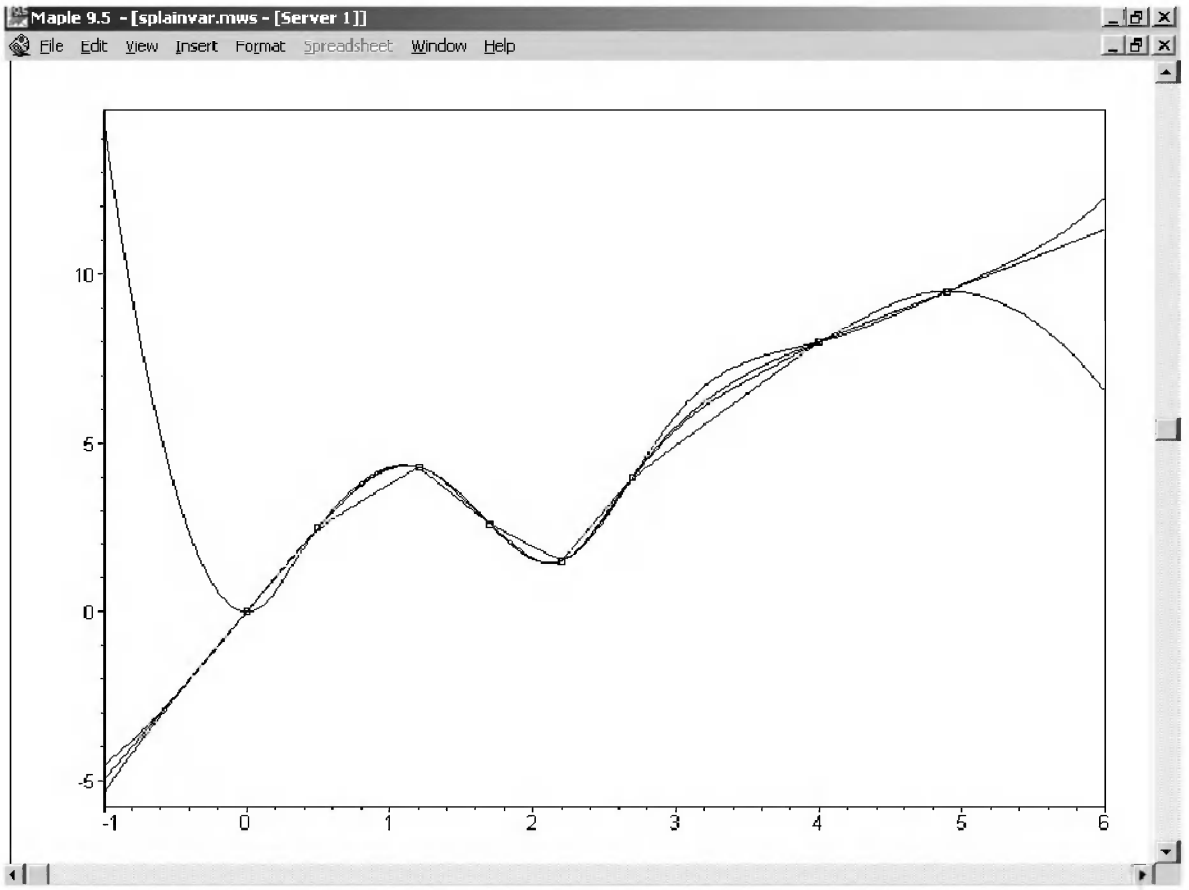


Рис. 6.13. Графики, построенные документом, представленным на рис. 6.12

Мы вернемся к рассмотрению сплайновой аппроксимации в конце этой главы при описании пакета расширения CurveFitting.

6.2.4. Состав пакета numapprox системы Maple

Для более глубоких и продвинутых операций аппроксимации служит специальный пакет расширения numapprox. Этот пакет содержит небольшое число безусловно очень важных функций:

```
> with(numapprox);
```

[*chebdeg*, *chebmult*, *chebpade*, *chebsort*, *chebyshev*, *confracform*, *hermite_pade*, *hornerform*, *infnorm*, *laurent*, *minimax*, *pade*, *remez*]

В их числе функции интерполяции и аппроксимации полиномами Чебышева, рядом Тейлора, отношением полиномов (Паде-аппроксимация) и др. Все они широко применяются не только в фундаментальной математике, но и при решении многих прикладных задач. Рассмотрим их, начиная с функций аппроксимации аналитических зависимостей.

6.2.5. Разложение функции в ряд Лорана

Для разложения функции `f` в ряд Лорана с порядком `n` в окрестности точки $x = a$ (или $x = 0$) служит функция `laurent`:

```
laurent(f, x=a, n)      laurent(f, x, n)
```

Представленный ниже пример иллюстрирует суть реализации разложения в ряд Лорана:

```
> laurent(f(x),x=0,4);
```

$$f(0)+D(f)(0)x+\frac{1}{2}(D^{(2)})(f)(0)x^2+\frac{1}{6}(D^{(3)})(f)(0)x^3+O(x^4)$$

```
> laurent(exp(x),x,5);
```

$$1+x+\frac{1}{2}x^2+\frac{1}{6}x^3+\frac{1}{24}x^4+O(x^5)$$

6.2.6. Паде-аппроксимация аналитических функций

Для осуществления Паде-аппроксимации в Maple используется функция `pade`:

```
pade(f, x=a, [m,n])     pade(f, x, [m,n])
```

Здесь `f` – аналитическое выражение или функция, `x` – переменная, относительно которой записывается аппроксимирующая функция, `a` – координата точки, относительно которой выполняется аппроксимация, `m`, `n` – максимальные степени полиномов числителя и знаменателя. Технику аппроксимации Паде непрерывной функции поясняет рис. 6.14.

На рис. 6.14 представлена аппроксимация синусоидальной функции, а также построены графики этой функции и аппроксимирующей функции. Под ними дан также график абсолютной погрешности для этого вида аппроксимации. Нетрудно заметить, что уже в интервале $[-\pi, \pi]$ погрешность резко возрастает на концах интервала аппроксимации.

Важным достоинством Паде-аппроксимации является возможность довольно точного приближения разрывных функций. Это связано с тем, что нули знаменателя у аппроксимирующего выражения способны приближать разрывы функций, если на заданном интервале аппроксимации число разрывов конечно. На рис. 6.15 представлен пример Паде-аппроксимации функции $\tan(x)$ в интервале от –4,5 до 4,5, включающем два разрыва функции.

Как видно из рис. 6.15, расхождения между функцией тангенса и ее аппроксимирующей функцией едва заметны лишь на краях интервала аппроксимации. Оба разрыва прекрасно приближаются аппроксимирующей функцией, и никакого выброса погрешности в точках разрыва нет. Такой характер аппроксимации подтверждается и графиком погрешности, которая лишь на концах интервала аппроксимации $[-4.0, 4.0]$ достигает значений 0,01 (около 1%).

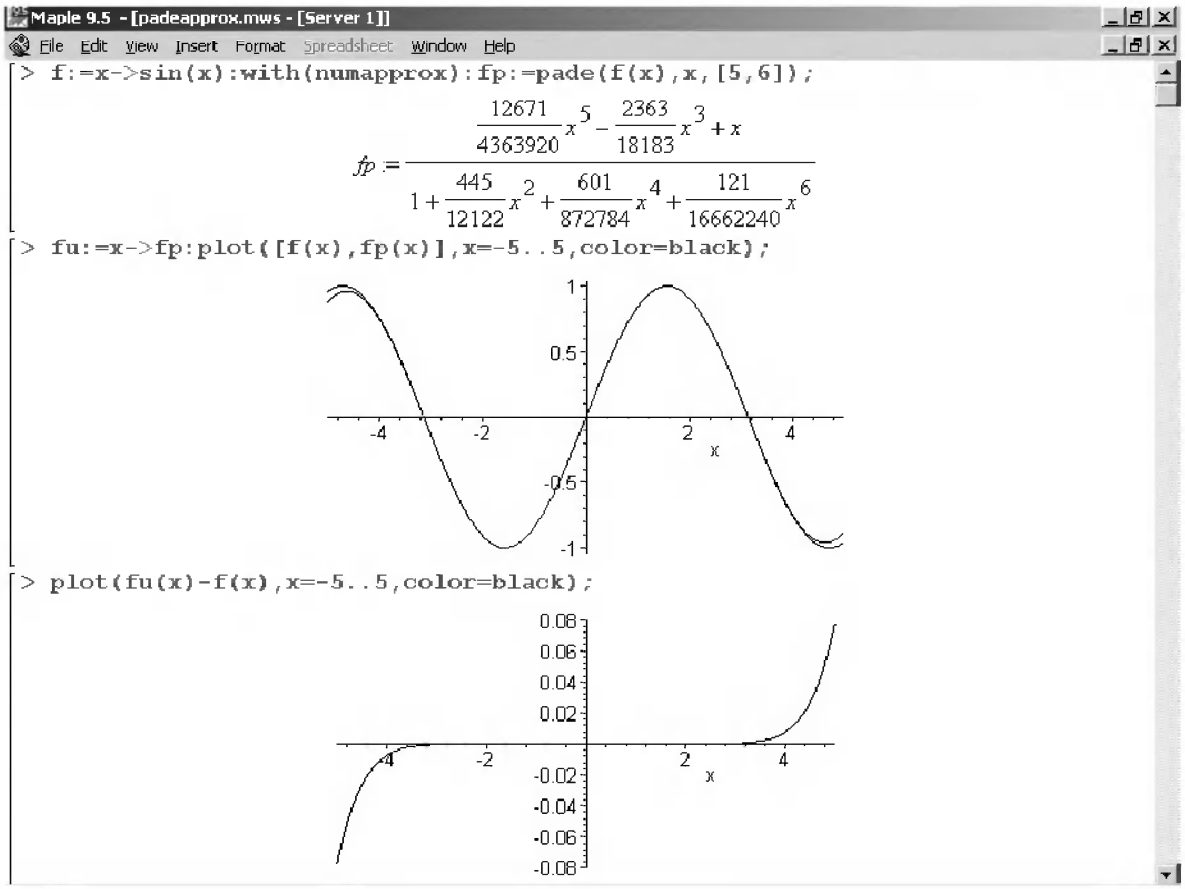


Рис. 6.14. Аппроксимация Паде для синусоидальной функции

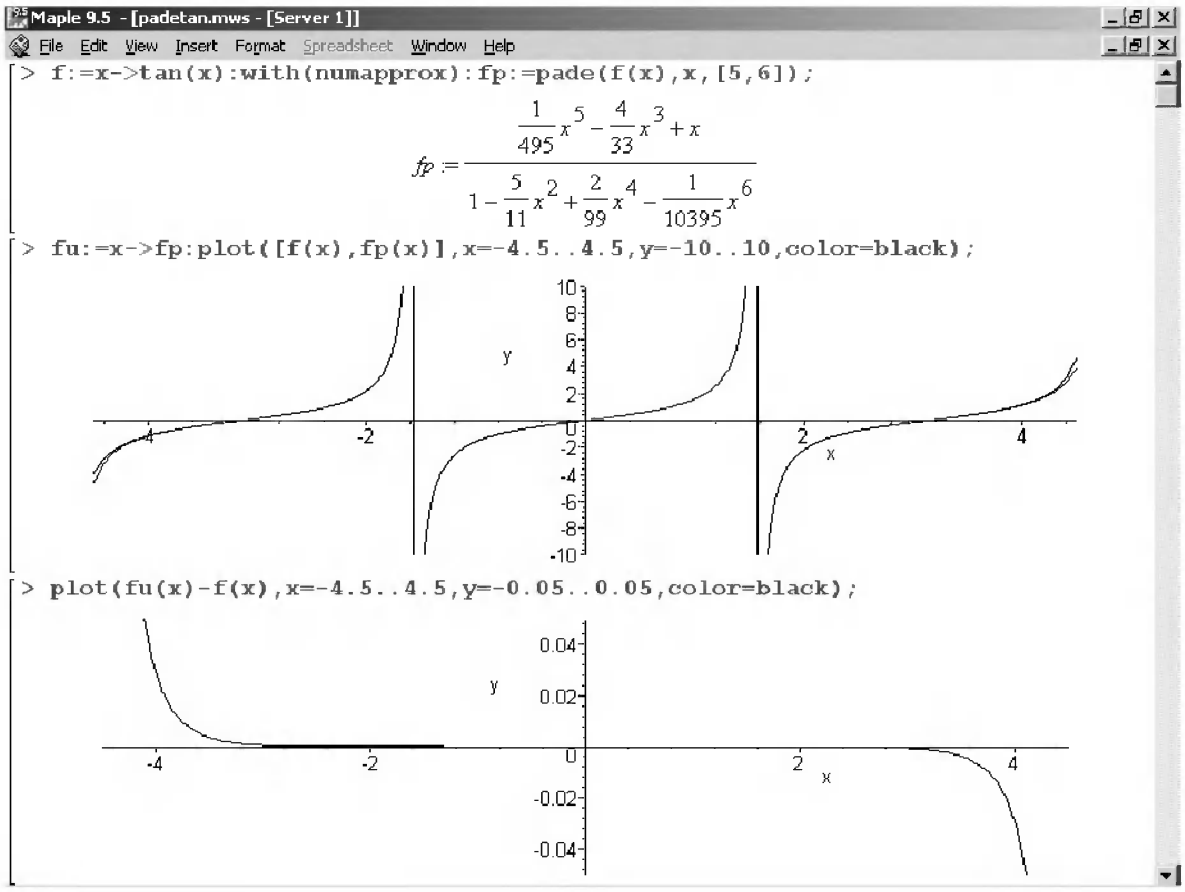


Рис. 6.15. Аппроксимация Паде для разрывной функции тангенса

6.2.7. Паде-аппроксимация с полиномами Чебышева

Для многих аналитических зависимостей хорошие результаты дает аппроксимация полиномами Чебышева. При ней более оптимальным является выбор узлов аппроксимации, что ведет к уменьшению погрешности аппроксимации.

В общем случае применяется Паде-аппроксимация, характерная представлением аппроксимирующей функции в виде отношения полиномов Чебышева. Она реализуется функциями `chebpade`:

```
chebpade(f, x=a..b, [m,n])     chebpade(f, x, [m,n])
chebpade(f, a..b, [m,n])
```

Здесь `a..b` задает отрезок аппроксимации, `m` и `n` – максимальные степени числителя и знаменателя полиномов Чебышева. Приведенный ниже пример показывает аппроксимацию Паде полиномами Чебышева для функции `f=cos(x)`:

```
> Digits:=10:chebpade(cos(x),x=0..1,5);
```

$$0.8235847380\,T(0, 2x-1) - 0.2322992716\,T(1, 2x-1)$$
$$- 0.5371511462\,T(2, 2x-1) + 0.002458235267\,T(3, 2x-1)$$
$$+ 0.0002821192574\,T(4, 2x-1) - 0.7722229156\,10^{-5}\,T(5, 2x-1)$$

```
> chebpade(cos(x),x=0..1,[2,3]);
```

$$0.8162435876\,T(0, 2x-1) - 0.1852356296\,T(1, 2x-1)$$
$$- 0.05170917481\,T(2, 2x-1))/T(0, 2x-1)$$
$$+ 0.06067214549\,T(1, 2x-1) + 0.01097466398\,T(2, 2x-1)$$
$$+ 0.0005311640964\,T(3, 2x-1))$$

6.2.8. Наилучшая минимаксная аппроксимация

Минимаксная аппроксимация отличается от Паде-аппроксимации минимизацией максимальной абсолютной погрешности во всем интервале аппроксимации. Она использует алгоритм Ремеза (см. ниже) и реализуется следующей функцией:

```
minimax(f, x=a..b, [m,n], w, 'maxerror')
minimax(f, a..b, [m,n], w, 'maxerror')
```

Здесь, помимо уже отмеченных параметров, `w` – процедура или выражение, `maxerror` – переменная, которой приписывается значение minimax-нормы. Ниже дан пример аппроксимации функции $\cos(x)$ в интервале $[-3, 3]$:

```
> minimax(cos(x),x=-3..3,[2,3],1,'minmax');
```

$$\frac{.5458304182 + (.5119634586\ 10^{-9} - .2308484266x)x}{.5217186403 + (.1496104420\ 10^{-9} - .1062847466x)x}$$

```
> minmax;
```

$$.04621605601$$

6.2.9. Наилучшая минимаксная аппроксимация по алгоритму Ремеза

Для получения наилучшей полиномиальной аппроксимации используется алгоритм Ремеза, который в Maple реализуется следующей функцией:

```
remez(w, f, a, b, m, n, crit, 'maxerror')
```

Здесь `w` – процедура, представляющая функцию $w(x) > 0$ в интервале $[a, b]$, `f` – процедура, представляющая аппроксимируемую функцию $f(x)$, `a` и `b` – числа, задающие интервал аппроксимации $[a, b]$, `m` и `n` – степени числителя и знаменателя аппроксимирующей функции, `crit` – массив, индексированный от 1 до $(m + n + 2)$ и представляющий набор оценок в критических точках (то есть точках максимума/минимума кривых погрешности), `maxerror` – имя переменной, которой присваивается минимаксная норма w abs$(f - r)$.

Следующий пример иллюстрирует применение данной функции для аппроксимации функции `erf(x)`:

```
> Digits:=12:w:=proc(x) 1.0 end;
```

$$w := \mathbf{proc}(x)\ 1.0\ \mathbf{end\ proc}$$

```
> f:=proc(x) evalf(erf(x)) end;
```

$$f := \mathbf{pros}(x)\ \mathrm{evalf}(\mathrm{erf}(x))\ \mathbf{end\ proc}$$

```
> crit:=array(1..7,[0,.1,.25,.5,.75,.9,1.]);
```

$$crit := [0, .1, .25, .5, .75, .9, 1.]$$

```
> remez(w,f,0,1,5,0,crit,'maxerror');
```

$$x \to 0.0000221268863 + (1.12678937620 + (0.018447321509 + (-0.453446232421 + (0.141246775527 + 0.00966355213050x)x)x)x)x$$

```
> maxerror;
```

$$0.0000221268894463$$

6.2.10. Другие функции пакета numapprox

Отметим назначение других функций пакета numapprox:

- `chebdeg(p)` – возвращает степень полинома Чебышева `p`;
- `chebmult(p, q)` – умножение полиномов Чебышева `p` и `q`;
- `chebsort(e)` – сортирует элементы ряда Чебышева;
- `confracform(r)` – преобразует рациональное выражение `r` в цепную дробь;
- `confracform(r, x)` – преобразует рациональное выражение `r` в цепную дробь с независимой переменной `x`;
- `hornerform(r)` – преобразует рациональное выражение `r` в форму Горнера;
- `hornerform(r, x)` – преобразует рациональное выражение `r` в форму Горнера с независимой переменной `x`;

- `infnorm(f, x=a...b, 'xmax')` – возвращает L-бесконечную норму функции на отрезке x [a, b];
- `infnorm(f, a...b, 'xmax')` – возвращает L-бесконечную норму функции на отрезке [a, b].

Действие этих функций очевидно, и читатель может самостоятельно опробовать их в работе.

6.3. Пакет приближения кривых CurveFitting

6.3.1. Общая характеристика пакета Curve Fitting

Появившийся в Maple пакет *приближения кривых* CurveFitting весьма полезен тем, кто занимается столь распространенной задачей, как приближение кривых. Он содержит ряд функций:

```
> with(CurveFitting);
[BSpline, BSplineCurve, Interactive, LeastSquares,
   PolynomailInterpolation, RationalInterpolation, Spline,
   ThieleInterpolation]
```

Доступ к функциям пакета возможен с помощью конструкций:

```
CurveFitting[function](arguments)   function(arguments)
```

Число функций пакета невелико, и все они описаны ниже.

6.3.2. Функция вычисления B-сплайнов Bsline

Функция `BSpline(k, v, opt)` служит для вычисления *B-сплайнов*. В отличие от обычных сплайнов, у которых точками стыковки сплайн-функций являются узловые точки, B-сплайны позволяют получить стыковку в произвольно заданных точках. Ввиду громоздкости вывода теоретическое обоснование приближения B-сплайнами опущено, и мы остановимся только на практической части такого приближения.

Указанная функция имеет следующие параметры: `k` – порядок сплайна (целое число), `v` – имя и `opt` – параметр в виде `knots=knotlist`, где `knotlist` – списокк из (k+1) элементов алгебраического типа. Используя функцию `CurveFitting [BSplineCurve]`, можно строить кривые B-сплайнов. Примеры применения этой функции представлены ниже:

```
> BSpline(3, x);
```

$$\begin{cases} 0 & x<0 \\ \frac{1}{2}x^2 & x<1 \\ -\frac{1}{2}+x-(x-1)^2 & x<2 \\ \frac{5}{2}-x+\frac{1}{2}(x-2)^2 & x<3 \\ 0 & 3\le x \end{cases}$$

```
> BSpline(2, x, knots=[0,a,2]);
```

$$\begin{cases} 0 & x<0 \\ \frac{x}{a} & x<a \\ \frac{-x+a}{2-a}+1 & x<2 \\ 0 & 2\le x \end{cases}$$

Как нетрудно заметить из этих примеров, функция `Bspline` возвращает результат в виде кусочных функций типа `piecewise`.

6.3.3. Функция построения B-сплайновых кривых BsplineCurve

Функция `BsplineCurve` служит для построения кривых B-сплайнов (то есть сплайнов с произвольными точками стыковки). Она может использоваться в формах:

```
BSplineCurve(xydata, v, opts)
BSplineCurve (xdata, ydata, v, opts)
```

Здесь:

- `xydata` – список, массив или матрица точек в форме `[[x1,y1],[x2,y2],...,[xn,yn]]`;
- `xdata` – список, массив или вектор значений независимой переменной `[x1,x2,...,xn]`;
- `ydata` – список, массив или вектор значений зависимой переменной в форме `[y1,y2,...,yn]`;
- `v` – имя независимой переменной;
- `opts` – необязательный параметр в форме одного или более выражений вида `order=k` или `knots=knotlist`.

Примеры применения функции `BSplineCurve` с порядком, заданным по умолчанию, и с третьим порядком (кубический B-сплайн) представлены на рис. 6.16.

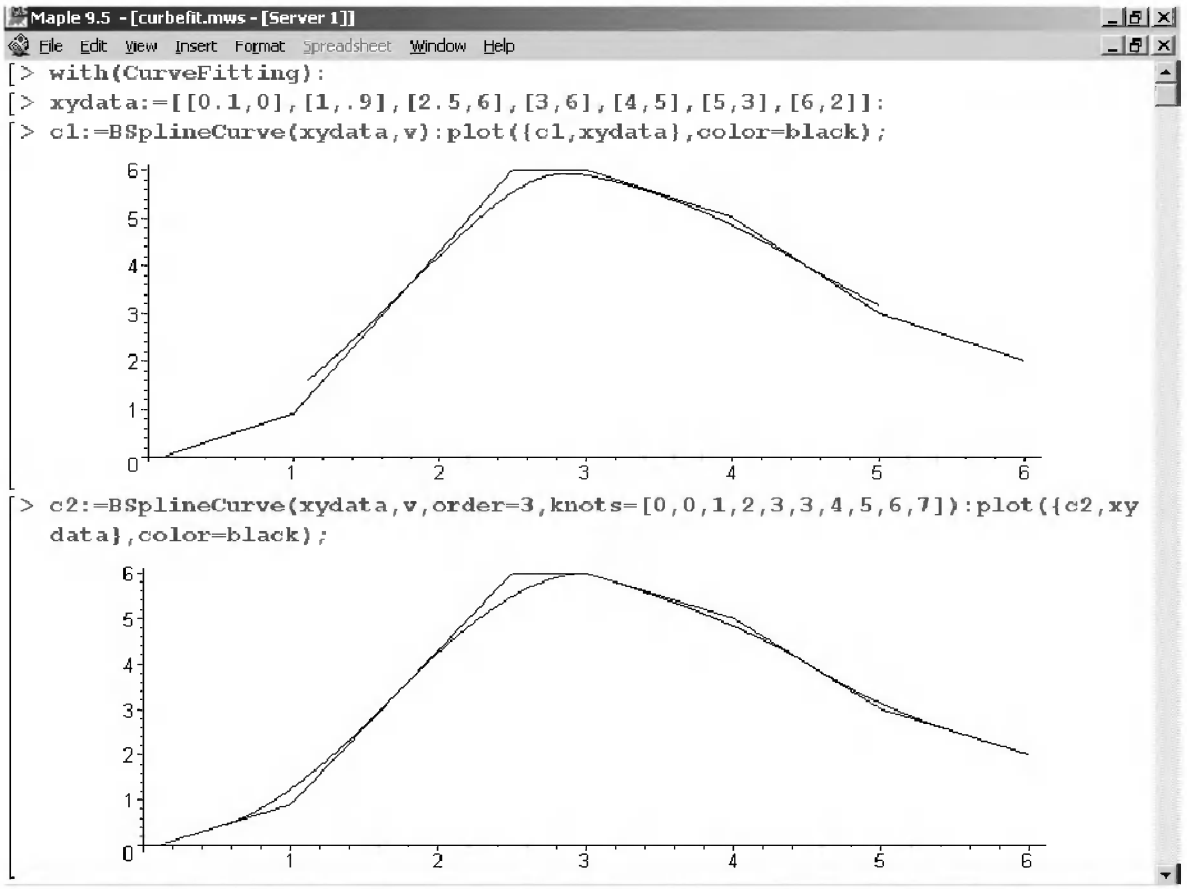


Рис. 6.16. Применение функции BSplineCurve

Следует отметить, что при малом числе точек стыковки аппроксимация B-сплайнами дает невысокую точность, что хорошо видно из рис. 6.16.

6.3.4. Сравнение полиномиальной и сплайновой аппроксимаций

Когда аппроксимируется гладкая функция, представленная парами данных с равномерным расположением узлом, то данные как полиномиальной, так и сплайновой аппроксимаций различаются незначительно. В этом случае применение куда более сложной сплайновой аппроксимации, как правило, кажется мало обоснованным.

Однако если точки данных расположены неравномерно, то применение полиномиальной аппроксимации может оказаться совершенно неприемлемым. Это отчетливо показывает пример, представленный на рис. 6.17. Здесь задана на первый взгляд (судя по расположению точек) не слишком сложная и чуть колебательная зависимость. Однако полиномиальная аппроксимация (представлена тонкой кривой), особенно вначале – в интервале первых трех точек, дает явно ошибочные сильные выбросы. А вот сплайновая аппроксимация (показана более жирной линией) ведет себя куда более приемлемо.

Причина лучшего поведения сплайновой аппроксимации здесь вполне очевидна – напоминая поведение гибкой линейки, сплайновая функция эффективно сглаживает выбросы кривой в промежутках между точками.

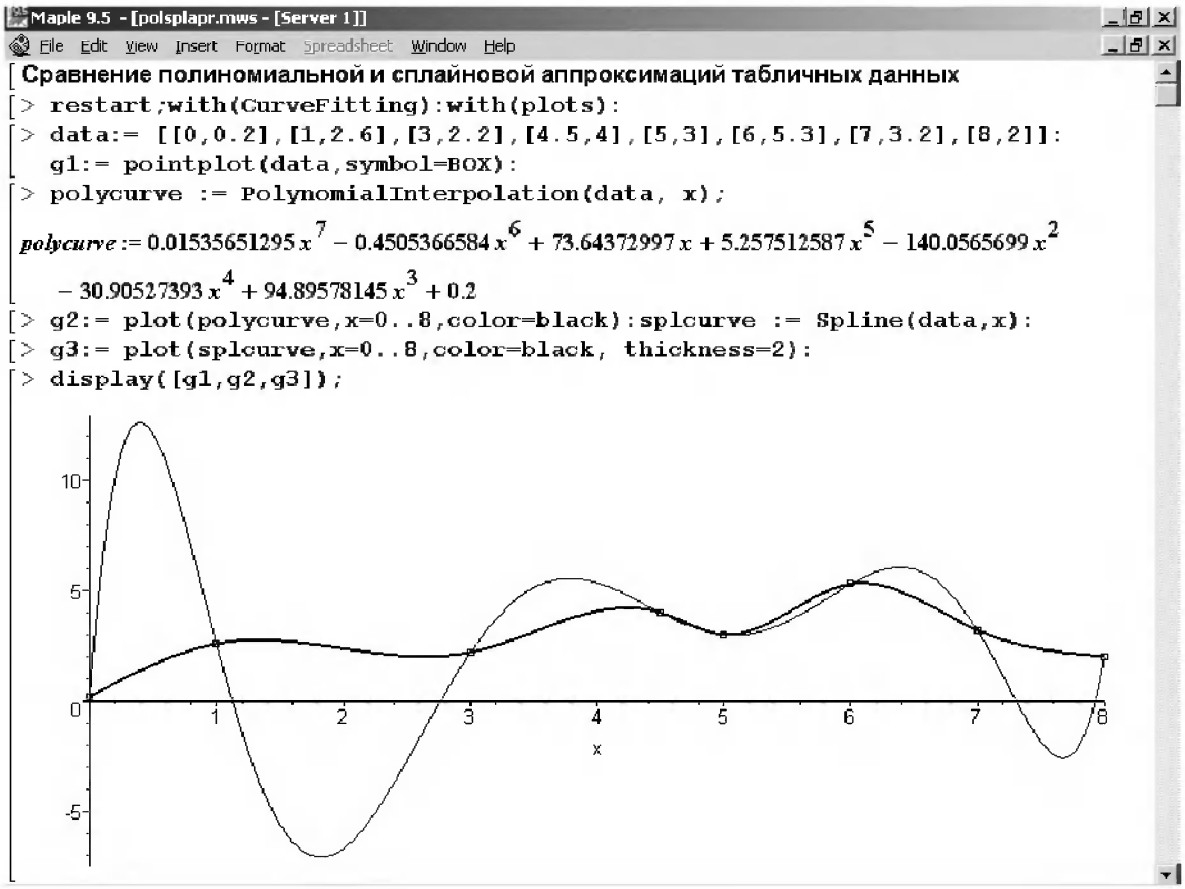


Рис. 6.17. Сравнение полиномиальной и сплайновой аппроксимаций для функции, заданной парами данных при неравномерном расположении узлов

6.3.5. Сплайновая аппроксимация при большом числе узлов

При большом числе узлов (десятки-сотни и выше) данные, представленные точками, выглядят нередко непредставительно. Например, на рис. 6.18 показан документ, иллюстрирующий сплайновую аппроксимацию функции синуса, представленной 31 отчетом, но без вывода графика сплайновой функции. Несмотря на равномерное расположение узлов, по графику точек невозможно определить, что это функция синуса.

Рисунок 6.19 отличается от рис. 6.18 только построением сплайновой функции, представленной графическим объектом g1 (на рис. 6.19 он исключен из параметров функции `display`). После построения графика сплайновой аппроксимирующей функции становится вполне ясным, что точки представляют функцию синуса, которая прекрасно представляется отрезками полиномов сплайн-функции.

Здесь полезно обратить внимание на то, что за пределами области узловых точек значения, возвращаемые сплайновой функцией в пакете CurveFitting, равны нулю. Так что экстраполяция по ней невозможна (в то же время функция spline такой возможностью обладает).

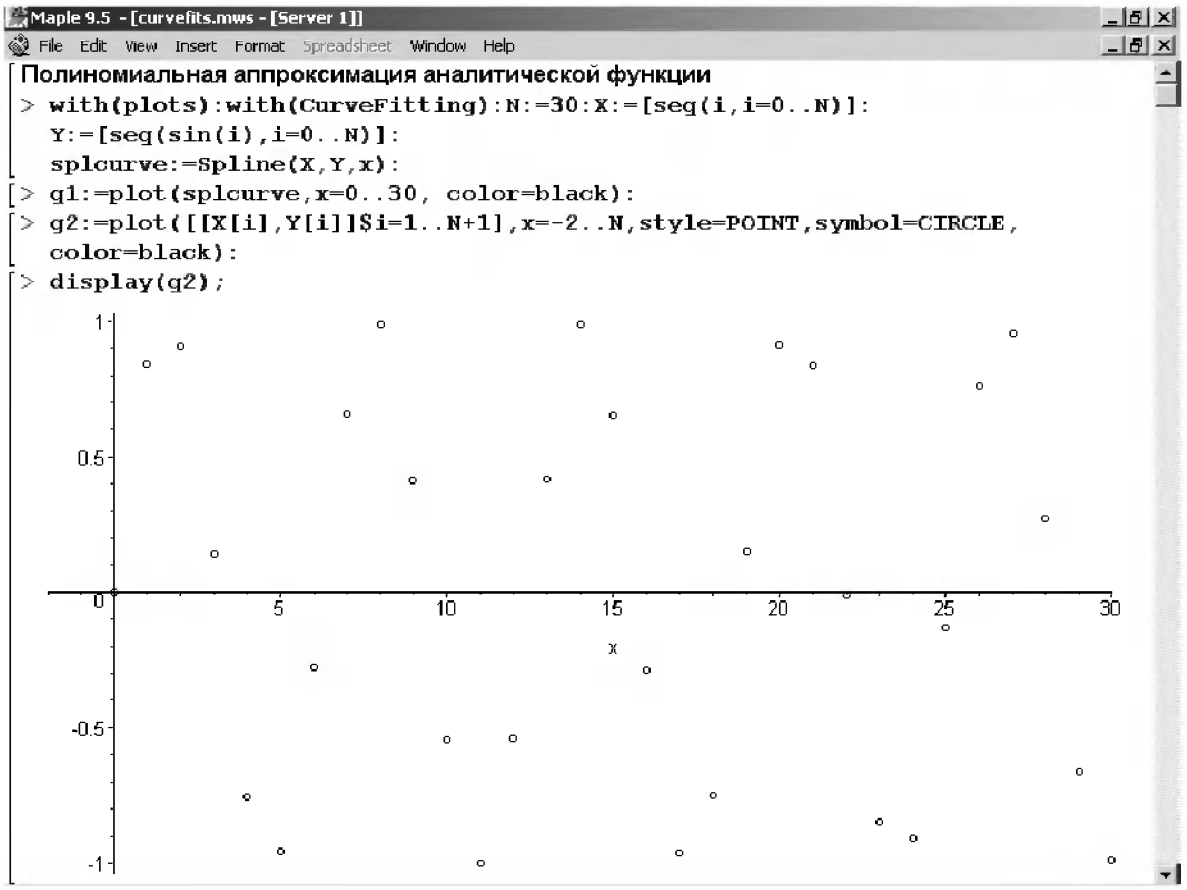


Рис. 6.18. Пример представления функции синуса 31 узловыми точками при равномерном расположении узлов

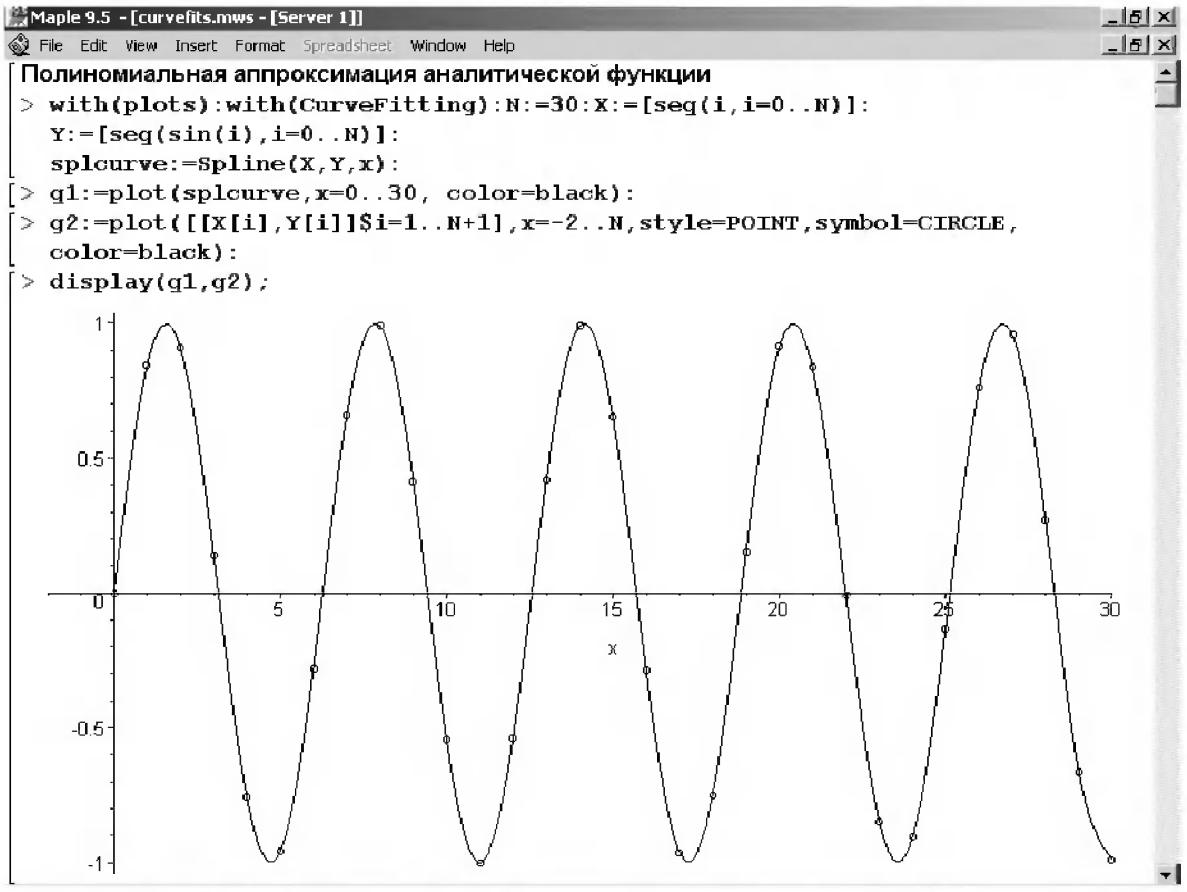


Рис. 6.19. Пример сплайновой аппроксимации синусоидальной функции

6.3.6. Функция полиномиальной аппроксимации

Функция `PolynomialInterpolation` реализует полиномиальную интерполяцию и может использоваться в виде:

```
PolynomialInterpolation (xydata, v)
PolynomialInterpolation(xdata, ydata, v)
```

Параметры функции были определены выше. Параметр v может быть как именем, так и численным значением. Примеры применения функции представлены ниже:

```
> with(CurveFitting):
PolynomialInterpolation([[0,0],[1,2],[2,4],[3,3]], z);
```

$$-\frac{1}{2}z^3+\frac{3}{2}z^2+z$$

```
> PolynomialInterpolation([0,2,5,8], [2,a,1,3], 3);
```

$$-\frac{1}{24}+\frac{5}{6}a$$

6.3.7. Функция рациональной аппроксимации

Функция рациональной интерполяции задается в виде:

```
RationalInterpolation (xydata, z, opts)
RationalInterpolation(xdata, ydata, z, opts),
```

где необязательный параметр `opts` задается выражениями `method=methodtype` или `degrees=[d1,d2]`. Функция возвращает результат в виде отношения двух полиномов. Параметр `methodtype` может иметь значения `'lookaround` или `subresultant`, задающие учет или пропуск сингулярных точек.

Пример применения функции `RationalInterpolation` (загрузка пакета опущена, но предполагается):

```
> xpoints := [0,1,2,3,4,-1]: ypoints := [0,3,1,3,a,1/11]:
f := RationalInterpolation(xpoints, ypoints, x);
```

$$f := -3\frac{x}{4x^2-17x+12}$$

```
> for i from 1 to 6 do normal(eval(f,x=xpoints[i])-ypoints[i]) end do;
```

$$0$$
$$0$$
$$0$$
$$0$$
$$-\frac{3}{2}-a$$
$$0$$

6.3.8. Функция вычисления обычных сплайнов Spline

Функция

Spline(xydata, v, opts) **Spline(xdata, ydata, v, opts)**

вычисляет обычные (не B-типа) сплайны. Примеры ее применения даны ниже:

```
> Spline([[0,1],[1,2],[2,5],[3,3]], x);
```

$$\begin{cases} 1+\frac{2}{15}x+\frac{13}{15}x^3 & x<1 \\ \frac{21}{5}-\frac{142}{15}x+\frac{48}{5}x^2-\frac{7}{3}x^3 & x<2 \\ -\frac{131}{5}+\frac{542}{15}x-\frac{66}{5}x^2+\frac{22}{15}x^3 & otherwise \end{cases}$$

```
> Spline([0,1,2,3], [1,2,5,3], v, degree=1);
```

$$\begin{cases} 1+v & v<1 \\ -1+3v & v<2 \\ 9-2v & otherwise \end{cases}$$

6.3.9. Функция аппроксимации непрерывными дробями

Функция `ThieleInterpolation` осуществляет интерполяцию на основе непрерывных дробей (Thiele's-интерполяцию). Она задается в виде:

ThieleInterpolation (xydata, v)
ThieleInterpolation(xdata, ydata, v)

Примеры применения данной функции представлены ниже:

```
> ThieleInterpolation([[1,3],[2,5],[4,75],[5,4]], x);
```

$$3+\frac{x-1}{\frac{1}{2}+\frac{x-2}{-\frac{1944}{77}+\frac{402}{77}x}}$$

```
> ThieleInterpolation([1,2,a], [2,4,3], 3);
```

$$2+\frac{2}{\frac{1}{2}+\frac{1}{\frac{1-a}{-\frac{3}{2}+a}+2}}$$

6.4. Выбор аппроксимации для сложной функции

6.4.1. Задание исходной функции и построение ее графика

Покажем возможности аппроксимации сложной функции средствами систем Maple на одном из комплексных примеров, давно помещенных в библиотеку пользователей системы Maple V R2, и переработанном для Maple 8 и более поздних версий. Воспользуемся для этого ранее описанными возможностями пакета numapprox, для чего прежде всего подключим его:

```
> restart:with(numapprox):
```

Будем искать приемлемую аппроксимацию для следующей, отнюдь не простой тестовой функции:

```
>  f := x -> int(1/GAMMA(t), t=0..x ) / x^2;
```

$$f := x \to \frac{\int_0^x \frac{1}{\Gamma(t)}\,dt}{x^2}$$

```
> plot(f,0..4,color=black);
```

График этой функции представлен на рис. 6.20. С первого взгляда это простой график, но тут как раз тот случай, когда простота обманчива. Вы сразу заметите, что график строится медленно, поскольку в каждой из множества его точек системе Maple приходится вычислять значение интеграла с подынтегральной функцией, содержащей гамма-функцию. Maple делает это по сложному и медленному алгоритму адаптивного численного интегрирования.

Итак, вычисление $f(x)$ по ее интегральному представлению совершенно не эффективно. Наша цель состоит в разработке процедуры вычислений, которая дала

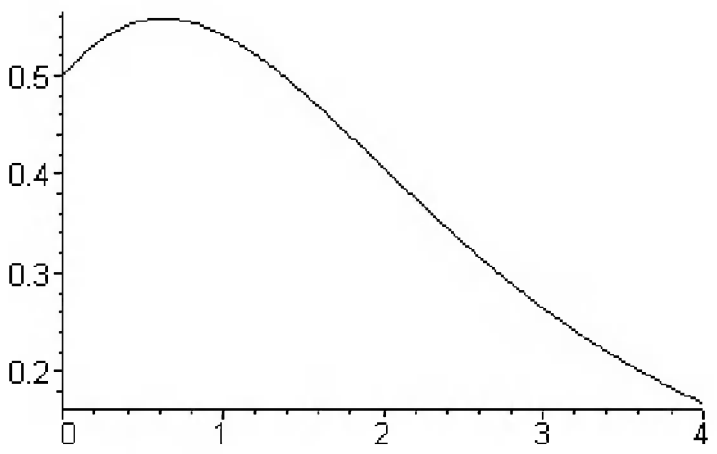


Рис. 6.20. График аппроксимируемой функции

бы 6 точных цифр результата в интервале [0..4] и требовала, по возможности, наименьшего числа арифметических операций для каждого вычисления. Втайне невредно помечтать о том, чтобы после аппроксимации время вычислений уменьшилось бы хотя бы в несколько раз. Что получится на деле, вы увидите чуть позже. А пока войдем в дебри практической аппроксимации.

6.4.2. Аппроксимация рядом Тейлора

Начнем с аппроксимации функции хорошо известным рядом Тейлора степени 8 относительно середины интервала (точки с x = 2):

```
> s := map(evalf, taylor(f(x), x=2, 9));
```

$$s := 0.4065945998 - 0.1655945998(x-2) + 0.00209790791(x-2)^2 + 0.01762626393(x-2)^3 - 0.006207547150(x-2)^4 + 0.00001414211(x-2)^8 + O((x-2)^9)$$

```
> TaylorApprox := convert(s, polynom):
```

Такой ряд позволяет использовать для вычислений только арифметические действия. Для удобства преобразуем аппроксимацию в функцию, чтобы она соответствовала форме, указанной для первоначальной функции $f(x)$. Тогда мы сможем построить график кривой ошибок для аппроксимации полиномом Тейлора:

```
> TaylorApprox := unapply(TaylorApprox, x);
```

$$TaylorApprox := x \rightarrow 0.7197837994 - 0.1565945998x + 0.00209790791(x-2)^2 + 0.01762626393(x-2)^3 - 0.006207547150(x-2)^4 + 0.000573358662(x-2)^5 + 0.00024331162(x-2)^6 - 0.00010010534(x-2)^7 + 0.00001414211(x-2)^8)$$

Кривая ошибок для аппроксимации полиномом Тейлора строится командой

```
> plot(f - TaylorApprox, 0..4,color=black);
```

и имеет вид, представленный на рис. 6.21. Эта кривая нас, прямо скажем, не слишком радует, поскольку погрешность в сотни раз превышает заданную.

Типичное свойство аппроксимации рядом Тейлора состоит в том, что ошибка мала вблизи точки разложения и велика вдали от нее. В данном случае самая большая ошибка имеет место в левой оконечной точке. Чтобы вычислить значение ошибки в точке $x = 0$, что ведет к делению на нуль (см. определение для $f(x)$), мы должны использовать значение предела:

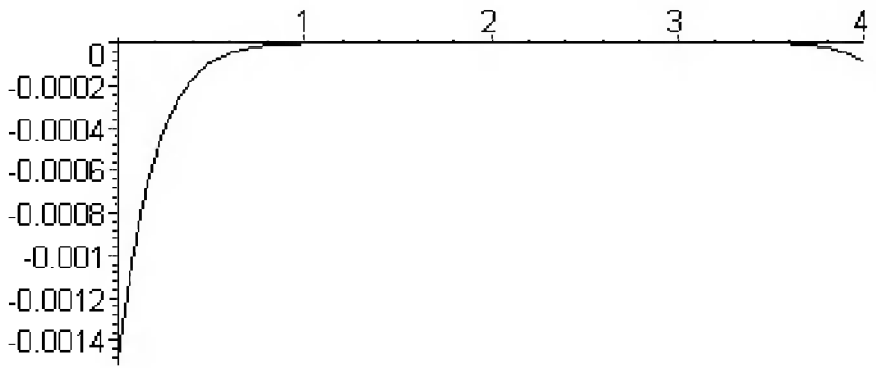


Рис. 6.21. Кривая погрешности при аппроксимации рядом Тейлора

```
> maxTaylorError := abs( limit(f(x), x=0) - TaylorApprox(0) );
```

$$maxTaylorError := 0.0015029608$$

Итак, в самом начале наших попыток мы потерпели полное фиаско, получив совершенно неприемлемое значение погрешности в сотни раз больше заданной.

6.4.3. Паде-аппроксимация

Теперь опробуем рациональную аппроксимацию Паде (Pade) функции $f(x)$ степени (4,4). Приближения по этому разложению будут аппроксимировать функцию более точно, и потому ошибки округления в вычислениях станут более заметными. Поэтому зададим вычисления с двумя дополнительными знаками точности:

```
> Digits := 12:
> s := map(evalf, taylor(f(x), x=2, 9)):
> PadeApprox := pade(s, x=2, [4,4]);
```

$$PadeApprox := (0.341034792604 + 0.0327799035348x - 0.00613783638188(x-2)^2 + 0.00452991113636(x-2)^3 - 0.000431506338862(x-2)^4)/(0.068484906786 + 0.465757546607x + 0.159149610837(x-2)^2 + 0.0266813683828(x-2)^3 + 0.00346967791444(x-2)^4)$$

```
> PadeApprox := unapply(PadeApprox, x):
```

Кривая ошибки для интервала [0,4] строится командой

```
> plot(f - PadeApprox, 0..4,color=black);
```

и имеет вид, показанный на рис. 6.22.

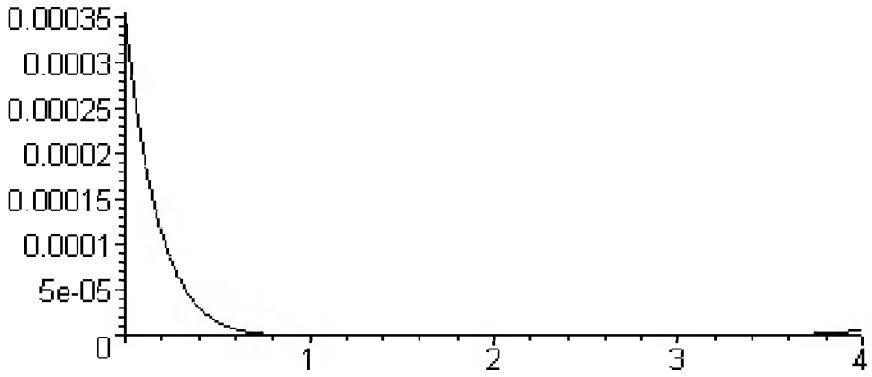


Рис. 6.22. Кривая погрешности при Паде-аппроксимации степени (4,4)

Как и при аппроксимации рядом Тейлора, ошибка здесь мала вблизи точки разложения и велика вдали от нее. Мы снова видим из графика, что для указанной функции самая большая ошибка – в левой оконечной точке. Однако максимальная ошибка в Паде-аппроксимации уже на порядок меньше, чем при аппроксимации полиномом Тейлора:

```
> maxPadeError := abs( limit(f(x), x=0) - PadeApprox(0) );
```

$$maxPadeError := 0.000353777322$$

Это успех, показывающий, что мы на верном пути. Но пока погрешность остается слишком большой по сравнению с заданной.

6.4.4. Аппроксимация полиномами Чебышева

Знатоки техники аппроксимации знают, что лучшие приближения на заданном интервале могут быть получены использованием разложения в ряд Чебышева. Это связано с тем, что ортогональные полиномы Чебышева позволяют получить аппроксимацию, погрешность которой в заданном диапазоне изменения аргумента распределена более равномерно, чем в предшествующих случаях. Выбросы погрешности на краях интервала аппроксимации в этом случае отсутствуют.

Разложим функцию $f(x)$ на [0,4] в ряд Чебышева с точностью $1*10^{-8}$. Это означает, что все члены с коэффициентами меньше, чем эта величина, будут опущены. Такая точность обеспечивается полиномом 13-ой степени:

```
> evalf( limit(f(x), x=0) );
```

$$.500000000000$$

```
> fproc := proc(x) if x=0 then 0.5 else evalf(f(x)) fi end:
> ChebApprox := chebyshev(fproc, x=0..4, 1E-8);
```

$$ChebApprox := 0.379206274272\mathrm{T}\left(0,\frac{x}{2}-1\right)-0.202632813998\mathrm{T}\left(1,\frac{x}{2}-1\right)$$
$$-0.0369064836430\mathrm{T}\left(2,\frac{x}{2}-1\right)+0.0370131431541\mathrm{T}\left(3,\frac{x}{2}-1\right)$$
$$-0.00888944143050\mathrm{T}\left(4,\frac{x}{2}-1\right)-0.000149789336636\mathrm{T}\left(5,\frac{x}{2}-1\right)$$
$$+0.000642974620794\mathrm{T}\left(6,\frac{x}{2}-1\right)-0.000170677949427\mathrm{T}\left(7,\frac{x}{2}-1\right)$$
$$+0.0000126917283881\mathrm{T}\left(8,\frac{x}{2}-1\right)+0.439874928738\ 10^{-5}\ \mathrm{T}\left(9,\frac{x}{2}-1\right)$$
$$-0.156284139876\ 10^{-5}\ \mathrm{T}\left(10,\frac{x}{2}-1\right)+0.204980540664\ 10^{-6}\ \mathrm{T}\left(11,\frac{x}{2}-1\right)$$
$$+0.456254277778\ 10^{-8}\ \mathrm{T}\left(12,\frac{x}{2}-1\right)-0.694323955261\ 10^{-8}\ \mathrm{T}\left(13,\frac{x}{2}-1\right)$$

Можно проверить для этого примера, что кривая ошибки при аппроксимации рядом Чебышева колеблется. Поскольку ряд Чебышева был оборван на члене степени 8 (как и полином ряда Тейлора), то максимальная ошибка оказалась все еще больше заданной.

Для последующих вычислений полезно заметить, что мы можем использовать процедуру для нахождения численных значений $f(x)$, которая будет намного эффективнее, чем прямое определение, которое требует численного интегрирования для каждого значения x. А именно определим процедуру численной оценки, основанную на разложении в ряд Чебышева степени 13, так как максимальная ошибка при такой аппроксимации меньше, чем 10^{-8}, и обеспечивает для нашей цели доста-

точную точность. Мы определим полином Чебышева `T(x)` из пакета `orthopoly` и затем для эффективной оценки преобразуем его в форму Горнера:

```
> F := hornerform( eval(subs(T=orthopoly[T],
        ChebApprox)) );
```

$$F := 0.499999998610 + (0.192405358503 + (-0.163971754264 + (-0.0083861432817 + (0.0277082269676 + (-0.00593172541573 + (-0.00132728874257 + (0.000910057654178 + (-0.000180351181100 + (0.57685696534\ 10^{-5} + (0.448885653549\ 10^{-5} + (-0.990274556116\ 10^{-6} + (0.925433855729\ 10^{-7} - 0.347161977631\ 10^{-8}x)x)x)x)x)x)x)x)x)x)x)x)x)x)x$$

```
> F := unapply(F, x):
```

Схема Горнера минимизирует число арифметических операций, заменяя операции возведения в степень операциями последовательного умножения.

6.4.5. Аппроксимация Чебышева-Паде

Теперь рассмотрим еще более точную рациональную аппроксимацию Чебышева-Паде. Это такая рациональная функция `r [m, n] (x)` с числителем степени `m` и знаменателем степени `n` такой же, как и для разложения в ряд Чебышева. Функция `r [m, n] (x)` согласуется с разложением в ряд ряда Чебышева `f(x)` членом степени `m+n`. Мы вычислим аппроксимацию Чебышева-Паде степени (4, 4), подобную обычной Паде-аппроксимации, успешно выполненной ранее:

```
> ChebPadeApprox := chebpade(F, 0..4, [4,4]);
```

$$ChebPadeApprox := x \to \left(0.285648386001\mathrm{T}\left(0,\frac{1}{2}x-1\right)\right.$$
$$+0.0896033595948\mathrm{T}\left(1,\frac{1}{2}x-1\right)-0.00626546540193\mathrm{T}\left(2,\frac{1}{2}x-1\right)$$
$$\left.+0.00537846740741\mathrm{T}\left(3,\frac{1}{2}x-1\right)-0.000414939186769\mathrm{T}\left(4,\frac{1}{2}x-1\right)\right)\Big/\Big($$
$$\mathrm{T}\left(0,\frac{1}{2}x-1\right)+0.879308974670\mathrm{T}\left(1,\frac{1}{2}x-1\right)+0.289575807084\mathrm{T}\left(2,\frac{1}{2}x-1\right)$$
$$\left.+0.048796334457\mathrm{T}\left(3,\frac{1}{2}x-1\right)+0.00650272206509\mathrm{T}\left(4,\frac{1}{2}x-1\right)\right)$$

Построим кривую ошибок:

```
> with(orthopoly, T):
> plot(F - ChebPadeApprox, 0..4,color=black);
```

Она представлена на рис. 6.23.

Максимальная ошибка и на этот раз имеет место в левой оконечной точке. Величина максимальной ошибки несколько меньше, чем ошибка при аппроксимации рядом Чебышева. Главное преимущество преставления в виде рациональной

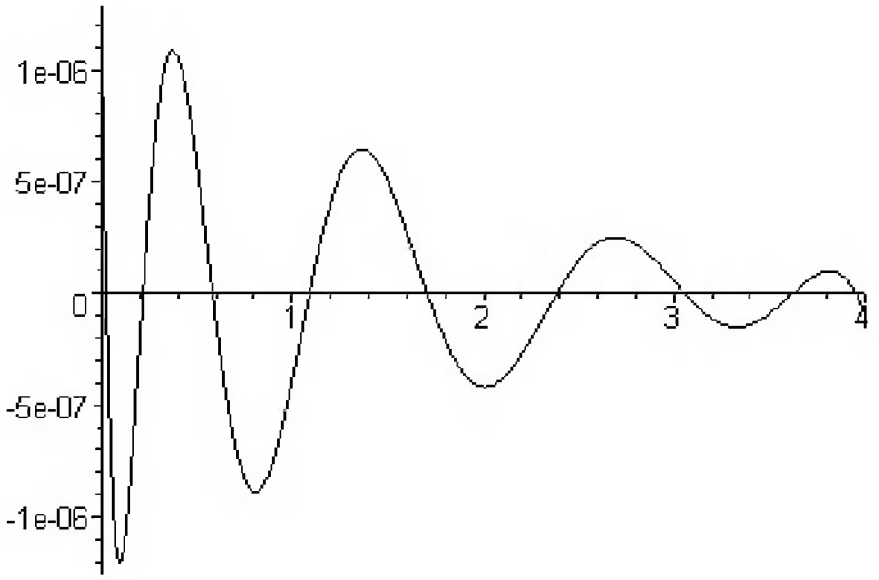


Рис. 6.23. Кривая ошибки при Паде-Чебышева рациональной аппроксимации

функции – высокая эффективность вычислений, которая может быть достигнута преобразованием в непрерывную (цепную) дробь (см. ниже). Однако полученная максимальная ошибка чуть-чуть больше заданной:

```
> maxChebPadeError := abs( F(0) - ChebPadeApprox(0) );
```

$$maxChebPadeError := 0.1236749\ 10^{-5}$$

Мы достигли впечатляющего успеха, и остается сделать еще один шаг в направлении повышения точности аппроксимации.

6.4.6. Минимаксная аппроксимация

Классический результат теории аппроксимации заключается в том, что минимакс как наилучшая аппроксимация рациональной функции степени (m, n) достигается, когда кривая ошибки имеет $(m+n+2)$ равных по величине колебаний. Кривая ошибки аппроксимации Чебышева-Паде имеет нужное число колебаний, но эта кривая должна быть выровнена (по амплитуде выбросов кривой ошибки), с тем чтобы обеспечить наилучшее минимаксное приближение. Эта задача решается с помощью функции `minimax`:

```
> MinimaxApprox := minimax(F, 0..4, [4,4], 1, "maxerror");
```

$$MinimaxApprox := x \rightarrow (0.174933018974 + (0.0833009600964 + (-0.0201933044764 + (0.00368158710678 - 0.000157698045886x)x)x)x)/(0.349866448284 + (0.031945251383 + (0.0622933780130 + (-0.0011478847868 + 0.00336343538021x)x)x)x)$$

Максимальная ошибка в аппроксимации `MinimaxApprox` дается значением переменной `maxerror`. Заметим, что мы наконец достигли нашей цели получения аппроксимации с ошибкой меньшей, чем $1 \cdot 10^{-6}$:

```
> maxMinimaxError := maxerror;
```

$$maxMinimaxError := 0.585028048949\ 10^{-6}$$

Построим график погрешности для данного типа аппроксимации:

```
> plot(F – MinimaxApprox,0..4,color=black);
```

График ошибки, представленный на рис. 6.24, показывает равные по амплитуде колебания.

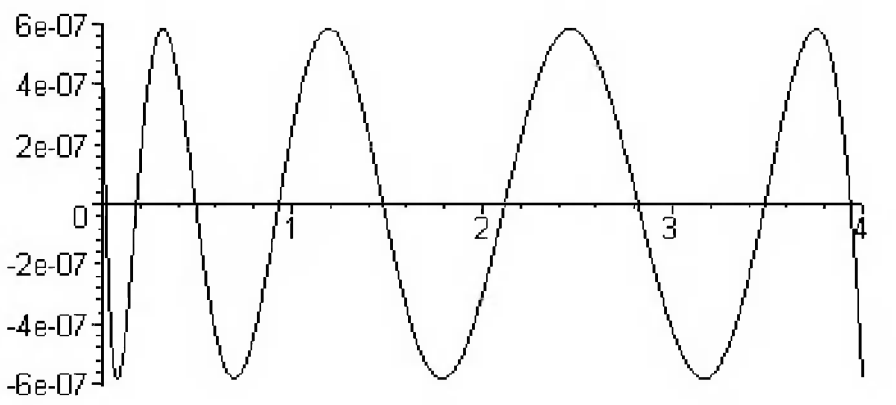


Рис. 6.24. График ошибки при минимаксной аппроксимации

Таким образом, мы блестяще добились успеха в снижении погрешности до требуемого и довольно жесткого уровня.

6.4.7. Эффективная оценка рациональных функций

Полиномы числителя и знаменателя в минимаксной аппроксимации уже выражены в форме Горнера (то есть в форме вложенного умножения). Оценка полиномом степени n в форме Горнера при n умножениях и n суммированиях – это наиболее эффективная схема оценки для полинома в общей форме. Однако для рациональной функции степени (m, n) мы можем делать кое-что даже лучше, чем просто представить выражения числителя и знаменателя в форме Горнера. Так, мы можем нормализовать рациональную функцию так, что полином знаменателя будет со старшим коэффициентом, равным 1. Мы можем также заметить, что вычисление рациональной функции степени (m, n) в форме Горнера требует выполнения всего $(m + n)$ сложений, $(m + n – 1)$ умножений и 1 деления. Другими словами, общий индекс действия есть

- $(m$ + n$)$ операций умножения/деления;
- $(m$ + n$)$ операций сложения/вычитания.

Вычисление рациональной функции можно значительно сократить и далее, преобразуя ее в непрерывную (цепную) дробь. Действительно, рациональная функция степени (m, n) может быть вычислена, используя только

- $\max(m, n)$ операций умножения/деления;
- $(m + n)$ операций сложения/вычитания.

Например, если m=n, тогда эта новая схема требует выполнения лишь половины числа действий умножения/деления по сравнению с предшествующим мето-

дом. Для рациональной функции MinimaxApprox вычисление в форме, выраженной выше, сводится к 9 действиям умножения/деления и 8 действиям сложения/вычитания. Число операций умножения/деления можно сократить до 8, нормализуя знаменатель к форме monic. Мы можем теперь вычислить непрерывную (цепную) дробь для той же самой рациональной функции. Вычисления по этой схеме, как это можно видеть из вывода Maple, сводятся только к 4 действиям деления и 8 действиям сложения/вычитания:

```
> MinimaxApprox := confracform(MinimaxApprox):
> lprint(MinimaxApprox(x));
-.468860043555e-1+ 1.07858988373/
(x+4.41994160718+16.1901836591/(x+4.29118998064+70.1943521765/
(x-10.2912531257+4.77538954280/(x+1.23883810079))))
```

6.4.8. Сравнение времен вычислений

Теперь определим время, необходимое для вычисления функции $f(x)$ в 1000 точек, используя первоначальное интегральное определение, и сравним его с временем, требующимся для схемы MinimaxApprox в виде непрерывной дроби. Сделаем это для системы Maple 8. Так как наше приближение будет давать только 6 точных цифр, мы также потребуем 6 точных цифр и от интегрального представления функции:

```
> Digits := 6: st := time():
> seq( evalf(f(i/250.0)), i = 1..1000 ):
> oldtime := time() - st;
```

$$oldtime := 4.075$$

В процессе вычислений с использованием представления рациональной функции в виде непрерывной дроби иногда требуется внести несколько дополнительных цифр точности для страховки. В данном случае достаточно внести две дополнительные цифры. Итак, новое время вычислений:

```
> Digits := 8: st := time():
> seq( MinimaxApprox(i/250.0), i = 1..1000 ):
> newtime := time() - st;
```

$$newtime := 0.342$$

Ускорение вычисления при аппроксимации есть:

```
> SpeedUp := oldtime/newtime;
```

$$SpeedUp := 11.915205$$

Мы видим, что процедура вычислений, основанная на MinimaxApprox, выполняется почти в 12 раз быстрее процедуры с использованием исходного интегрального определения. Это серьезный успех, полностью оправдывающий время, потерянное на предварительные эксперименты по аппроксимации и ее оптимизации! Заметим, однако, что при применении новых реализаций Maple и современных ПК достигнутый выигрыш может быть менее значительным.

6.4.9. Преобразование в код ФОРТРАНа или C

Один из поводов разработки эффективной аппроксимации для вычисления математической функции заключается в создании библиотек подпрограмм для популярных языков программирования высокого уровня, таких как ФОРТРАН или C. В Maple имеются функции преобразования на любой из этих языков. Например, мы можем преобразовывать формулу для минимаксной аппроксимации в код ФОРТРАНа:

```
> fortran(MinimaxApprox(x));
```

$$fortran\left(-0.0468860043555+1.07858988373\Big/\left(x+4.41994160718+\frac{16.1901836591}{x+4.29118998064+\dfrac{70.1943521765}{x-10.2912531257+\dfrac{4.77538954280}{x+1.23883810079}}}\right)\right)$$

Аналогично можно преобразовать выражения в коды других языков программирования.

6.5. Регрессия в Maple

6.5.1. Функция реализации метода наименьших квадратов LeastSquares

Функция `LeastSquares` в пакете CurveFitting служит для реализации аппроксимации в Maple по методу наименьших квадратов. При этом методе происходит статистическая обработка данных (самих по себе или представляющих аналитическую функцию) исходя из минимума *среднеквадратической погрешности* для всех отсчетов. Эта функция реализуется в формах:

```
LeastSquares (xydata, v, opts)
LeastSquares(xdata, ydata, v, opts)
```

Все входящие в нее параметры были определены выше (см. параметры функции `BSplineCurve`). Параметр `opts` задается в форме выражений `weight=wlist`, `curve=f` или `params=pset`.

Следующие примеры иллюстрируют применение функции `LeastSquares`:

```
> with(CurveFitting):
LeastSquares([[0,.5],[1,2],[2,4],[3,8]], v);
```

$$-.500000000000 + 2.44999999999999974v$$

```
> LeastSquares([0,1,2,3], [1,2,4,6], v, weight=[1,1,1,10]);
```

$$\frac{95}{146}+\frac{259}{146}v$$

```
> LeastSquares([0,1,3,5,6], [1,-1,-3,0,5], v, curve=a*v^2+b*v+c);
```

$$\frac{856}{637}+\frac{33}{49}v^2-\frac{2231}{637}v$$

6.5.2. Функция fit для регрессии в пакете stats

Для проведения регрессионного анализа служит также функция `fit` из пакета `stats`, которая вызывается следующим образом:

```
stats[fit,leastsquare[vars,eqn,parms]](data)
```

или

```
fit[leastsquare[vars,eqn,parms]](data),
```

где `data` – список данных, `vars` – список переменных для представления данных, `eqn` – уравнение, задающее аппроксимирующую зависимость (по умолчанию линейную), `parms` – множество параметров, которые будут заменены вычисленными значениями.

6.5.3. Линейная и полиномиальная регрессия с помощью функции fit

На приведенных ниже примерах показано проведение регрессии с помощью функции `fit` для зависимостей вида $y(x)$:

```
> with(stats):Digits:=5;
```

$$\text{Digits} := 5$$

```
> fit[leastsquare[[x,y]]]([[1,2,3,4],[3,3.5,3.9,4.6]]);
```

$$y = 2.4500 + .52000x$$

```
> fit[leastsquare[[x,y], y=a*x^2+b*x+c]]([[1,2,3,4],
[1.8,4.5,10,16.5]]);
```

$$y = 0.9500000000x^2 + 0.2100000000x + 0.5500000000$$

В первом примере функция регрессии не задана, поэтому реализуется простейшая линейная регрессия, а функция `fit` возвращает полученное уравнение регрессии для исходных данных, представленных списками координат узловых точек. Это уравнение аппроксимирует данные с наименьшей среднеквадратичной погрешностью. Во втором примере задано приближение исходных данных степенным многочленом второго порядка. Вообще говоря, функция `fit` обеспечива-

ет приближение любой функцией в виде полинома, осуществляя полиномиальную регрессию.

Рисунок 6.25 показывает регрессию для одних и тех же данных полиномами первой, второй и третьей степеней с построением их графиков и точек исходных данных.

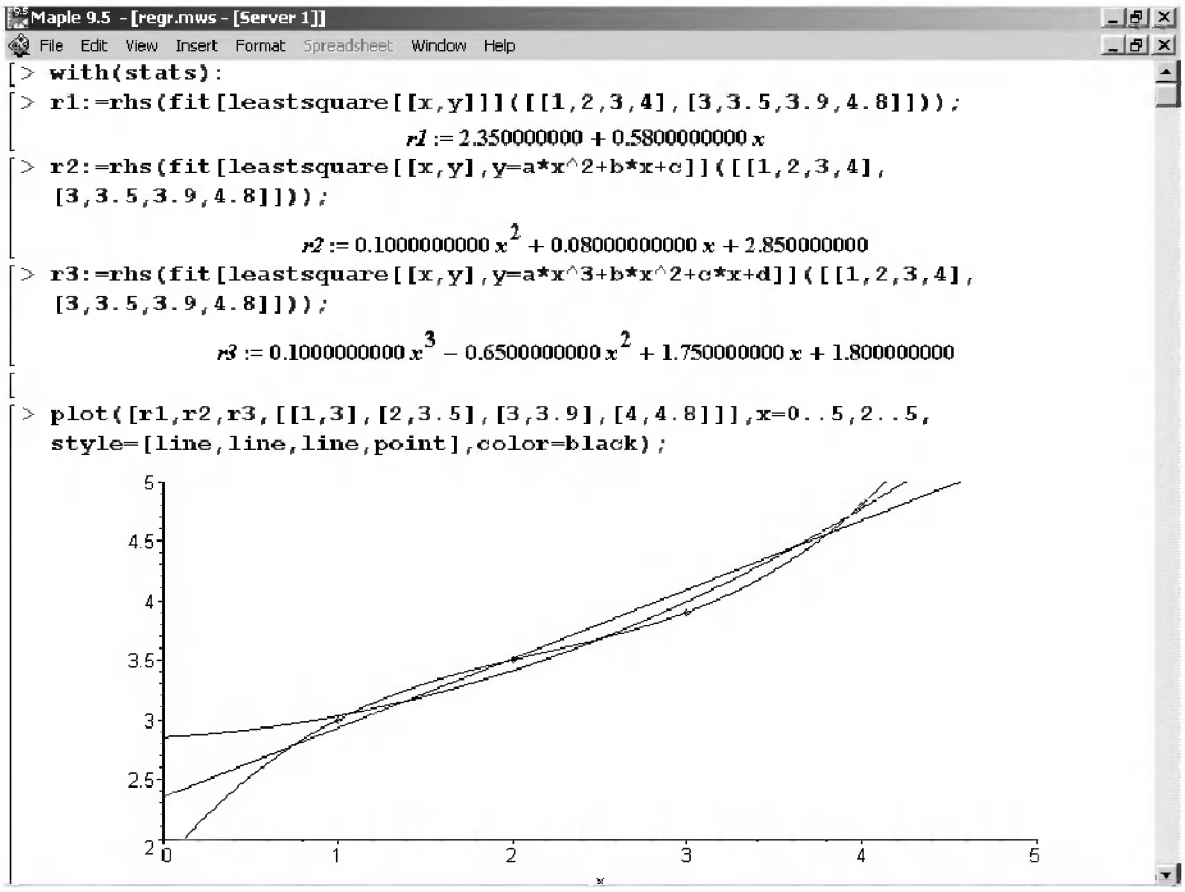


Рис. 6.25. Примеры регрессии полиномами первой, второй и третьей степеней

Нетрудно заметить, что лишь для полинома третьей степени точки исходных данных точно укладываются на кривую полинома, поскольку в этом случае (4 точки) регрессия превращается в полиномиальную аппроксимацию. В других случаях точного прохода линии регрессии через узловые точки нет, но обеспечивается минимум среднеквадратической погрешности для всех точек – следствие реализации метода наименьших квадратов.

Применение регрессии обычно оправдано при достаточно большом числе точек исходных данных. При этом регрессия может использоваться для *сглаживания* данных.

6.5.4. Регрессия для функции ряда переменных

Функция `fit` может обеспечивать регрессию и для функций нескольких переменных. При этом надо просто увеличить размерность массивов исходных данных. В качестве примера ниже приведен пример регрессии для функции двух переменных

```
> f:=fit[leastsquare[[x,y,z],z=a+b*x+c*y,{a,b,c}]]\
([[1,2,3,5,5],[2,4,6,8,8],[3,5,7,10,Weight(15,2)]]);
```

$$f := z = 1 + 13/3x - 7/6y$$

```
> fa:=unapply(rhs(f),x,y);
```

$$fa := (x, y) \rightarrow 1 + 13/3x - 7/6y$$

```
> fa(1.,2.);
```

$$2.999999999$$

```
> fa(2,3);
```

$$37/6$$

В данном случае функция регрессии задана в виде $z = a + b\,x + c\,y$. Для получения вычисляемого выражения для построения графика она преобразуется в функцию двух переменных `fa(x,y)` путем отделения правой части выражения для функции `f`. После этого возможно вычисление значений функции `fa(x,y)` для любых заданных значений `x` и `y`.

6.5.5. Линейная регрессия общего вида

Функция fit может использоваться и для выполнения линейной регрессии общего вида:

$f(x) = a*f1(x) + b*f2(x) + c*f3(x)+....$

Функция такой регрессии является линейной комбинацией ряда функций $f1(x)$, $f2(x)$, $f3(x)$,..., причем каждая их них может быть и нелинейной, например экспоненциальной, логарифмической, тригонометрической и т. д. Пример линейной регрессии общего вида представлен на рис. 6.26.

6.5.6. Нелинейная регрессия

К сожалению, функция `fit` неприменима для нелинейной регрессии. При попытке ее проведения возвращается структура процедуры, но не результат регрессии – см. пример ниже:

```
> fit[leastsquare[[x,y], y=a*2^(x/b),{a,b}]]([[1,2,3,4],
[1.1,3.9,9.5,16.25]]);
```

$$fit_{leastsquare[x,y],y=a2^{\left(\frac{x}{b}\right)},\{a,b\}}([[1, 2, 3, 4], [1.1, 3.9, 9.5, 15.25]])$$

Прямых функций для выполнения нелинейной регрессии общего вида в ранних версиях Maple не было. Но большинство нелинейных зависимостей удается свести к линейным с помощью простых линеаризирующих преобразований. На рис. 6.27 показан пример экспоненциальной регрессии $f(x) = a*e^{b*x}$, которая (благодаря логарифмированию точек y) сводится к линейной регрессии. Детали преобразований даны в документе рис. 6.27. Используя другие преобразования, этот документ легко приспособить для выполнения других видов нелинейной регрессии, например степенной или логарифмической.

Кроме того, на интернет-сайте корпорации Waterloo Maple можно найти файлы simplenl.mws и gennlr.mws с процедурами и примерами линейной и нелиней-

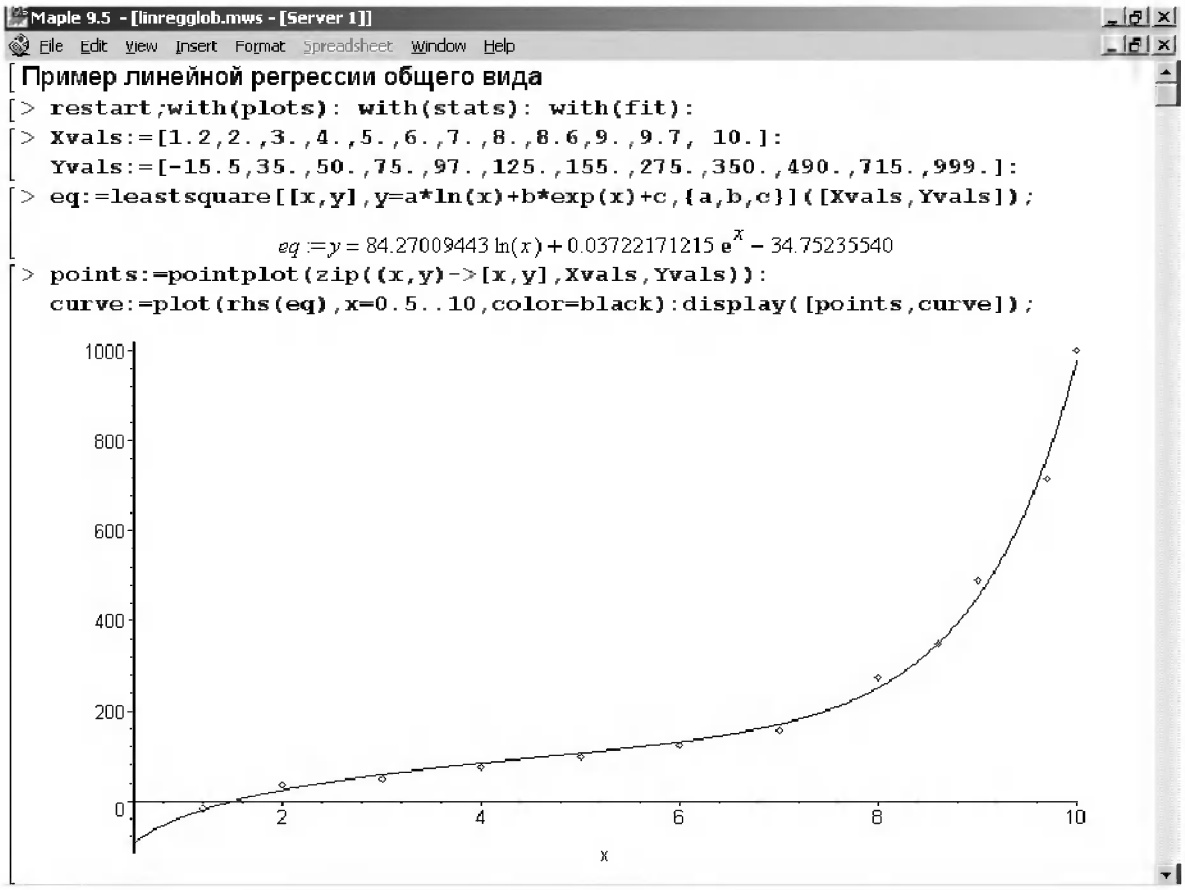


Рис. 6.26. Пример выполнения линейной регрессии общего вида

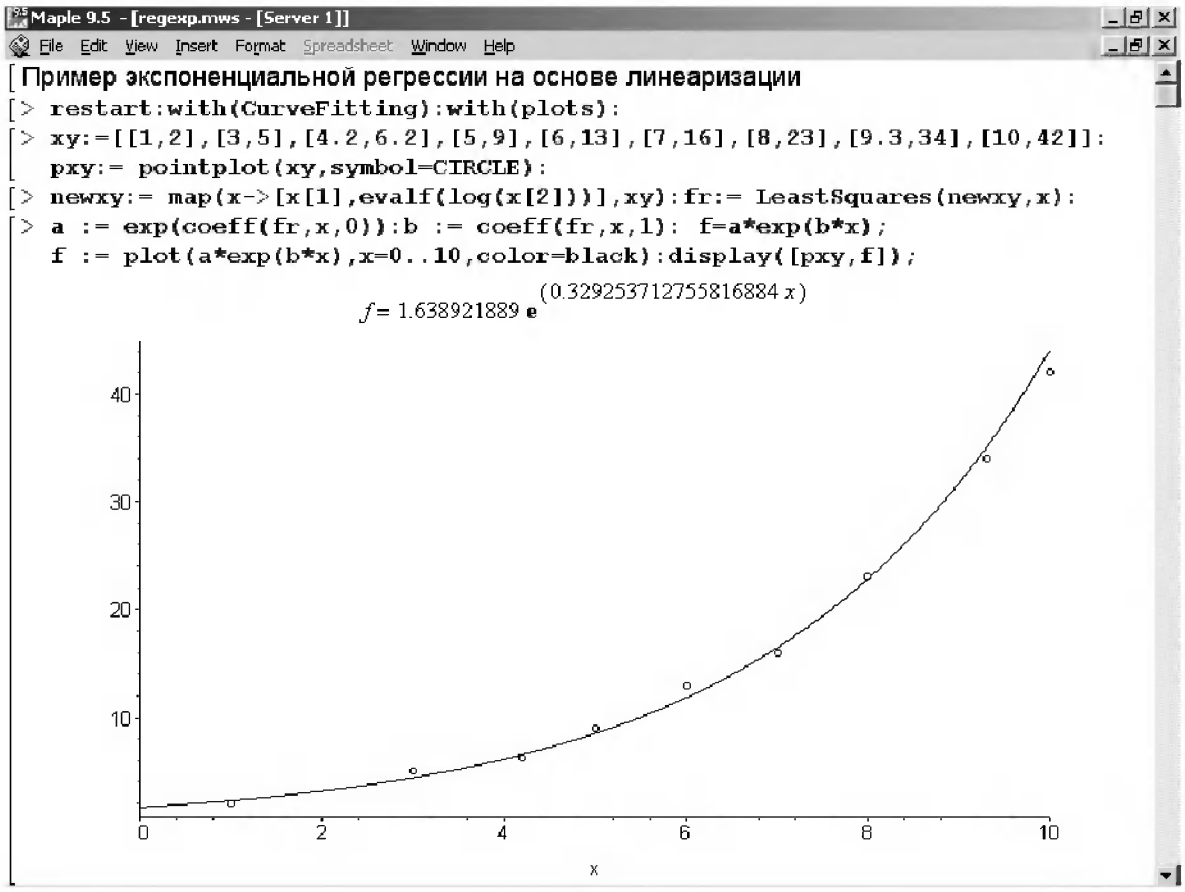


Рис. 6.27. Пример экспоненциальной регрессии

ной регрессий общего вида. Интересная реализация нелинейной регрессии для кусочной функции дается в файле nonelinearpiecewise.mws. К сожалению, эти документы слишком громоздки для описания их в данной книге.

6.5.7. Сплайновая регрессия с помощью функции BSplineCurve

Функция BSplineCurve из пакета CurveFitting может использоваться для реализации *сплайновой регрессии*. Пример этого представлен на рис. 6.28. Опция order задает порядок B-сплайнов, который на 1 меньше заданного целого значения.

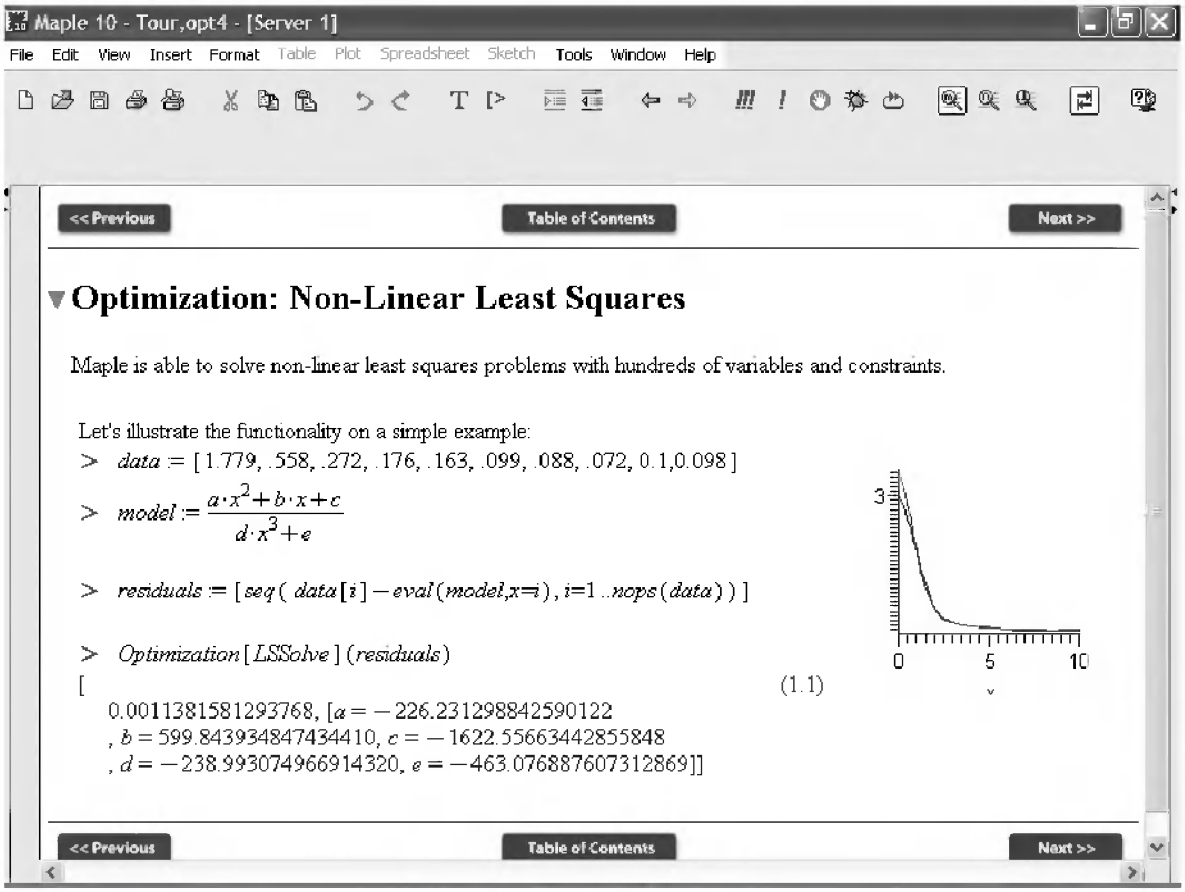


Рис. 6.28. Пример выполнения сплайновой регрессии B-сплайнами

Функция `BsplineCurve` выглядит несколько недоделанной. Так, при `order=3` и 4 кривая регрессии не дотягивает до концевых точек, а при установке `order=1` все точки соединяются отрезками прямых – в том числе концевые. Так что использовать эту функцию для экстраполяции нельзя.

6.6. Аппроксимация в системе Mathematica 4/5

Система Mathematica 4/5 имеет сравнимый с Maple 9.5/10, хотя и немного иной, набор средств для проведения интерполяции и аппроксимации.

6.6.1. Аппроксимация разложением аналитической функции в ряд

Простейший способ аппроксимации аналитической функции в системе Mathematica – разложение функциональной зависимости в ряд с помощью функции `series` (см. пример на рис. 6.29). В этом примере использована функция `Collect` для получения результата разложения в обычной форме, допускающей вычисления.

На рис. 6.29 представлены графики синусоиды, построенной по аналитическому выражению, и график разложения ее в ряд Тейлора в окрестности точки x0 = 2. Хорошо заметно расхождение за пределами области, примыкающей к опорной точке функции. Как отмечалось, погрешность уменьшается, если x0 = 0 (ряд Маклорена).

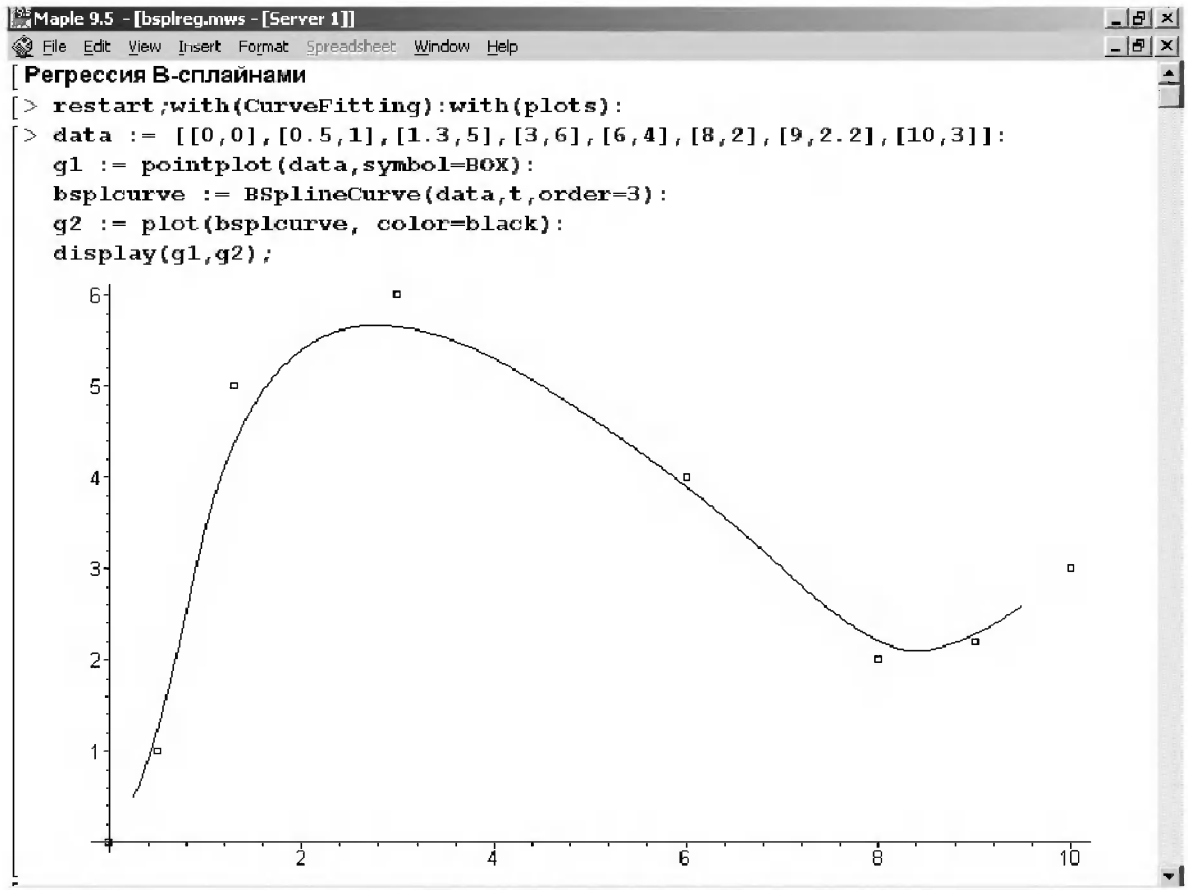


Рис. 6.29. Представление синусоидальной функции рядом Тейлора с графической иллюстрацией его точности

6.6.2. О разложении в ряд при большом числе членов

Все системы компьютерной алгебры (включая Mathematica) способны вести вычисления с применением рациональных чисел в виде отношения целых чисел. При этом они используют аппарат точной арифметики, что сводит вычислитель-

ную погрешность к нулю. В результате погрешность разложения в ряд может быть очень малой и число членов ряда можно резко увеличить.

Рисунок 6.30 показывает разложение функции синуса относительно точки x = 0 при удержании в разложении 32 членов ряда. Как показывает график синусоиды и график ее разложения в ряд, по крайней мере 4 периода синусоиды вычисляются на глазок практически точно. На это указывает и график абсолютной погрешности приближения, представленный ниже. Этот график позволяет судить о погрешности куда более точно – из него видно, что при двух периодах синусоиды (по периоду слева и справа от точки x = 0) погрешность действительно очень мала.

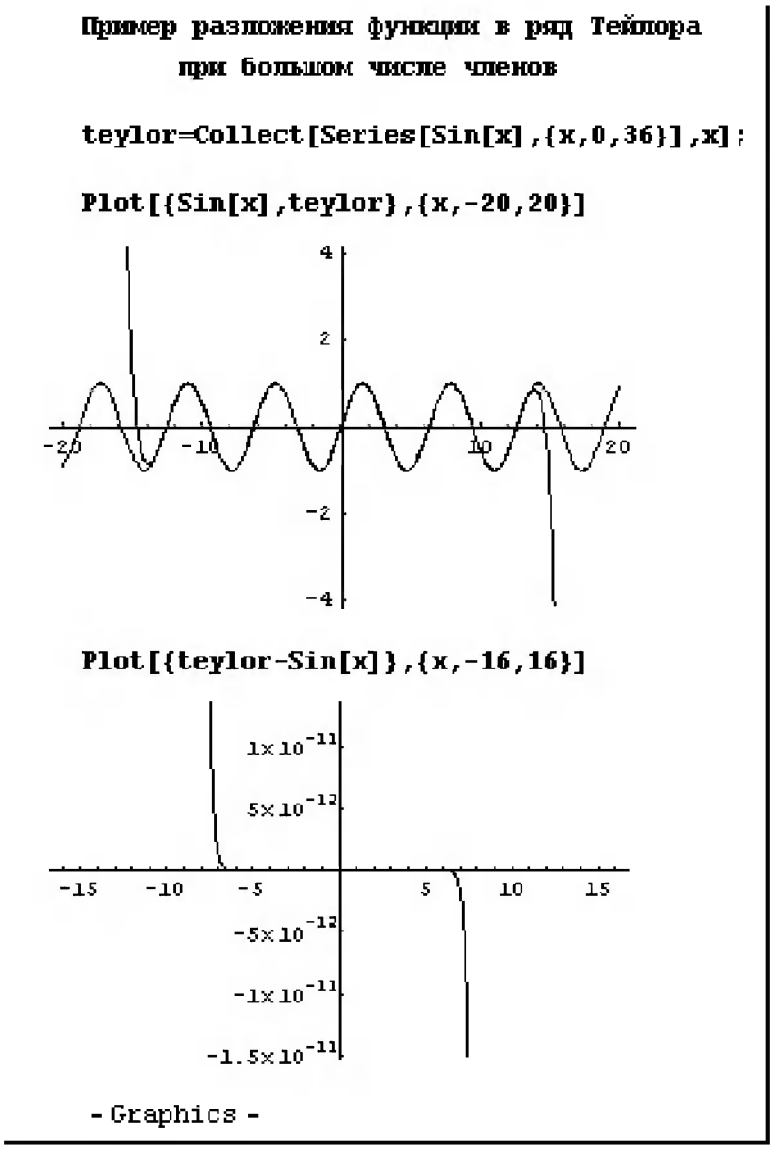


Рис. 6.30. Пример разложения функции синуса в ряд Маклорена при 32 членах ряда

Отмеченная возможность может дать «второе дыхание» методам приближения функций по их разложениям в ряд. Обратите внимание на то, что вывод длинного ряда на рис. 6.30 заблокирован точкой с запятой. Переменная-функция teylor обеспечивает вычисление функции синуса по ее разложению в ряд.

6.6.3. Полиномиальная интерполяция

Для реализации полиномиальной аппроксимации (интерполяции) в СКМ Mathematica 4/5 используются следующие функции:

- **InterpolatingFunction[range, table]** – возвращает объект – интерполирующую функцию, позволяющую вычислять промежуточные значения в заданном диапазоне **range** для таблицы **table**.
- **InterpolatingPolynomial[data, var]** – возвращает полином по переменной **var**, значения которого в узловых точках точно совпадают с данными из списка **data**. Он может иметь форму {{x1, f1}, {x2, f2}, ...} или {f1, f2, ...}, тогда во втором случае xi принимают значения 1, 2, ... Вместо fi может быть{fi, dfi, ddfi, ...}, указывая производные в точках xi.
- **Interpolation[data]** – конструирует объект **InterpolatingFunction.**
- **InterpolationOrder** – опция к **Interpolation**, которая указывает степень подходящего полинома.

Применение основной функции **Interpolation** поясняет следующий пример:

```
data=Table[{x,x^2+1},{x,1,5}]
 {{1,2},{2,5},{3,10},{4,17},{5,26}}
funi=Interpolation[data]
 InterpolatingFunction[{{1,5}},<>]
{funi[1.5],funi[3],funi[4.5]}
 {3.25,10,21.25}
```

Таким образом, на заданном отрезке изменения x функция **Interpolation** позволяет найти любое промежуточное значение функции **funi[x]**, в том числе значения в узловых точках.

Теперь рассмотрим полиномиальную аппроксимацию, при которой полином ищется в явном виде и используется в качестве приближающей функции для данных, представленных списком координат узловых точек некоторой зависимости. Степень полинома в этом случае всегда на 1 меньше числа узловых точек интерполяции или аппроксимации. Пример на рис. 6.31 иллюстрирует технику проведения полиномиальной аппроксимации с применением интерполирующего степенного многочлена.

Как и следовало ожидать, степень аппроксимирующего многочлена равна трем, поскольку заданы всего четыре пары данных. На рис. 6.31 представлено также сравнение данных полиномиальной аппроксимации с исходными данными. Исходные данные построены на графике в виде точек, а зависимость, представленная аппроксимирующим полиномом, построена на графике сплошной линией.

Полезно обратить внимание на технику построения графика аппроксимирующей функции одновременно с построением узловых точек. На рис. 6.31 для этого используются два графических объекта: g1 – график полинома, g2 – график узловых точек, представленных списком data, и строящийся графической функцией ListPlot с опцией, задающей размеры точек. Функция **Show[g1, g2]** строит объекты g1 и g2 как порознь, так и совместно. Построения отдельно объектов из рис. 6.6 удалены, оставлен только график полинома вместе с узловыми точками. Этот прием мы будем в последующем использовать неоднократно.

■ Полиномиальная аппроксимация с визуализацией

```
In[93]:= data := {{0, .9}, {2, 8.1}, {3, 17}, {4, 33}}

In[95]:= p[x_] = Collect[InterpolatingPolynomial[data, x], x]

Out[95]= 0.9 + 2.74167 x - 0.4625 x^2 + 0.445833 x^3

In[96]:= g1 := Plot[p[x], {x, 0, 4}]

In[100]:= g2 := ListPlot[data, PlotStyle -> {PointSize[0.02]}]

In[101]:= Show[g1, g2, PlotRange -> {0, 35}];
```

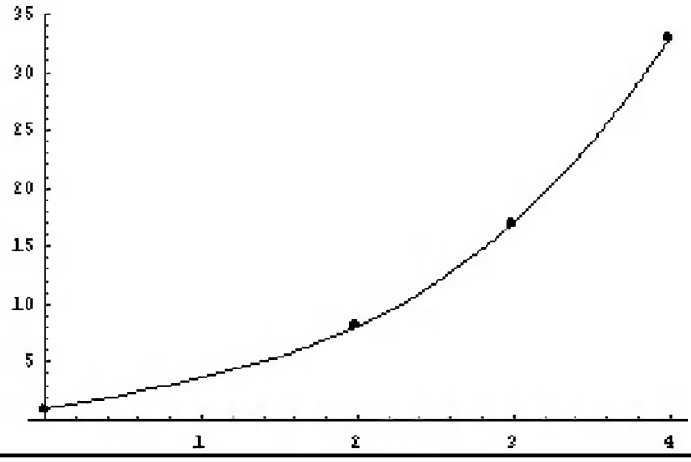


Рис. 6.31. Полиномиальная аппроксимация таблично заданных данных

6.6.4. Полиномиальная аппроксимация специальных функций

Рассмотрим полиномиальную аппроксимацию функции Бесселя, для чего вначале зададим саму функцию в интервале [0,10], построим таблицу ее 11 значений и график функции.

```
f[x_]=BesselJ[1,x]+1;
data=Table[{x,N[f[x]]},{x,0,10}];
g1:=Plot[f[x],{x,0,10}];g1
```

График функции представлен на рис. 6.32. Сама функция Бесселя колеблется относительно нуля, что заметно усложняет аппроксимацию. Поэтому в данном случае к ее значению прибавлен член 1 (его всегда можно вычесть из аппроксимирующего выражения и получить функцию, приближающую функцию Бесселя).

Теперь выполним полиномиальную аппроксимацию функции f(x) и построим (функцией Show) график исходных точек и график полинома:

```
g2:=ListPlot[data,PlotStyle->{PointSize[0.02]}]
p[x_]=Collect[InterpolatingPolynomial[data,x],x]
1. + 0.495892x + 0.0128994x^2 - 0.0794126x^3 + 0.0123736x^4 - 0.00303646x^5 +
0.00167845x^6 - 0.000382565x^7 + 0.0000406462x^8 - 2.08648×10^-6 x^9 + 4.22174×10^-8 x^10
```

```
ip = InterpolatingPolynomial[{3, 6, 4, {7, -1}, 9}, x]
```

$$3+\left(3+\left(-\frac{5}{2}+\left(\frac{5}{3}+\left(-\frac{59}{36}+\frac{35}{36}\,(-4+x)\right)(-4+x)\right)(-3+x)\right)(-2+x)\right)(-1+x)$$

```
ipe[x_] = Expand[ip]
```

$$-\frac{443}{3}+\frac{2957\,x}{9}-\frac{3055\,x^2}{12}+\frac{3275\,x^3}{36}-\frac{61\,x^4}{4}+\frac{35\,x^5}{36}$$

```
dipe[x_] = D[ipe[x], x]
```

$$\frac{2957}{9}-\frac{3055\,x}{6}+\frac{3275\,x^2}{12}-61\,x^3+\frac{175\,x^4}{36}$$

```
dipe[4]
-1
Show[Plot[ipe[x], {x, 0, 6}], ListPlot[{3, 6, 4, 7, 9}]]
```

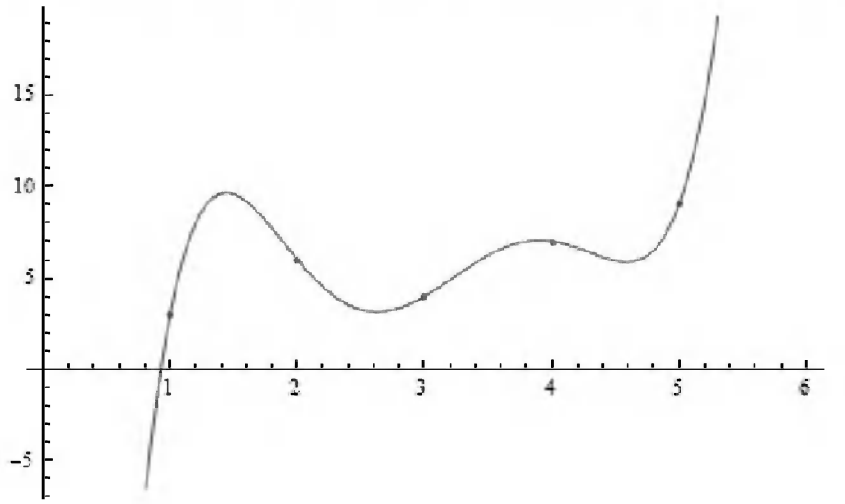


Рис. 6.32. График функции f(x)

```
g3:=Plot[p[x],{x,0,10}]
Show[g3,g2]
```

Как видно из рис. 6.33, график полинома степени 10 очень похож на график аппроксимируемой функции (рис. 6.32) и практически точно (в пределах видимости) проходит через узловые точки.

Судить о точности аппроксимации можно по графику абсолютной погрешности, которую можно представить разностью p(x)–f(x). Для построения такого графика (рис. 6.34) воспользуемся функцией **Plot**:

```
Plot[(p[x]-f[x]),{x,0,10},PlotRange→{-0.0008,0.0001}]
```

Результат аппроксимации вполне удовлетворительный. Максимальная погрешность менее 0.001 (или 0,1%). Для ряда практических расчетов (но далеко не для всех) эта погрешность вполне приемлема.

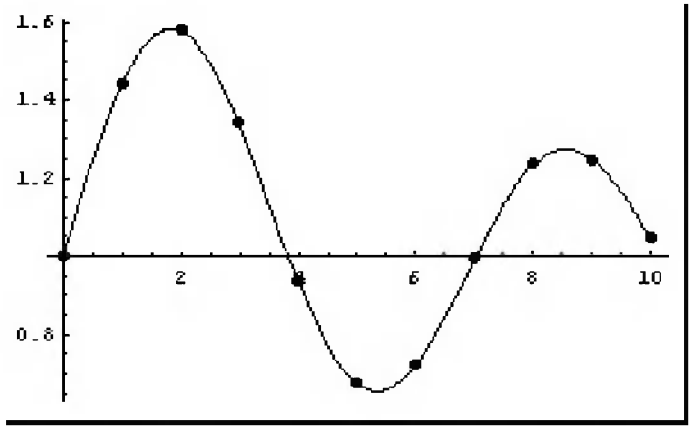


Рис. 6.33. График полинома и узловые точки

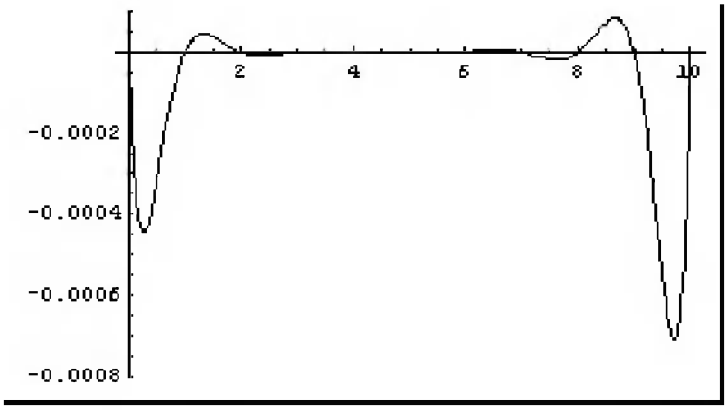


Рис. 6.34. График абсолютной погрешности при аппроксимации функции Бесселя полиномом десятой степени

6.6.5. Полиномиальная аппроксимация при большом числе узлов

Просто увеличение числа узлов и степени полинома приводит обычно к неутешительным результатам. Это хорошо видно из рис. 6.35, где показан график аппроксимирующего полинома при степени n = 40 (41 точка функции f(x), использованной в предшествующем примере).

Mathematica, однако, позволяет задавать вычисления при аппроксимации с достаточно большим числом цифр – порядка степени аппроксимирующего полинома. Разумеется, этой возможностью надо пользоваться осмотрительно, поскольку увеличение числа цифр ведет к увеличению времени вычислений и рано или поздно вызывает численную нестабильность.

Выполним, к примеру, аппроксимацию для 21 узла (n = 20), установив функцией SetPrecision вычисления аппроксимирующего полинома с 20 точными цифрами:

```
f[x_]=BesselJ[1,x]+1;
data=Table[{x,N[f[x],50]},{x,0,10,0.5}];
```

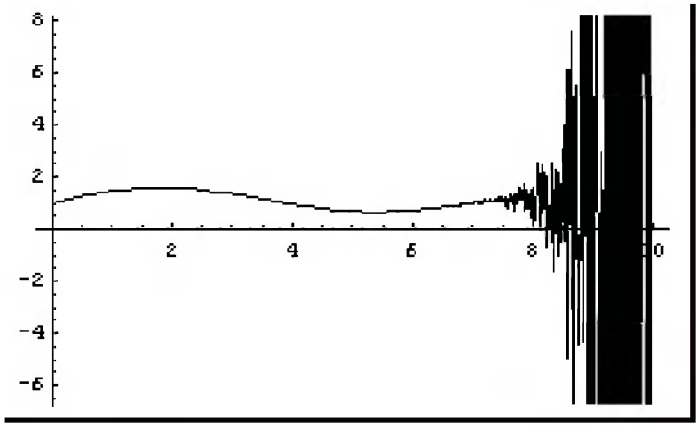


Рис. 6.35. График полинома степени 40, иллюстрирующий возникновение числовой неустойчивости в правой части – при больших x

```
g1:=Plot[f[x],{x,0,10}]
p[x_]=SetPrecision[Collect[InterpolatingPolynomial[data,
x ],x],20]
1.0000000000000000000 +
 0.50000000026808300024x - 2.1946079883927716025×10^-9x^2 -
 0.06249999200941254440 3x^3 - 1.7338956400433230831×10^-8x^4 +
 0.00260419196642820574 91x^5 - 2.6520381936664922480×10^-8x^6 -
 0.00005423265833077900 2056x^7 - 1.2572612631091120302×10^-8x^8 +
 6.8412544115568677809×10^-7x^9 + 2.2426515004772608460×10^-9x^10 -
 4.9758073921924942709 2×10^-9x^11 - 1.6405048247087269161×10^-10x^12 +
 6.5645938318937200218×10^-11x^13 - 5.0007652405250705692×10^-12x^14 +
 4.6925914016242405225×10^-13x^15 - 5.9718776421275391973×10^-14x^16 +
 4.8572376755578655131×10^-15x^17 - 2.2237845150851505265×10^-16x^18 +
 5.4431086706688516340×10^-18x^19 - 5.6368826561019998150×10^-20x^20
g3:=Plot[p[x],{x,0,10}]
Show[g3,g2]
```

В выводе этого фрагмента представлен полученный аппроксимирующий полином степени n=20. График полинома и точек функции f(x) представлен на рис. 6.36. От графика рис. 6.35 он отличается лишь большей вдвое густотой узловых точек.

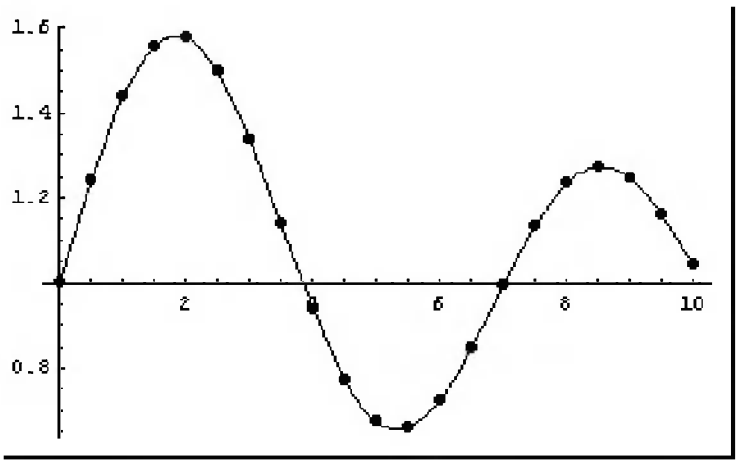


Рис. 6.36. График полинома двадцатой степени

Рисунок 6.37 показывает график погрешности для данного случая. Он построен применением следующей команды:

```
Plot[(p[x]-f[x]),{x,0,10},PlotRange->{-10^-11,
1.5*10^-10}]
```

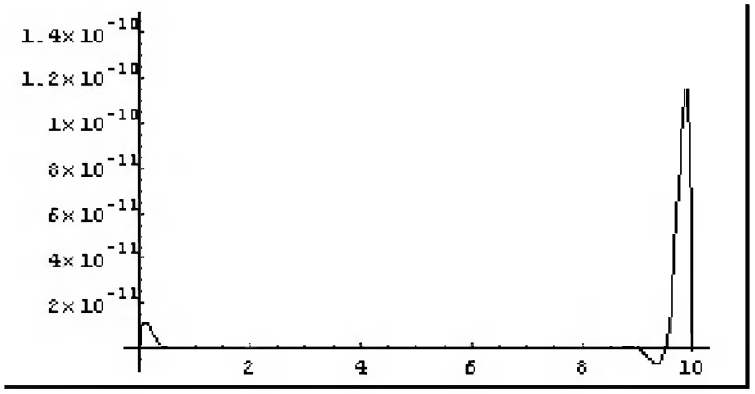


Рис. 6.37. График абсолютной погрешности при аппроксимации смещенной на 1 функции Бесселя при степени полинома n = 20

Сравнив этот график с графиком, представленным на рис. 6.34, можно сделать вывод о том, что увеличение порядка полинома всего вдвое ведет к уменьшению погрешности примерно на 7 порядков!

6.6.6. Рациональная интерполяция и аппроксимация

Подпакет Approximations встроенного пакета расширения NumericalMath системы Mathematica 4/5 содержит ряд функций для *рациональной аппроксимации* аналитических функций. Для рациональной интерполяции и аппроксимации функций по заданным значениям абсцисс служит функция:

- **RationalInterpolation[f,{x,m,k},{x_1,x_2,…,x_{m+k+1}}]** – возвращает аппроксимирующее функцию f выражение в виде отношения полиномов со степенью полинома числителя m и знаменателя k в абсциссах, заданных списком {x_1,x_2,...,x_{m+k+1}}.

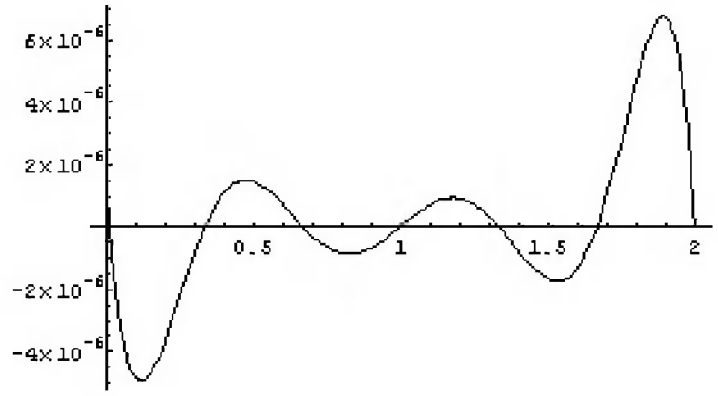
Пример на применение этой функции для аппроксимации зависимости $\sin(x)/x+1$представлен на рис. 6.38. Обратите внимание на первую строку этого примера – в ней задана загрузка подпакета Approximations из встроенного пакета расширения NumericalMath.

Нетрудно заметить, что кривая погрешности явно осциллирует и погрешность оказывается максимальной вблизи краев интервала аппроксимации. Несмотря на малое число явно заданных узловых точек (их 7 и точка при x = 0 с особенностью не используется), погрешность аппроксимации мала и не превышает $8*10^{-6}$.

```
<< NumericalMath`Approximations`
ri1 = RationalInterpolation[ Sin[x] / x + 1, {x, 2, 4},
  {1 / 100, 1 / 3, 2 / 3, 1, 4 / 3, 5 / 3, 2}]
```

$$\frac{2. - 0.575467\,x + 0.151927\,x^2}{1 - 0.287672\,x + 0.158877\,x^2 - 0.0228971\,x^3 + 0.00777953\,x^4}$$

```
Plot[ri1 - (Sin[x] / x + 1), {x, 0, 2}]
```




```
- Graphics -
```

Рис. 6.38. Пример рациональной аппроксимации зависимости sin(x)/x+1

Представленная функция может использоваться и в иной форме:

```
RationalInterpolation[f,{ x, m, k},{ x, xmin, xmax}]
```

В данном случае выбор абсцисс осуществляется автоматически в интервале от xmin до xmax. В отличие от первого случая, когда абсциссы могли быть расположены неравномерно, в данном случае расположение их будет равномерным. На рис. 6.39 приведен пример аппроксимации функции синуса в интервале от $-\pi$ до π.

При рациональной аппроксимации можно задать опции WorkingPrecision и Bias со значениями по умолчанию $MachinePrecision и 0 соответственно. Опция Bias обеспечивает автоматическую расстановку узлов интерполяции. При удачном подборе значения Bias обеспечивается симметрирование выбросов погрешности, дающее наименьшее ее значение в пиках. На рис. 6.40 приведен пример интерполяции (аппроксимации) экспоненциальной функции в интервале изменения x от 0 до 2. Погрешность аппроксимации в данном случае меньше 10^{-6}.

Из приведенных примеров видна высокая эффективность рациональной аппроксимации даже при умеренной степени полиномов числителя и знаменателя. Имеющиеся в системе Mathematica средства позволяют до предела упростить проведение такой аппроксимации и легко оценивать области ее применения и порядок получаемой погрешности.

```
<< NumericalMath`Approximations`

ri2 = RationalInterpolation[Sin[x], {x, 5, 6}, {x, -Pi, Pi}]

(-1.7499 x 10^-16 + 1. x + 3.98929 x 10^-14 x^2 -
   0.128641 x^3 - 4.03859 x 10^-15 x^4 + 0.00276805 x^5) /
 (1 + 4.01691 x 10^-14 x + 0.0380273 x^2 + 2.25378 x 10^-15 x^3 +
   0.000770917 x^4 + 1.76917 x 10^-16 x^5 + 0.0000107609 x^6)

Plot[ri2 - Sin[x], {x, -Pi, Pi}]
```

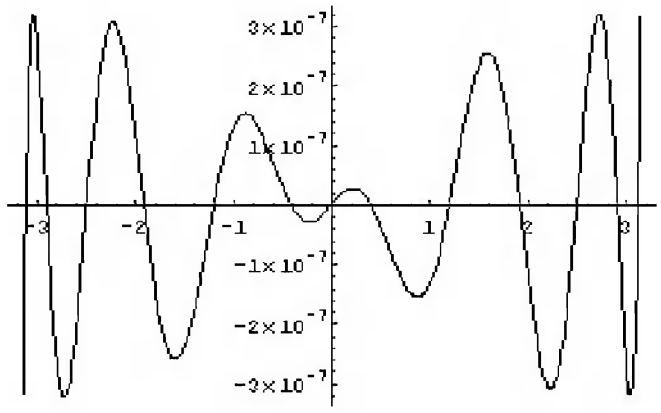


Рис. 6.39. Пример рациональной аппроксимации синусоидальной функции

```
<< NumericalMath`Approximations`

ri3 = RationalInterpolation[Exp[x], {x, 2, 4}, {x, -1, 2},
  Bias -> 0.3]

1.00001 + 0.364451 x + 0.0414444 x^2
-------------------------------------------------
1 - 0.635528 x + 0.177014 x^2 - 0.0259959 x^3 + 0.00171154 x^4

Plot[ri3 - Exp[x], {x, -1, 2}]
```

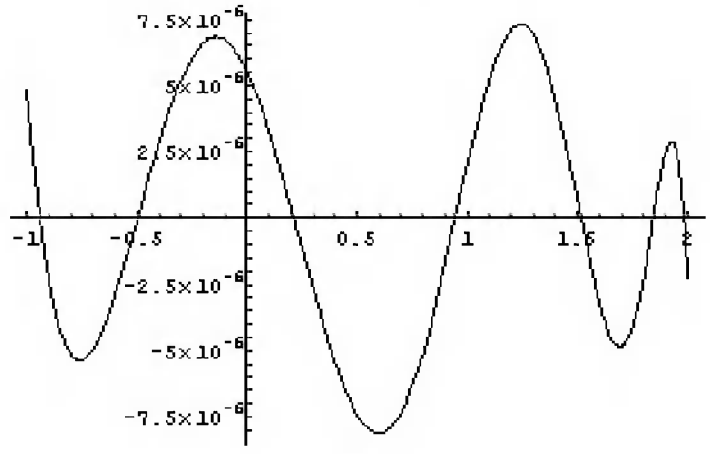


Рис. 6.40. Пример аппроксимации экспоненты при выборе опции Bias->0.3

6.6.7. Рациональная Паде-аппроксимация

В подпакете Pade встроенного в Mathematica 4/5 пакета расширения Calculus определены две функции для *рациональной аппроксимации Паде*:

- **Pade[f,{x,x0,m,k}]** – возвращает выражение для аппроксимации Паде функции f(x) в окрестностях точки x0 в виде отношения двух полиномов степеней m и k.
- **EconomizedRationalApproximation[f,{x,{xmin,xmax},m,k}]** – возвращает выражение для осуществления *экономичной рациональной аппроксимации* функции f(x) в интервале {xmin, xmax} в виде отношения двух полиномов степеней m и k.

На рис. 6.41 представлен пример на аппроксимацию Паде функции синуса с построением графика исходной функции (темная линия) и аппроксимирующей функции (более светлая линия).

```
<< Calculus`Pade`;

f1=Pade[Sin[x], {x, 0, 5, 6}]
```

$$\frac{x - \frac{2369\,x^3}{18183} + \frac{12671\,x^5}{4363920}}{1 + \frac{445\,x^2}{12122} + \frac{601\,x^4}{872784} + \frac{121\,x^6}{16662240}}$$

```
g1 = Plot[{f1, Sin[x]}, {x, -6, 6},
   PlotStyle -> {{GrayLevel[0.4]}, {GrayLevel[0]}}];
```

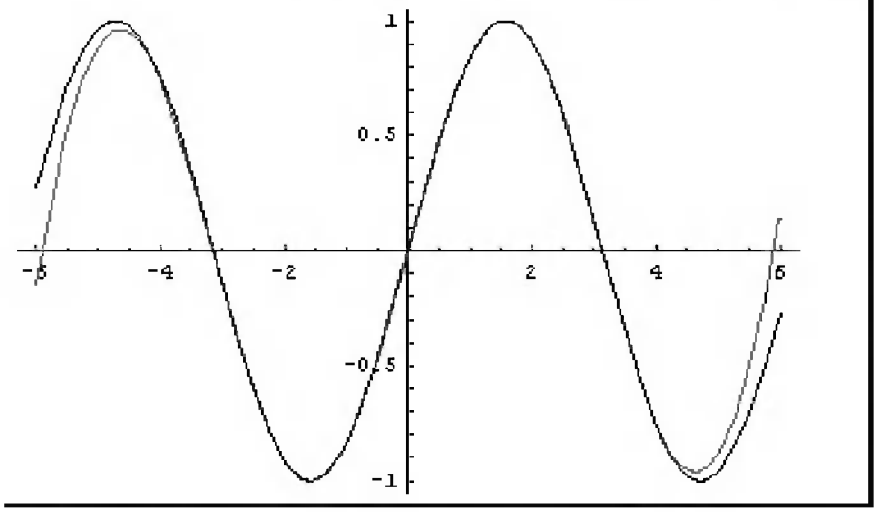


Рис. 6.41. Пример осуществления аппроксимации Паде

Зависимость относительной погрешности от значения независимой переменной x представлена на рис. 6.42.

Пример осуществления *экономичной рациональной аппроксимации* для синусоидальной зависимости показан на рис. 6.43. Здесь также дана графическая визуализация аппроксимации в виде наложенных друг на друга исходной и аппроксимирую-

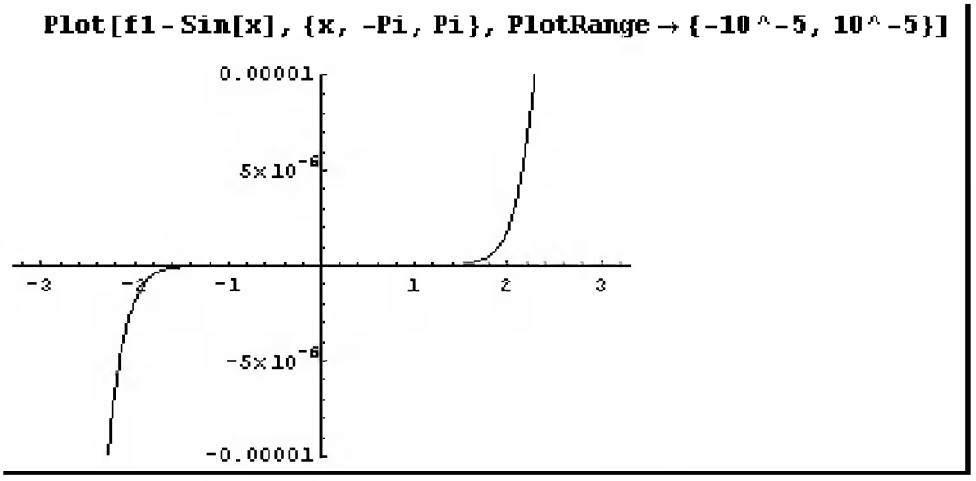


Рис. 6.42. График погрешности при аппроксимации синусоидальной функции

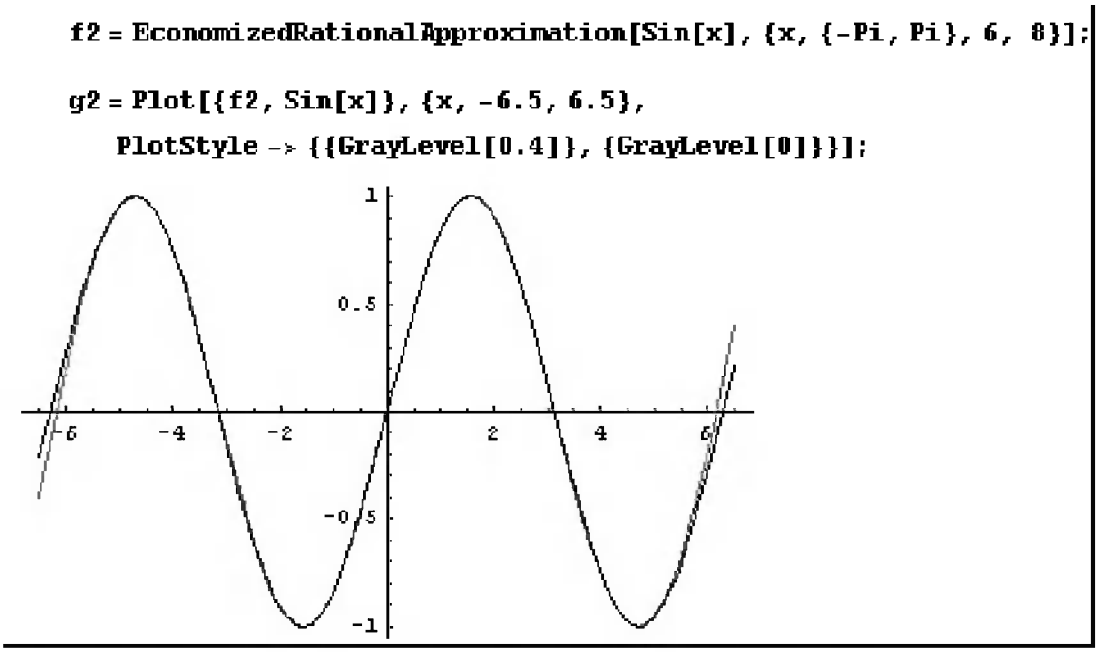


Рис. 6.43. Пример осуществления экономичной рациональной аппроксимации

щей функций. Нетрудно заметить, что область видимого на глаз совпадения исходной и аппроксимирующей функций заметно расширилась.

Экономичная рациональная аппроксимация обычно позволяет получить приемлемую погрешность при меньшей степени полиномов числителя и знаменателя аппроксимирующей функции. В ограниченной области {xmin,xmax} эта аппроксимация нередко позволяет получить погрешность менее сотых долей процента – см. рис. 6.44. Здесь показан график погрешности в виде разности между значениями аппроксимирующей и аппроксимируемой функций.

Несмотря на обширные возможности в выборе средств аппроксимации, все же надо отметить, что они уступают таковым у конкурента системы Mathematica – Maple, где функций для осуществления аппроксимации больше.

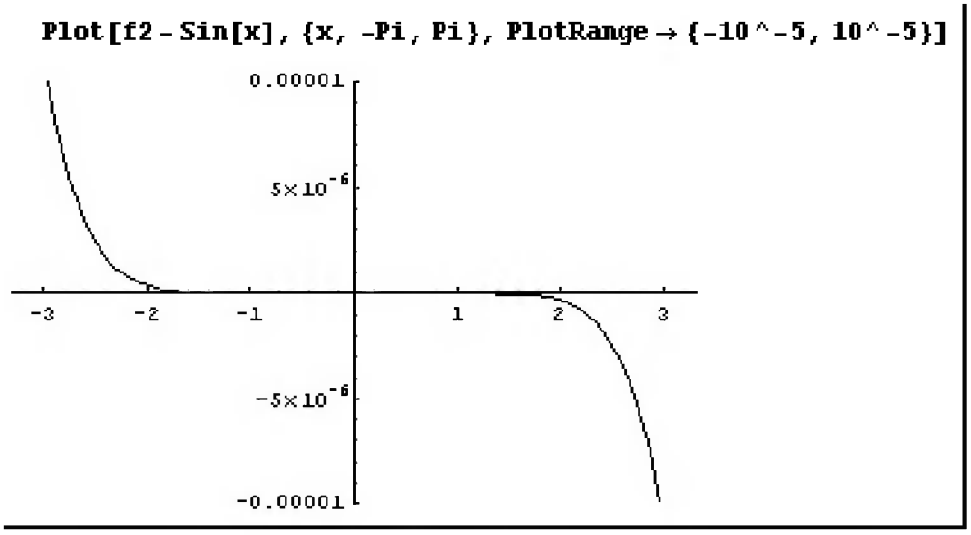


Рис. 6.44. График погрешности экономичной рациональной аппроксимации

6.6.8. Оптимизация аппроксимации

Как уже отмечалось в теоретической преамбуле, для оптимизации аппроксимации целесообразно применять схему Горнера. Подпакет Horner из встроенного пакета расширения Algebra в системе Mathematica 4/5 (В Mathematica 3 подпакет входил в пакет расширения NumericalMath) реализует схему Горнера для полиномиальных выражений. Для этого в нем есть функция:

- **Horner[poly]** – устанавливает полином poly в форму Горнера;
- **Horner[poly, vars]** – устанавливает полином ряда переменных vars в форму Горнера.

Примеры преобразования полиномов в схему Горнера:

```
<< Algebra'Horner'
Horner[ a x^3 + b x^2 + c x + d, x ]
 (d + x(x + x(b + ax)))
Horner[ x^(1/3) + x + x^(3/2) ]
```

$$\left(1+\left(1+\sqrt{x}\right)x^{2/3}\right)x^{1/3}$$

Схема Горнера может использоваться и для отношения полиномов:

Horner[poly1/poly2] и **Horner[poly1/poly2,vars1,vars2]**.

Эти функции можно использовать для улучшенного представления аппроксимации Паде, что демонстрирует следующий пример:

```
<< Calculus'Pade'
approx = Pade[ Exp[Log[x]-x], x, 0, 3, 2]
```

$$\frac{x-\frac{x^2}{2}+\frac{x^3}{12}}{1+\frac{x}{2}+\frac{x^2}{12}}$$

```
Horner[ approx ]
```

$$\frac{x\left(1+\left(-\frac{1}{2}+\frac{x}{12}\right)x\right)}{1+\left(\frac{1}{2}+\frac{x}{12}\right)x}$$

Переход к схеме Горнера дает ряд преимуществ перед обычным вычислением полиномов: уменьшается время вычислений, повышается точность вычислений, уменьшается вероятность расхождения численных методов, в которых используются полиномы.

6.6.9. Минимаксная аппроксимация

В подпакете Approximations встроенного пакета расширения NumericalMath минимаксная аппроксимация реализуется следующей функцией:

- **MiniMaxApproximation [f,{x,{xmin,xmax},m,k}** – возвращает рациональную функцию аппроксимации f при степени полиномов числителя и знаменателя {m,k} и в интервале изменения x от xmin до xmax.
- **MiniMaxApproximation[f,approx,{x,{xmin,xmax},m,k}** – возвращает рациональную функцию аппроксимации f при степени полиномов числителя и знаменателя {m,k} и в интервале изменения x от xmin до xmax с возможностью выбора метода аппроксимации approx.

Эта аппроксимация использует интерактивный алгоритм вычислений. Они начинаются с первого шага, на котором используется функция **RationalInterpolation**. Затем аппроксимация последовательно улучшается применением алгоритма Ремеза, лежащего в основе этого вида аппроксимации.

Функция **MiniMaxApproximation** возвращает два списка: первый – с координатами абсцисс, при которых наблюдается максимальная погрешность, и второй – рациональную функцию аппроксимации. На рис. 6.45 представлен пример на аппроксимацию функции $\exp(x^2)$. Там же построен график погрешности аппроксимации.

График погрешности минимаксной аппроксимации наглядно показывает ее главное свойство – кривая погрешности при наличии колебаний обеспечивает равенство амплитуд колебаний. Погрешность в последнем примере не превышает $6\cdot10^{-7}$, что свидетельствует о высокой и в ряде случаев даже избыточной точности.

Следует отметить, что малость абсолютной ошибки для ряда функций (например, тригонометрических) может приводить к большим относительным погрешностям в точках, где функции имеют нулевые значения. Это может привести к отказу от выполнения аппроксимации вследствие исчерпания числа итераций (опция MaxIteration 20). Подобный случай наблюдается, например, при исполнении такой команды:

```
MiniMaxApproximation[Cos[x], {x, {1, 2}, 2, 4}]
```

Делением функции на (x-Pi/2) можно исключить эту ситуацию и получить нужные приближения – рис. 6.46.

График относительной погрешности для этого примера представлен на рис. 6.47. Обратите внимание на то, что в этом примере погрешность аппроксимации не превышает $7\cdot10^{-10}$.

```
<<NumericalMath`Approximations`

mmlist = MiniMaxApproximation[Exp[x^2],
          {x, {0, 2}, 4, 6}]

{{0., 0.0500479, 0.193169, 0.410482, 0.676581, 0.964579,
  1.2487, 1.50587, 1.718, 1.87389, 1.96841, 2.},
 {(1. - 0.420736 x + 0.559096 x^2 -
     0.149632 x^3 + 0.0797694 x^4) /
   (1 - 0.420675 x - 0.441899 x^2 + 0.277403 x^3 + 0.00083104
     x^4 - 0.0270927 x^5 + 0.00451102 x^6), -6.21987 * 10^-7}}

mmfunc = mmlist[[2, 1]]

(1. - 0.420736 x + 0.559096 x^2 - 0.149632 x^3 + 0.0797694 x^4) /
 (1 - 0.420675 x - 0.441899 x^2 + 0.277403 x^3 +
   0.00083104 x^4 - 0.0270927 x^5 + 0.00451102 x^6)

Plot[1 - mmfunc/Exp[x^2], {x, 0, 2}]
```

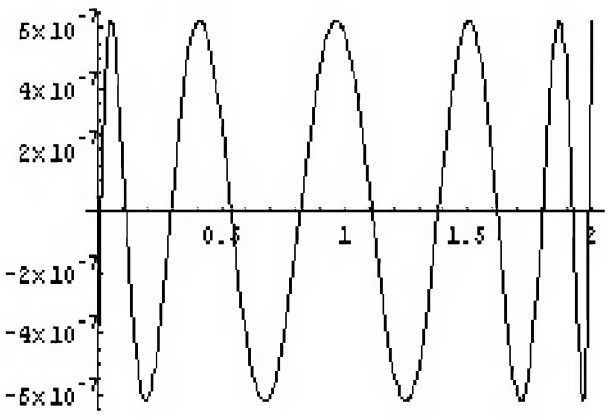


Рис. 6.45. Минимаксная аппроксимация функции exp(x^2) и график погрешности при ней

В заключение отметим, что данный пакет имеет функции для *общей рациональной интерполяции*, с помощью которой можно аппроксимировать функции, заданные параметрически:

- **GeneralRationalInterpolation[{f_x,f_y},{t,m,k},{t_1,t_2,…,t_{n+k+1}}]** – дает рациональную интерполяцию для параметрически заданных функций для списка значений параметра t.
- **GeneralRationalInterpolation[{f_x,f_y},{t,m,k},{t,tmin,tmax}]** – дает рациональную интерполяцию для параметрически заданных функций при автоматическом выборе значений параметра t.

```
<< NumericalMath`Approximations`

MiniMaxApproximation[Cos[x], {x, {1, 2}, 2, 4}]

MiniMaxApproximation::van :
 Failed to locate the extrema in 20 iterations. The function
   Cos[x] may be vanishing on the interval (1, 2) or the
   WorkingPrecision may be insufficient to get convergence.

MiniMaxApproximation[Cos[x], {x, {1, 2}, 2, 4}]

MiniMaxApproximation[Cos[x] / (x - Pi / 2),
  {x, {1, 2}, 2, 4}][[2, 1]]

                    -0.636483 - 0.284196 x + 0.0904596 x^2
-------------------------------------------------------------------
1 - 0.191361 x + 0.0771022 x^2 - 0.0103059 x^3 + 0.00164047 x^4

mmacos = %N[x - Pi / 2]

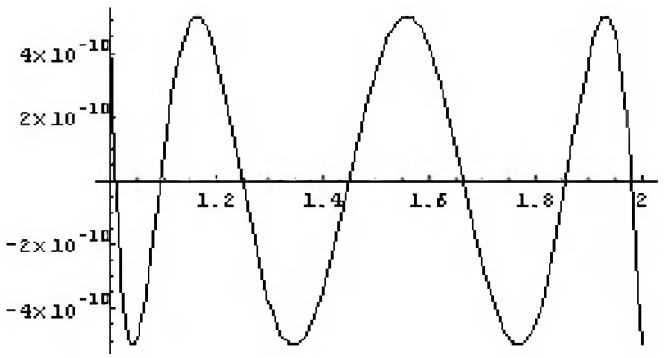
                  (-1.5708 + x) (-0.636483 - 0.284196 x + 0.0904596 x^2)
-------------------------------------------------------------------
1 - 0.191361 x + 0.0771022 x^2 - 0.0103059 x^3 + 0.00164047 x^4
```

Рис. 6.46. Пример минимаксной аппроксимации функции косинуса

```
mmacos = %N[x - Pi / 2]

                  (-1.5708 + x) (-0.636483 - 0.284196 x + 0.0904596 x^2)
-------------------------------------------------------------------
1 - 0.191361 x + 0.0771022 x^2 - 0.0103059 x^3 + 0.00164047 x^4

Plot[1 - mmacos / Cos[x], {x, 1, 2}]
```




```
- Graphics -
```

Рис. 6.47. График погрешности минимаксной аппроксимации косинуса

С помощью других функций можно осуществить также *общую минимаксную интерполяцию*, обычно обеспечивающую минимальную погрешность:

- **GeneralMiniMaxInterpolation[{f_x,f_y},{t,(tmin,tmax},m,k},x]** – дает рациональную минимаксную интерполяцию для параметрически заданных функций с параметром t.
- **GeneralMiniMaxInterpolation[{f_x,f_y},approx,{t,(tmin,tmax},m,k},x]** – дает рациональную минимаксную интерполяцию для параметрически заданных функций для списка значений параметра t с указанием метода аппроксимации approx.
- **GeneralMiniMaxInterpolation[{f_x,f_y,g},{t,(tmin,tmax},m,k},x]** – дает рациональную минимаксную интерполяцию для параметрически заданных функций при автоматическом выборе значений параметра t и с ошибкой, заданной g(t).

Примеры на применение этого довольно редкого вида аппроксимации можно найти в справке системы Mathematica.

6.6.10. Сплайновая интерполяция и аппроксимация

Подпакет SplineFit встроенного пакета расширения NumericalMath системы Mathematica 4/5 содержит функцию для проведения сплайновой интерполяции и экстраполяции:

- **SplineFit[data,type]** – возвращает сплайн-функцию для данных data и типа сплайн-аппроксимации type – по умолчанию это кубический сплайн Cube (другие типы Bezier (B-сплайны) и CompositeBezier). Сплайн-функция имеет формат SplineFunction[type, range, interval].

Рисунок 6.48 показывает пример сплайн-интерполяции для зависимости $y(x)$, представленной пятью парами точек в прямоугольной системе координат. На нем построены также графики аппроксимирующей функции и исходных точек.

Специфика сплайн-регрессии по функции **SplineFit** заключается в преобразовании значений как xi, так и yi. Это позволяет представлять сплайнами в общем виде параметрически заданные функции, что поясняет рис. 6.49.

С помощью функции InputForm можно вывести детальные данные о примененных сплайн-функциях, что показано в конце ноутбука рис. 6.49.

6.6.11. Функция регрессии Fit

Для решения задач регрессии в СКМ Mathematica 4/5 используется функция **Fit**:

- **Fit[data, funs, vars]** – для списка данных **data** ищет приближение методом наименьших квадратов в виде линейной комбинации функций **funs** переменных **vars**. Данные **data** могут иметь форму {{x1,y1,...,f1}, {x2,y2,...,f2}, ...}, где число координат x,y,... равно числу переменных в списке **vars**. Также данные **data** могут быть представлены в форме {f1, f2, ...} с одной координа-

```
<<NumericalMath`SplineFit`

data={{1,2},{1.8,3.5},{3,4.5},{4,3},{5,4}}

{{1, 2}, {1.8, 3.5}, {3, 4.5}, {4, 3}, {5, 4}}

spl=SplineFit[data,Cubic]

SplineFunction[Cubic, {0., 4.}, <>]

spl[3.99]

{4.98979, 3.98161}

g1=ParametricPlot[spl[x],{x,0,4},PlotRange->All,Compiled->False];

g2=ListPlot[data,PlotStyle->{PointSize[0.02]}];Show[g1,g2]
```

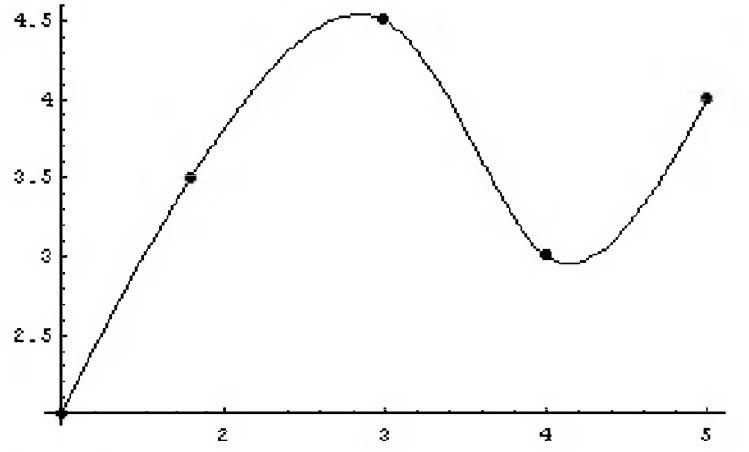


Рис. 6.48. Пример сплайн-интерполяции для зависимости y(x), заданной списком координат своих узловых точек

той, принимающей значения 1,2,.... Аргумент **funs** может быть любым списком функций, которые зависят только от объектов **vars**.

Следующие примеры показывают приближение исходных данных степенными полиномами второй и третьей степеней и линейной комбинацией двух функций:

```
Fit[{{0, 0.9}, {2, 8.1}, {3, 17}, {4, 33}}, {1, x, x^2}, x]
0.997273 - 1.40864 x + 2.33409 x^2
Fit[{{0, 0.9}, {2, 8.1}, {3, 17}, {4, 33}}, {1, x, x^2, x^3}, x]
0.9 + 2.74167 x - 0.4625 x^2 + 0.445833 x^3
Fit[{{0, 0.9}, {2, 8.1}, {3, 17}}, {x^2, Exp[x], x}, x]
0.9 E^x + 2.89276 x - 1.08392 x^2
```

Здесь в первом примере выполняется полиномиальная регрессия со степенью многочлена 2. Максимальная степень на 1 меньше числа пар исходной зависимости – при такой степени регрессия вырождается в обычную полиномиальную аппроксимацию. Она рассматривалась выше. В нашем случае число пар равно 4, так что вырождение наступает при степени полинома, равной 3, – второй пример.

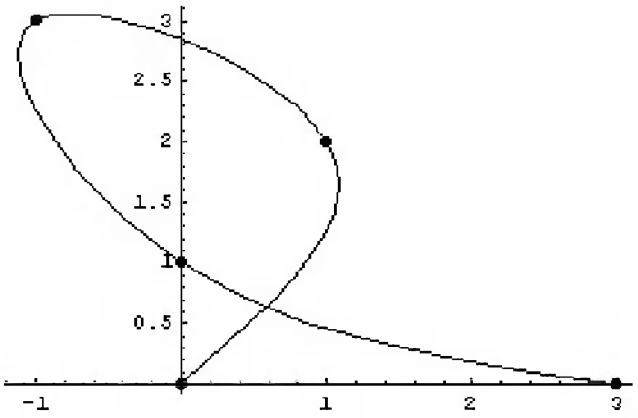
```
data = {{0,0},{1,2},{-1,3},{0,1},{3,0}};

spl=SplineFit[data,Cubic]

SplineFunction[Cubic, {0., 4.}, <>]

g1=ParametricPlot[spl[x],{x,0,4},PlotRange->All,Compiled->False];

g2=ListPlot[data,PlotStyle->{PointSize[0.02]}];Show[g1,g2]
```




```
- Graphics -

InputForm[spl]

SplineFunction[Cubic, {0., 4.}, {{0, 0}, {1, 2}, {-1, 3}, {0, 1}, {3, 0}},
 {{{0, 111/56, 0, -55/56}, {0, 57/28, 0, -1/28}},
  {{1, -27/28, -165/56, 107/56}, {2, 27/14, -3/28, -23/28}},
  {{-1, -9/8, 39/14, -37/56}, {3, -3/4, -18/7, 37/28}},
  {{0, 69/28, 45/56, -15/56}, {1, -27/14, 39/28, -13/28}}}]
```

Рис. 6.49. Пример сплайн-интерполяции параметрически заданной функции

Третий пример представляет данные линейной комбинацией квадратичной, экспоненциальной и линейной функций.

Рисунок 6.50 показывает несколько иной путь проведения полиномиальной регрессии – исходные данные заданы объектом-списком data. Представлены два варианта регрессии – для степени полинома 1 (линейная регрессия) и 2 (квадратичная регрессия).

В конце документа рис. 6.50 дано построение графиков аппроксимирующих полиномов и точек исходных данных. Заметно, что при регрессии графики полиномов проходят в середине «облака» исходных точек и не укладываются на них точно.

6.6.12. Линейная регрессия

В подпакете LinearRegression встроенного пакета расширения Statistics СКМ Mathematica 4/5 имеются расширенные функции для проведения *линейной регрессии общего вида* – в дополнение включенной в ядро функции **Fit**. Прежде всего это функция **Regress**:

```
Регрессия линейная и квадратичная с визуализацией

data := {{0, .9}, {1.1, 3}, {2, 8.1}, {3, 17}, {4, 33}}

p1[x_] = Fit[data, {1, x}, x]

-3.51203 + 7.87724 x

g1 := Plot[p1[x], {x, 0, 4}]

p2[x_] = Fit[data, {1, x, x^2}, x]

1.16201 - 1.24508 x + 2.27624 x^2

g2 := Plot[p2[x], {x, 0, 4}]

gd := ListPlot[data, PlotStyle -> {PointSize[0.02]}]

Show[g1, g2, gd, PlotRange -> {-5, 35}];
```

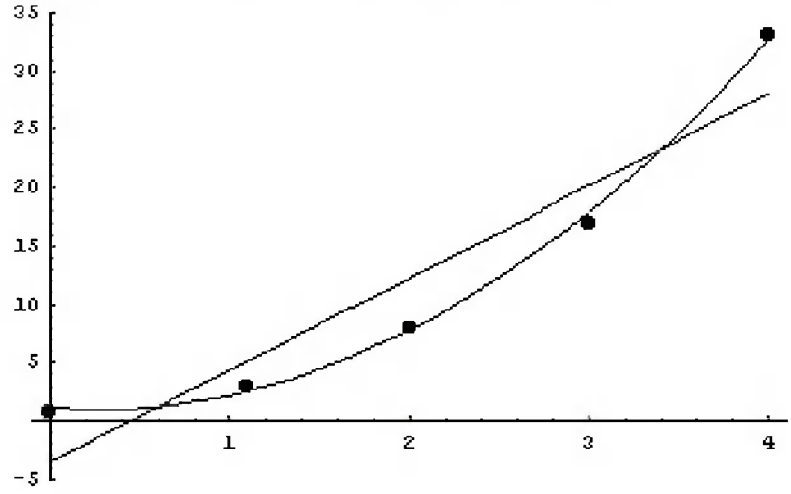


Рис. 6.50. Полиномиальная регрессия с графическим выводом

- **Regress[data,{I,x,x^2},x]** – осуществляет регрессию данных data, используя квадратичную модель.
- **Regress[data,{I,x_1,x_2,x_1x_2},{x_1,x_2}]** – осуществляет регрессию, используя в ходе итераций зависимость между переменными x_1 и x_2.
- **Regress[data,{f_1,f_2,...},vars]** – осуществляет регрессию, используя модель линейной регрессии общего вида с уравнением регрессии, представляющим линейную комбинацию функций f_i от переменных vars.

Данные могут быть представлены листом ординат {y_1,y_2,...} или листом {{x11,x12,...,y1},{x21,x22,...,y2},...}. Пример применения функции Regress представлен на рис. 6.51. Нетрудно заметить, что функция выводит детальный отчет по проведенной регрессии.

На рис. 6.52 показан еще один пример проведения регрессии с графической визуализацией с помощью функции **MultipleListPlot**. Применение этой функции позволило построить линии интервала разброса данных, в котором располагаются точки данных для регрессии.

```
<<Statistics`LinearRegression`

data = {{0.05, 90}, {0.09, 97}, {0.14, 107}, {0.17, 124}, {0.18, 142},
   {0.21, 150}, {0.23, 172}, {0.25, 189}, {0.28, 209}, {0.35, 253}};

(regress = Regress[data, {1, x, x^2}, x];
Chop[regress, 10^(-5)])
```

```
                                Estimate      SE          TStat       PValue
{ParameterTable → 1             73.2869       11.2362     6.52237     0.000327186
                  x             170.573       120.586     1.41453     0.200111    ,
                  x^2           1034.55       298.131     3.47012     0.0104042

 RSquared → 0.982579, AdjustedRSquared → 0.977601, EstimatedVariance → 62.1793,

                                DF      SumOfSq     MeanSq      FRatio      PValue
 ANOVATable → Model             2       24548.8     12274.4     197.404     0       }
              Error             7       435.255     62.1793                         }
              Total             9       24984.1
```

```
func = Fit[data, {1, x, x^2}, x]

73.2869 + 170.573 x + 1034.55 x^2

Options[Regress]

{RegressionReport → SummaryReport,
 IncludeConstant → True, BasisNames → Automatic,
 Weights → Automatic, Tolerance → Automatic, ConfidenceLevel → 0.95}
```

Рис. 6.51. Пример применения функции Regress

Пакет линейной регрессии содержит ряд и иных функций, с которыми можно ознакомиться с помощью справочной базы данных системы Mathematica.

6.6.13. Нелинейная регрессия

В подпакете NonlinearFit пакета статистических вычислений Statistica содержатся важные функции для выполнения *нелинейной регрессии общего вида*:

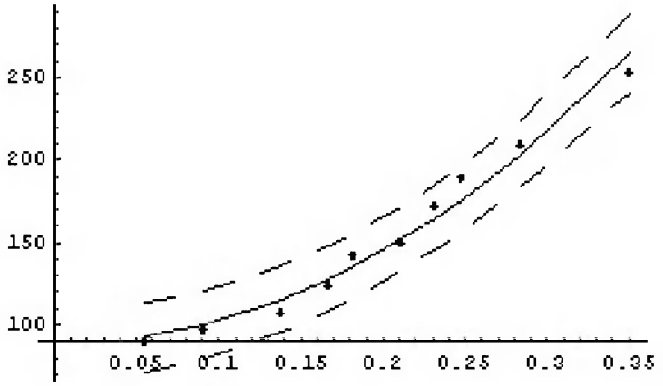
- **NonlinearFit[data,model,variables,parameters]** – выполняет регрессию по заданной модели (формуле) model с переменными variables и параметрами parameters для заданных данных data.
- **NonlinearRegress[data,model,variables,parameters]** – выполняет регрессию по заданной модели (формуле) model с переменными variables и параметрами parameters для заданных данных data с выдачей листа диагностики.

Данные могут быть представлены листом ординат {y1,y2,...} или листом {{x11,x12,...,y1},{x21,x22,...,y2},...}. В ходе регрессии минимизируются заданные параметры, так что заданная модель регрессии приближает данные с минимальной среднеквадратической погрешностью.

```
<<Statistics`LinearRegression`

data = {{0.055, 90}, {0.091, 97}, {0.138, 107},
 {0.167, 124}, {0.182, 142}, {0.211, 150},
 {0.232, 172}, {0.248, 189}, {0.204, 209},
 {0.351, 253}};

MultipleListPlot[
 data, predicted, lowerCI, upperCI,
 SymbolShape -> {PlotSymbol[Diamond], None, None, None},
 PlotJoined -> {False, True, True, True},
 PlotStyle -> {Automatic, Automatic,
 Dashing[{.05, .05}], Dashing[{.05, .05}]}
]
```




```
- Graphics -
```

Рис. 6.52. Пример проведения регрессии с графической визуализацией

На рис. 6.53 показан пример выполнения логарифмической регрессии. При ней модель представлена выражением a*Log[b*x]. Результатом действия функции **NonlinearFit** является уравнение регрессии в виде этой модели с найденными значениями параметров a и b. Представлена также визуализация регрессии в виде графика функции – модели и исходных точек.

Отчет по нелинейной регрессии, который формирует функция NonlinearRegression, представлен на рис. 6.54 для примера, изображенного на рис. 6.53.

Более детальные данные об опциях и обозначениях в отчетах нелинейной регрессии можно найти в справочной базе данных.

6.6.14. Нелинейная регрессия в Mathematica 6

Функция нелинейной регрессии **FindFit**, которая была и в системе Mathematica 5.2, в Mathematica 6 [170] усовершенствовалась и стала вполне полноценным и простым средством для проведения нелинейной регрессии произвольного вида. Функция задается в виде:

```
FindFit[data,expr,pars,vars]     FindFit[data,{expr,cons},pars,vars]
```

```
<< Statistics`NonlinearFit`

data:={{1,1},{2,1.3},{3,1.716},{4,1.8},{5,2.048},{6,2.167}}

fr=NonlinearFit[data, a*Log[b*x],{x},{a,b}]

0.665503 Log[4.11893 x]

g1:=Plot[fr,{x,1,6}]

g2:=ListPlot[data,PlotStyle->{PointSize[0.02]}]

Show[g1,g2]
```

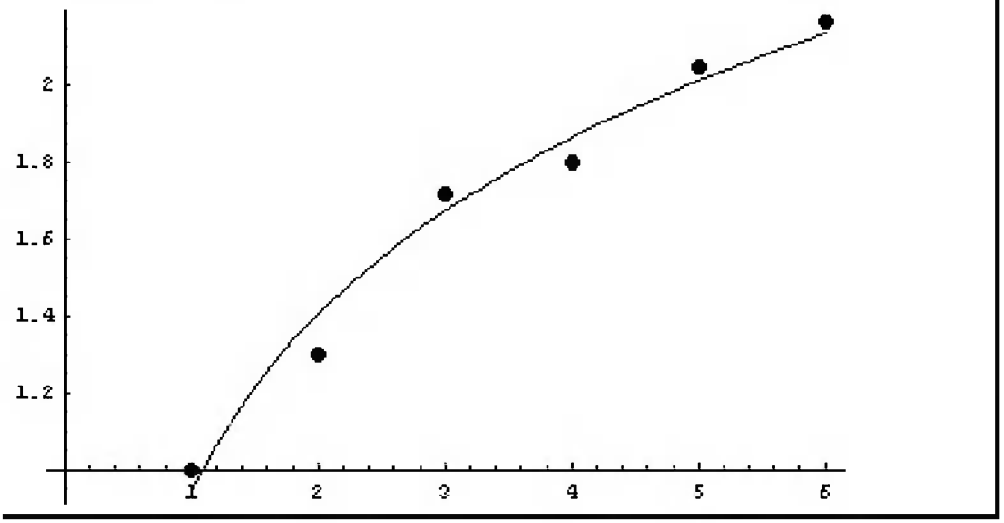


Рис. 6.53. Пример логарифмической регрессии

```
NonlinearRegress[data, a*Log[b*x],{x},{a,b}]

{BestFitParameters → {a → 0.665503, b → 4.11893},

                          Estimate      Asymptotic SE      CI
 ParameterCITable → a     0.665503      0.0504167          {0.525524, 0.805482},
                    b     4.11893       0.806289           {1.88031, 6.35754}

 EstimatedVariance → 0.00558058,

                                  DF      SumOfSq        MeanSq
              Model               2       17.7425        8.87126
 ANOVATable → Error               4       0.0223223      0.00558058,
              Uncorrected Total   6       17.7648
              Corrected Total     5       0.994689

 AsymptoticCorrelationMatrix → {{1.        -0.972212},
                                {-0.972212  1.      }},

                                        Curvature
 FitCurvatureTable → Max Intrinsic      3.94364×10^-16
                     Max Parameter-Effects   2.07792
                     95. % Confidence Region 0.379478}
```

Рис. 6.54. Отчет о проведении нелинейной регрессии

Здесь data – список данных, expr – выражение, представляющее функцию регрессии, cons – ограничительные условия, pars – список искомых параметров, переменная или список переменных.

Пример проведения нелинейной регрессии с помощью функции **FindFit** с графической визуализацией регрессии путем построения функции регрессии и точек исходных данных показан на рис. 6.55. Хорошо видно, что кривая функции регрессии проходит в облаке исходных точек. При этом суммарное среднеквадратичное отклонение всех точек равно нулю.

```
Clear[a, b, x]

data = {{1.0, 0.92}, {2.1, 0.68}, {2.9, 0.5}, {3.7, 0.3}, {5.0, 0.1}};

model = Cos[a x] Exp[-b x];

fit = FindFit[data, model, {a, b}, x]

{a → 0.290623, b → 0.0928508}

modelf = Function[{x}, Evaluate[model /. fit]]

Function[{x}, e^(-0.0928508 x) Cos[0.290623 x]]

Plot[modelf[x], {x, 0, 6}, Epilog → Map[Point, data]]
```

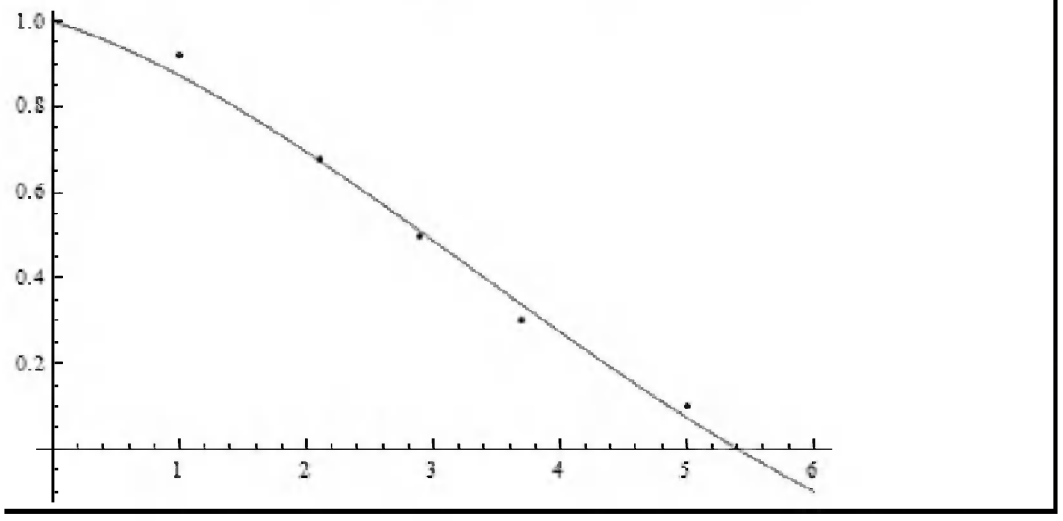


Рис. 6.55. Пример выполнения нелинейной регрессии

6.6.15. Полиномиальная регрессия

В подпакете PolynomialFit пакета NumericalMath определена функция для полиномиальной регрессии для полинома степени n без явного указания x^i:

- **PolynomialFit[data,n]** – возвращает полином степени **n**, обеспечивающий наилучшее среднеквадратичное приближение для данных, представленных data. Если **data** представлено списком ординат функции, то абсциссы форми-

руются автоматически с шагом 1. Если data представлен списком координат {xi,yi}, то полином наилучшим образом приближает зависимость $y_i(x_i)$.

Ниже представлен пример на применение функции полиномиальной аппроксимации:

```
<<NumericalMath'PolynomialFit'
p = PolynomialFit[ { 1,3.9,4.1,8.9,16,24.5,37,50 } ,3]
   (FittingPolynomial[<>,  3 ])
p[5.]
 25.
Expand[p[x]]
 0.×10^-11 x + 1.00000000000x^2 + 0.×10^-13 x^3
```

6.6.16. Тригонометрическая регрессия

Многие выражения содержат периодические тригонометрические функции, например sin(x) или cos(x). Помимо обычного спектрального представления выражений, подпакет TrigFit пакета NumericalMath имеет функции для *тригонометрической регрессии*:

- **TrigFit[data, n, x]** – дает тригонометрическую регрессию для данных data с cos(n x) и sin(n x) и с периодом 2π.
- **TrigFit[data, n, {x, L}]** – дает тригонометрическую регрессию для данных data с cos(2πnx/L) и sin(2πnx/L) и с периодом L.
- **TrigFit[data, n,{x, x0, x1}]** – дает тригонометрическую регрессию для данных data с cos(2πn(x-x0/(x1-x0)) и sin(2πn(x-x0/(x1-x0)) и с периодом (x1−x0).

Примеры тригонометрической регрессии даны ниже:

```
<<NumericalMath'TrigFit'
data = Table[1+Sin[x]+3Cos[2x]+3 Sin[3x],
          {x, 0, 2Pi-2Pi/7, 2Pi/7}];
TrigFit[data, 0, x]
 1.
TrigFit[data, 1, {x, L}]
```

$$\left(1. + 0.\,\mathrm{Cos}\left[\frac{2\pi x}{L}\right] + 1.\,\mathrm{Sin}\left[\frac{2\pi x}{L}\right]\right)$$

```
Fit[Transpose[{Range[0, 2Pi-2Pi/7, 2Pi/7], data}],
              {1, Cos[x], Sin[x]}, x]
 1. + 3.4766×10^-16 Cos[x] + 1. Sin[x]
TrigFit[data, 3, {x, x0, x1}]
```

$$1. + 0.\,\mathrm{Cos}\left[\frac{2\pi\,(x-x0)}{-x0+x1}\right] + 3.\,\mathrm{Cos}\left[\frac{4\pi\,(x-x0)}{-x0+x1}\right] -$$

$$4.44089\times10^{-16}\,\mathrm{Cos}\left[\frac{6\pi\,(x-x0)}{-x0+x1}\right] + 1.\,\mathrm{Sin}\left[\frac{2\pi\,(x-x0)}{-x0+x1}\right] +$$

$$4.44089\times10^{-16}\,\mathrm{Sin}\left[\frac{4\pi\,(x-x0)}{-x0+x1}\right] + 3.\,\mathrm{Sin}\left[\frac{6\pi\,(x-x0)}{-x0+x1}\right]$$

Этот вид регрессии применяется редко.

6.7. Средства интерполяции и аппроксимации системы Mathcad

В СКМ Mathcad возможности интерполяции и аппроксимации намного меньше, чем в системах Maple или Mathematica. Но есть и приятные исключения. Удобно организована сплайновая интерполяция с продолжением (экстраполяцией по первому или последнему сплайну). Есть ряд функций нелинейной регрессии, в том числе общего вида. И наконец, есть уникальная функция predict для прогноза авторегрессионным методом Бурга (Burge). Иногда его называют методом Берга.

6.7.1. Одномерная линейная интерполяция и экстраполяция

Для *кусочно-линейной интерполяции* в СКМ Mathcad используется следующая функция `linterp(VX,VY,x)`. Для заданных векторов абсцисс `VX` и ординат `VY` узловых точек и заданного аргумента `x` функция `linterp` возвращает значение функции при ее кусочно-линейной аппроксимации (интерполяции). При экстраполяции используются отрезки прямых, проведенных через две крайние точки.

6.7.2. Одномерная сплайновая интерполяция и экстраполяция

Для осуществления сплайновой интерполяции и экстраполяции Mathcad предлагает четыре встроенные функции. Три из них служат для получения векторов вторых производных сплайн-функций при различных видах продолжения (экстраполяции):

- `cspline(VX, VY)` – возвращает вектор **VS** вторых производных с продолжением по кубическому полиному;
- `pspline(VX, VY)` – возвращает вектор **VS** вторых производных с продолжением по параболической кривой;
- `lspline(VX, VY)` – возвращает вектор **VS** вторых производных с линейным продолжением.

Наконец, четвертая функция `interp(VS, VX, VY, x)` возвращает значение $y(x)$ для заданных векторов **VS, VX, VY** и заданного значения x.

Таким образом, сплайн-интерполяция проводится в два этапа. На первом с помощью функции `cspline`, `pspline` или `lspline` отыскивается вектор вторых производных функции `y(x)`, заданной векторами `VX` и `VY` ее значений (абсцисс и ординат). Затем на втором этапе для каждой искомой точки вычисляется значение `y(x)` с помощью функции `interp`.

На рис. 6.56 показан фрагмент документа Mathcad, иллюстрирующий применение описанных функций для линейной и сплайновой интерполяции при трех

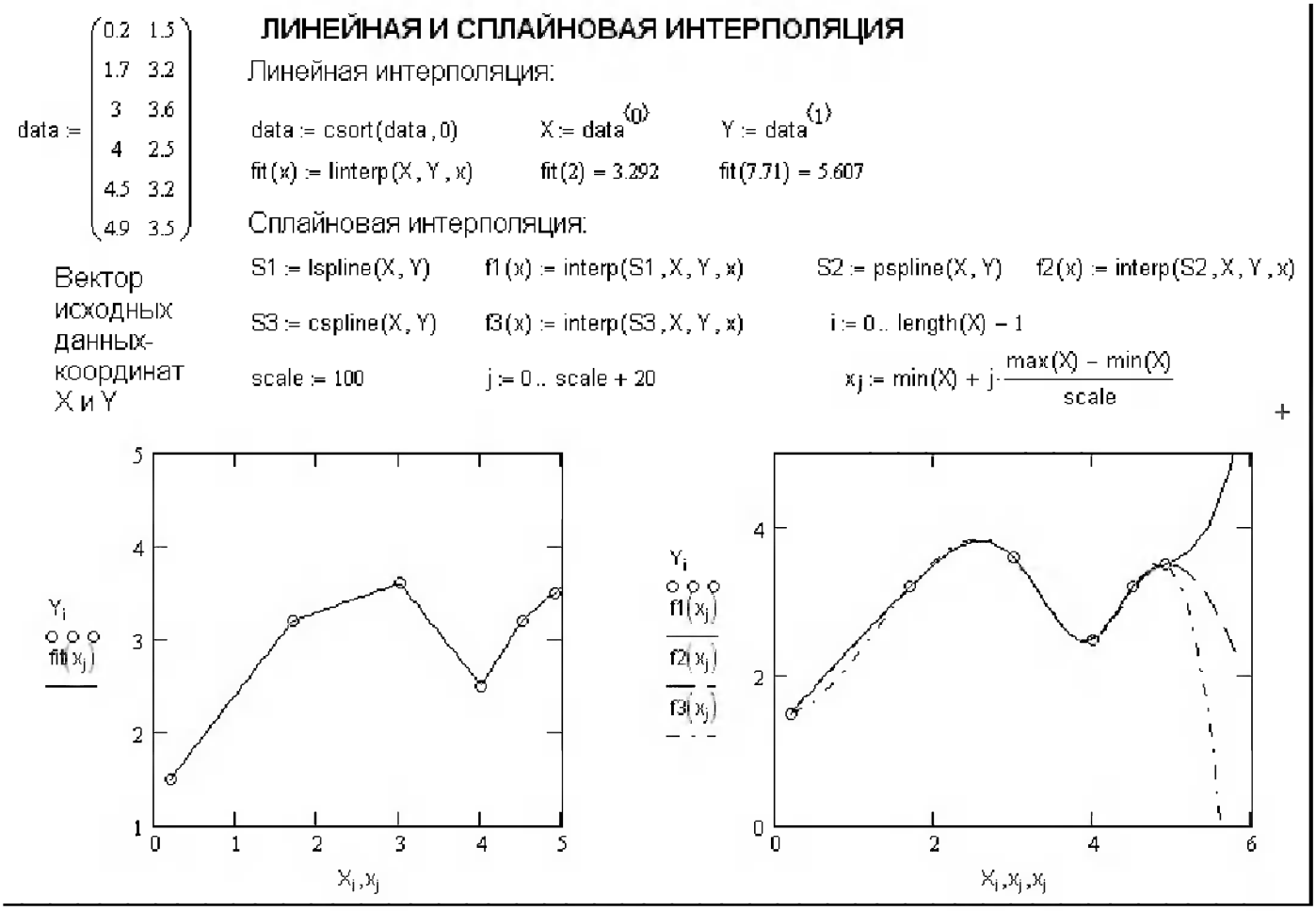


Рис. 6.56. Линейная и сплайновая интерполяция

видах сплайнов – функций f1, f2 и f3. Они соответствуют сплайнам с линейным, квадратическим и кубическим продолжением.

Как отмечалось, сплайновая интерполяция может использоваться для экстраполяции с тремя видами продолжения функций – линейным, параболическим и кубическим. Такая экстраполяция представлена на рис. 6.56 отрезками кривых в интервале x от 5 до 6. Нетрудно заметить, что качество экстраполяции довольно низкое и сильно зависит от того, какие сплайны выбраны: 1 – с линейным продолжением, 2 – с параболическим продолжением и 3 – с кубическим продолжением (эти цифры относятся соответственно к переменным Sn и функциям fn). Низкое качество экстраполяции оправдывает название «продолжение функций».

Может показаться, что в связи с низким качеством экстраполяции само ее существование практически бесполезно. Но это далеко не так! При использовании сплайновой интерполяции для приближения нелинейных зависимостей в задачах моделирования приближение функций за пределами области узловых точек позволяет избежать разрывов функции, которые способны резко исказить ход моделирования. Другое дело, что продолжение едва ли уместно использовать для решения серьезных задач предсказания и прогноза.

6.7.3. Одномерная B-сплайновая интерполяция и экстраполяция

В Mathcad введена и функция аппроксимации *B-сплайнами* `bspline(VX,VY,U,n)`, где `U` – вектор координат точек сшивки, n=1, 2 или 3 – степень полиномов. Размер вектора `U` на $(n-1)$ меньше размера векторов x и y. B-сплайновая аппроксимация имеет продолжение и позволяет задавать сшивку сплайн-функций в точках, отличных от узловых. Крайняя левая точка сшивки должна быть левее, а правая – правее конечных узловых точек. На рис. 6.57 представлены примеры интерполяции и продолжения с применением сплайн-функций с линейным ($n = 1$), квадратичным ($n = 2$) и кубическим ($n = 3$) продолжением.

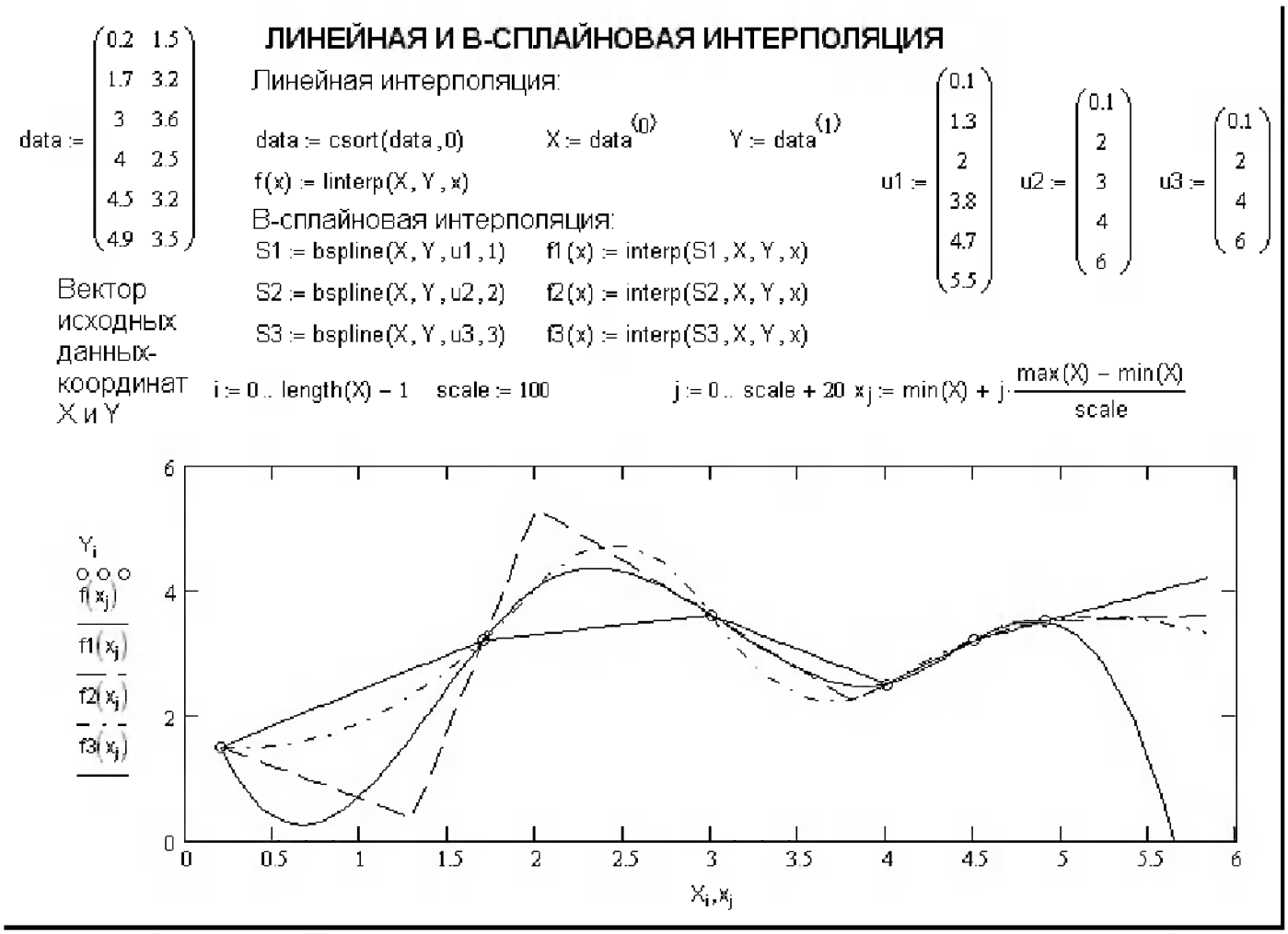


Рис. 6.57. Линейная и B-сплайновая интерполяция и экстраполяция

Разумеется, качество B-сплайновой интерполяции сильно зависит от правильного положения точек сшивки и, конечно, от набора исходных узловых точек. Как видно из рис. 6.57, эта интерполяция может использоваться и для экстраполяции. Однако хорошей назвать экстраполяцию и в этом случае трудно.

6.7.4. Двумерная линейная и сплайновая интерполяция

В Mathcad 12 имеется возможность выполнения двумерной сплайновой интерполяции. Это позволяет существенно повысить представительность сложных графиков функций, в том числе контурных (рис. 6.58).

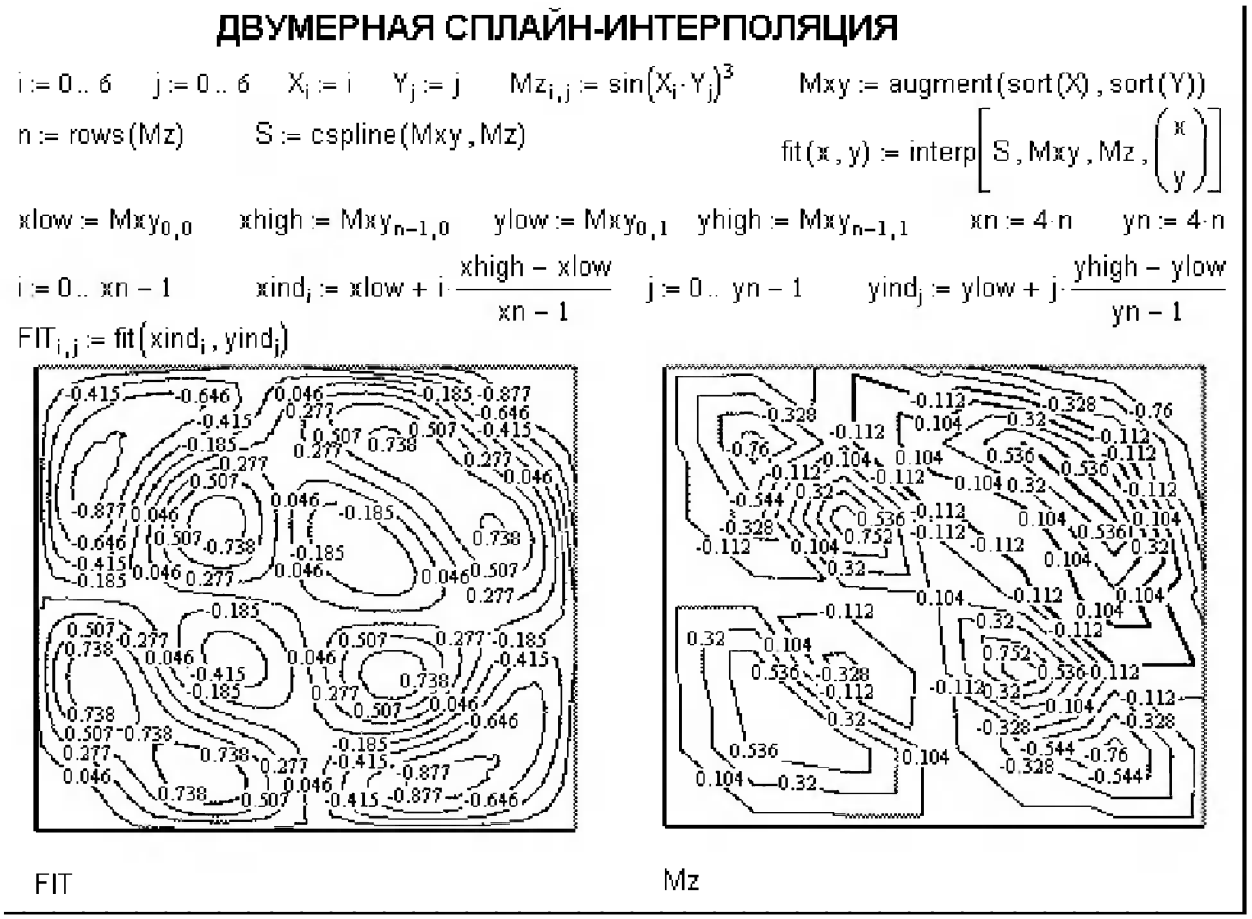


Рис. 6.58. Пример применения двухмерной сплайновой интерполяции для построения контурного графика сложной поверхности

На этом рисунке слева показан контурный график после двумерной сплайновой интерполяции, а справа – без нее (с применением линейной интерполяции). Достоинства двумерной сплайновой интерполяции очевидны и в том случае, если представить графики рис. 6.58 в виде поверхностей, просто переформатировав их.

6.7.5. Интерполяция и экстраполяция функций по Лагранжу

Для интерполяции и экстраполяции по формуле Лагранжа можно воспользоваться документом, показанным на рис. 6.59. Эта формула использует только данные об узловых точках функции. Замечательно, что в таблице узлы могут быть записаны в произвольном порядке!

Рис. 6.59. Пример интерполяции табличных данных с помощью обобщенной формулы Лагранжа

Недостатками обобщенной формулы Лагранжа являются ее сложность и отсутствие явного выражения для аппроксимирующего полинома. Тем не менее это единая аппроксимирующая формула, и в ряде случаев ее применение удобно и полезно, например если число узловых точек приходится часто менять. Иногда при вычислениях по этой формуле может происходить переполнение разрядной сетки чисел, и тогда вычисления будут сопровождаться ошибкой.

6.7.6. Полиномиальная аппроксимация

Несложно выполнить и *полиномиальную аппроксимацию* таблично заданной функции. Пример этого представлен на рис. 6.60. Здесь аппроксимируется N-образная вольтамперная характеристика (ВАХ) туннельного диода. После этого можно выполнять вычисления ВАХ как в узловых точках, так и между ними и даже немного за пределами области задания узловых точек (экстраполяция).

К сожалению, для более сложных видов полиномиальной аппроксимации (рациональной или Паде, Чебышевской, минимаксной и др.) специальных средств в обычных версиях Mathcad нет. Но их можно найти в расширении Numeric Recipes или реализовать программно.

Рис. 6.60. Полиномиальная аппроксимация ВАХ туннельного диода

6.7.7. Линейная регрессия

Для проведения линейной регрессии система Mathcad имеет следующие функции:

- `corr(VX,VY)` – возвращает скаляр – коэффициент корреляции Пирсона;
- `intercrpt(VX,VY)` – возвращает значение параметра `a` (смещение линии регрессии по вертикали);
- `slope(VX,VY)` – возвращает значение параметра `b` (угловой коэффициент линии регрессии).

На рис. 6.61 показан фрагмент документа Mathcad с примером проведения линейной регрессии для данных, представленных значениями элементов в векторах `VX` и `VY`.

Как видно из рисунка, линия регрессии проходит в «облаке» исходных точек с максимальным среднеквадратичным приближением к ним. Чем ближе коэффициент корреляции к 1, тем точнее представленная исходными точками зависимость приближается к линейной.

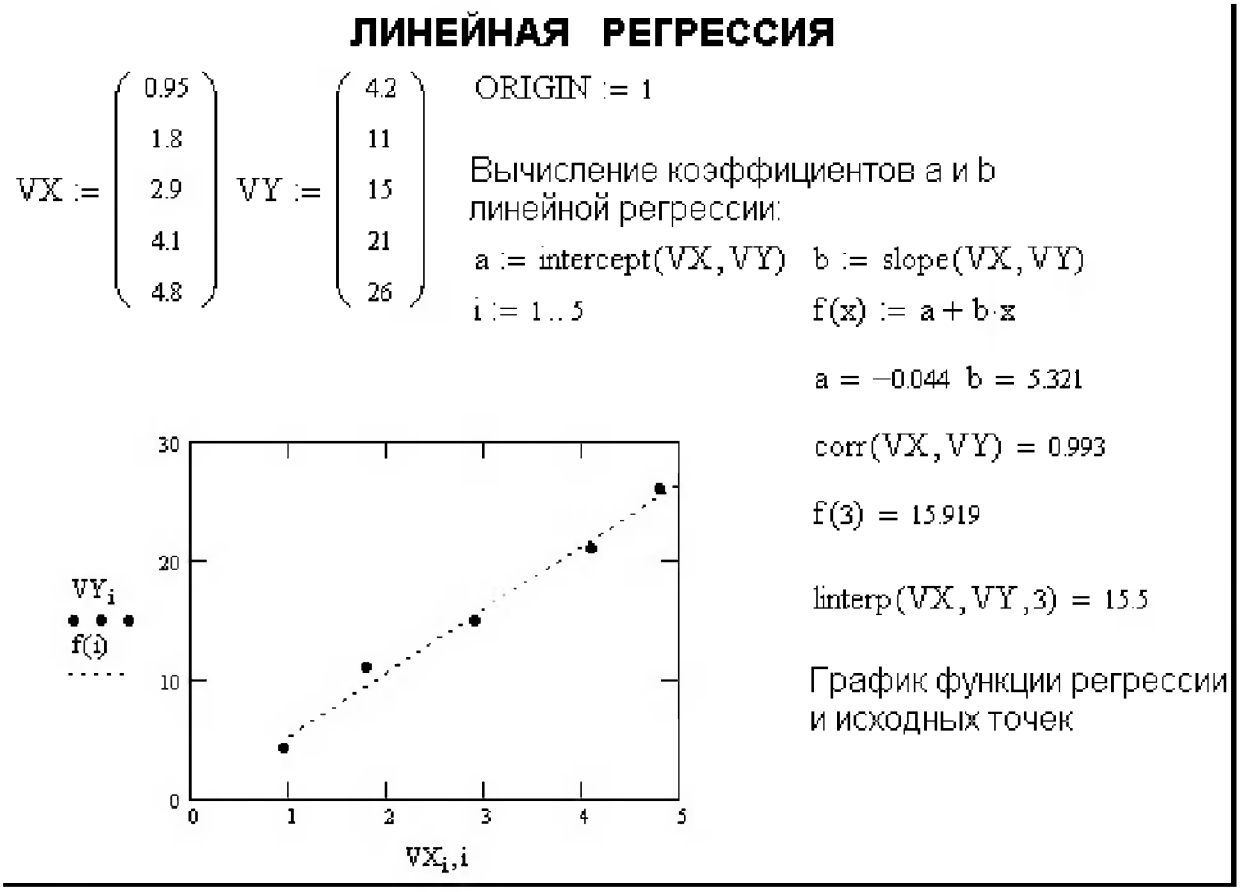


Рис. 6.61. Линейная регрессия

6.7.8. Реализация линейной регрессии общего вида

В Mathcad реализована возможность выполнения линейной регрессии общего вида. При ней заданная совокупность точек приближается к функции вида:

$$F(x,K1,K2,...,Kn) = K1 \cdot F1(x) + K2 \cdot F2(x) + ... + Kn \cdot Fn(x).$$

Таким образом, функция регрессии является линейной комбинацией функций `F1(x)`, `F2(x)`, ..., `Fn(x)`, причем сами эти функции могут быть нелинейными, что резко расширяет возможности такой аппроксимации и распространяет ее на многие нелинейные функции.

Для реализации линейной регрессии общего вида используется функция `linfit(VX,VY,F)`, возвращающая вектор `K` коэффициентов линейной регрессии общего вида, при котором среднеквадратичная погрешность приближения «облака» исходных точек, координаты которых хранятся в векторах `VX` и `VY`, оказывается минимальной. Вектор `F` должен содержать функции `F1(x)`, `F2(x)`, ..., `Fn(x)`, записанные в символьном виде.

Рисунок 6.62 поясняет проведение линейной регрессии общего вида с применением функции `linfit`. Процедура проведения вычислений настолько проста, что не нуждается в особых комментариях.

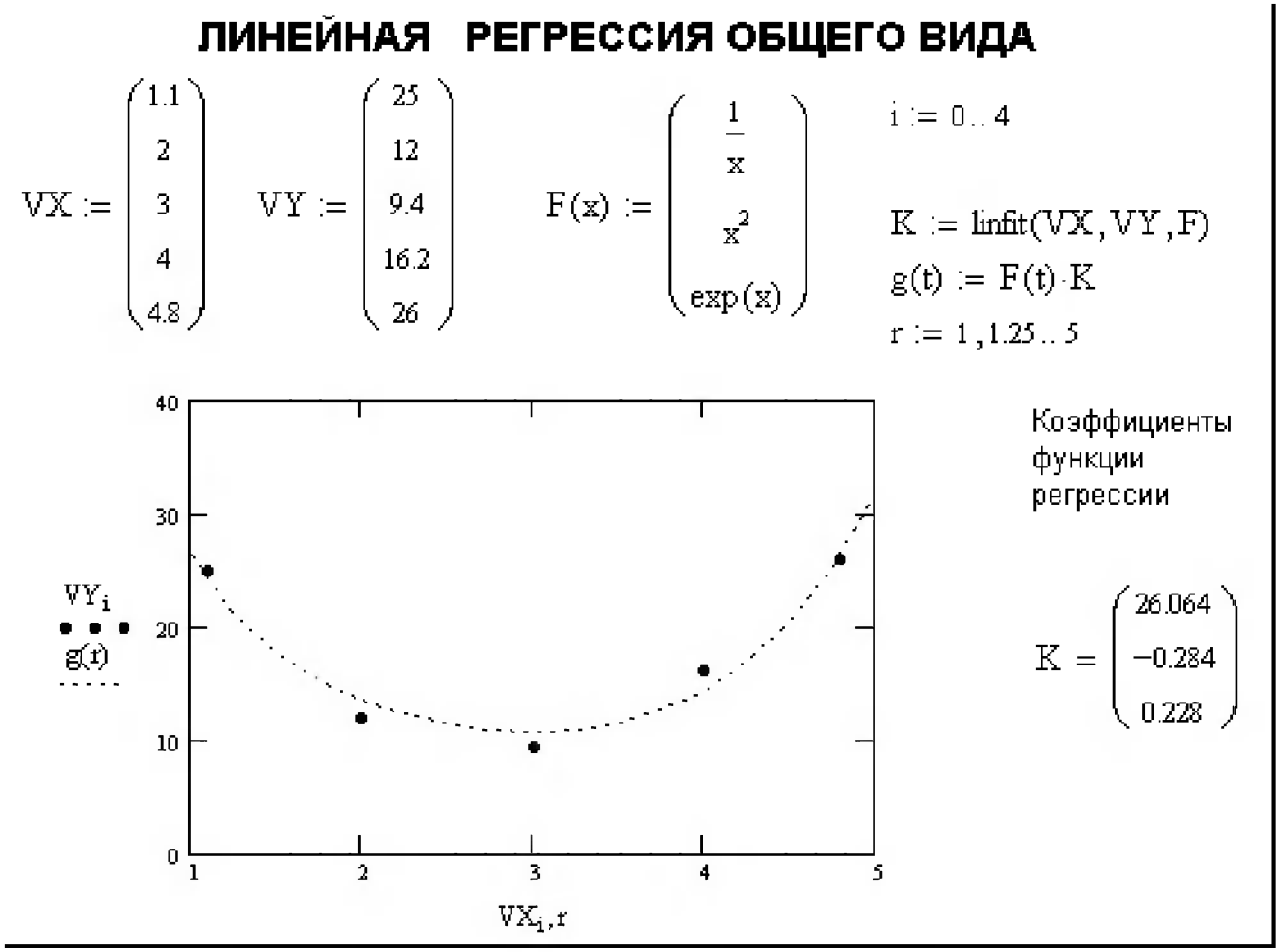


Рис. 6.62. Пример проведения линейной регрессии общего вида

Расположение абсцисс точек исходного массива в векторе VX может быть любым, но они должны идти в порядке возрастания. Вектор ординат VY должен содержать ординаты точек с абсциссами, указанными в векторе VX.

6.7.9. Реализация одномерной полиномиальной регрессии

В Mathcad введена и функция для обеспечения полиномиальной регрессии при произвольной степени полинома регрессии. Регрессия осуществляется функцией `regress(VX,VY,n)`, которая возвращает вектор VS, запрашиваемый функцией `interp(VS,VX,VY,x)` и содержащий коэффициенты многочлена n-й степени, который наилучшим образом приближает «облако» точек с координатами, хранящимися в векторах VX и VY.

На рис. 6.63 показан фрагмент документа Mathcad с примером выполнения полиномиальной регрессии. Для вычисления коэффициентов полинома регрессии используется функция `submatrix`.

Функция `regress` создает единственный приближающий полином, коэффициенты которого вычисляются по всей совокупности заданных точек. Иногда по-

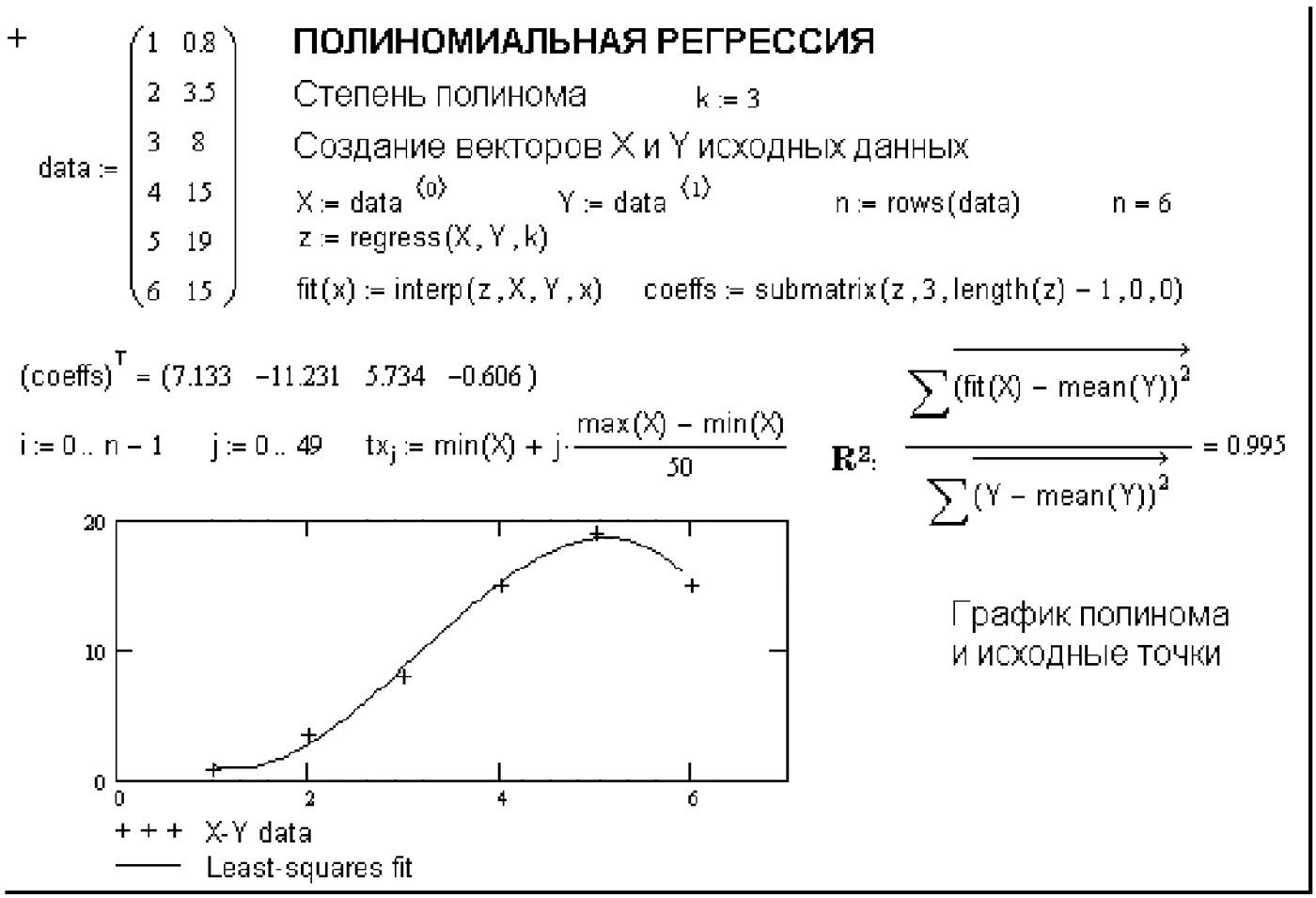


Рис. 6.63. Полиномиальная регрессия в системе Mathcad

лезна другая функция полиномиальной регрессии, дающая локальные приближения отрезками полиномов второй степени `loess(VX,VY,span)`, которая возвращает вектор `VS`, используемый функцией `interp(VS,VX,VY,x)` для наилучшего приближения данных векторов `VX` и `VY` отрезками полиномов второй степени. Аргумент `span>0` указывает размер локальной области приближаемых данных (рекомендуемое начальное значение – 0,75). Чем больше `span`, тем сильнее сказывается сглаживание данных. При больших значениях `span` эта функция приближается к функции `regress(VX,VY,2)`.

На рис. 6.64 показан фрагмент документа Mathcad с примером приближения сложной функции со случайным разбросом ее значений с помощью совокупности отрезков полиномов второй степени (функция `loess`) для двух значений параметра `span`.

Из рисунка нетрудно заметить, что при значении `span = 0.05` отслеживаются характерные случайные колебания значений функции, тогда как уже при `span = 0.5` кривая регрессии становится практически гладкой. К сожалению, из-за отсутствия простого описания аппроксимирующей функции в виде отрезков полиномов этот вид регрессии широкого применения не нашел.

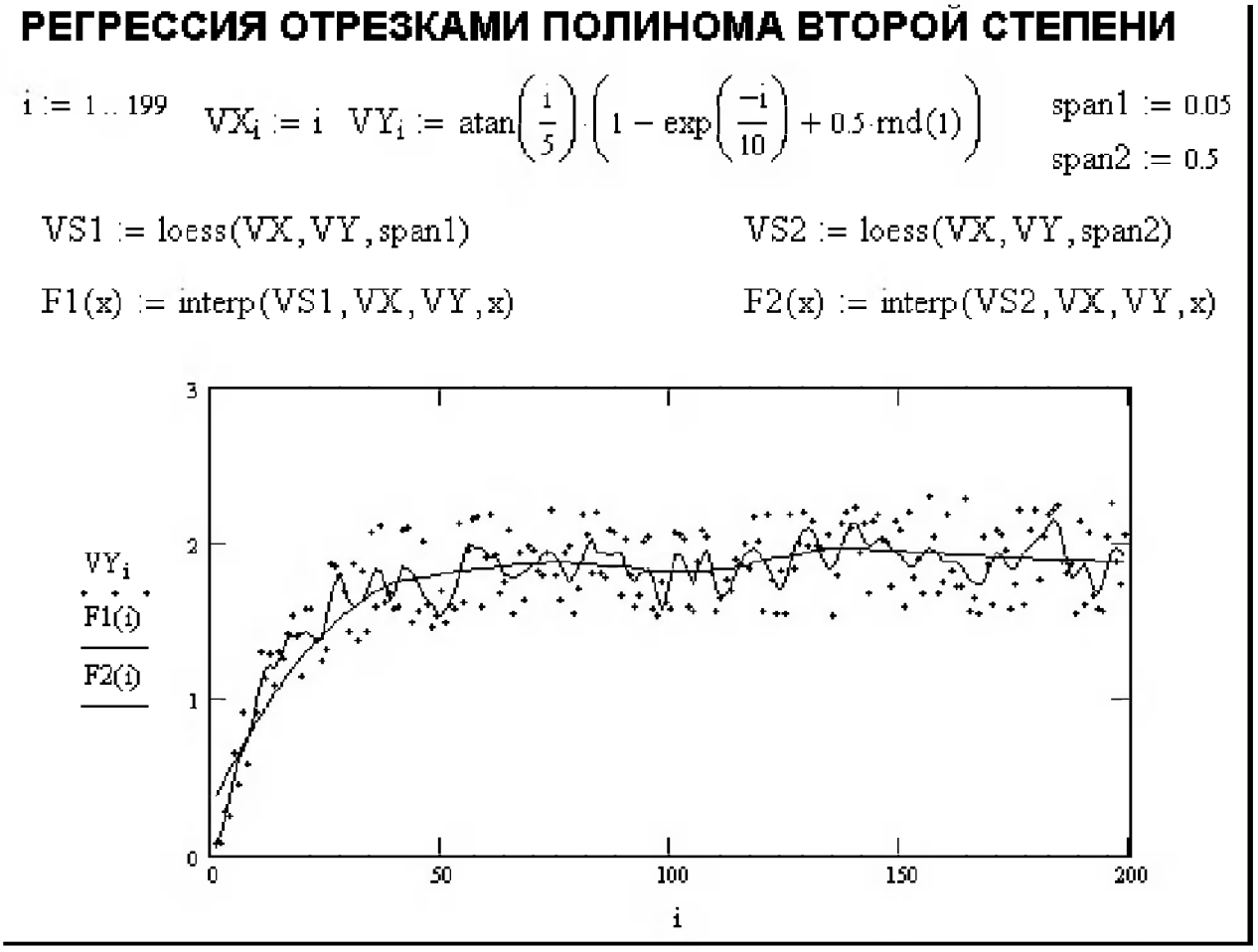


Рис. 6.64. Пример регрессии отрезками полиномов второй степени

6.7.10. Проведение многомерной регрессии

Mathcad позволяет выполнять также многомерную регрессию. Самый типичный случай ее использования – приближение поверхностей в трехмерном пространстве. Их можно описать, задав массив значений высот `z`, соответствующих двумерному массиву `Mxy` координат точек (x,y) на горизонтальной плоскости.

Новых функций для этого не задано. Используются уже описанные ранее функции, но в несколько иной форме:

- `regress(Mxy,Vz,n)` – возвращает вектор, запрашиваемый функцией `interp(VS,Mxy,Vz,V)` для вычисления многочлена `n`-й степени, который наилучшим образом приближает точки множества `Mxy` и `Vz` (`Mxy` – матрица размера m × 2, содержащая координаты `x` и `y`, `Vz` – `m`-мерный вектор, содержащий `z`-координаты, соответствующие `m` точкам, указанным в `Mxy`);
- `loess(Mxy,Vz,span)` – аналогична `loess(VX,VY,span)`, но в многомерном случае;
- `interp(VS,Mxy,Vz,V)` – возвращает значение `z` по заданным векторам `VS` (создается функцией `regress` или `loess`) и `Mxy`, `Vz` и `V` (вектор координат `x` и `y` заданной точки, для которой находится `z`).

Пример многомерной интерполяции был приведен выше. В целом многомерная регрессия применяется сравнительно редко из-за сложности сбора исходных данных.

6.7.11. Проведение нелинейной регрессии общего вида

Для проведения нелинейной регрессии общего вида в Mathcad используется функция `genfit(VX,VY,VS,F)`, которая возвращает вектор `K` параметров функции `F`, дающий минимальную среднеквадратичную погрешность приближения функцией `F(x,K1,K2,…,Kn)` исходных данных.

Вектор `F` должен быть вектором с символьными элементами, причем они должны содержать аналитические выражения для исходной функции и ее производных по всем параметрам. Вектор `VS` должен содержать начальные значения элементов вектора `K`, необходимые для решения системы нелинейных уравнений регрессии итерационным методом.

На рис. 6.65 показан фрагмент документа Mathcad с примером выполнения нелинейной регрессии общего вида с помощью нелинейной функции F(x,a,b) = a·exp(-b·x) + a·b.

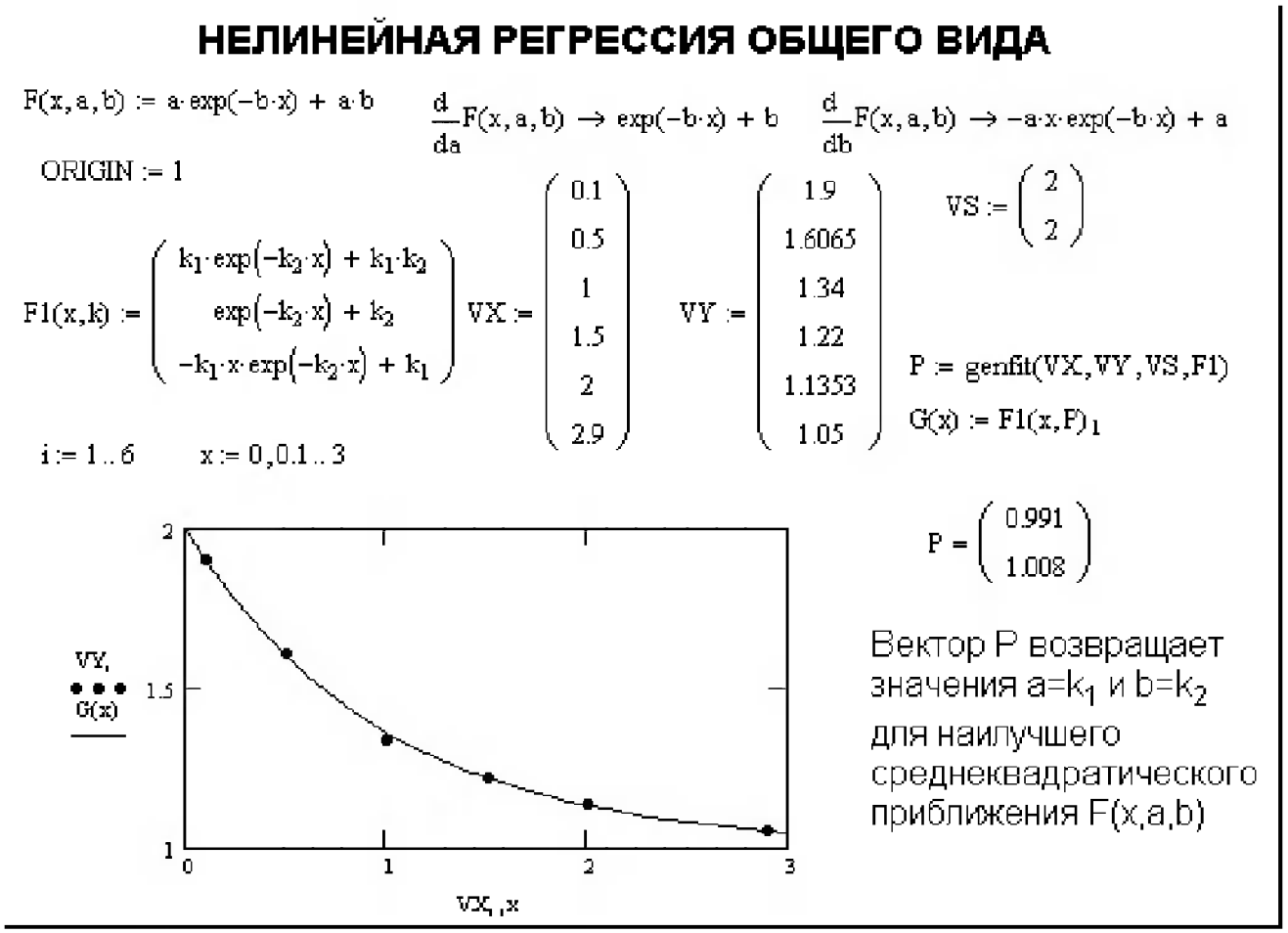


Рис. 6.65. Пример выполнения нелинейной регрессии общего вида

При решении этой задачи возникают две проблемы. Во-первых, надо вычислить значения производных по переменным a и b. Как показано на рисунке, это сделано средствами символьных операций (первая строка документа после заголовка), что наглядно иллюстрирует пользу от таких операций.

Вторая проблема связана с необходимостью применения функции genfit в ее стандартном виде. Поэтому пришлось заменить параметр a на k1, а параметр b – на k2. Остальные операции в примере достаточно очевидны.

6.7.12. Новые функции для проведения регрессии в Mathcad

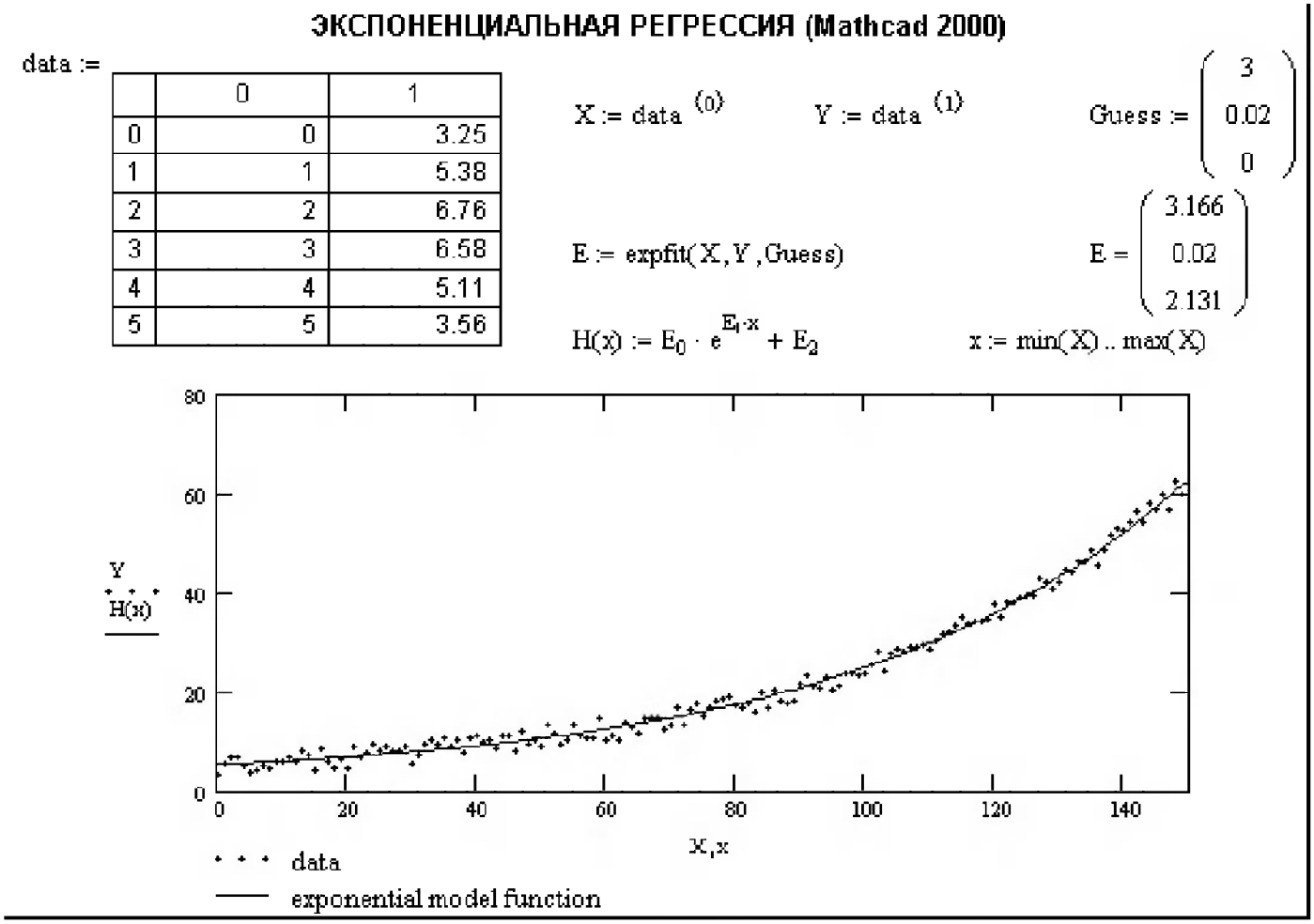
В последние версии Mathcad, начиная с Mathcad 2000, был введен ряд новых функций регрессии:

- expfit(vx,vy,vg) – возвращает вектор, содержащий коэффициенты (a, b и c) аппроксимирующего выражения вида a·e(b·x) + c, график которого лучшим образом приближается к точкам, координаты которых хранятся в векторах vx и vy (вектор vg содержит первое приближение к решению);
- lgsfit(vx,vy,vg) – то же, но для выражения a/(1 + b·e(-c·x));
- logfit(vx,vy) – то же, но для выражения a·ln(x + b) + c (начального приближения не требуется);
- medfit(vx,vy) – то же, но для выражения a + bx (начального приближения не требуется);
- pwrfit(vx,vy,vg) – то же, но для выражения a·xb + c (вектор vg содержит первое приближение к решению);
- sinfit(vx,vy,vg) – то же, но для выражения a·sin(x + b) + c.

6.7.13. Пример выполнения экспоненциальной регрессии

Поскольку все частные виды регрессии с помощью приведенных выше функций Mathcad выполняются по одной схеме, ограничимся двумя примерами ее проведения. Фрагмент документа Mathcad с примером на экспоненциальную регрессию приведен на рис. 6.66.

Исходные данные представлены матрицей data, имеющей два столбца. Нулевой столбец содержит координаты точек исходных данных, а первый – их абсциссы. Перед вычислением задаются начальные приближения параметров экспоненциальной регрессии – вектор Guess. Сама регрессия реализуется одним выражением с функцией expfit. На рисунке представлены график функции регрессии и исходные точки.



	0	1
0	0	3.25
1	1	5.38
2	2	6.76
3	3	6.58
4	4	5.11
5	5	3.56

Рис. 6.66. Пример экспоненциальной регрессии

6.7.14. Пример выполнения синусоидальной регрессии

Еще один вид регрессии – синусоидальной – представляет фрагмент документа Mathcad, показанный на рис. 6.67. Здесь исходные векторы данных формируются путем добавления к значениям синусоидальной функции случайных чисел. Регрессия реализуется функцией `sinfit`.

Рекомендуется разобраться в небольших отличиях при реализации регрессии в приведенных выше примерах.

6.7.15. Приближение данных рядом Фурье

Помимо полиномов, существует и такой единообразный метод представления самых различных зависимостей, как ряд Фурье. Когда коэффициенты Фурье вычисляются методом прямоугольников, то ряд Фурье фактически приближает ис-

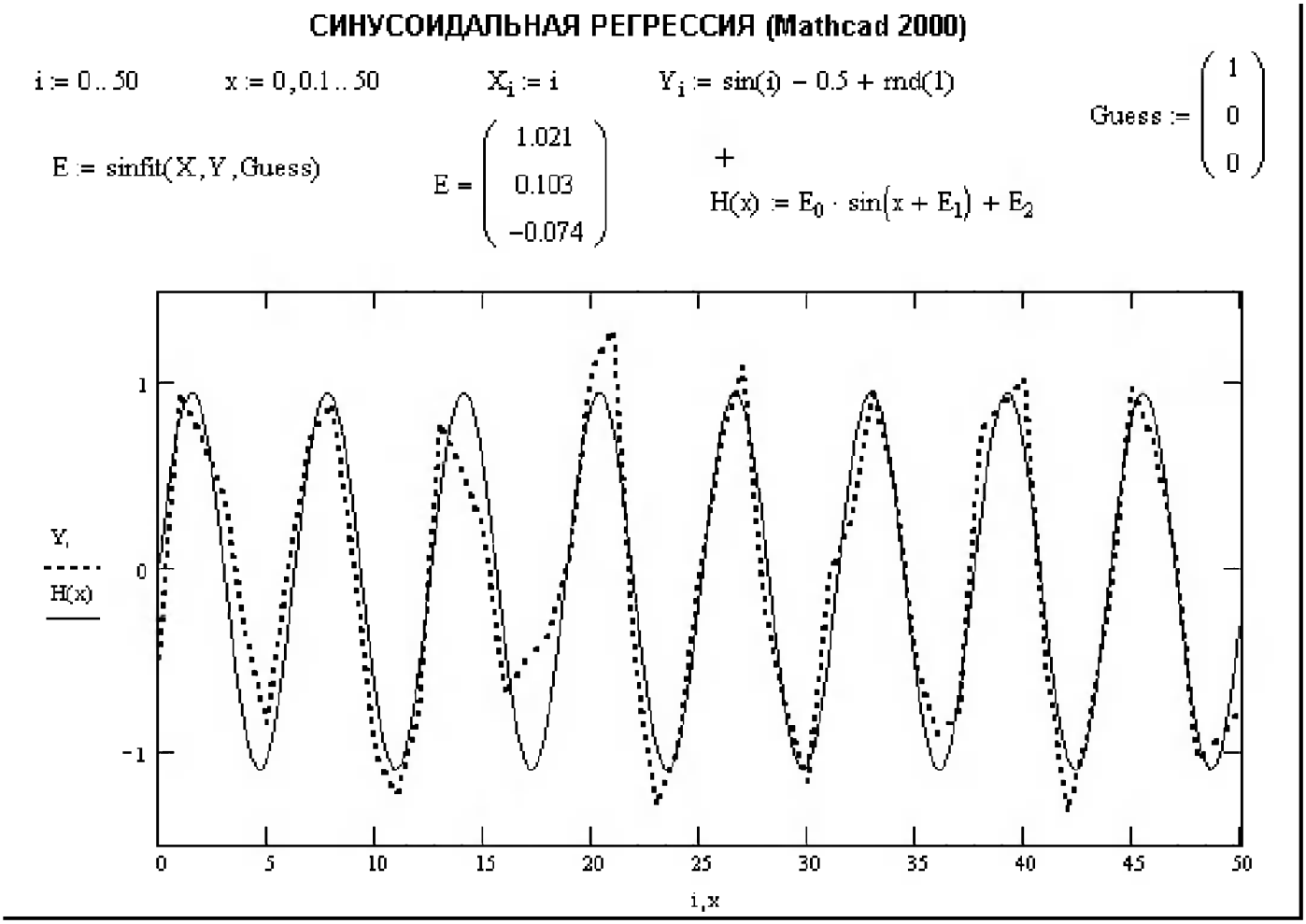


Рис. 6.67. Пример синусоидальной регрессии

ходную зависимость (облако ее узловых точек) с наименьшей среднеквадратической погрешностью.

На рис. 6.68 представлено начало документа, в котором реализовано приближение зашумленных данных рядом Фурье. Показано задание исходной зависимости $f(x)$, вектора ее значений Y0 и вектора зашумленных данных Y.

Далее на рис. 6.69 представлены вычисление коэффициентов Фурье методом прямоугольников (программные модули) и построение ограниченного числом гармоник nmax = 10 ряда Фурье. Нетрудно заметить, что за исключением концевых точек ряд Фурье неплохо приближает облако точек исходной зависимости. Поскольку мы имеем функцию приближения в явном виде, можно говорить о полноценной аппроксимации, равно как и о довольно эффективном сглаживании данных. Из рис. 6.69 ясно, что спектр Фурье быстро затухает, что позволяет отбросить высшие гармоники и тем самым обеспечить эффективное сглаживание данных.

Большой «ложкой дегтя» в таком приближении оказывается злополучный эффект Гиббса. Он проявляется как плохое схождение ряда Фурье на концах отрезка приближения (в нашем случае в точках с x = 0 и x = 2). При этом наблюдается

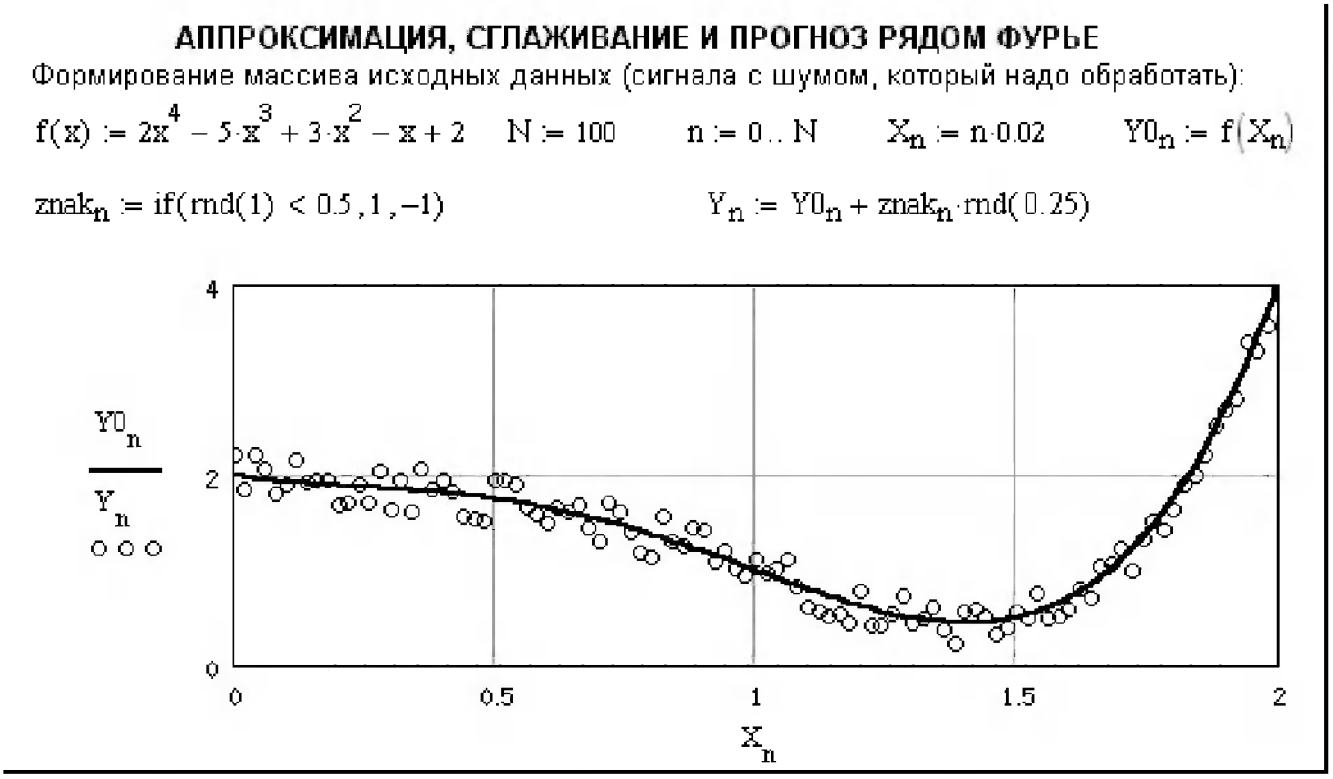


Рис. 6.68. Начало документа с примером приближения данных рядом Фурье

совершенно недопустимый выброс кривой аппроксимации и волнообразный ее ход даже в центре графика.

6.7.16. Улучшение сходимости приближения рядом Фурье

Одним из главных способов улучшения сходимости рядов Фурье является устранение разрывов исходных функций в начале и в конце интервала приближения. Если данные представлены отдельными точками (узлами), то для этого достаточно использовать итерационную формулу, представленную в начале рис. 6.70.

В результате пересчета точек исходной зависимости получается новая зависимость с нулями в конце интервала приближения. Такую зависимость можно представить рядом Фурье с только синусными членами, что дает сразу несколько важных достоинств:

- заметно сужается спектр Фурье, что позволяет использовать меньшее число гармоник;
- ряд Фурье быстро сходится;
- проявление эффекта Гиббса резко уменьшается;
- появляется возможность экстраполяции (продолжения по синусам).

Так, в нашем случае оказалось достаточным ограничить число гармоник ряда Фурье значением 5 (на основании приведенной спектрограммы).

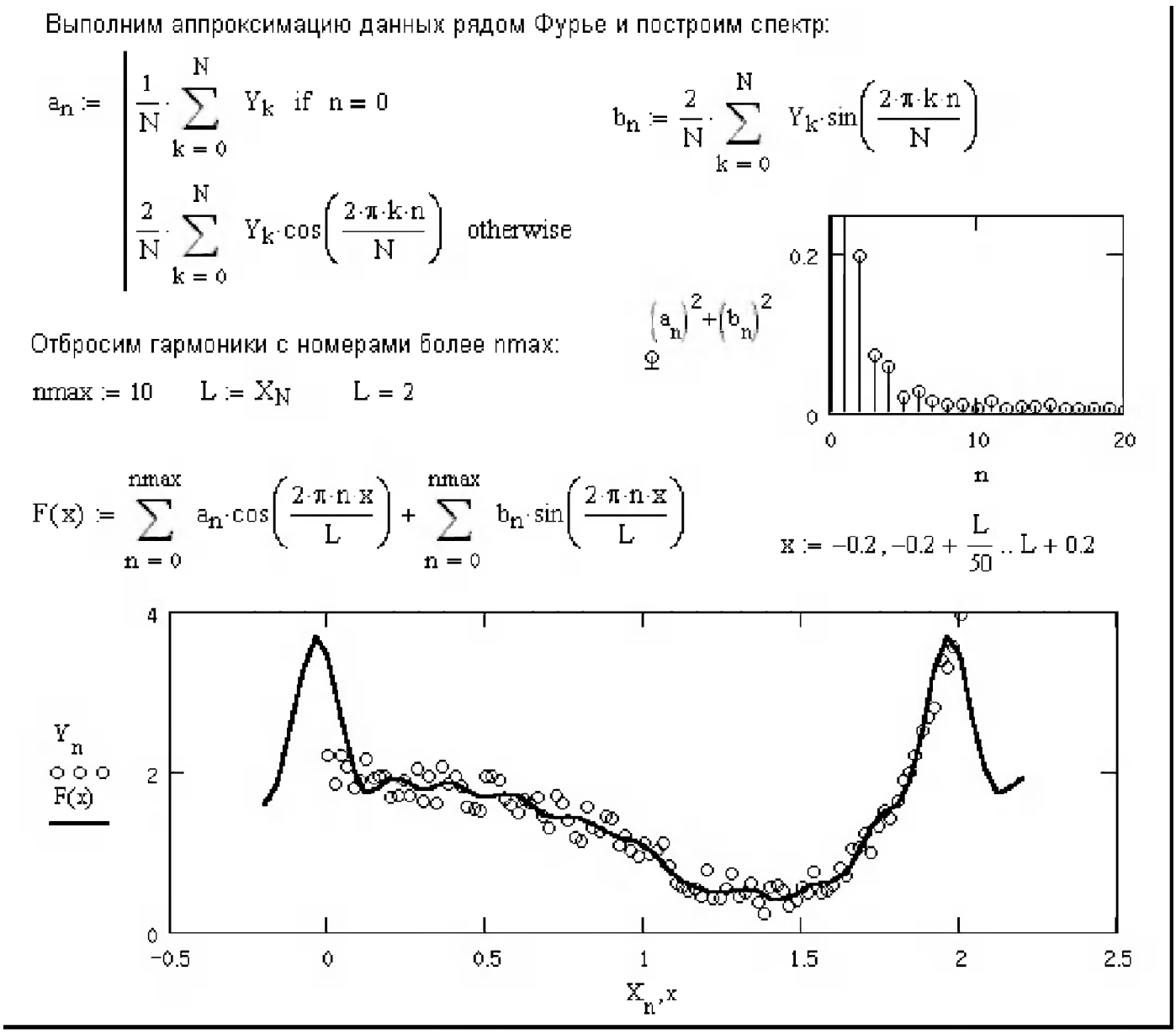


Рис. 6.69. Приближение данных рядом Фурье

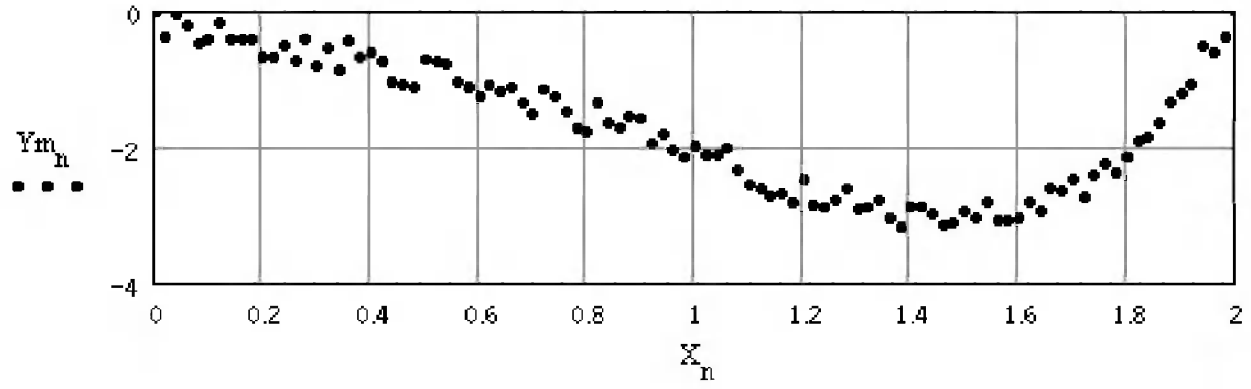
6.7.17. Эффективное приближение данных рядом Фурье

Теперь после преобразования данных можно получить весьма эффективное приближение их рядом Фурье с малым числом гармоник – рис. 6.71 (верхний график).

Используя далее обратную формулу преобразования, можно вернуть данным исходные значения и получить кривую регрессии, показанную на рис. 6.71 снизу. Нетрудно заметить, что полученная аппроксимирующая функция вполне удовлетворительно справляется с экстраполяцией.

Для улучшения аппроксимации модифицируем исходные данные, так, чтобы крайние точки имели близкое к нулю значения:

$$Ym_n := Y_n - Y_0 - \frac{Y_N - Y_0}{L} \cdot X_n$$



Используем для аппроксимации ряд Фурье с нечетным продолжением (по синусам) за края интервала. Аппроксимирующая функция при этом не будет иметь разрывов функции и ее производной, и ряд будет сходиться значительно лучше.

$$bm_n := \frac{2}{N} \cdot \sum_{k=1}^{N-1} Ym_k \cdot \sin\left(\frac{\pi \cdot k \cdot n}{N}\right)$$

Спектр функции узкий.
Можно ограничиться
числом членов рядя Фурье $nmax := 5$

$$Gm(x) := \sum_{n=1}^{nmax} bm_n \cdot \sin\left(\frac{\pi \cdot n \cdot x}{L}\right)$$

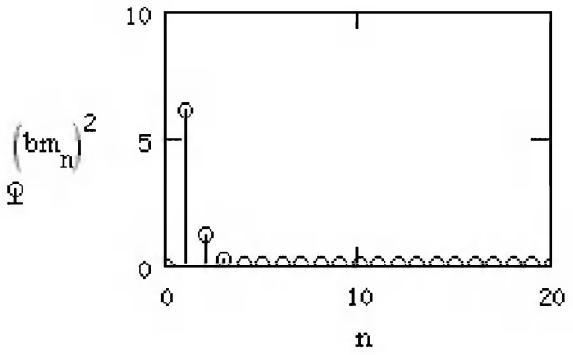


Рис. 6.70. Часть документа с примером приближения данных рядом Фурье с устранением разрывов в начале и конце отрезка приближения

6.8. Аппроксимация и регрессия в СКМ Derive и MuPAD

6.8.1. Функция FIT в Derive

Обработка числовых данных изначально не была главной целью систем символьной математики, ориентированных в основном на аналитические вычисления. Поэтому средства системы Derive поначалу могут показаться издевательски слабыми, ибо на уровне ядра они представлены единственной встроенной функцией **FIT** для проведения регрессии. Но не будем осуждать создателей Derive и разберемся в том, что же все-таки дает эта система в обработке численных данных.

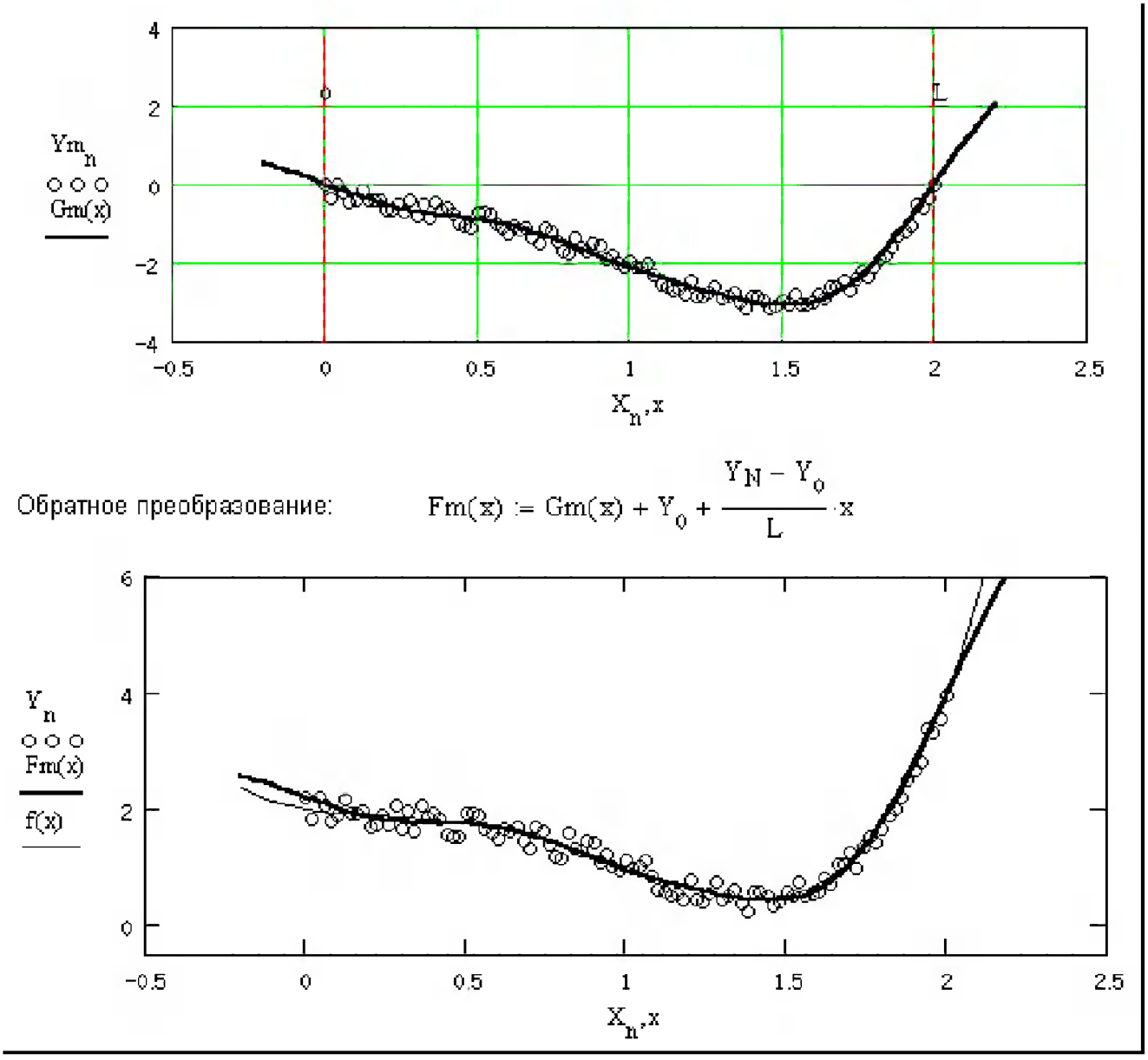


Рис. 6.71. Окончание документа с примером приближения данных рядом Фурье (сверху показано приближение преобразованной, а снизу – восстановленной исходной зависимости)

Функция **FIT(m)** выполняет обработку матрицы m методом наименьших квадратов. Под этим емким определением лежит возможность решения задач на регрессию различных зависимостей – от простейшей линейной регрессии до линейной и нелинейной регрессий общего вида. Продемонстрируем это на ряде конкретных примеров.

Линейная регрессия

Для решения этой задачи с помощью функции **FIT(m)** надо создать матрицу m, имеющую два столбца. Верхняя строка матрицы должна содержать переменную x и функцию $y(x)$ – в нашем случае это $a \cdot x + b$. Остальные строки должны содержать пары значений x_i и y_i, то есть исходные данные (например, экспериментальные). Вычисление функции **FIT(m)** с помощью команды approX даст иско-

мую аппроксимирующую функцию в виде формулы с численными значениями параметров *a* и *b*. Следующий пример поясняет это:

```
1:   "Реализация метода наименьших квадратов"
2:   "Линейная регрессия"
         /  x   a x + b \
         |               |
         |  2     5.5    |
         |               |
         |  4     6.3    |
3:   FIT |               |
         |  6     7.2    |
         |               |
         | 10     8.6    |
         |               |
         \  8      8     /
4:   "Используем команду approX"
5:   0.395 x + 4.75
```

Итак, в данном случае облако точек исходной зависимости приближено функцией 0,369*x*+4,75.

Логарифмическая регрессия

Столь же просто можно провести логарифмическую регрессию:

```
1:   "Реализация метода наименьших квадратов"
2:   "Логарифмическая регрессия"
         / x   a + b LOG (x, 10) \
         |                       |
         | 1           1         |
         |                       |
3:   FIT | 2         1.45        |
         |                       |
         | 3         1.7         |
         |                       |
         \ 4         1.9         /
4:   0.645501 LN (x) + 0.999640
```

Облако точек в данном случае приближено логарифмической функцией 0,645501 *ln*(*x*) + 0,99964.

Полиномиальная регрессия

Другой пример показывает использование функции **FIT(m)** для полиномиальной регрессии:

```
1:   "Реализация метода наименьших квадратов"
2:   "Полиномиальная регрессия"
         /       3      2           \
         | x   a x  + b x  + c x + d |
         |                           |
         | 1            1            |
         |                           |
```

```
3:   FIT | 2            4          |
         |                         |
         | 3          8.5          |
         |                         |
         \ 4           8          /
                  3          2
4:   - 1.08333 x  + 7.25 x  - 11.1666 x + 6
```

В этом примере облако точек приближено полиномом третьей степени, представленным в строке #4.

Линейная регрессия общего вида

Универсальность заложенных в реализацию функции **FIT(m)** алгоритмов позволяет проводить и более сложные вычисления. Так, можно осуществить линейную регрессию общего вида, когда отыскиваются коэффициенты *a*,*b*,*c*,..., обеспечивающие приближение облака исходных точек комбинацией функций:

$y(x) = a*f1(x) + b*f2(x) + c*f3(x) + ...,$

с наименьшей среднеквадратической погрешностью.

Ниже представлен пример для зависимости

```
          2      -x
y(x) = a*x  + b*e   + c/x:

1:   "Реализация метода наименьших квадратов"
2:   "Линейная регрессия общего вида"
         /           2                  c  \
         |   x   a x  + b EXP (x) + - |
         |                            x  |
         |                               |
         |   1            4.73           |
         |                               |
3:   FIT |   2            14.5           |
         |                               |
         | -0.5           -1.15          |
         |                               |
         |  -10            100           |
         |                               |
         \  -1            0.368         /
                x             2    0.980392
4:   1.31098 #e  + 1.00101 x  + -----
                                   x
```

Здесь надо обратить внимание на то, что выбранная функция регрессии должна быть линейна относительно искомых параметров, но может быть нелинейной относительно переменной x.

Регрессия для функции нескольких переменных

Функцию **FIT(m)** можно использовать и для приближения по методу наименьших квадратов функций ряда переменных. При этом соответственно увеличивается число столбцов матрицы *m*. Пусть данные надо описать зависимостью

$z(x,y) = a \cdot x + b \cdot exp(y) + c.$

Это можно сделать следующим образом:

```
1:   "Реализация метода наименьших квадратов"
2:   "Функция двух переменных"
         / x  y  a x + b EXP (y) + c \
         |                           |
         | 1  1          9.43        |
         |                           |
3:   FIT | 2  1          10.4        |
         |                           |
         | 1  2          18.8        |
         |                           |
         \ 2  2          19.8        /
                 y
4:   2.00930 #e  + 0.985000 x + 2.97562
```

Внимание! *Единственная встроенная функция для проведения регрессии* ***FIT*** *в системе Derive обладает довольно обширными возможностями – с ее помощью можно провести регрессию практически любого вида. Более того, если в этой функции задать полином со степенью n – 1, где n – число пар данных, то будет осуществлена полиномиальная аппроксимация данных. Функция FIT Derive – пример удачного создания многофункционального средства для решения задач интерполяции, аппроксимации и регрессии.*

6.8.2. Аппроксимация Паде в Derive

Реализация аппроксимации Паде в среде Derive возможна после загрузки файла внешнего расширения approx.mth. Он содержит функцию **PADE**:

PADE(y,x,x0,n,d)

и дает аппроксимацию вблизи $x = x0$ отношением двух многочленов, причем n – степень числителя и d – степень знаменателя ($n = d$ или $d - 1$). Приведенный ниже пример иллюстрирует технику проведения рациональной аппроксимации:

```
1: "Аппроксимация Паде"
          2
2: PADE (x  + 1, x, 1, 3, 3)
    2
3: x  + 1
4: PADE (SIN (x), x, 1, 2, 2)
                            10 2            11              8
     0.00306523 (3.94074 10  x -1.27412 10  x+9.35916 10 )
5: — ---------------------------
                       7  2             7                8
            3.21290 10  x — 9.08822 10  x + 3.75921 10
```

В строке 2 этого примера задана аппроксимация Паде для функции (x^2 + 1). Поскольку эта функция – сама по себе полином, то функция PADE выдала исход-

ную функцию. Зато аппроксимация функции $\sin(x)$ ведет к выдаче отношения полиномов в полном виде.

6.8.3. Интерполяция и аппроксимация в MuPAD

Заметно более скромными, чем в системе Derive, являются возможности обработки численных данных системы MuPAD. Для интерполяции по Лагранжу в ней служит функция

lagrange(xlist, ylist, var[, coeffRing]),

где **xlist** и **ylist** – списки значений x и y, var – идентификатор переменной выходного выражения, coeffRing – необязательный параметр в форме Cat::Ring или опция выходного выражения. Следующие примеры поясняют применение этой функции:

```
lagrange([1,2,3,4],[2,5,10,17],x);
                                2
                          poly(x  + 1, [x])
lagrange([1,2],[3,4],y);
                           poly(y + 2, [y])
```

Возможна также двумерная интерполяция для прямоугольной области размера $m \times n$:

lagrange([xlist, ylist],[zlist_1, zlist_2,...,zlist_m], [xvar, yvar]).

Она поясняется следующим примером:

```
lagrange([[1,2,3],[3,5]],[[6,10],[24,-3],[2,3]],[x,y],Expr);
              2                    2
poly((59/4)·x ·y + (-257/4)·x  + (-239/4)·x·y + (1029/4)·x + 47·y
— 193, [x, y])
```

Как нетрудно заметить, результатом обработки данных является полином, представленный особым типом данных – poly.

6.8.4. Фурье-аппроксимация в Derive

Малые математические системы обладают ограниченными возможностями в решении задач спектрального анализа и синтеза. К примеру, Derive вообще не обладает средствами для этого, включенными в ядро системы. Впрочем, в библиотечном файле int_apps.mth есть функция для разложения произвольной зависимости $y(t)$ в тригонометрический ряд Фурье:

- **FOURIER (y,t,t1,t2,n)** – возвращает выражение в виде суммы из n гармоник тригонометрического ряда Фурье, приближающего функцию $y(t)$ в интервале t от t=$t1$ до $t2$.

Пример на применение этой функции дан ниже:

```
1: "Разложение функции в ряд Фурье"
              2
2: FOURIER (x , x, 0, 1, 2)
```

```
   COS (4 pi x)    SIN (4 pi x)   COS (2 pi x)   SIN (2 pi x~
   ------------ - ------------ + ------------ - ------------~
3:       2             2 pi            2              pi    ~
     4 pi                              pi                    ~
   ~)     1
   ~- + --
   ~      3
   ~
```

Разумеется, базовые средства Derive позволяют реализовать различные приемы спектрального анализа и синтеза по известным алгоритмам. В качестве примера рассмотрим вычисление коэффициентов Берга – задача, которая уже решалась средствами Mathcad.

Пусть требуется вычислить относительную амплитуду n-ой гармоники для отрезка синусоиды с углом отсечки *theta* (или, проще, *t*). Для этого известны три разные формулы для коэффициентов Берга, дающих относительную амплитуду, при n, равных 0, 1 и *n*>=2. Ниже показано конструирование функции **BERG(n,t)**, вычисляющей коэффициенты Берга при любых целых *n* и углах *t* (в радианах):

```
1: "Вычисление коэффициентов Берга"
                SIN (t) - t COS (t)
2: A0 (t) := --------------------
                pi (1 - COS (t))
               t - SIN (t) COS (t)
3: A1 (t) := --------------------
               pi (1 - COS (t))
                 2
                - (SIN (n t) COS (t) - n COS (n t) SIN (t))
                 pi
4: AN (n, t) := ------------------------------------------
                                 2
                          n (n  - 1) (1 - COS (t))
5: A01 (n, t) := IF (n > 0.5, A1 (t), A0 (t))
6: BERG (n, t) := IF (n > 1.5, AN (n, t), A01 (n, t))
         /    pi \
7: BERG |0, --|
         \    6  /
8: 0.110598
         /    pi \
9: BERG |1, --|
         \    6  /
10 0.215223
          /    pi \
11: BERG |2, --|
          \    2  /
12: 0.212206
```

В этом примере в строках 2–4 заданы три исходные функции A0, A1 и AN, вычисляющие коэффициенты Берга для *n* = 0, 1 и любых n, равных или больших 2. В строке 5 определена функция A01, выбирающая с помощью функции IF функ-

цию A0 при $n = 0$ или A1 при $n = 1$. Далее в строке 6 определена искомая функция BERG(n,t), которая, также с помощью функции IF, выбирает функцию A01, если n меньше 2, и AN, если n больше или равно 2. Таким образом, основная функция **BERG(n,t)** вычисляет коэффициенты Берга при любых целых n и углах t – в строках 7–12 показаны примеры этого.

Благодаря наличию функций суммирования задачи синтеза колебаний по их гармоникам в системе Derive решаются легко, но для ограниченного числа гармоник. Отсутствие средств быстрого преобразования Фурье лишает Derive возможностей по серьезной обработке данных спектральными методами. Но, видимо, это и не нужно в системе, ориентированной на пользователей с умеренными потребностями в компьютерной математике.

6.8.5. Спектральный анализ и синтез в системе MuPAD

Возможности спектрального анализа и синтеза в системе MuPAD также ограничены. Имеются лишь две функции для осуществления классического БПФ:

- **fft(list,n)** – прямое быстрое преобразование Фурье над вектором – списком list, имеющим 2n элементов, результат – вектор того же размера;
- **ifft(list,n)** – обратное быстрое преобразование Фурье над вектором – списком list, имеющим 2n элементов, результат – вектор того же размера.

Применение их вполне очевидно – см. аналогичные функции для системы Mathcad и примеры их применения.

6.9. Экстраполяция и прогноз

6.9.1. Основы экстраполяции и прогноза

Экстраполяция в большинстве случаев означает вычисление значений аппроксимирующей данные функции по полученному интерполирующему или аппроксимирующему выражению за пределами отрезка интерполяции $[a,b]$, то есть вне $a \le x \le b$. Вычисление при $x < a$ называют левосторонней экстраполяцией, или экстраполяцией назад, а при $x > b$ – правосторонней экстраполяцией, или экстраполяцией вперед.

Прогноз, в принципе, не обязательно требует нахождения явной аппроксимирующей зависимости и может осуществляться и другими методами, например с помощью итерационных методов.

Можно подметить одну печальную закономерность: чем проще метод прогноза и чем большую теоретическую погрешность он имеет, тем чаще он применяется на практике и кажется более надежным. В повседневной жизни мы часто пользуемся *тривиальными*, или *наивными*, методами прогноза вперед – по последнему значению и по математическому ожиданию. Например, взглянув на небо и на термометр, мы наивно делаем вывод о том, что дождя не будет, а температура будет той,

которую показывает градусник снаружи окна. О том, сколь часто такие «прогнозы» нарушаются, каждый нередко ощущал «на своей шкуре».

Более просвещенные пользователи могут попытаться сделать прогноз по математическому ожиданию, которое несложно вычислить для *стационарных прогнозируемых процессов*, статистические параметры во времени постоянны. Однако погрешность таких прогнозов лишь немного меньше, чем при прогнозе по последнему значению. К тому же при этом надо знать статистические характеристики прогнозируемого процесса.

Сплайны оказались малопригодными для экстраполяции вследствие локального характера приближений. Экстраполяция при этом обычно производится по первой или последней сплайн-функции, что дало основание называть такое приближение *продолжением*. В зависимости от типа сплайна подобное продолжение может быть линейным, квадратическим или кубическим. Оно никоим образом не связано с истинным характером функции, по которой осуществляется экстраполяция, и по существу задается произвольным.

Гораздо лучшие результаты при экстраполяции и прогнозе можно ожидать при использовании регрессионных методов, когда прогноз осуществляется по уравнению регрессии. Можно отметить две важные причины этого:

- уравнение регрессии строится на основе информации о всех узловых точках исходной зависимости;
- одновременно с построением уравнения регрессии осуществляется статистическая обработка массива исходных данных (сглаживание), что позволяет в ряде случаев уменьшить случайные ошибки данных.

В теории временных рядов эквивалентом регрессии по сути является вычисление *тренда*. Поэтому можно вести речь о *трендовых методах предсказания*.

Известны также *фильтровые методы предсказания*. При них исходная зависимость представляется как входной сигнал фильтра. Предсказание является как бы «звоном» фильтра по окончании сигнала. Замечательно, что этот метод предсказания в определенной мере пригоден для предсказания зависимостей, имеющих колебательные компоненты. Для целей прогноза могут использоваться фильтры Винера и Калмана. Их реализации есть, к примеру, в системе MATLAB.

Для предсказания сложных зависимостей можно использовать простейший метод линейного предсказания. При этом методе полагается, что очередной отсчет вектора данных y_N можно рассматривать как функцию ряда предшествующих отсчетов, помноженных на весовые коэффициенты, число которых определяет порядок метода модели. Например, если он равен 3, то $y_N = f(y_{N-1}, y_{N-2}, y_{N-3})$. Соответственно следующий отсчет определяется как $y_{N+1} = f(y_N, y_{N-1}, y_{N-2})$ и т. д.

Функцию для *линейного предсказания по трем предшествующим отсчетам* можно записать в виде:

$$f(y_{N-1}, y_{N-2}, y_{N-3}) = a\,y_{N-1} + b\,y_{N-2} + c\,y_{N-3},$$

где коэффициенты a, b и c находятся из решения следующей системы уравнений:

$$y_N = y_{N-1}\,a + y_{N-2}\,b + y_{N-3}\,c;$$
$$y_{N+1} = y_N\,a + y_{N-1}\,b + y_{N-2}\,c;$$
$$y_{N+2} = y_{N+1}\,a + y_{N+2}\,b + y_{N+3}\,c.$$

Эта методика предсказания дает неплохие результаты для предсказания эмпирических детерминированных зависимостей.

Для прогноза эмпирических (как детерминированных, так и зашумленных, или, в общем, статистических) зависимостей неплохо зарекомендовали себя *авторегрессионные методы*. В них наряду с упомянутыми коэффициентами вводятся коэффициенты автокорреляции, которые уточняются тем или иным итерационным методом. Детальное описание этого процесса для функции **predict**, реализующей метод Бурга (или Берга по звучанию имени Burge), и можно найти в электронной книге Data Analysis Extension Pack в разделе Linear Prediction.

6.9.2. Линейное предсказание в системе Mathcad

Весьма интересной является включенная в систему Mathcad (начиная с шестой версии) *функция предсказания* (экстраполяции) `predict(data,k,N)`, где `data` – вектор данных, `k` – число точек с конца вектора, используемых для предсказания, и `N` – число точек предсказания. Она по ряду заданных и равномерно расположенных точек позволяет рассчитать некоторое число `N` последующих точек, то есть, по существу, осуществляет экстраполяцию произвольной (но достаточно гладкой и предсказуемой) зависимости. Эта функция в новых версиях Mathcad реализует метод Бурга, описанный выше.

На рис. 6.72 показан фрагмент документа Mathcad с примером применения функции предсказания (экстраполяции) для «чистой» (то есть не засоренной шумами и погрешностями) аналитической зависимости. Исходная функция (синусоида с экспоненциальным нарастанием) задана своими 100 точками (тонкая линия). Область предсказания определена еще на 400 точек (жирные линии для 19 и 37 последних точек). Кроме того, построена линия для положения верхушек экспоненциально нарастающего сигнала. Видно, что при $k = 19$ предсказание превосходное даже для всех 400 точек, тогда как при $k = 37$ предсказание хорошо для 300 точек, а затем наступает «разболтка» и точки предсказания дают заметно большую амплитуду предсказанных значений, чем это есть на самом деле у предсказываемой функции.

В общем случае предсказание выглядит очень неплохо для аналитических зависимостей, содержащих гладкие компоненты, синусоиды и косинусоиды, экспоненты и т. д. Порой, например при 19 точках, предсказание выглядит как сенсационное – многие исследователи в области предсказания зависимостей рады получить прогноз на несколько точек вперед. Разумеется, это верно далеко не всегда!

Большой интерес представляет применение функции предсказания в случае зашумленных зависимостей. В этом случае точность предсказания неизбежно резко ухудшается.

Есть достаточно эффективный способ повысить точность предсказания и в этом случае – для этого надо предварительно сгладить зашумленную зависимость. Пример такого подхода дан на рис. 6.73. Тут исходная зависимость в виде

ПРИМЕНЕНИЕ ФУНКЦИИ ПРЕДСКАЗАНИЯ

Задаем 100 точек исходной зависимости: $k := 0 .. 100$ $data_k := \sin\left(\frac{k}{5}\right) \cdot \exp\left(\frac{k}{200}\right)$

С помощью функции predict задаем вычисление еще 400 точек:

p1 := predict(data, 19, 400) p2 := predict(data, 37, 400)

Строим график исходной и экстраполируемой зависимости: $i := 0, 1 .. 400$

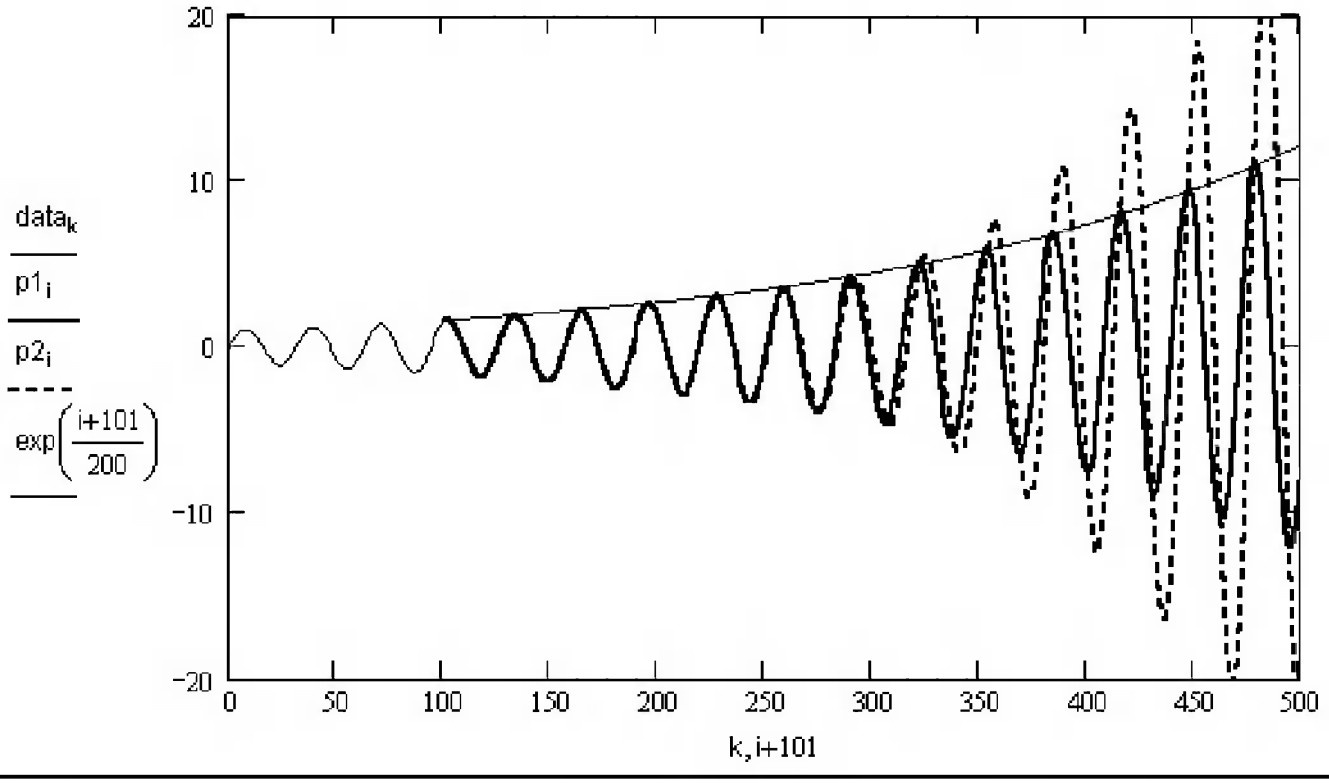


Рис. 6.72. Применение функции предсказания (экстраполяции)

экспоненциально затухающей синусоиды зашумлена шумом с равномерным распределением, создаваемым генератором случайных чисел (функция **rnd**). Для сглаживания используется функция medsmooth, реализующая сглаживание по методу скользящей медианы.

Еще один пример прогноза по зашумленной сложной зависимости представлен на рис. 6.74. Здесь шумовая компонента имеет нормальный закон распределения шумовой компоненты. Исходная функция представлена синусоидой, возведенной в кубическую степень. Интересно, что особенность формы такой функции (горизонтальные «ступеньки» при переходе через 0) явно заметна и в кривой прогноза.

6.9.3. Функции предсказания пакета Numeric Recipes

Для системы Mathcad создан пакет расширения Numeric Recipes, реализующий ряд численных методов вычислений. В этом пакете есть три функции, которые служат для решения задач экстраполяции данных и предсказания:

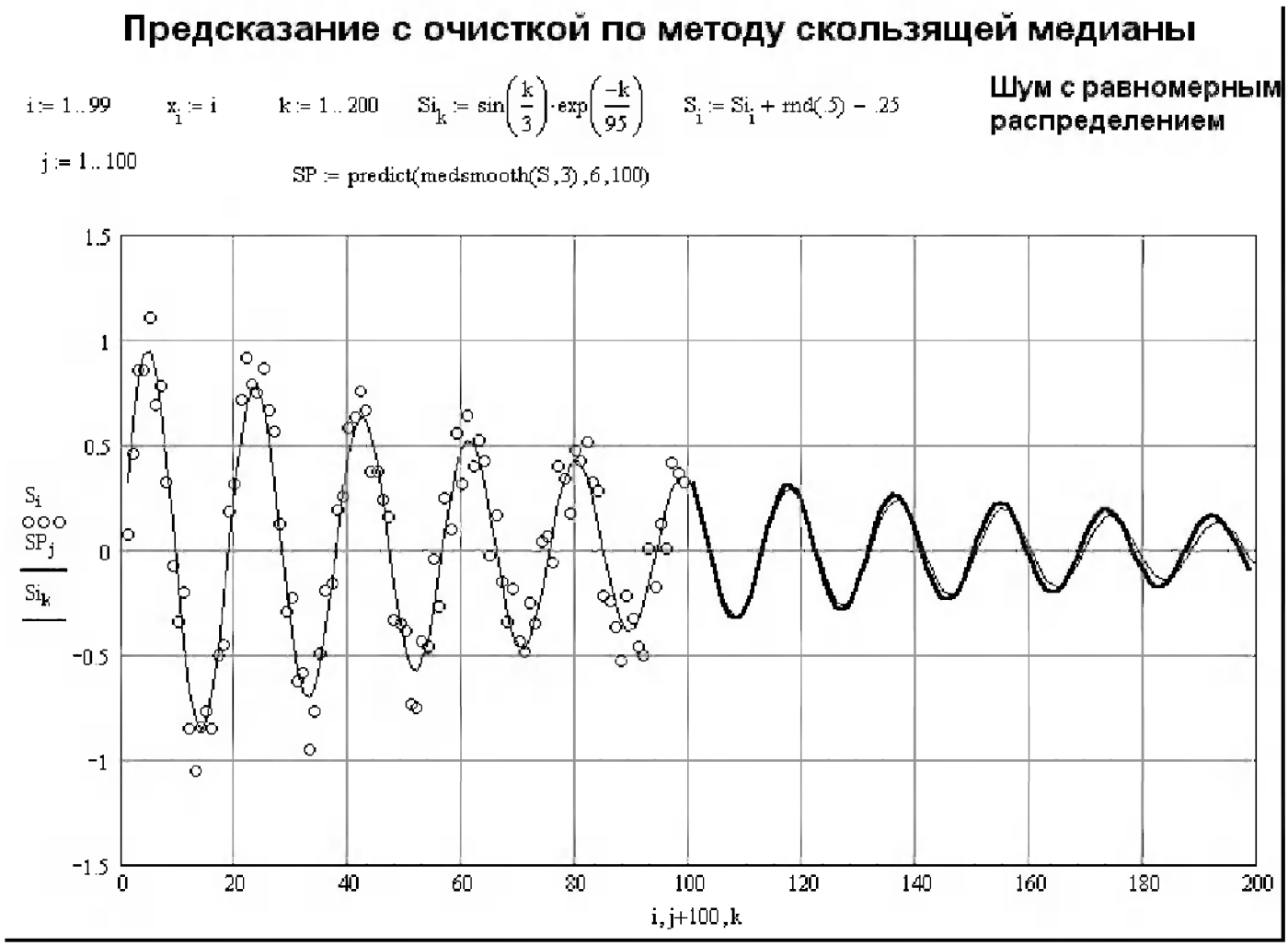


Рис. 6.73. Применение функции предсказания после сглаживания данных

- `memcof(v,m)` – возвращает вектор с `m` коэффициентами для функции однократного линейного предсказания данных, хранящихся в векторе `v`;
- `fixrts(d)` – модифицирует вектор коэффициентов функции линейного предсказания;
- `predic(v,d,n)` – возвращает вектор `n` значений предсказанных данных для исходного вектора данных `v`, используя для этого вектор коэффициентов функции предсказания `d`.

Эти функции используются для организации предсказания значений вектора `v` исходных данных, имеющего `n` элементов. Вспомогательная функция `memcof` создает вектор `d` с длиной m, который содержит коэффициенты d_j, которые используются для предсказания (n+1)-го значения по формуле

$$\sum_{j=0}^{m-1} d_j \cdot v_{n-j-1}.$$

Функция `fixrts` используется для минимизации погрешности предсказания. Она возвращает вектор уточненных коэффициентов d_j. Они получаются решением следующего уравнения для корней характеристического полинома:

$$z^m - \sum_{j=0}^{m-1} d_j \cdot z^{m-j-1} = 0,$$

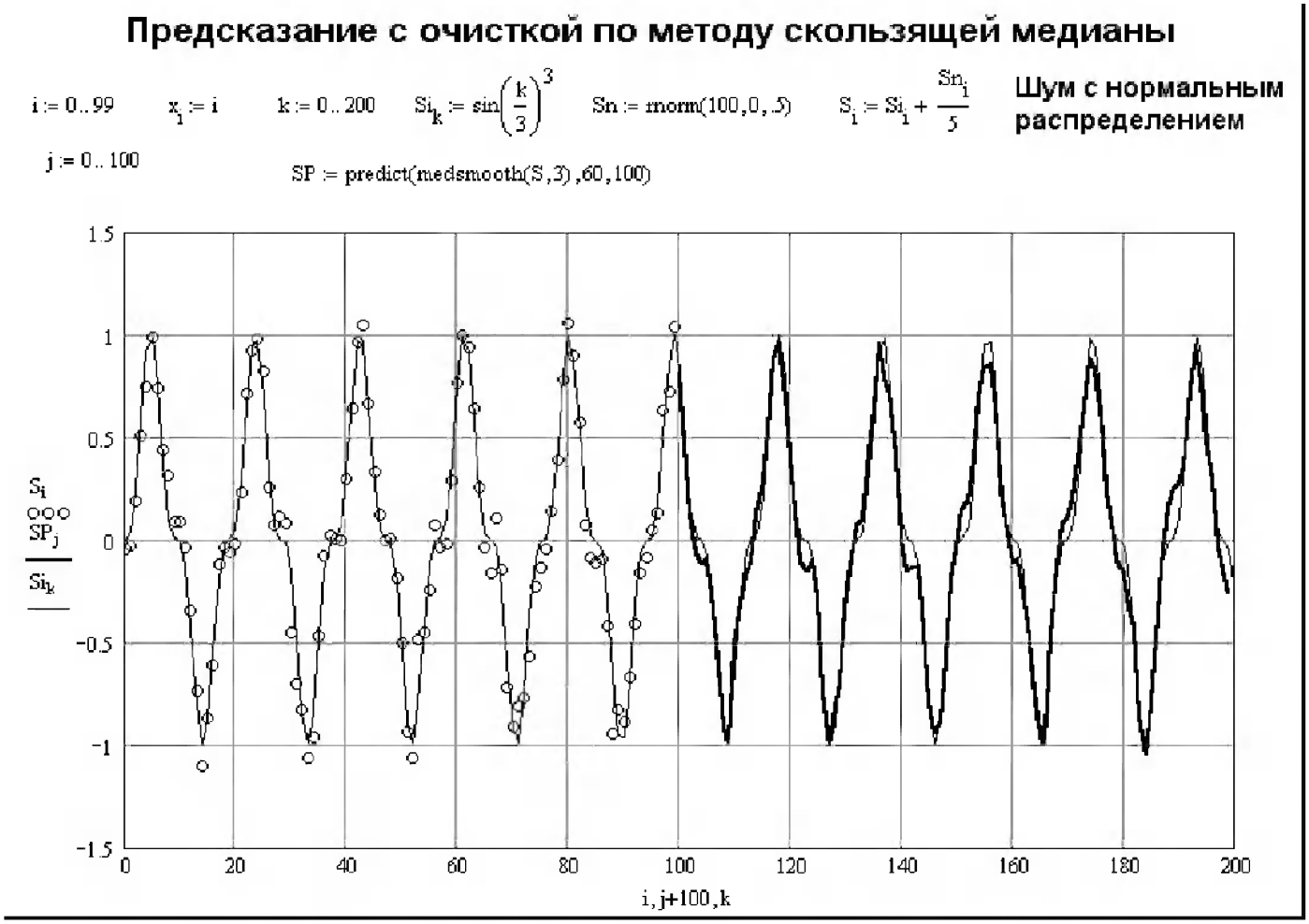


Рис. 6.74. Применение функции предсказания после сглаживания данных

лежащих в пределах единичной окружности $1/(\bar{z})$. Этот вектор также имеет длину m и может использоваться для более точного предсказания уже ряда точек.

Напоминаем, что Mathcad имеет встроенную функцию `predict`, которая справляется с описанными выше задачами предсказания. Она реализует усовершенствованный *метод Бурга*. Однако детали работы функции `predict` скрыты от пользователя, в то время как приведенные выше функции подробно описаны в электронной книге Numeric Recipes, что делает их применение более прозрачным для пользователя, решающего собственные задачи предсказания.

6.9.4. Функции предсказания пакета расширения Signal Processing

Пакет расширения Signal Processing системы Mathcad добавляет к ней еще две функции предсказания:

- `yulew(s,N)` – возвращает вектор коэффициентов для предсказания *методом Юле-Уокера* порядка `N (1<N<length(s))`;
- `burg(s,N)` – предсказание методом Бурга порядка `N (1<N<length(s))`.

Эти функции также основаны на применении линейной модели авторегрессии, при которой каждый последующий отсчет рассматривается как линейная

комбинация предшествующих отсчетов. Предсказываемая функция может быть нелинейной и иметь весьма сложный характер. Функции обеспечивают создание коэффициентов авторегрессионной модели порядка N для вектора данных действительного типа `s`, имеющего `n` значений ($n > N$). Функции возвращают вектор с длиной `N+1`, представляющий коэффициенты предсказания. Если вектор `s` содержит n значений x_i, где $i = 0..n-1$, то очередное n-ое значение можно найти по формуле

$$k := 1..N \qquad x_n = \sum_k -c_k \cdot x_{n-k}.$$

Метод Юле-Уокера основан на вычислении параметров (коэффициентов) линейной модели авторегрессии, обеспечивающей минимум среднеквадратической погрешности прямого предсказания. Сигнал `s` рассматривается как выходной сигнал модели авторегрессии, на вход которой поступают данные с белым шумом. С позиций теории фильтров коэффициенты c_k – это коэффициенты передаточной характеристики всеполюсного фильтра N-го порядка с бесконечной импульсной характеристикой (IIR). Предсказание представляет собой реакцию такого фильтра, после того как на него был подан сигнал, точки которого есть значения элементов вектора `v` (словами практиков, это «звон» фильтра).

На рис. 6.75 представлен фрагмент документа Mathcad с примером предсказания для зашумленной экспоненциально спадающей синусоиды.

Нетрудно заметить, что точки исходного сигнала неплохо укладываются на кривую предсказания, полученную с помощью метода Юле-Уокера. Но до идеального предсказания, разумеется, далеко, и оно существенно ухудшается при наличии шумовой компоненты сигнала или в случае, когда последующие значения функции нелинейно зависят от предшествующих значений.

Метод Бурга также основан на вычислении параметров линейной модели авторегрессии, но обеспечивающей минимум среднеквадратической погрешности как для прямого, так и для обратного предсказаний. Сигнал `s` рассматривается как выходной сигнал модели авторегресии, на вход которой поступает белый шум.

Применение метода Бурга иллюстрирует фрагмент документа Mathcad, представленный на рис. 6.76. Здесь в качестве тестового использован случайный сигнал. Не стоит забывать, что функция лишь возвращает коэффициенты авторегрессионной модели сигнала. В данном примере для получения отклика фильтра использована функция `response`.

Точность предсказаний существенно зависит от порядка метода, который задается параметром `N`. Функция `burg` с формулой предсказания фактически дублирует функцию `predict` системы Mathcad. Дополнительные данные о прогнозе, в том числе и назад, и с использованием непараметрического подхода, можно найти в примерах системы Mathcad 12.

Предсказание методом Юле-Уокера

$n := 0..100$ $\quad X_n := \sin\left(\frac{2\cdot\pi\cdot n}{12}\right)\cdot\exp\left(\frac{-n}{50}\right) + \mathrm{rnd}(0.2) - 0.1$

$M := 50$ $\quad m := 0..M-1$ $\quad y_m := X_m$

$N := 5$ $\quad c := \mathrm{yulew}(y, N)$ $\quad i := M..100$ $\quad c = \begin{pmatrix} 1 \\ -1.13 \\ 0.45 \\ -0.102 \\ 0.219 \\ 0.12 \end{pmatrix}$

$k := 1..N$ $\quad y_0 := 0$ $\quad y_i := -\left[\sum_k c_k\cdot(i-k\geq 0)\cdot y_{i-k}\right]$

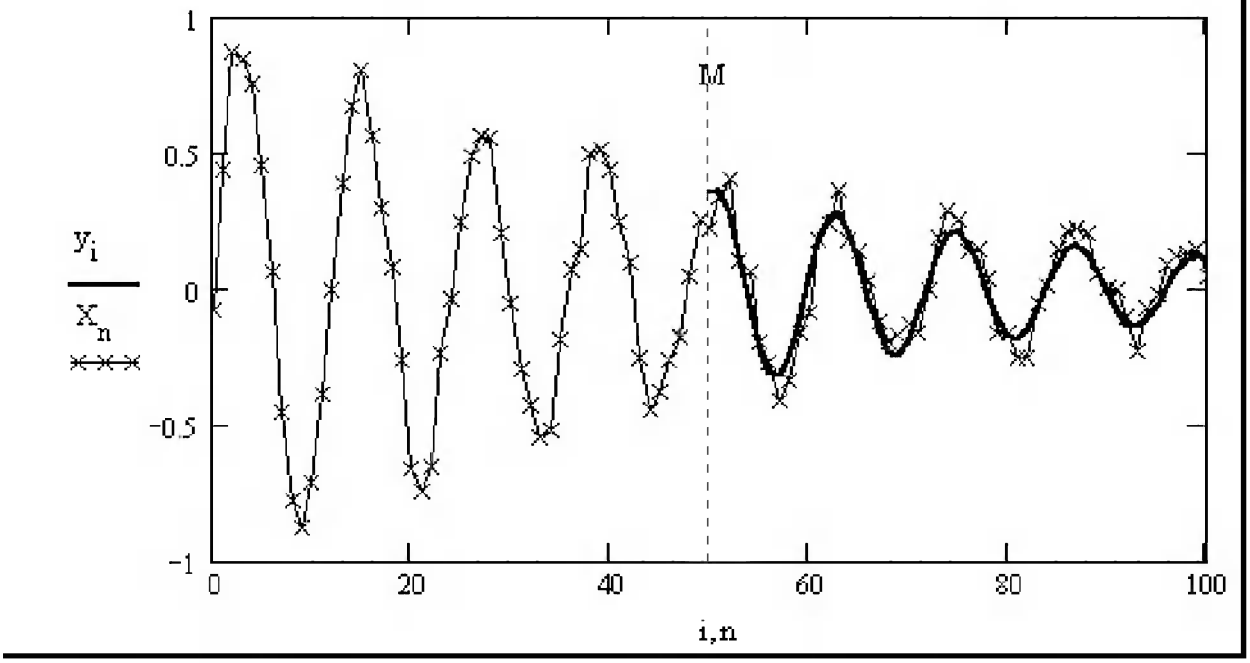


Рис. 6.75. Пример предсказания методом Юле-Уокера

6.9.5. Предсказание по закону Мура на основе нелинейной регрессии

На заре разработки микропроцессоров один из основателей корпорации Intel Гордон Мур высказал предположение, что количество транзисторов на кристалле микропроцессора будет удваиваться за некоторый промежуток времени [169]. В апреле 2005 года исполнилось 40 лет с момента этого уникального предсказания, ставшего знаменем могучей корпорации Intel, ныне тратящей на научные исследования около 5 млрд долл. в год [153].

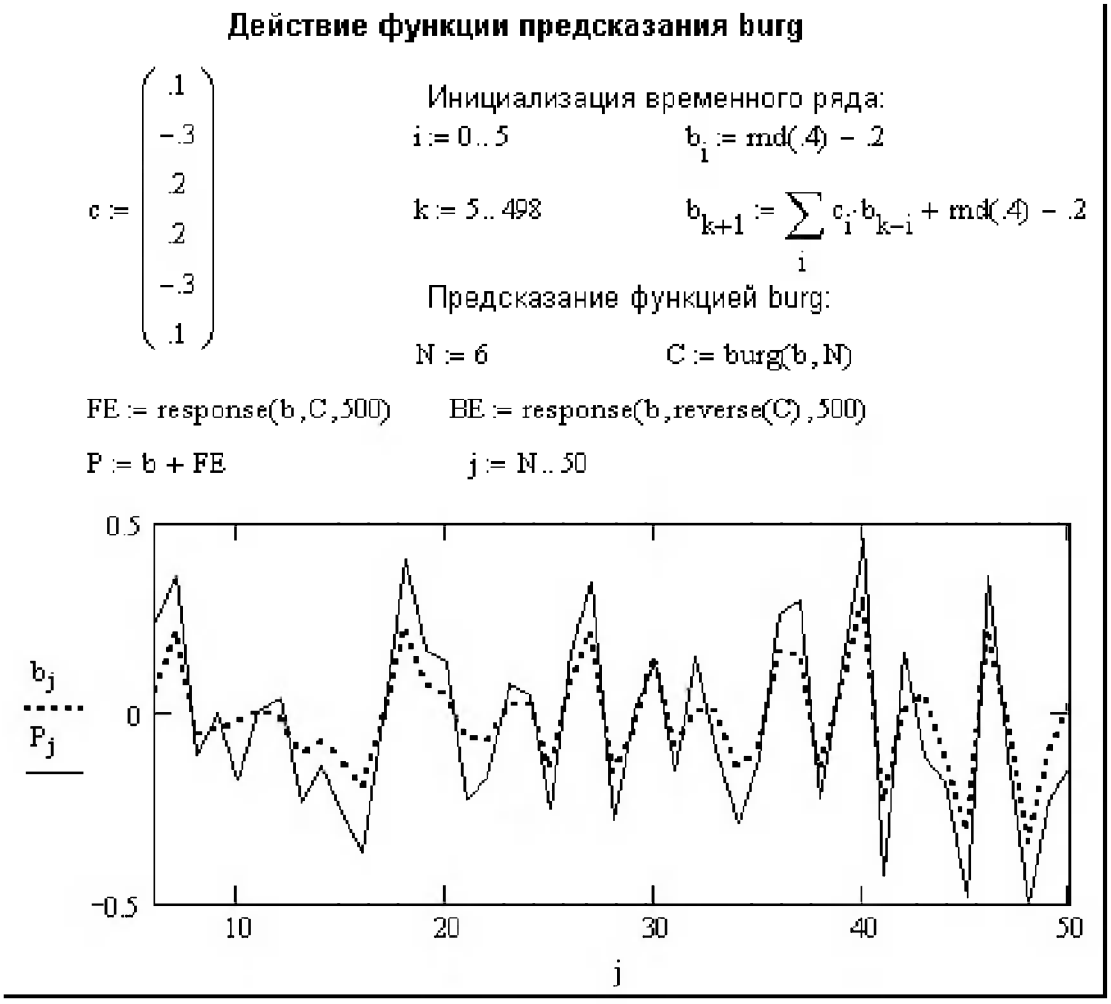


Рис. 6.76. Пример предсказания методом Бурга

Зависимость, описывающая «*закон Мура*», относится к классу степенных (экспоненциальных) зависимостей и может быть представлена одной из следующих формул:

$$N(y) = N \cdot 2^{\frac{y}{yy}}$$

– для нарастающей степенной зависимости;

$$N(y) = N \cdot 2^{\frac{-y}{yy}} + a$$

– для спадающей степенной зависимости.

В случае спадающей временной зависимости обычно приходится использовать дополнительный параметр a – значение $N(y)$ при $y \to \infty$.

Итак, зависимость, лежащая в основе «закона Мура», является достаточно простой двухпараметрической (или трехпараметрической) монотонной функцией одной переменной y (время в годах, относительно первого года, который мы считаем нулевым). Рассмотрим метод наименьших квадратов для нахождения параметров

N и yy приближающей функции в общем виде на примере приближающей функции с двумя параметрами $F(y,N,yy)$. Тогда для n исходных точек имеем:

$$F(y_i,N,yy)=\bar{f}_i,\ i=1,2,...,n.$$

Сумма квадратов разностей соответствующих значений функций f и F будет иметь вид:

$$\sum_{i=1}^{n}[f_i-F(y_i,N,yy)]^2=\phi(N,yy).$$

Эта сумма является функцией $\phi(N,yy)$ двух переменных – начального значения искомой зависимости N и времени удвоения yy. Задача сводится к отысканию параметров N и yy, обеспечивающих ее минимальное значение. Для этого используем необходимое условие экстремума:

$$\frac{\partial\phi}{\partial N}=0,\quad \frac{\partial\phi}{\partial yy}=0$$

или

$$\sum_{i=1}^{n}[f_i-F(y_i,N,yy)]\cdot F'_N(y_i,N,yy)=0,$$

$$\sum_{i=1}^{n}[f_i-F(y_i,N,yy)]\cdot F'_{yy}(y_i,N,yy)=0.$$

Входящие в эти выражения производные можно вычислить аналитически, что будет сделано чуть позже с помощью СКМ.

Решив представленную систему из двух уравнений с двумя неизвестными N и yy, получим конкретный вид искомой функции $F(y,N,yy)$. Заметим, что изменение количества искомых параметров не приведет к изменению сущности метода, а отразится только на числе уравнений для нелинейной регрессии.

Для найденной эмпирической формулы в соответствии с исходными табличными данными можно найти сумму квадратов отклонений

$$\sigma=\sum_{i=1}^{n}\varepsilon_i^{\,2},$$

которая для заданного вида приближающей функции и ее найденных параметров (параметры N и yy) будет наименьшей. Мерой близости исходной и полученной зависимостей может служить *коэффициент корреляции*.

Описанная методика определения параметров N и yy является вариантом нелинейной регрессии общего вида. Она позволяет отыскивать нужные параметры при произвольном (не обязательно равноотстоящем) расположении узловых точек исходных данных. К недостаткам метода относится необходимость решения системы нелинейных уравнений итерационными методами, возникновение при

расчетах очень больших чисел, что может привести к переполнению разрядной сетки применяемой СКМ.

6.9.6. Закон Мура для числа транзисторов на кристаллах микропроцессоров

Попытаемся выяснить, насколько справедлив «закон Мура» для микропроцессоров – основных изделий корпорации Intel [153]. Ниже представлена таблица, дающая представление о динамике роста числа транзисторов (в тысячах штук) на кристалле микропроцессоров корпорации Intel с момента появления в 1971 году первого микропроцессора 4004. Таблица охватывает тридцатилетний период разработки процессоров корпорацией Intel и представляет лишь небольшую выборку из числа созданных Intel изделий.

Тип микропроцессора	Тысяч транзисторов	Год разработки	Параметр y
4004	N0 = 2,3	1971	0
8008	3,5	1972	1
8080	6	1974	3
8088	29	1979	8
286	134	1982	11
386	275	1986	15
486	1200	1989	18
Pentium	3500	1993	22
Pentium PRO	5500	1995	24
Pentium II	7500	1997	26
Pentium III	9500	1999	28
Pentium IV	42 000	2000	29
Pentium IV M	75 000	2001	30

Данные приведенной таблицы впечатляют сами по себе. Но насколько они соответствуют представленной формуле? И возможен ли по ним прогноз? Попробуем ответить на эти вопросы.

На рис. 6.77 представлен документ системы Mathcad 12 с математической иллюстрацией «закона Мура». В левом верхнем углу документа задана формула «закона Мура» и в аналитическом виде вычислены ее частные производные по искомым параметрам N и yy. Затем заданы векторы $F1$ (функции и ее производных, нужных для реализации алгоритма нелинейной регрессии), числа лет, прошедших с 1971 года, Vy и числа тысяч транзисторов на кристалле процессора VN. С помощью функции `genfit`, использующей эти данные, вычислены параметры $N0$ и yy. Начальные условия, сильно влияющие на точность регрессии, задаются вектором VS – они содержат стартовые значения параметров N (в тысячах штук) и значения yy.

Левый график задает число транзисторов как функцию от параметра yy (время удвоения) в линейном масштабе. При этом расчетный график имеет типично экс-

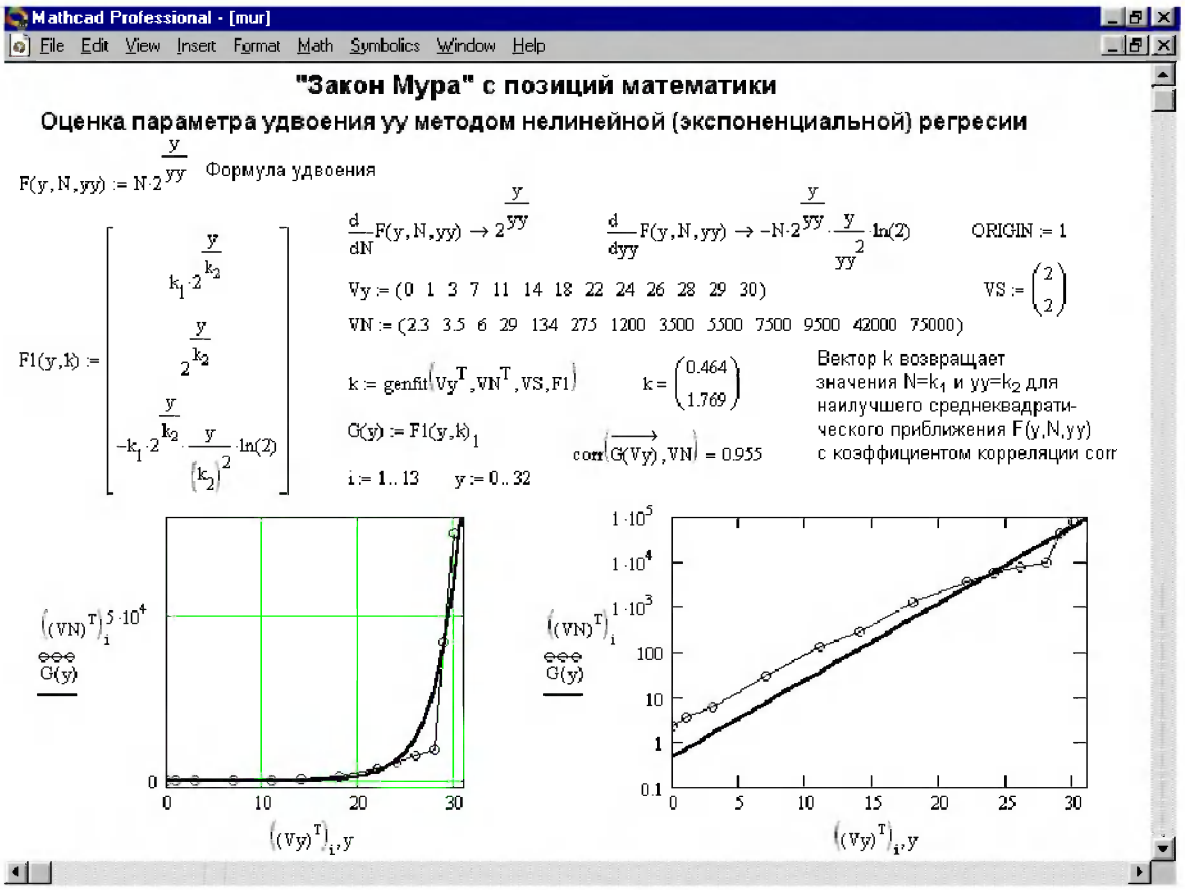


Рис. 6.77. Нелинейная регрессия для зависимости, описывающей увеличение числа транзисторов на кристалле процессора корпорации Intel от времени

поненциальный вид. Он показывает особенно резкое нарастание числа транзисторов в микропроцесорах начиная с 90-х годов прошлого века. До этого времени график мало представителен.

Экспоненциальный рост числа транзисторов (и иных параметров микросхем) от времени очень чувствителен к параметру *yy*. Его приближенные значения от 1 до 2 лет ведут к чудовищным просчетам! Да и сама процедура нелинейной регрессии для такой зависимости оказывается очень чувствительной к ошибкам машинных расчетов. В этом нетрудно убедиться, слегка меняя исходные данные или начальные приближения для *y* и *N*. Тем не менее близкое к единице значение коэффициента корреляции corr=0.955 говорит о том, что заданная зависимость при полученных значениях *N*0 и *yy* не так уж и плохо соответствует исходным парам данных.

Характерной особенностью нелинейной регрессии оказываются большая погрешность в начальной области расчетов и резкое отличие расчетного параметра *N* в первые годы от реальных значений числа транзисторов на кристаллах первых микропроцессоров. Так, выброс вниз даже небольшого числа исходных точек в правой области графика (где число транзисторов очень велико) ведет к тому, что большинство исходных точек в левой части графика (где число транзисторов мало) располагается сверху расчетной зависимости – что прекрасно видно из правого графика рис. 6.77.

Устранить этот недостаток можно добрым «дедовским» методом – взяв за левую точку прямой графика в логарифмическом масштабе точку первого отсчета и подобрав крутизну прямой на глаз по наилучшему положению в облаке исходных точек (отсчетов). Считая за нулевой 1971 год и за N0 = 2,3 тысячи число транзисторов первого микропроцессора 4004, попробуем методом проб подобрать с помощью Mathcad логарифмическую прямую, на которую хорошо укладываются данные за первые годы развития процессоров и которая исходит из точки (0, N0). Результат представлен на рис. 6.78 сплошной тонкой линией. Названия процессоров на графике проставлены с помощью графического редактора.

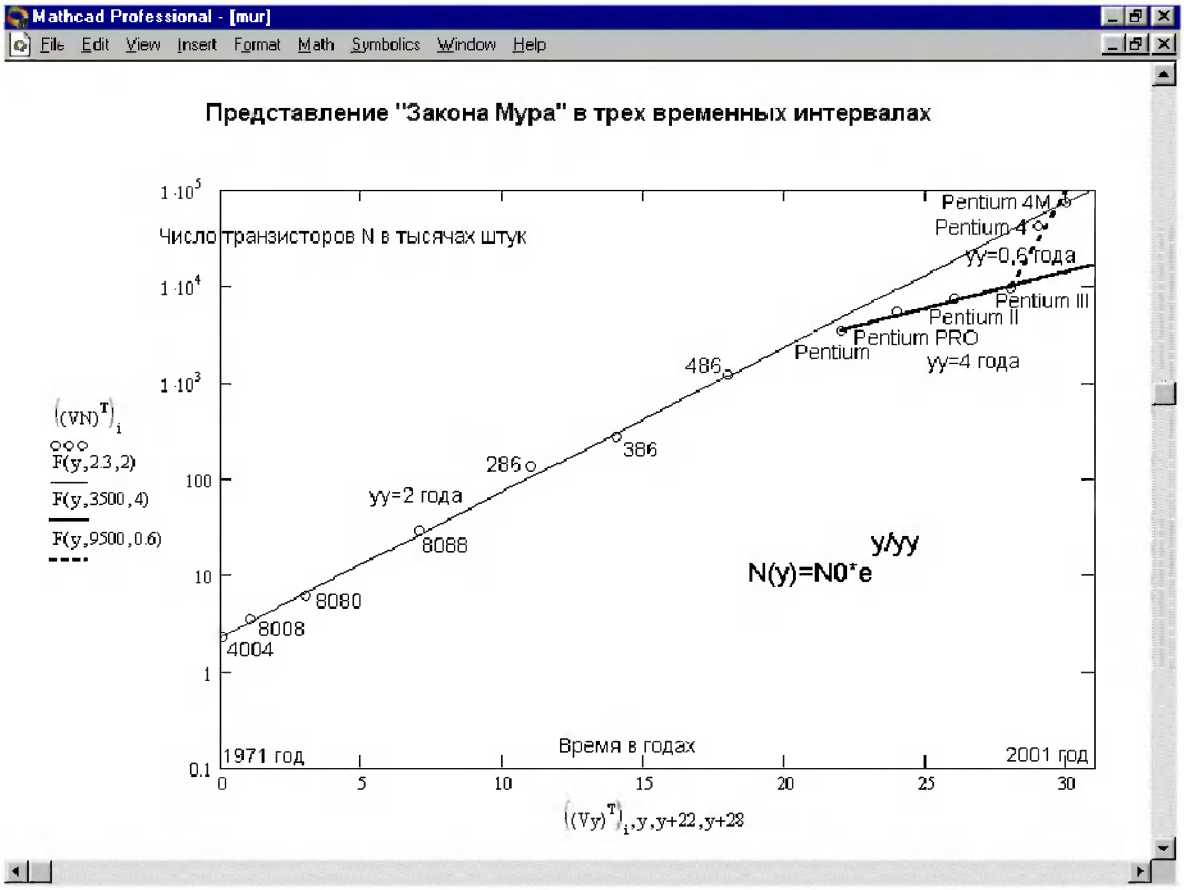


Рис. 6.78. Результат представления «закона Мура» отрезками прямых в логарифмическом масштабе

Результат оказывается просто поразительным. Оказывается, что целых 22 года число транзисторов и впрямь увеличивалось вдвое за каждые yy = 2 года. При этом исходные точки укладываются почти точно на представляющую их приближенную зависимость. Таким образом, проведенная специалистами Intel коррекция «закона Мура» была вполне обоснованной и довольно точной. Однако до года или даже до полутора лет время удвоения *yy* за этот период никогда не падало. Следовательно, строго математически, начальные прогнозы Мура были очень неточны, что нисколько не умаляет их рекламного и эмоционального значения.

В целом, усредняя параметр *yy* на весь тридцатилетний период развития микропроцессоров, можно признать, что данные нелинейной регрессии достаточно

корректны. Последуем за специалистами Intel и попытаемся дать прогноз роста числа транзисторов на кристалле микросхем на основании нашего приближения «закона Мура» выражением (1). Это показано на рис. 6.79 для первого десятилетия (рисунок слева) и для следующего десятилетия (рисунок справа).

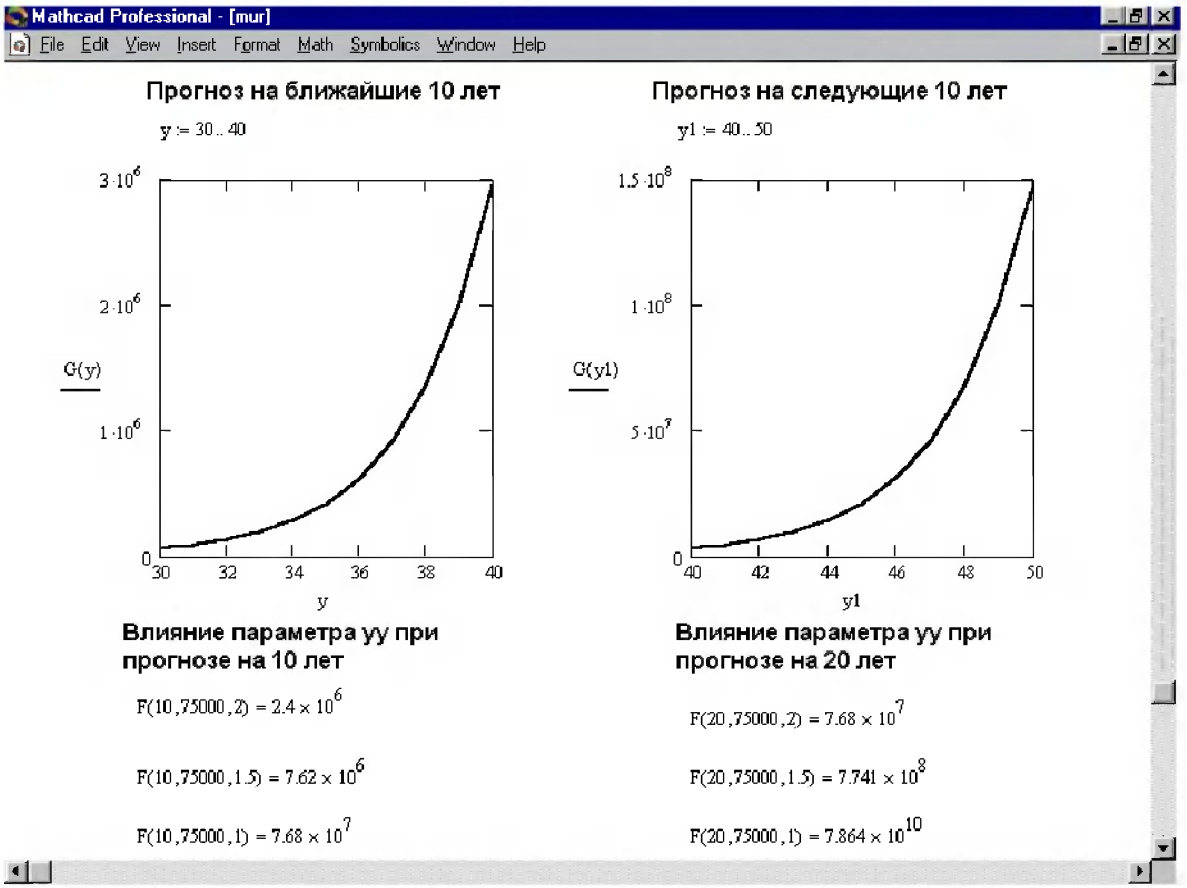


Рис. 6.79. Прогноз роста числа транзисторов на кристалле микросхемы на первые и на последующие десять лет по данным нелинейной регрессии

Прогноз на первые десять лет выглядит более или менее реалистичным. Можно ожидать появления микропроцессоров, на кристалле которых будет до 3 млрд транзисторов. Но вот прогноз на следующее десятилетие выглядит скорее фантастическим, чем реальным, – судя по нему, к 2020 году число транзисторов на кристалле достигнет примерно 140 млрд!

Весной 2003 года Гордон Мур сам признал, что необходима и дальнейшая коррекция его «закона». В частности, такая коррекция необходима в связи с развитием новых технологий в производстве микросхем – а именно, электронно-ионной технологии с применением рентгеновских излучений с малой длиной волны. Надо не забывать, что математически «закон Мура» законом не является – скорее это попытка корпорации Intel всей своей мощью выдержать определенный темп развития микропроцессоров, описанный простой и наглядной зависимостью, предложенной Муром, что ничуть не приуменьшает социально-экономическую значимость этого «закона».

Глава 7

Статистические вычисления

Мир СКМ сейчас насыщен *статистическими системами*, например такими, как Statistica, STADIA или StatGraphics [119–121, 139–142]. Они прекрасно подходят для решения численных задач статистической обработки обширных массивов данных и хорошо приспособлены для многовариантных расчетов. В связи с этим в этой главе мы рассмотрим только средства статистических вычислений, имеющиеся в системах, относящихся к компьютерной алгебре.

7.1. Некоторые положения статистики

7.1.1. Эксперименты, события и другие понятия статистики

Создание некоторой системы условий называют *испытанием*, или *экспериментом*. Если до осуществления эксперимента его результаты нельзя точно предсказать, то эксперимент называют *вероятностным*, *случайным* или *стохастическим*. В ходе эксперимента происходят *факты* или *события A*, наступление которых можно наблюдать. Изучением законов, которым подчиняются случайные события, занимается *теория вероятности*.

События могут быть достоверными, невозможными и случайными. В последнем случае события могут наступать или не наступать. Пара событий может быть несовместной, если наступление одного события исключает другое (например, падение монеты на ту или иную сторону). События могут быть взаимно противоположными, если они несовместны и одно из них наступает. Возможны объединения (суммы) и пересечения событий.

Случайные события характеризуются *вероятностью события* $P(A)$, которую оценивают числом от 0 (событие не наступает) до 1 (при 1 событие непременно наступит). Если число элементарных исходов некоторого эксперимента равно n, а событию A благоприятствует m исходов, то *классическая вероятность* события A будет $P(A) = m/n$. Пусть на тарелке лежит 10 белых и $m = 5$ красных черешен. Значит, $n = 10 + 5 = 15$. Какова вероятность, что мы возьмем наугад красную черешню? Она равна $P(A) = m/n = 5/15 = 1/3$.

Классическое определение вероятности неприемлемо, если события не являются равновозможными. Например, игральный кубик со скошенными некоторыми гранями не имеет равновозможных вариантов выпадения. В таких случаях пользуются *статистической вероятностью* событий. Пусть при n экспериментов событие A наступило m раз. Это число называют абсолютной частотой события A, а $P^*(A) = m/n$ называют относительной частотой события. Вероятностью события A называют число $P(A)$, около которого группируются значения относительной частоты события A при большом числе экспериментов (испытаний).

Математическая статистика – это наука о методах систематизации и использования статистических данных для получения научных и практических выводов. Она решает следующие типовые задачи:

- получение данных о различных перестановках и сочетаниях объектов (задачи комбинаторики);
- получение усредняющих данных об объектах, например успеваемости студентов или их росте и весе;
- сравнение эффективности разных технологий и процессов;
- решение задач приближения и аппроксимации экспериментальных зависимостей;
- сглаживание данных с большими случайными ошибками (шумами);
- прогноз некоторых событий (например, курса валют и др.).

Есть и множество других, в том числе сложных, задач статистики, но в рамках данной книги мы ограничимся только перечисленными выше задачами. При этом мы будем рассматривать некоторую совокупность данных, называемую *генеральной совокупностью*, а также выборки данных из нее, именуемые *выборочными совокупностями*. Как правило, данные мы будем представлять в виде *вариационного ряда*, при котором данные используются в порядке возрастания результатов наблюдения.

7.1.2. Решение задач комбинаторики

К числу элементарных задач статистики относятся задачи *комбинаторики*. Рассмотрим основные из них.

Перестановкой n объектов называют их расположение в определенном порядке. Число перестановок задается как значение факториала $P_n = n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1 = n!$, для вычисления которого в Mathcad есть оператор !. Например, число перестановок для 10 предметов есть 10! = 3628800. Значения факториала быстро растут с ростом n.

Размещением некоторой части m из множества n элементов называется их расположение в определенном порядке. Число размещений обозначают как

$$A_n^m = n \cdot (n-1) \cdot \ldots \cdot (n-m) \cdot (n-m+1) = \frac{n!}{(n-m)!}.$$

Пример: сколько вариантов набора двух разных цифр возможно на диске телефона, имеющем 10 цифр? Ответ: $A_2^{10} = 10 \cdot 9 = 90$.

Сочетанием m элементов из множества из n элементов называют любую часть элементов (подможества) этого множества

$$C_n^m = \frac{A_n^m}{m!}.$$

Пример: сколько способов выбора делегации из $m = 3$ возможно из группы, насчитывающей $n = 10$ человек. Имеем

$$C_{10}^3 = \frac{10 \cdot 9 \cdot 8}{1 \cdot 2 \cdot 3} = 120.$$

Многие функции статистики требуют вычисления факториалов больших чисел, что уже отмечалось в главах 1 и 3.

7.1.3. Дискретные и непрерывные случайные величины

В теории вероятностей *случайной величиной* называют переменную величину, которая в зависимости от исхода испытания случайно принимает какое-либо одно значение из множества возможных значений. Случайные величины могут быть *дискретными* (отличающимися на определенную и постоянную величину) и *непрерывными*. Примерами дискретных случайных величин является последняя цифра номера телефона или число студентов в группе. Примерами непрерывных величин является вес людей или температура воздуха. Эти параметры имеют непрерывно изменяющиеся значения, порой отличающиеся очень незначительно.

Дискретные случайные величины задаются своими значениями и их вероятностями, например в виде следующей таблицы:

X	x_1	x_2	x_3	...	x_{n-1}	x_n
P	p_1	p_2	p_3	...	p_{n-1}	p_n

В сумме вероятности дискретной случайной величины равны 1. Математическим ожиданием дискретной случайной величины называют значение

$$M(X) = x_1 \cdot p_1 + x_2 \cdot p_2 + ... + x_n \cdot p_n.$$

Если дискретных случайных величин достаточно много, то математическое ожидание их приближенно равно среднему значению

$$M(X) \approx x = (x_1 + x_2 + ... x_n)/n.$$

Мерой «рассеивания» дискретных случайных величин могло бы служить отклонение случайных величин от их математического ожидания. Но, имея разные знаки, отклонения часто взаимно компенсируются. Поэтому мерой «рассеивания» принято считать *квадрат отклонений* случайной величины X (вы, вероятно, подметили, что большими буквами обозначаются случайные величинами, а малыми – из значения).

Дисперсией $D(X)$ дискретной случайной величины X называется математическое ожидание квадрата отклонения случайной величины X от ее математического ожидания, то есть $D(X) = M[X - M(X)]^2$. А средним квадратичным отклонением называют корень квадратный из дисперсии

$$\sigma(X) = \sqrt{D(X)}.$$

Непрерывные случайные величины могут принимать любые значения на том или ином отрезке. Поэтому их закон распределения нельзя описать в виде таблицы. *Функцией распределения* (или интегральной функцией распределения) непрерывных случайных величин называется функция $F(x)$, равная вероятности того, что случайная величина X приняла значение, меньшее x: $P(X) = \{(X < x)$. Функция $F(x)$ всегда монотонно растущая, и ее значения лежат в пределах от 0 до 1. Если значения x лежат в пределах от $-\infty$ до $+\infty$, то $F(-\infty) = 0$ и $F(+\infty) = 1$.

Плотностью вероятности (или дифференциальной функцией распределения) случайной величины называют функцию $f(x) = F'(x)$. Вероятность попадания непрерывной случайной величины в интервал $(a;b)$ равна интегралу от $f(x)$ в пределах от a до b:

$$P(a < x < b) = \int_a^b f(x)dx = F(b) - F(a).$$

Математическим ожиданием $M(X)$ непрерывной случайной величины X с плотностью вероятности $f(x)$ называют величину несобственного интеграла

$$M(X) = \int_{-\infty}^{\infty} x \cdot f^* x)dx,$$

а *дисперсией* случайной величины с математическим ожиданием a именуют величину:

$$D(X) = \int_{-\infty}^{\infty} (x-a)^2 f(x)dx.$$

Как и в случае дискретных величин, дисперсия является мерой отклонения случайных величин от их математического ожидания.

7.1.4. Законы распределения и статистические функции

В статистической обработке данных генеральной или выборочной совокупности используются различные законы распределения непрерывных случайных величин. Один из самых простых законов – равномерный. Он соответствует постоянному значению $f(x) = C$ на отрезке $[a,b]$ с единичной площадью зависимости $f(x)$. Отсюда следует, что $C = 1/(b - a)$.

Одним из самых распространенных является *нормальный закон распределения*:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(x-a)}{2\sigma^2}},$$

где a – математическое ожидание и σ – среднеквадратичное отклонение. Интегральная функция распределения для него:

$$F(x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{(x-a)^2}{2\sigma^2}}.$$

Значимость нормального закона распределения вероятностей для практики вычислений не стоит переоценивать – очень часто нет достаточных оснований считать, что этот закон соблюдается.

7.2. Статистические вычисления в системе Maple 10/11

Системы класса Maple не относятся к числу «гигантов статистики». Тем не менее проведение статистических расчетов в Maple возможно и в ряде случаев весьма целесообразно – например, когда они являются частью исследовательского проекта, выполняемого на основе применения системы Maple.

7.2.1. Пакет статистических вычислений Statistics системы Maple

Основная часть статистических расчетов в системе Maple 10/11 сосредоточена в пакете `Statistics` (в старых версиях был пакет stats). Для загрузки пакета используется команда:

```
with(Statistics):
```

Заменив знак «:» на «;», можно получить довольно большой список всех функций этого пакета. Для детального знакомства с функциями пакета в справке можно активизировать гиперссылку Statistics[Commands].

Поскольку Maple – прежде всего система компьютерной алгебры, то большинство вычислительных функций пакета Statistics задано в символьном (аналитическом) виде. Это удобно при выполнении расчетов в области теоретической статистики. Для получения результатов в численном виде надо использовать соответствующие функции преобразования.

В состав средств статистических вычислений Maple 10/11 включен также пакет расширения ProcessControl Package. Он реализует численные методы статистической обработки информации, которые прямого отношения к компьютерной алгебре не имеют. В связи с этим данный пакет не рассматривается.

7.2.2. Генерация случайных чисел с заданным распределением

Часто, например при реализации *методов Монте-Карло*, возникает необходимость в *генерации чисел с заданным законом распределения вероятности*. Для этого в пакете Statistics есть функции практически для всех других известных распределений. Их почти четыре десятка:

Bernoulli	**Beta**	**Binomial**
Cauchy	**ChiSquare**	**DiscreteUniform**
Distribution	**EmpiricalDistribution**	**Erlang**
Error	**Exponential**	**FRatio**
Gamma	**Geometric**	**Gumbel**
Hypergeometric	**InverseGaussian**	**Laplace**

Logistic	**LogNormal**	**Maxwell**
Moyal	**NegativeBinomial**	**NonCentralBeta**
NonCentralChiSquare	**NonCentralFRatio**	**NonCentralStudentT**
Normal	**Pareto**	**Poisson**
Power	**ProbabilityTable**	**Rayleigh**
StudentT	**Triangular**	**Uniform**
VonMises	**Weibull**	

Примеры на рис. 7.1 демонстрируют технику получения случайных чисел с заданным законом распределения – в данном случае нормальным. В первом примере осуществляется вывод формулы нормального распределения вероятности, а во втором примере – вывод 4 случайных чисел с нормальным распределением. В обоих примерах используется функция **RandomVariable**, создающая объект статистических вычислений.

```
with(Statistics) : X := RandomVariable(Normal(mu, sigma)) : PDF(X, u);
```

$$\frac{\frac{1}{2}\,\frac{\sqrt{2}\;e^{-\frac{1}{2}\,\frac{(u-\mu)^2}{\sigma^2}}}{\sqrt{\pi}\;\sigma}}{2}$$

```
R := RandomVariable(Normal(0, 0.5)) : Sample(R, 4);
```

```
[ 0.751510855909014897  -.527023971396303526  -0.0233459456894332049  -.382300522070510962 ]
```

Рис. 7.1. Примеры вывода формулы нормального распределения и генерации ряда случайных чисел с таким распределением

7.2.3. Графика статистического пакета Statistics системы Maple

Статистический пакет `Statistics` имеет свою библиотечку для построения статистических графиков. Представлены 20 функций для построения таких графиков:

AreaChart	**BarChart**	**BoxPlot**
BubblePlot	**ColumnGraph**	**CumulativeSumChart**
DensityPlot	**ErrorPlot**	**FrequencyPlot**
Histogram	**KernelDensityPlot**	**LineChart**
NormalPlot	**PieChart**	**PointPlot**
ProbabilityPlot	**ProfileLikelihood**	**ProfileLogLikelihood**
QuantilePlot	**ScatterPlot**	**SurfacePlot**

Назначение каждой функции ясно из ее названия. Довольно часто для визуализации статистических вычислений используется построение гистограмм и графиков тех или иных распределений. На рис. 7.2 показан пример, в котором задано

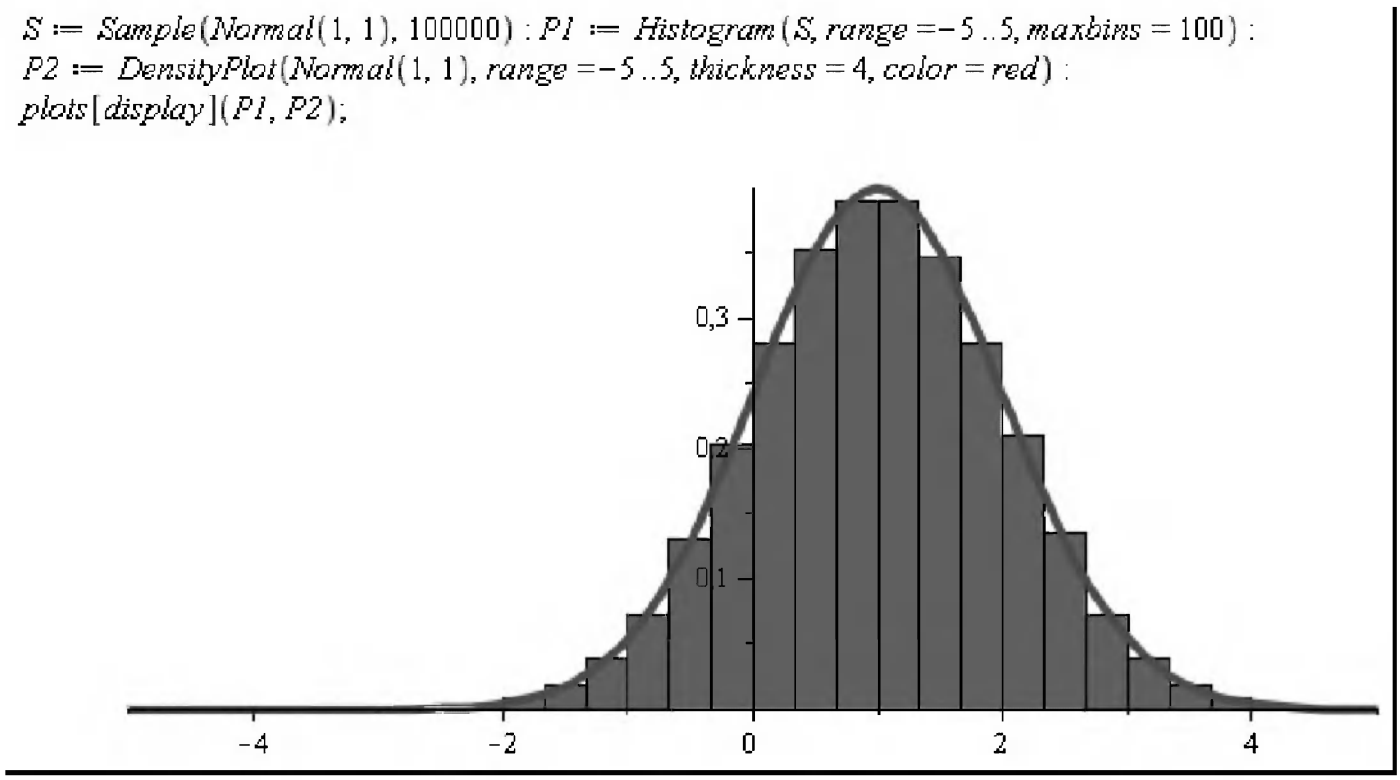


Рис. 7.2. Пример построения гистограммы и кривой нормального распределения

построение 100 000 случайных чисел с нормальным распределением и построение гистограмм их распределения и теоретической кривой нормального распределения. Нетрудно заметить, что данные моделирования случайных чисел и теоретического описания их распределения хорошо соответствуют друг другу.

7.2.4. Ассистент интерактивного статистического анализа данных

В Maple 10/11 введен ассистент Data Analysis интерактивного статистического анализа данных, представленных одномерным массивом. Ниже представлен пример задания массива из 10 чисел и обращения к ассистенту:

```
> V1 := Vector([1, 2, 3, 4, 5, 6, 3, 8, 5, 3]);
V2 := Vector([seq(evalhf(sin(i)), i = 1 .. 1000)]);
with(Statistics);
InteractiveDataAnalysis(V1, V2);
```

При исполнении этих команд появляется окно ассистента, показанное на рис. 7.3. В поле Dataset имеется список из двух массивов, обозначенных как 1 и 2, в поле Qantities of Interest приведены имена вычисленных статистических параметров и их значения, а в поле Preview Dataset – гистограмма выделенного массива (в данном случае V1, обозначенный в списке как 1). Открытое меню Graph иллюстрирует возможные типы графиков, которыми можно представлять заданный массив.

Все виды графиков, имеющиеся в позиции `Graph` меню окна ассистента, реализованы соответствующими функциями пакета Statistics. Пример построения

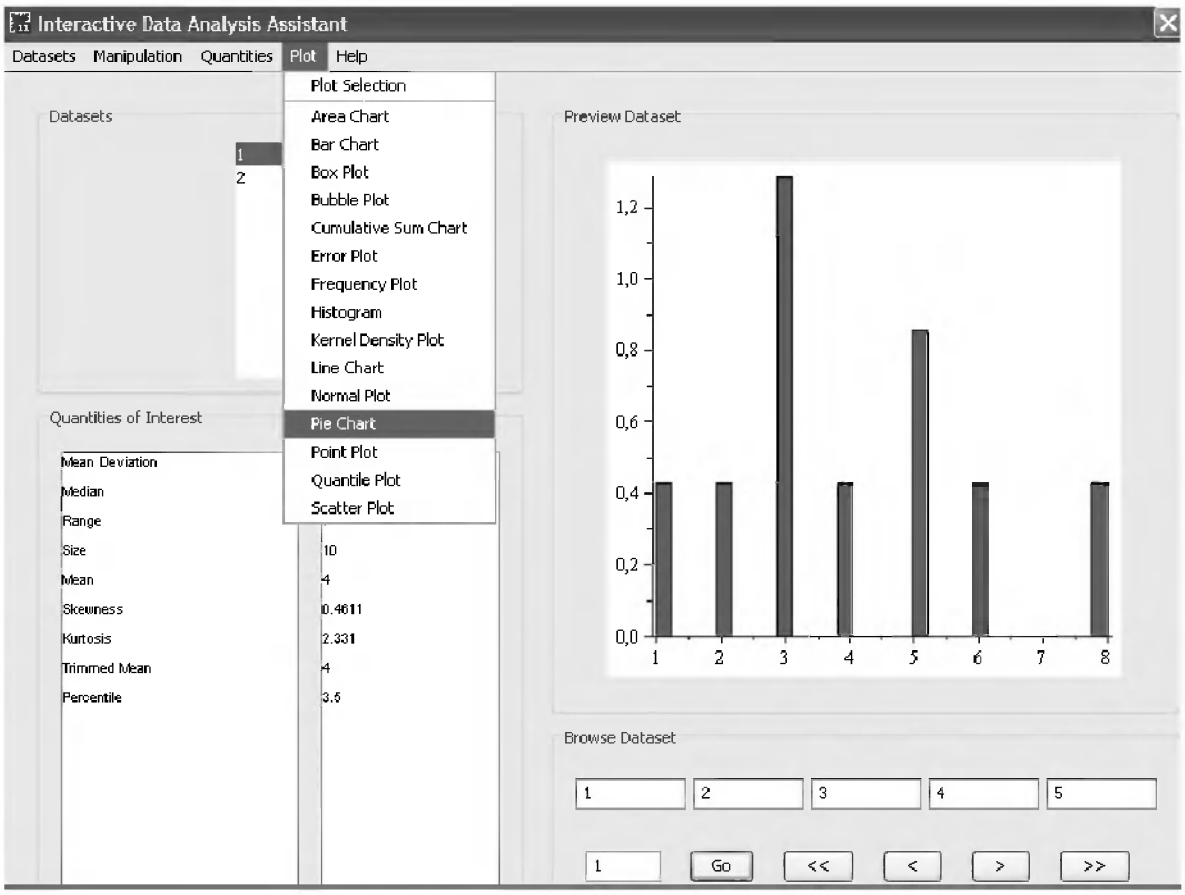


Рис. 7.3. Основное окно ассистента интерактивного статистического анализа данных

гистограммы и кривой нормального распределения уже приводился. Ассистент позволяет также строить круговые диаграммы, в том числе с удалением выбранного сегмента.

Читатель, выполняющий серьезные статистические расчеты, должен, разумеется, всерьез ознакомиться со справкой по пакету Statistics, поскольку выше были приведены лишь самые основные сведения об этом пакете и правила его применения. Они преследовали цель начального знакомства с пакетом. Одним из самых серьезных применений статистических вычислений является приближение функций и данных по методу наименьших квадратов (или по методу регрессии). Этот вид вычислений будет достаточно подробно описан в дальнейшем.

7.2.5. Пакет RandomTools для создания случайных объектов

Пакет для работы со случайными объектами RandomTools служит для расширения базовых возможностей системы Maple в части генерации различных случайных объектов, таких как числа различных форматов, векторы, матрицы, строковые символы, таблицы и т. д. Они образно названы `Flavor` (в буквальном переводе «букет (вина)»), что подчеркивает возможную сложность структуры создаваемых объектов.

Список функций пакета выдает команда его загрузки:

```
> with(RandomTools);
```

Пакет позволяет создавать различные случайные оъекты, например векторы, матрицы, полиномы и т. д. Для пользователей компьютерной алгебры этот мощный статистический пакет особого интереса не представляет, так что ограничимся упоминанием о его существовании. Читатели, интересующиеся пакетом, могут ознакомиться с ним по справке. Возможности пакета заметно различаются в разных версиях системы Maple.

7.3. Средства статистики в СКМ Mathematica 4/5

7.3.1. Комбинаторика и ее функции

Средства комбинаторики в системе Mathematica 4/5 представлены в пакете дискретной математики DiscreteMath. Эти средства уникальны – представлены около 450 функций комбинаторики и работы с графами. Описание большинства из них выходит за рамки тематики данной книги.

Подпакет Combinatorica задает определение ряда функций комбинаторики и теории графов. Следует отметить, что ввиду обилия функций даже в справочной системе даны примеры на лишь избранные функции. Для ознакомления с назначением каждой функции достаточно задать команду **?Имя_функции**. Примеры применения некоторых функций комбинаторики представлены ниже:

```
<<DiscreteMath'Combinatorica'
?Permute
 Permute[l, p] permutes list
    l according to permutation p. More...
MinimumChangePermutations[{a,b,c}]
 {{a,b,c},{b,a,c},{c,a,b},{a,c,b},{b,c,a},{c,b,a}}
Map[RankPermutation, Permutations[{1,2,3,4}]]
{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23}
```

Для большинства читателей подпакет комбинаторики содержит явно избыточное число функций. Большинство из них просты, и заинтересованный читатель может легко воспользоваться ими по мере необходимости.

7.3.2. Пакет расширения Statistics системы Mathematica

В ядре системы Mathematica практически нет статистических функций. Зато пакет расширения Statistics имеет множество функций, охватывающих практически все расчеты теоретической и прикладной статистики. Пакет расширения Statistics содержит следующие подпакеты:

- ConfidenceIntervals – функции доверительных интервалов;
- ContinuousDistributions – функции непрерывных распределений;
- DataManipulation – манипуляции с данными;
- DataSmoothing – сглаживание данных;
- DescriptiveStatistics – статистика распределений;
- DiscreteDistributions – функции дискретных распределений;
- HypothesisTests – проверка статистических гипотез;
- LinearRegression – линейная регрессия (см. главу 6);
- MultiDescriptiveStatistics – статистика множественных распределений;
- MultinormalDistribution – функции множественных нормальных распределений;
- NonlinearFit – нелинейная регрессия;
- NormalDistribution – функции нормального распределения;
- Common – данные общего характера.

Для работы каждого из подпакетов требуется его загрузка в память компьютера с помощью команды

```
>>Statistics'Имя_подпакета'
```

Имена подпакетов расширения статистики приведены выше.

7.3.3. Манипуляции с данными – DataManipulation

Статистические данные обычно бывают представлены в виде списков данных – как одномерных, так и двумерных (таблиц и матриц) и даже многомерных. Большая часть функций, обеспечивающих манипуляции с данными, сосредоточена в подпакете DataManipulation.

Данные могут вводиться в строках ввода или считываться из файлов с помощью функции **ReadList**. Для манипуляций с данными могут использоваться многие функции ядра системы, описанные ранее, – в частности, все функции обработки списков. Подпакет DataManipulation дает ряд удобных функций. Ниже представлена первая группа таких функций:

- **Column[data,n]** – возвращает n-ый столбец списка data;
- **Column[data,{n1,n2,...}]** – возвращает список из ni столбцов списка данных;
- **ColumnTake[data,spec]** – возвращает столбцы списка data с данной спецификацией spec;
- **ColumnDrop[data,spec]** – удаляет столбцы списка data с данной спецификацией;
- **ColumnJoin[data1,data2,...]** – объединяет столбцы списков datai;
- **RowJoin[data1,data2,...]** – объединяет строки списков datai;
- **DropNonNumeric[data]** – удаляет из списка data нечисловые элементы;
- **DropNonNumericColumn[data]** – удаляет из списка data столбцы с нечисловыми элементами.

Примеры применения этих функций:

```
<<Statistics'DataManipulation'
data = {{a, 7}, {b, 2}, {c, 3}, {d, w}, {e, 5}, {f, 6}}
 {{a,7},{b,2},{c,3},{d,w},{e,5},{f,6}}
col2 = Column[data, 2]
 {7,2,3,w,5,6}
newdata = DropNonNumeric[col2]
 {7,2,3,5,6}
```

Полезны также следующие функции подпакета:

- **BooleanSelect[list,sel]** – удаляет из **list** элементы, которые дают True при тестировании **sel**;
- **TakeWhile[list,pred]** – удаляет из list все элементы, начиная с того, для которого **pred** дает True;
- **LengthWhile[list,pred]** – возвращает число элементов, которые были удалены после того, как **pred** дало значение True (отсчет с начала листа).

Примеры на применение этих функций:

```
TakeWhile[col2, NumberQ]
 {7  ,2  ,3}
LengthWhile[col2, NumberQ]
 3
```

Ряд функций служит для подготовки данных с целью построения гистограмм:

- **Frequancies[list]** – готовит данные для представления частотной гистограммы;
- **QuantileForm[list]** – дает отсортированные данные для представления квантилей;
- **CumulativeSums[list]** – дает кумулятивное суммирование данных листа.

Для подготовки гистограмм могут использоваться и следующие функции:

```
BinCounts[data,{min,max,dx}]        RangeCounts[data,{c1,c2,...}]
CategoryCounts[data,{e1,e2,...}]    BinLists[data,{min,max,dx}]
RangeLists[data,{c1,c2,...}]        CategoryLists[data,{e1,e2,...}]
```

С достаточно простыми примерами их работы можно ознакомиться по справочной системе Mathenatica с описанием данного подпакета.

7.3.4. Стандартная обработка массива данных

В подпакете DescriptiveStatistics сосредоточены наиболее важные и общеизвестные функции по статистической обработке массивов (списков) данных:

- **CentralMoment(data,r)** – возвращает центральный момент данных data порядка r;
- **Mean[data]** – возвращает среднее значение данных data;
- **MeanDeviation[data]** – возвращает среднее отклонение данных;
- **Median[data]** – возвращает центральное значение (медиану) данных;

- **MedianDeviation[data]** – возвращает абсолютное отклонение (от медианы) данных;
- **Skewness[data]** – возвращает коэффициент асимметрии данных (эксцесс);
- **StandardDeviation[data]** – возвращает стандартное отклонение данных;
- **GeometricMean[data]** – возвращает геометрическое среднее данных;
- **HarmonicMean[data]** – возвращает гармоническое среднее данных;
- **RootMeanSquare[data]** – возвращает среднеквадратическое значение;
- **Quantile[data,q]** – возвращает q-й квантиль;
- **InterpolatingQuantile[data,q]** – возвращает q-й квантиль, используя при вычислениях интерполяцию данных;
- **VarianceData[data]** – возвращает среднеквадратическое отклонение данных.

Мы не приводим определений этих функций, поскольку при символьных списках **data** их легко получить именно в том виде, который реализован в системе Mathematica:

```
<<Statistics'DescriptiveStatistics'
ds={x1,x2,x3}
{x1, x2, x3}
Mean[ds]
```

$$\frac{1}{3}\ (x1 + x2 + x3)$$

```
MeanDeviation[ds]
```

$$\frac{1}{3}\left(\text{Abs}\left[x1 + \frac{1}{3}(-x1 - x2 - x3)\right] + \text{Abs}\left[x2 + \frac{1}{3}(-x1 - x2 - x3)\right] + \text{Abs}\left[x3 + \frac{1}{3}(-x1 - x2 - x3)\right]\right)$$

```
Variance[ds]
```

$$\frac{1}{2}\left(\left(x1 - \frac{1}{3}(x1 + x2 + x3)\right)^2 + \left(x2 - \frac{1}{3}(x1 + x2 + x3)\right)^2 + \left(x3 - \frac{1}{3}(x1 + x2 + x3)\right)^2\right)$$

```
Skewness[ds]
```

$$\sqrt{3}\left(\left(x1 + \frac{1}{3}(-x1 - x2 - x3)\right)^3 + \left(x2 + \frac{1}{3}(-x1 - x2 - x3)\right)^3 + \left(x3 + \frac{1}{3}(-x1 - x2 - x3)\right)^3\right) \Big/ \left(\left(x1 + \frac{1}{3}(-x1 - x2 - x3)\right)^2 + \right.$$

$$\left(x2 + \frac{1}{3}(-x1 - x2 - x3) \right)^2 + \left(x3 + \frac{1}{3}(-x1 - x2 - x3) \right)^2 \Bigg)^{3/2}$$

Следующие примеры поясняют действие этих функций при обработке уже конкретных численных данных:

```
data:={10.1,9.6,11,8.2,7.5,12,8.6,9}
CentralMoment[data,2]
 1.9525
Mean[data]
 9.5
MeanDeviation[data]
 1.175
Median[data]
 9.3
MedianDeviation[data]
 0.95
Skewness[data]
 0.374139
StandardDeviation[data]
 1.4938
GeometricMean[data]
 9.39935
HarmonicMean[data]
 9.30131
RootMeanSquare[data]
 9.60221
Quantile[data,1]
 12.
InterpolatingQuantile[data,1]
 InterpolatingQuantile[{10.1,9.6,11,8.2,7.5,12,8.6,9},1]
Variance[data]
 2.23143
```

В этом случае результатом действия функций обычно являются числа в формате плавающей точки. С рядом других, менее распространенных функций этого подпакета можно ознакомиться с помощью справочной системы. Там же даны примеры на их применение.

7.3.5. Линейное сглаживание и фильтрация

В подпакете DataSmoothing определены функции для *сглаживания данных*, имеющих большой случайный разброс. К таким данным обычно относятся результаты ряда физических экспериментов, например по энергии элементарных частиц, или сигналы, поступающие из космоса. Для того чтобы выделить полезную информацию из таких данных с большим уровнем шумов, и применяется процедура сгла-

живания. Она может быть линейной (например, усреднение по ряду точек) или нелинейной (например, экспоненциальной).

В данном подпакете определены следующие функции сглаживания:

- **MovingAverage[data,r]** – сглаживание данных **data** методом скользящего среднего для **r** точек;
- **MovingMedian[data,r]** – сглаживание данных **data** по медиане для r точек (опция RepeatedSmoothing->True используется для повторного сглаживания);
- **LinearFilter[data,{c_0,c_1,...,c_{r-1}}]** – линейная фильтрация (**c_j** – весовые множители).

Применение сглаживания усреднением иллюстрирует рис. 7.4. На нем задан массив (таблица) из 500 случайных точек с равномерным распределением и создан графический объект из этих точек в виде кружков малого диаметра. Затем выполнена операция сглаживания (по 12 смежным точкам) и создан графический объект сглаженных точек в виде кружков большего диаметра. Для сопоставления оба объекта построены на одном графике функцией **Show**.

Нетрудно заметить, что сглаженные точки группируются вокруг среднего значения, равного 0.5, тогда как исходные точки разбросаны практически равномерно по всему полю рисунка. Остальные функции сглаживания можно использовать аналогичным образом.

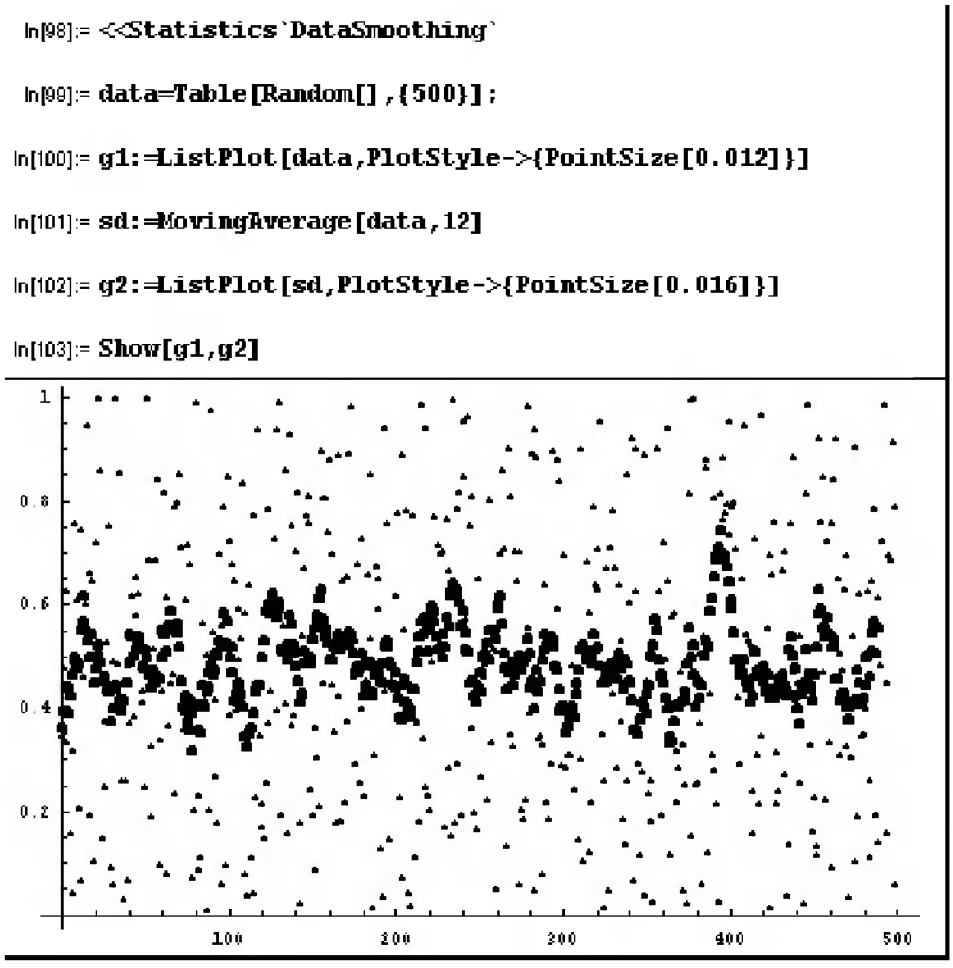


Рис. 7.4. Пример линейного сглаживания данных из 500 точек

7.3.6. Экспоненциальное сглаживание

Для нелинейного экспоненциального сглаживания имеется следующая функция:

- **ExponentialSmoothing[data,a]** – дает экпоненциальное (нелинейное) сглаживание с параметром **a**, задающим степень сглаживания.

Эффективность нелинейного (экспоненциального) сглаживания демонстрирует рис. 7.5, документ которого построен по тому же принципу, что и показанный на рис. 7.4. Нетрудно заметить, что нелинейное сглаживание в данном случае явно более эффективно.

```
<<Statistics`DataSmoothing`

data=Table[Random[],{500}];

g1:=ListPlot[data,PlotStyle->{PointSize[0.012]}]

sd:=ExponentialSmoothing[data,0.05]

g2:=ListPlot[sd,PlotStyle->{PointSize[0.016]}]

Show[g1,g2]
```

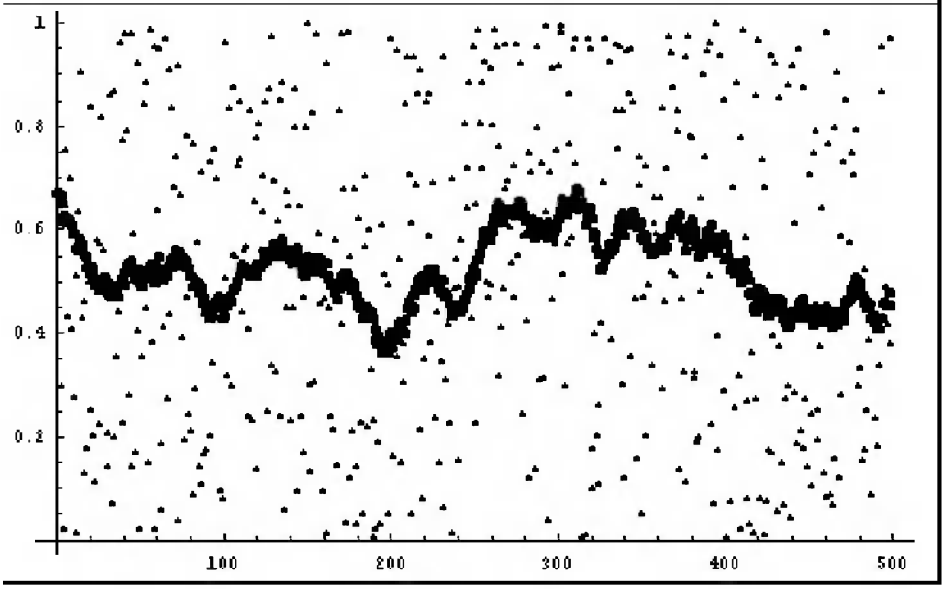


Рис. 7.5. Пример экспоненциального сглаживания

Выбор метода сглаживания зависит от решаемых пользователем задач и остается за ним. Сильное сглаживание ведет к потере быстрых компонент сигнала. А сравнение рис. 12.2 и 12.3 показывает, что для одного и того же массива случайных чисел вид сглаженной зависимости получается разным, что свидетельствует о необходимости очень внимательного отношения к применению этой операции.

7.3.7. Функции нормального распределения

Подпакет NormalDistribution содержит хорошо известные функции *нормального распределения* вероятностей и родственные им функции следующих распределений:

- **NormalDistribution[mu,sigma]** – нормальное распределение;
- **StudentT Distribution[r]** – T-распределение Стьюдента;
- **ShiSquareDistribution[r]** – хи-квадрат распределение;
- **FRatioDistribution[r1,r2]** – F-распределение.

Для этих и многих других непрерывных распределений заданы также функции плотности распределения, среднего, среднеквадратического отклонения, стандартного отклонения, вычисления коэффициента асимметрии и др. Рисунок 7.6 иллюстрирует получение выражения для плотности нормального распределения pdf и получения графика плотности этого распределения со смещенной вершиной.

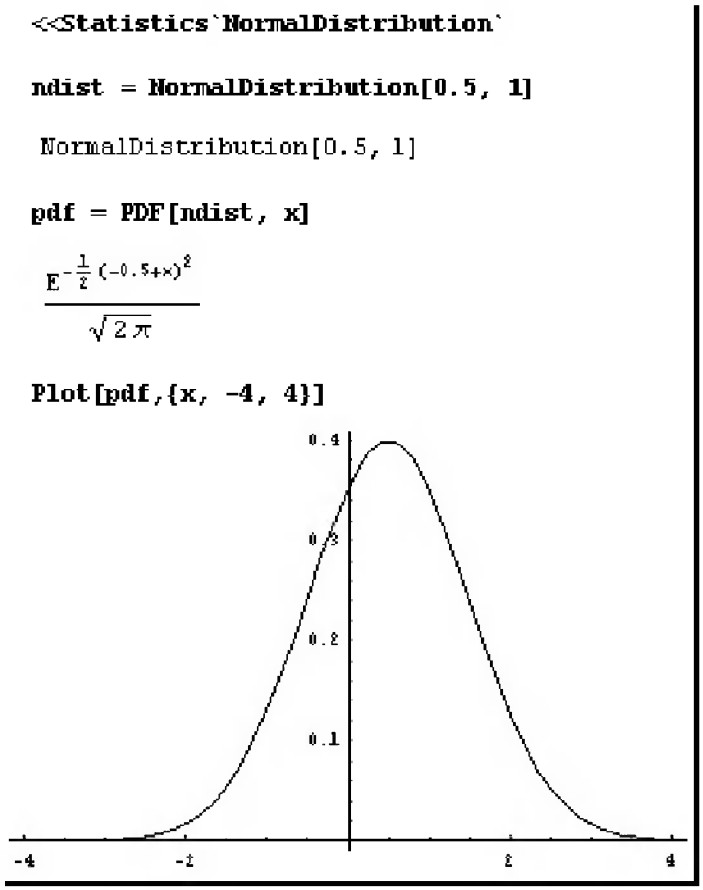


Рис. 7.6. Пример работы с функцией нормального распределения

Аналогичным образом легко построить графики других функций распределения.

7.3.8. Функции непрерывного распределения

Целый ряд функций, подобных вышеописанным, задан и в пакете ContinuiusDistribution для ряда функций непрерывного распределения. Прежде всего это функции плотности вероятности непрерывного распределения:

- **BetaDistribution[α,β]** – непрерывное бета-распределение;
- **CauchyDistribution[a,b]** – распределение Коши;
- **ChiSquareDistribution[n]** – хи-квадрат распределение;
- **ExponentialDistribution[γ]** – экспоненциальное распределение;
- **ExtremeValueDistribution[α,β]** – распределение Фишера-Типпета;
- **FratioDistribution[n1,n2]** – F-распределение;
- **GammaDistribution[α,γ]** – гамма-распределение;
- **NormalDistribution[μ,σ]** – нормальное распределение (Гауссиана);
- **LaplaceDistribution[μ,β]** – Лапласа (двойное экспоненциальное) распределение;
- **LogNormalDistribution[μ,σ]** – логарифмическое нормальное распределение;
- **LogisticDistribution[μ,β]** – логистическое распределение;
- **RayleighDistribution[σ]** – Релея распределение;
- **StudentDistribution[n]** – распределение Стьюдента;
- **UniformDistribution[min,max]** – равномерное распределение;
- **WeibullDistribution[α,β]** – распределение Вейбулла.

Для всех этих законов (**dist**) определен ряд статистических функций. Из них наиболее важными являются следующие функции:

- **PDF[dist,x]** – возвращает значение плотности вероятности распределения при аргументе **x** и заданном законе распределения **dist**.
- **CDF[dist,x]** – возвращает значение кумулятивной функции распределения – интеграла от функции плотности распределения.
- **Quantile[dist,q]** – возвращает q-й квантиль для заданного закона распределения.
- **Domain[dist]** – возвращает пределы значений переменной.
- **Mean[dist]** – возвращает среднее значение данных data.
- **Varians[dist]** – возвращает дисперсию для заданного закона распределения.
- **StandardDeviation[dist]** – возвращает стандартное отклонение для заданного закона распределения.
- **Skewness[dist]** – возвращает коэффициент асимметрии для заданного закона распределения.
- **Kurtosis[dist]** – возвращает значение коэффициента эксцесса.
- **KurtosisExess[dist]** – возвращает эксцесс (остроту пика).
- **CharacteristicFunction[]dist,t]** – возвращает характеристическую функцию $\varphi(t)$ для заданного закона распределения.
- **ExpextedValue[f,dist]** – возвращает ожидаемое значение чистой функции **f**.

- **ExpextedValue[f,dist,x]** – возвращает ожидаемое значение чистой функции **f** в точке x.
- **Random[dist]** – создает псевдослучайное число с заданным распределением.
- **RandomArray[dist,]** – создает массив с размерностью **dims** псевослучайных чисел с заданным распределением.

Приведем примеры применения некоторых из этих функций для экспоненциального закона распределения:

```
<<Statistics'ContinuousDistributions'
edist = ExponentialDistribution[1]
  ExponentialDistribution[1]
CDF[edist, 10]
  1 - 1/e^10
cdfunction = CDF[edist, x]
  1 - e^-x
Random[edist]
  1.75426
Random[edist]
  0.499882
RandomArray[gdist, 10]
{0.404935,0.460706,2.20041,0.349605,0.245138,0.329001,0.900709,
0.231028,0.454624,0.413024}
```

В этом пакете представлен также ряд менее распространенных статистических функций, например для нецентрированных законов распределения – хи-квадрат, Стьюдента и Фишера. Определение большинства этих функций представлено далее в разделе 8.3.2, в котором эти функции описаны для массовой СКМ Mathcad.

7.3.9. Функции дискретного распределения

Пакет DiskreteDistribution содержит функции для дискретного распределения вероятностей:

- **BernoulliDistribution[p]** – дискретное распределение Бернулли со средним p;
- **BinomialDistribution[n,p]** – биномиальное распределение;
- **DiscreteUniformDistribution[n]** – дискретное равномерное распределение;
- **GeometricDistribution[p]** – геометрическое распределение;
- **HyperGeometricDistribution[n,ns,nt]** – гипергеометрическое распределение;
- **NegativeBinomial Distribution[r,p]** – отрицательное биномиальное распределение;
- **PoissonDistribution[mu]** – распределение Пуссона со средним mu.

В этом подпакете есть также набор статистических функций (PDF, CDF и др.), подобных приведенным для подпакета непрерывных распределений. В связи с этим ограничимся несколькими примерами, приведенными ниже:

```
<<Statistics'DiscreteDistributions'
pdist = PoissonDistribution[0.5]
 PoissonDistribution[0.5]
Mean[pdist]
 0.5
Quantile[pdist, .85]
 1
ExpectedValue[x^3, pdist, x]
 5
RandomArray[bdist, {3, 3}]
 {{11,10,9},{11,16,8},{15,13,10}}
```

Подпакеты непрерывных и дискретных распределений охватывают наиболее распространенные законы распределения. В разделе 8.3.2 даны определения ряда этих функций.

7.3.10. Графика пакета Statistica системы Mathematica

В подпакете StatisticsPlots есть ряд функций для построения статистических данных:

- **BoxWhiskerPlot[data]** – построение «ящика с усами»;
- **ParetoPlot[data]** – построение диаграммы Парето;
- **QuantilePlot[list1,list2]** – построение графиков квантилей;
- **PairwiseScatterPlot[matrix]** – построение точечной диаграммы для многовариантных данных.

Построение «ящика с усами» для ста случайных чисел (рис. 7.7) иллюстрирует следующий пример:

```
<<Statistics'StatisticsPlots'
dat = Table[Random[], {100}];
BoxWhiskerPlot[dat]
```

Следующий пример демонстрирует построение пяти «ящиков с усами» (рис. 7.8):

```
datm = Table[Random[], {100}, {5}];
BoxWhiskerPlot[datm]
```

Параметры «ящиков с усами» можно менять с помощью опций. Так, обычно «ящики с усами» строятся для квантилей, отстоящих от медианы на величину 0,25. Опция BoxQuantile позволяет изменить это значение. Опция BoxOrientation с начальной установкой Vertical задает вертикальное расположение ящиков, тогда как эта опция с установкой Horizontal позволяет строить горизонтально расположенные «ящики с усами». С другими опциями можно познакомиться по справке.

Пример построения диаграммы Парето представлен на рис. 7.9. Диаграмма представляет собой одновременное построение гистограммы и интегральной (кумулятивной) кривой для данных. В этом примере интересно представление данных в виде массива символов.

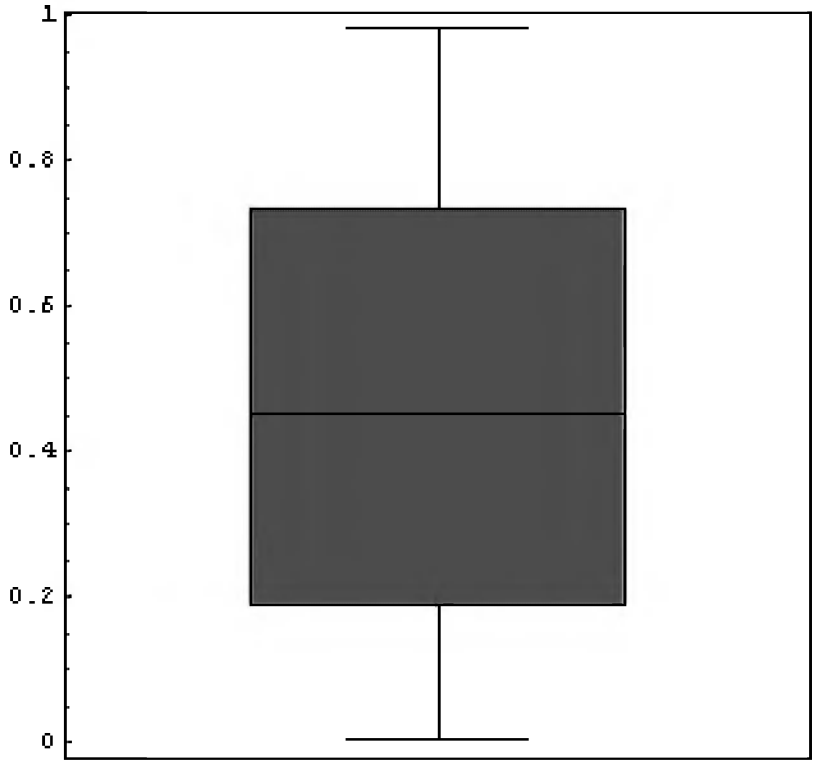


Рис. 7.7. Диаграмма типа «ящик с усами»

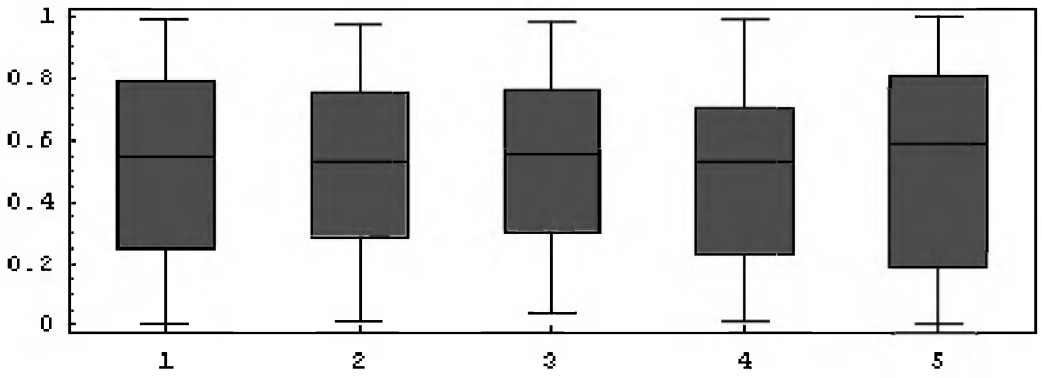


Рис. 7.8. Построение пяти «ящиков с усами»

```
ParetoPlot[{a, b, c, d, d, d, e, d, e, e, f, a, b, c}]
```

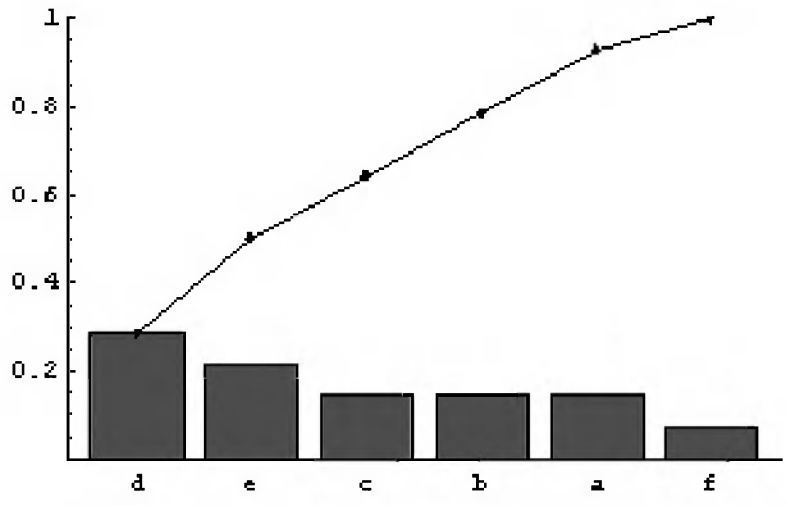


Рис. 7.9. Пример построения диаграммы Парето

7.3.11. Другие функции пакета Statistics

Для выполнения дисперсионного анализа служит подпакет ANOVA дисперсионного анализа. Он содержит основную функцию

- **ANOVA[data]** – односторонний дисперсионный анализ.
- **ANOVA[data,mofel,factor]** – общий дисперсионный анализ со спецификацией model и factors.

Вывод результатов этого анализа довольно громоздок, и его можно найти в примерах справки по данному подпакету.

Функции для оценки доверительных интервалов сосредоточены в подпакете ConfidencelInterval. Приведем примеры применения некоторых функций этого подпакета:

```
<<Statistics'ConfidenceIntervals'
data1 = {2.0, 1.25, 0.7, 1.0, 1.1, 3.2, 3.2,
 3.3, 2.1, 0.3};
data2 = {1.8, 0.2, 1.5, 1.9, 1.1, 3.0, 2.3,
 0.9, 2.4, 1.0};
MeanCI[data2]
 MeanCI[{1.8,0.2,1.5,1.9,1.1,3.,2.3,0.9,2.4,1.}]
MeanDifferenceCI[data1, data2, EqualVariances -> True]
 {-0.72087,1.13087}
MeanDifferenceCI[data1, data2]
 {-0.726116,1.13612}
VarianceCI[data1]
 {0.588256,4.14394}
VarianceRatioCI[data1, data2, ConfidenceLevel -> .9]
 {0.559735,5.65632}
mean = Mean[data1]
 1.815
se = StandardErrorOfSampleMean[data1]
 0.352613
StudentTCI[mean, se, Length[data1] — 1,
ConfidenceLevel -> 0.9]
 {1.16862,2.46138}
```

В подпакете HypothesTests сосредоточено сравнительно небольшое число хорошо известных функций для выполнения тестов *проверки статистических гипотез*. Загрузка пакета и проведение теста на среднее значение показаны ниже:

```
<<Statistics'HypothesisTests'
data1 = 34, 37, 44, 31, 41, 42, 38, 45, 42, 38;
MeanTest[data1, 34, KnownVariance -> 8]
  OneSidedPValue→ 3.05394r10^-9
```

Пакеты LinearRegression и NonlinearFit содержат средства для выполнения линейной регрессии общего вида и нелинейной регрессии. Все функции этих подпакетов были подробно описаны в главе 6.

У специалистов в области статистики интерес вызовут подпакеты MultiDescriptiveStatistics и MultinormalDistribution с многочисленными функциями мно-

жественных распределений. Они позволяют оценивать статистические характеристики объектов, описываемых функциями многих переменных. Рисунок 7.10 поясняет загрузку подпакета MultinormalDistribution, получение выражения для плотности нормального распределения по двум переменным $x1$ и $x2$ и получение трехмерного графика для плотности такого распределения.

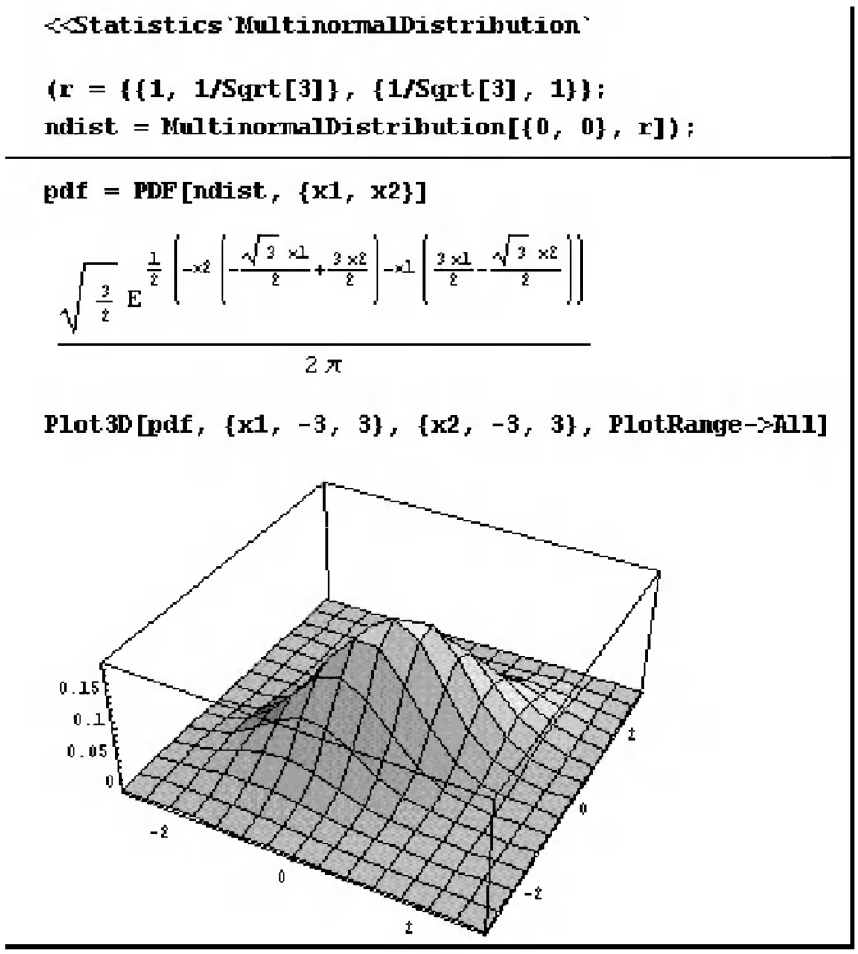


Рис. 7.10. Получение аналитического выражения и графика нормального распределения по двум переменным

Подпакет Common содержит перечисление всех определений, вошедших в расширение Statistics, с указанием директорий, в которые они входят.

7.4. Статистика в СКМ Mathcad

СКМ Mathcad имеет хорошо организованные средства для статистических вычислений, ориентированные на инженерные и научно-технические расчеты [37–53]. По некоторым показателям, например удобству предоставления статистических функций, возможности проведения нелинейной регрессии и др., Mathcad превосходит описанные выше средства более мощных СКМ.

7.4.1. Функции комбинаторики в СКМ Mathcad

В системе Mathcad имеются следующие основные функции комбинаторики:

- `permut(n,m)` – вычисление числа размещений;
- `combin(m,n)` – вычисление числа сочетаний.

Примеры применения этих функций:

`permut(10,2)=90` **`combin(10,3)=120.`**

Эти функции способны решать простые задачи комбинаторики. При решении сложных задач возможно переполнение разрядной сетки компьютера. В этом случае нужно переходить к вычислениям функций комбинаторики прямо через факториалы, используя аппарат точной арифметики (оператор символьного вывода →).

7.4.2. Генерация случайных чисел с равномерным распределением

Mathcad имеет функцию `rnd(x)` для генерации случайных чисел с равномерным распределением на отрезке $[0, x]$.

Функция `hist(int, A)` возвращает вектор частот попадания данных массива `A` в заданные вектором `int` интервалы. Вектор `int` должен содержать значения границ, число попаданий данных из вектора `M` должно подсчитываться. Если строится гистограмма из N элементов, то вектор `int` должен содержать $N+1$ элементов. Функция возвращает вектор из N элементов, числовые значения которых можно использовать для графического построения гистограмм.

Приведем следующий пример: надо создать вектор из 1000 случайных чисел с равномерным распределением на отрезке [0,10], построить график точек, представляющих эти числа, и гистограмму из 10 столбцов. Документ, осуществляющий эти операции, представлен на рис. 7.11. Равномерность заполнения точками прямоугольника рисунка и примерно одинаковая высота всех столбиков гистограммы указывают, что и впрямь распределение чисел близко к равномерному.

7.4.3. Функции статистики в Mathcad

В Mathcad имеется ряд наборов функций, относящихся к наиболее распространенным законам распределения. Характер функции для каждого закона распределения задается первой буквой их имени:

- `d` (Density) – плотность вероятности $f(x)$;
- `p` (Probality) – функция распределения $F(x)$;
- `q` (Quantil) – инверсная функция распределения – *квантиль*;
- `r` (Random) – вектор случайных чисел.

Квантили функций распределения случайных величин позволяют по заданной вероятности вычислить такое значение x, при котором вероятность равна или

Рис. 7.11. Работа со случайными числами и оценка их параметров в системе Mathcad

меньше заданного значения p. А функции, начинающиеся с буквы r, служат для генерации случайных чисел с заданным законом распределения.

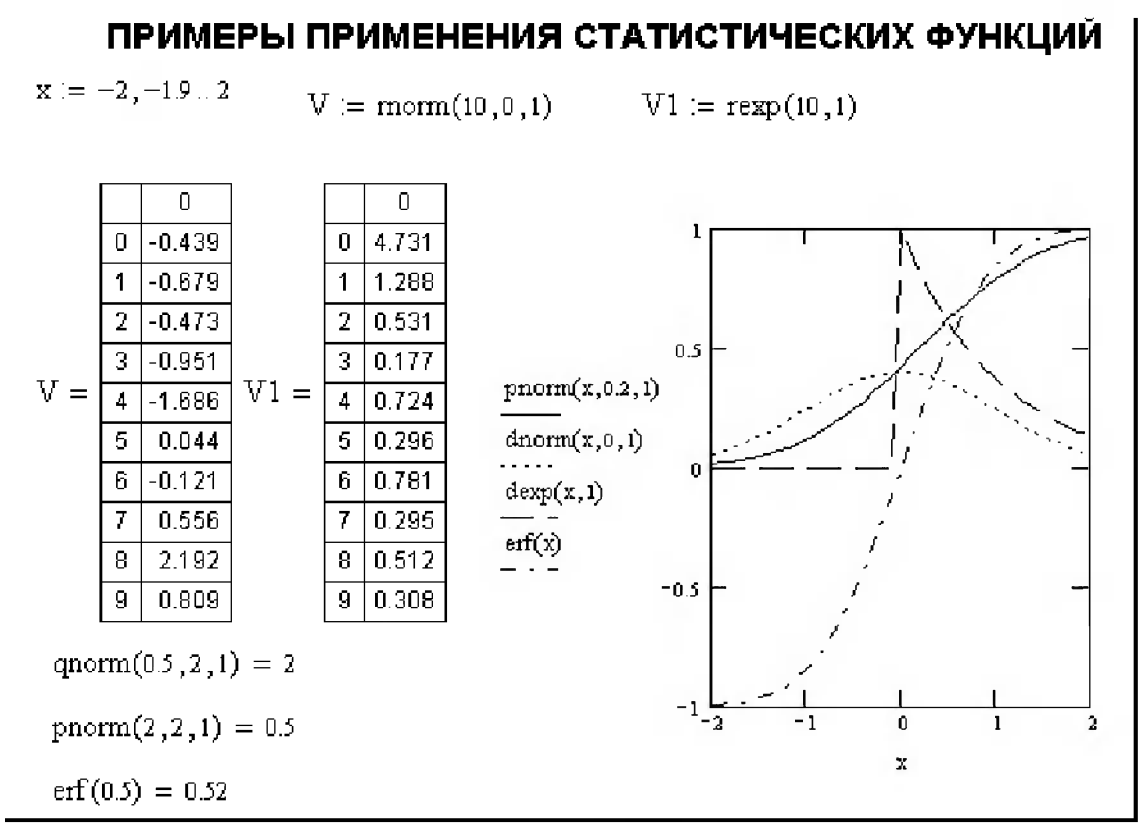
В качестве примера приведем функции нормального и экспоненциального распределений:

- `dnorm(x,μ,σ)` `pnorm(x,μ,σ)` `qnorm(p,μ,σ)` `rnorm(m,μ,σ)` – нормальное распределение ($\mu = a$ – среднее значение, $\sigma > 0$ – среднеквадратичное отклонение);
- `dexp(x,r)` `pexp(x,r)` `qexp(p,r)` `rexp(m,r)` – экспоненциальное распределение (r, $x>0$).

Статистические расчеты – весьма обширная область математики. К статистическим функциям общего характера в системе Mathcad относятся следующие функции скалярного аргумента x:

- `cnorm(x)` – кумулятивная нормальная функция – подобна функции `pnorm (x,0,1)`, описанной выше;
- `erf(x)` – функция ошибок (или интеграл вероятности) $erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$;
- `cerf(x)` – дополнительная функция ошибок (1-$erf(x)$).

Приведем несколько примеров применения статистических функций, представленных выше. Примеры представлены на рис. 7.12.



	0
0	-0.439
1	-0.679
2	-0.473
3	-0.951
4	-1.686
5	0.044
6	-0.121
7	0.556
8	2.192
9	0.809

	0
0	4.731
1	1.288
2	0.531
3	0.177
4	0.724
5	0.296
6	0.781
7	0.295
8	0.512
9	0.308

Рис. 7.12. Примеры применения статистических функций

Следующая группа функций относится к вычислению основных статистических параметров одного массива данных (матрицы размера $m \times n$ или вектора):

- `mean(A)` – возвращает среднее значение элементов массива `A`: $mean = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1} A_{i,j}$;
- `gmean(A)` – возвращает гармоническое среднее значение элементов массива `A`: $gmean = (\prod_{i=0}^{m-1}\prod_{j=0}^{n-1} A_{i,j})^{1/(mn)}$;
- `hmean(A)` – возвращает геометрическое среднее значение элементов массива `A`: $mean = (\frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\frac{1}{A_{i,j}})^{-1}$;
- `median(A)` – возвращает медиану массива `A`;
- `mode(A)` – возвращает значение наиболее часто повторяющегося элемента массива, если он явно есть, в противном случае дает сигнал ошибки;
- `stdev(A)` – задает стандартное отклонение элементов массива `A` – $\sqrt{\mathrm{var}(A)}$;
- `Stdev(A)` – задает выборочное стандартное отклонение элементов массива A – $\sqrt{Var(A)}$;
- `var(A)` – возвращает так называемую смещенную оценку дисперсии (вариацию) для элементов массива `A`: $\mathrm{var} = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\left|A_{i,j} - mean(A)\right|^2$;

- `Var(A)` – возвращает несмещенную оценку дисперсии для элементов массива `A` с иной, чем у функции `var(A)`, нормировкой:

$$\mathrm{var} = \frac{1}{mn-1}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\left|A_{i,j} - mean(A)\right|^2$$

- `kurt(A)` – возвращает значение эксцесса (остроты кривой распределения):

$$kurt = \frac{mn(mn+1)}{(mn-1)(mn-2)(mn-3)}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\left(\frac{A_{i,j} - mean(A)}{Stdev(A)}\right)^4 - \frac{3(mn-1)^2}{(mn-2)(mn-2)}.$$

Эти функции часто используются для подсчета частоты используемых в речи звуков, букв и слов, данных о людях и др.

Функция mode нуждается в некоторых пояснениях. *Модой* в статистике называют варианту, которая встречается в выборке чаще всего. Действие функции mode очевидно из приведенных ниже примеров:

Массив	**mode**
1, 2, 3, 4, 5	нет
1, 2, 3, 3, 4, 5, 6	3
1, 2, 2, 3, 4, 5, 5, 5, 6, 7, 8	2, 5

Полезно также вспомнить о понятии медианы. Это варианта, которая делит отсортированный ряд вариант на две выборки с равным числом вариант в каждой.

Если число элементов выборки нечетно, то есть определяется как 2k+1, то медианой будет (k+1)-й элемент. При четном числе элементов в выборке медианой будет полусумма (среднее арифметическое) k-го и (k+1)-го элементов.

Когда речь идет о некоторых зависимостях, данные обычно представляются двумя и более массивами. Степенью связи зависимостей является *коэффициент корреляции* `corr(A, B)` – коэффициент корреляции Пирсона двух массивов `A` и `B` (*m*×*n* элементов). Если коэффициент корреляции близок к 0, то данные (зависимости) не согласованы, а при близости его к 1 данные согласованы.

7.4.4. Функции вычисления плотности распределения вероятности

Функции вычисления плотности вероятности распределения в Mathcad представлены следующим набором:

- `dbeta(x,s1,s2)` – бета-распределение (`s1`, `s2>0` – параметры формы, `0<x<1`);
- `dbinom(k,n,p)` – биномиальное распределение (возвращает значение вероятности `P(x = k)`, где `n` и `k` – целые числа, причем `0≤k≤n` и `0≤p≤1`);
- `dcauchy(x,l,s)` – распределение Коши (`l` – параметр разложения, `s>0` – параметр масштаба);
- `dchisq(x,d)` – хи-квадрат распределение (`x`, `d>0`, причем `d` – число степеней свободы);
- `dexp(x,r)` – экспоненциальное распределение (`r`, `x>0`);

- `dF(x,d1,d2)` – распределение Фишера (`d1`, `d2>0` – числа степеней свободы, `x>0`);
- `dgamma(x,s)` – гамма-распределение (`s>0` – параметр формы, `x≥0`);
- `dgeom(k,p)` – геометрическое распределение (`0<p≤1` – вероятность успеха в отдельном испытании, `k` – целое неотрицательное число);
- `dlnorm(x,mm,ss)` – логарифмическое нормальное распределение (mm – натуральный логарифм среднего значения, `ss>0` – натуральный логарифм среднеквадратичного отклонения, `x>0`);
- `dlogis(x,l,s)` – логистическое распределение (`l` – параметр разложения, `s>0` – параметр масштаба);
- `dnbinom(k,n,p)` – отрицательное биномиальное распределение (`n>0` и `k>0` – целые числа, `0<p≤1`);
- `dnorm(x,mm,ss)` – нормальное распределение (mm – среднее значение, `ss>0` – среднеквадратичное отклонение);
- `dpois(k,ll)` – распределение Пуассона (`ll>0`, `k` – целое неотрицательное число);
- `dt(x,d)` – распределение Стьюдента (`d>0` – число степеней свободы, `x` – вещественное число);
- `dunif(x,a,b)` – равномерное распределение (`a` и `b` – граничные точки интервала, причем `a<b` и `a≤x≤b`);
- `dweibull(x,s)` – распределение Вейбулла (`s>0` – параметр формы).

Обратите внимание, что имена у всех этих функций начинаются с буквы d – затем следует название собственно функции.

7.4.5. Функции распределения

Функции распределения дают вероятность того, что случайная величина будет иметь значения, меньшие или равные определенной величине. Для Mathcad функции распределения начинаются с имени p и в остальном записываются аналогично функциям, представленным в разделе 7.4.4. Например, `pbeta(x,s1,s2)` – значение в точке `x` функции бета-распределения.

7.4.6. Квантили распределения

Следующая группа задает обращения (квантили) функций распределения случайных величин. Они позволяют по заданной вероятности вычислить такое значение `x`, при котором вероятность равна или меньше заданного значения `p`. Имена этих функций начинаются с буквы q. Например, `qbeta(p,s1,s2)` – квантили обратного бета-распределения с параметрами формы `s1` и `s2`.

7.4.7. Функции создания случайных чисел с различными законами распределения

Последняя группа статистических функций служит для создания векторов со случайными числами с определенными законами их распределения. Они начинаются с буквы r, например:

- `rbeta(x, б1, б2)` – бета-распределение;
- `rnorm(x,a,ss)` – нормальное распределение и т. д.

Обратите внимание на то, что обозначения параметров приведенных функций отличаются от принятых в описании Mathcad. Это сделано для того, чтобы согласовать их с приведенной ниже табл. 7.1, в которой даны аналитические определения законов распределения.

Таблица 7.1. Законы распределения и функции Mathcad для создания векторов с заданными законами распределения

Название распределения	Функция Mathcad	Функция плотности распределения
Бета-распределение	Rbeta	$f_{\beta(\alpha_1,\alpha_2)}(x) = \begin{cases} \dfrac{\Gamma(\alpha_1+\alpha_2)}{\Gamma(\alpha_1)\ \ (\alpha_2)} \cdot x^{\alpha_1-1}(1-x)^{\alpha_2-1} & \text{при} \quad 0 \le x \le 1; \\ 0 \quad \text{для остальных значений } x \end{cases}$
Биномиальное	Rbinom	$f(m) = [\dfrac{n!}{m! \cdot (n-m)!}] \cdot p^m \cdot q^{n-m}$
Коши	rcauchy	$f(x) = \dfrac{1}{\pi} \cdot \dfrac{c}{c^2 + (x-a)^2}, \quad -\infty < x < \infty$
Хи-квадрат	Rchisq	$f_{\chi^2(m)}(x) = \dfrac{1}{2^{\frac{m}{2}}\Gamma\left(\dfrac{m}{2}\right)} \cdot x^{\frac{m}{2}-1} e^{-\frac{x}{2}}, \ x > 0$
Экспоненциальное	Rexp	$p(x,\theta) = \theta\, e^{-\theta x} \ (x \ge 0)$, где $\theta > 0$ – параметр распределения
Фишера	rF	$f_{F(m_1,m_2)}(x) = \dfrac{\Gamma\left(\dfrac{m_1+m_2}{2}\right) m_1^{\frac{m_1}{2}} m_2^{\frac{m_2}{2}}}{\Gamma\left(\dfrac{m_1}{2}\right)\Gamma\left(\dfrac{m_2}{2}\right)} \cdot \dfrac{x^{\frac{m_1}{2}-1}}{(m_1 x + m_2)^{\frac{m_1+m_2}{2}}}$ при $0 \le x \le \infty$

Таблица 7.1. Законы распределения и функции Mathcad для создания векторов с заданными законами распределения (окончание)

Название распределения	Функция Mathcad	Функция плотности распределения
Гамма-распределение	rgamma	$\gamma_{(a,b)}(x)=\begin{cases}\dfrac{b^a}{\Gamma(a)}x^{a-1}e^{-bx} & \text{при } 0\le x\le\infty\\ 0 \text{ при } x<0\end{cases}$
Геометрическое	Rgeom	$f(x)=p(1-p)^{x-1}$
Логнормальное	Rlnorm	$f(x)=\dfrac{1}{\sqrt{2\pi}\cdot\sigma x}e^{\frac{(\ln x-\ln a)^2}{2\sigma^2}}$
Отрицательное биномиальное	rnbinom	$p_k=P\{X=k\}=C^k_{r+k-1}p^r(1-p)^k$ где $0<p<1,\ r>0$
Нормальное	Rnorm	$f(x)=\dfrac{1}{\sigma\sqrt{2\pi}}\cdot e^{-\frac{(x-a)^2}{2\sigma^2}}$, где a, σ – параметры распределения, $\sigma>0$
Пуассона	Rpois	$f(x)=\dfrac{\lambda^x\cdot e^{-\lambda}}{x!}$
Стьюдента	Rt	$f_t(x)=\dfrac{1}{\sqrt{\pi m}}\cdot\dfrac{\Gamma(\frac{m+1}{2})}{\Gamma(\frac{m}{2})}\cdot(1+\dfrac{x^2}{m}),\ (-\infty<x<\infty)$, m – число степеней свободы, Г – гамма-функция
Равномерное	Runif	$f_\xi(x)=\begin{cases}\dfrac{1}{b-a} & \text{при } a\le x\le b;\\ 0 \text{ при } x<a \text{ и } x>b\end{cases}$ [a, b] – отрезок распределения
Вейбулла	rweibull	$f_\xi(t)=\lambda_0\,\alpha t^{\alpha-1}\,e^{-\lambda 0\,t\hat{}\alpha},\ t\ge 0$

7.4.8. Примеры статистических вычислений в системе Mathcad

Средства системы Mathcad позволяют решать широкий круг статистических вычислений. Рассмотрим несколько примеров этого.

На рис. 7.12 показан фрагмент документа Mathcad с примерами построения графиков различных статистических функций и задания наборов чисел с различным распределением.

Следующая пара примеров иллюстрирует вычисление медианы для векторов, содержащих нечетное и четное число элементов:

```
V:=(1 2 3 4 5)          median(V)=3
V:=(1 2 3 4 5 6)        median(V)=3.5
```

ПРИМЕРЫ ПРИМЕНЕНИЯ СТАТИСТИЧЕСКИХ ФУНКЦИЙ

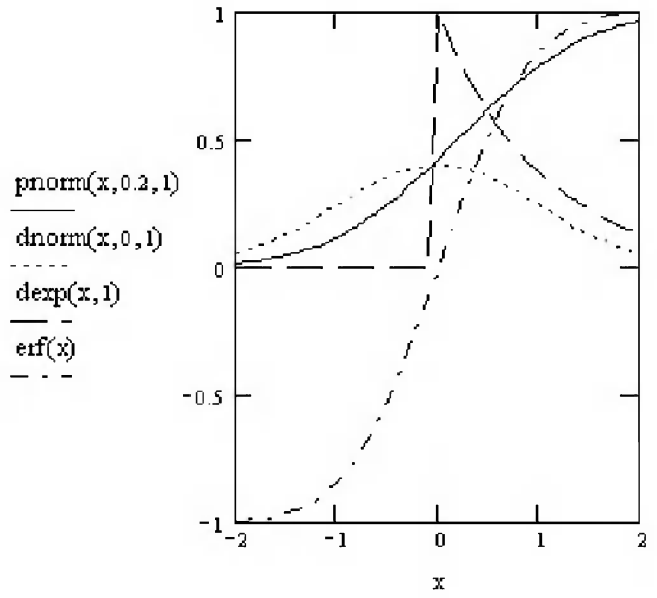
$x := -2, -1.9 .. 2$ $\quad V := rnorm(10,0,1)$ $\quad V1 := rexp(10,1)$

$V =$

	0
0	-0.439
1	-0.679
2	-0.473
3	-0.951
4	-1.686
5	0.044
6	-0.121
7	0.556
8	2.192
9	0.809

$V1 =$

	0
0	4.731
1	1.288
2	0.531
3	0.177
4	0.724
5	0.296
6	0.781
7	0.295
8	0.512
9	0.308



$qnorm(0.5,2,1) = 2$

$pnorm(2,2,1) = 0.5$

$erf(0.5) = 0.52$

Рис. 7.12. Примеры работы с функциями статистики системы Mathcad 12

А теперь рассмотрим решение задачи, которую мы уже рассматривали, – пусть в группе студентов 25 человек, а в году 365 дней. Какова вероятность того, что ни у кого из них дни рождения не совпадут? Решение задачи разными способами дано на рис. 7.13.

Вначале представлена попытка решить задачу «в лоб», просто вычислив выражение для вероятности, содержащее факториал числа 1000. Как и следовало ожидать, выражение на экране дисплея покраснело от стыда (на рисунке этого, увы,

$$p := \frac{365!}{(340! \cdot 25! \cdot 365^{25})}$$

Found a number with a magnitude greater than 10^307
while trying to evaluate this expression.

$$\frac{365!}{(340! \cdot 25! \cdot 365^{25})} \rightarrow \frac{4346431391691900527592549036258453258}{15631431481949933625718603881602115365807218581438064575195312 5}$$

$$\frac{365!}{(340! \cdot 25! \cdot 365^{25})}\ \text{float},10 \rightarrow .2780571566\text{e-}25$$

Рис. 7.13. Пример решения статистической задачи на несовпадение дат рождения

не видно), и при наведении на него мыши появился комментарий, свидетельствующий о том, что превышено максимальное значение чисел в ходе вычислений – оно оказалось больше, чем 10^{307}.

Затем предпринята попытка вычислить нужное выражение с применением оператора символьного вывода →. На сей раз что-то вычислено, но результат представлен в виде рационального чисеа. У истинных математиков это должно вызвать бурный восторг, но большинству «простых смертных» хотелось бы получить чего-нибудь попроще и понятнее. Воспользовавшись символьной функцией float, мы, наконец, получим обычный результат, кстати, полностью совпадающий с полученным ранее в системе Maple 9.5. Удивляться тут совсем не к чему – мы уже знаем, что в Mathcad используется ядро символьной математики системы Maple.

Найти вероятность того, что при 50 и 200 бросаниях монеты число падений на каждую сторону будет одинаково (вероятность p = 0.5). Эта задача решается с применением функции дискретного биноминального распределения dbinom:

dbinom(25,50,0.5)=0.112 **dbinom(100,200,0.5)=0,056**

Квантили биноминального распределения позволяют решить следующую задачу: сколько раз за 100 бросков выпадет орел, если вероятность этого события равна 0,25:

qbinom(0.25,100,0.5)=47

Геометрическое распределение часто используется для оценки попадания в цель при стрельбе, например из пушки. Пусть вероятность попадания в цель равна 0,15. Какова вероятность того, что цель будет поражена при десятом и сотом выстреле. Решение этой задачи следующее:

dgeom(5,0.15)=0.067 **dgeom(10,0.15)=0.03**

Какова вероятность поражения цели до 5 и 10 выстрела, если вероятность поражения цели равна 0,15. Ответ таков:

pgeom(5,0.15)=0.623 **pgeom(10,0.15)=0.833**

Читатель может найти множество и других примеров выполнения статистических расчетов.

7.4.9. Функции сглаживания данных

Данные большинства экспериментов имеют случайные составляющие, поэтому часто возникает необходимость статистического сглаживания данных. Система Mathcad не только имеет функции для реализации наиболее известных методов сглаживания, но и позволяет реализовать некоторые частные алгоритмы сглаживания, нередко хорошо проясняющие смысл этой операции.

Ряд функций Mathcad предназначен для выполнения операций сглаживания данных различными методами. В их названии имеется слово `smooth` (гладкий):

- `medsmooth(VY,n)` – для вектора с `m` действительными числами возвращает `m`-мерный вектор сглаженных данных по методу скользящей медианы,

параметр n задает ширину окна сглаживания (n должно быть нечетным числом, меньшим m);

- ksmooth(VX,VY,b) – возвращает n-мерный вектор сглаженных данных VY, вычисленных на основе распределения Гаусса. VX и VY – n-мерные векторы действительных чисел. Параметр b (полоса пропускания) задает ширину окна сглаживания (b должно в несколько раз превышать интервал между точками по оси x);
- supsmooth(VX,VY) – возвращает n-мерный вектор сглаженных данных VY, вычисленных на основе использования процедуры линейного сглаживания методом наименьших квадратов по правилу k-ближайших соседей с адаптивным выбором k. VX и VY – n-мерные векторы действительных чисел. Элементы вектора VX должны идти в порядке возрастания.

На рис. 7.14 показан фрагмент документа Mathcad с примерами применения функций сглаживания, указанных выше. Из этих функций наиболее сильное сглаживание обеспечивает функция supsmooth. Однако при ней есть риск потери быстрых деталей сглаживаемой зависимости (в приведенных примерах для сгла-

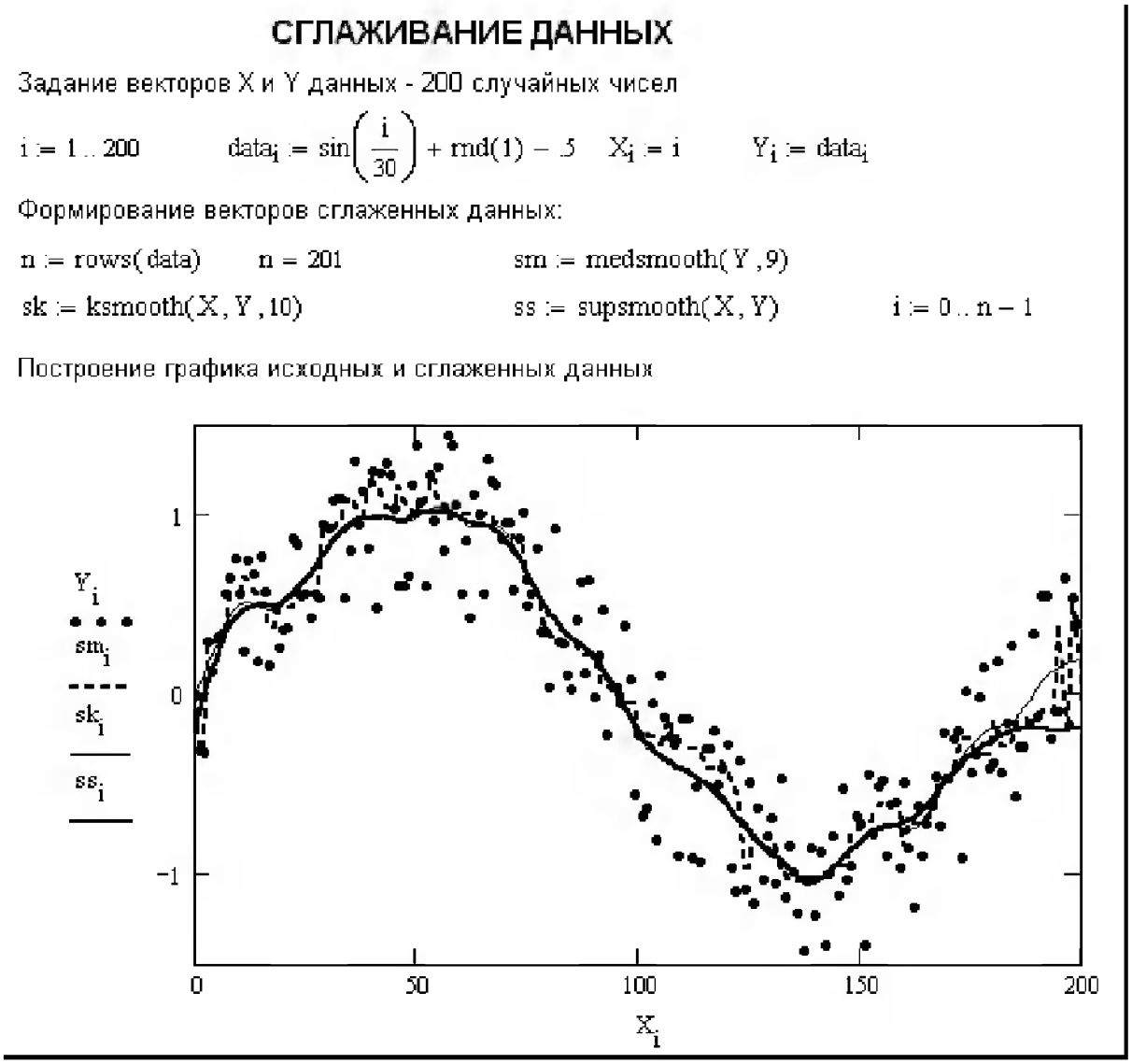


Рис. 7.14. Сглаживание данных с помощью функций сглаживания

живания используется синусоидальная функция с шумами, созданными генератором случайных чисел). Все функции сглаживания возвращают вектор сглаженных значений VY, размер которого соответствует размеру вектора VY.

В показанном на рисунке фрагменте документа для получения исходного массива точек зашумленной синусоиды используется генератор случайных чисел с равномерным распределением. При этом как расположение точек, так и вид кривых сглаживания могут отличаться от приведенных при иной настройке генератора случайных чисел. Но в любом случае будет хорошо видно, что кривые сглаживания намного более гладкие, чем кусочно-линейная функция, соединяющая точки друг с другом в последовательном порядке. При сглаживании бывает полезно применение функции `sort(Y)`, сортирующей данные векторов, что иногда уменьшает погрешности численного алгоритма сглаживания. Рекомендуется просмотреть примеры применения функций сглаживания в справке Mathcad.

7.4.10. Линейное сглаживание по пяти точкам

Иногда полезно знать, по каким формулам конкретно выполняется сглаживание. Рисунок 7.15 иллюстрирует решение классической задачи линейного сглаживания по пяти точкам. Термин «линейное» означает, что при сглаживании используется линейная функция. Приведены формулы сглаживания как для «центральных» точек, так и для крайних с учетом особой обработки краевых точек.

Линейное сглаживание по пяти точкам является частным случаем сглаживания по методу скользящей медианы – при нем каждая точка данных обрабатывается как среднее значение ее и ближайших точек. При этом происходит перемещение ансамбля обрабатываемых точек от начала вектора с ними к его концу.

7.4.11. Нелинейное сглаживание по семи точкам

Нелинейное сглаживание по семи точкам реализует документ, показанный на рис. 7.16. Увеличение числа обрабатываемых точек позволяет повысить степень сглаживания.

ЛИНЕЙНОЕ СГЛАЖИВАНИЕ ПО ПЯТИ ТОЧКАМ

$i := 0..100$ $y_i := \sin\left(\frac{i}{15}\right) + rnd(1) - .5$ Создание вектора исходных данных (101 случайное число)

$n := length(y) - 1$

Формулы линейного сглаживания по 5 точкам:

$ys_0 := 0.2\cdot(3\cdot y_0 + 2\cdot y_1 + y_2 - y_4)$ $ys_1 := 0.1\cdot(4\cdot y_0 + 3\cdot y_1 + 2\cdot y_2 + y_3)$

$ys_{n-1} := 0.1\cdot(y_{n-3} + 2\cdot y_{n-2} + 3\cdot y_{n-1} + 4\cdot y_n)$

$ys_n := 0.2\cdot(3\cdot y_n + 2\cdot y_{n-1} + y_{n-2} - y_{n-4})$

$i := 2..n-2$ $ys_i := 0.2\cdot(y_{i-2} + y_{i-1} + y_i + y_{i+1} + y_{i+2})$ $i := 0..n$

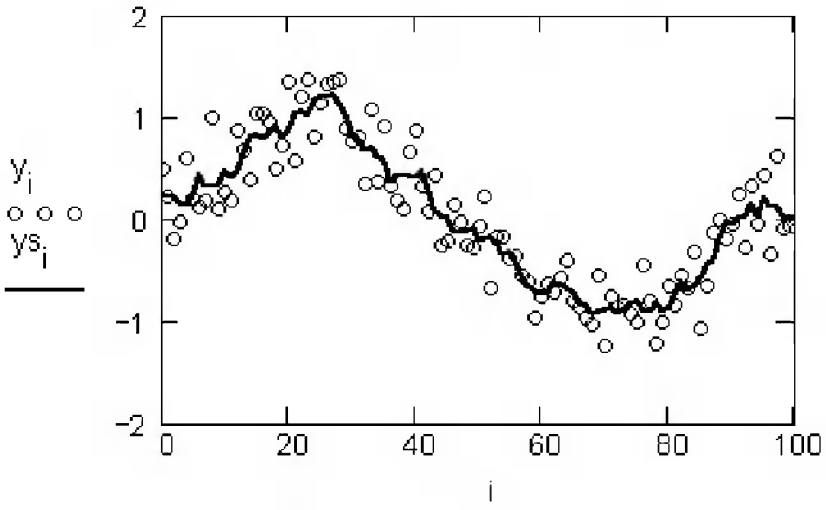


График показывает исходные точки и сглаживающую их функцию

Рис. 7.15. Линейное сглаживание по пяти точкам

Результат линейного сглаживания зашумленной синусоиды показан на рис. 7.17. Сравнение его с рис. 7.15 показывает, что нелинейное сглаживание несколько лучше, чем линейное. Но чуда нет!

Вопрос о повторном применении сглаживания оказался спорным. Существует мнение, что повторное сглаживание чрезмерно искажает исходные данные. В дальнейшем мы рассмотрим комбинированное сглаживание, в котором используется возможность сглаживания с помощью вейвлетов.

НЕЛИНЕЙНОЕ СГЛАЖИВАНИЕ ПО СЕМИ ТОЧКАМ

$i := 0..100 \quad y_i := \sin\left(\frac{i}{15}\right) + rnd(1) - 0.5$ Задание вектора исходных данных (101 случайное число)

$n := length(y) - 1$

Формулы нелинейного сглаживания по 7 точкам

$$ys_0 := \frac{39 \cdot y_0 + 8 \cdot y_1 - 4 \cdot (y_2 + y_3 - y_4) + 4 \cdot y_5 - 2 \cdot y_6}{42}$$

$$ys_1 := \frac{8 \cdot y_0 + 19 \cdot y_1 + 16 \cdot y_2 + 6 \cdot y_3 - 4 \cdot y_4 - 7 \cdot y_5 + 4 \cdot y_6}{42}$$

$$ys_2 := \frac{-4 \cdot y_0 + 16 \cdot y_1 + 19 \cdot y_2 + 12 \cdot y_3 + 2 \cdot y_4 - 4 \cdot y_5 + y_6}{42}$$

$$ys_{n-2} := \frac{y_{n-6} - 4 \cdot y_{n-5} + 2 \cdot y_{n-4} + 12 \cdot y_{n-3} + 19 \cdot y_{n-2} + 16 \cdot y_{n-1} - 4 \cdot y_n}{42}$$

$$ys_{n-1} := \frac{4 \cdot y_{n-6} - 7 \cdot y_{n-5} - 4 \cdot y_{n-4} + 6 \cdot y_{n-3} + 16 \cdot y_{n-2} + 19 \cdot y_{n-1} + 8 \cdot y_n}{42}$$

$$ys_n := \frac{-2 \cdot y_{n-6} + 4 \cdot y_{n-5} + y_{n-4} - 4 \cdot y_{n-3} - 4 \cdot y_{n-2} + 8 \cdot y_{n-1} + 39 \cdot y_n}{42}$$

$i := 3..n-3 \quad ys_i := \frac{7 \cdot y_i + 6 \cdot (y_{i+1} + y_{i-1}) + 3 \cdot (y_{i+2} + y_{i-2}) - 2 \cdot (y_{i+3} + y_{i-3})}{21}$

$i := 0..n$

Рис. 7.16. Нелинейное сглаживание по семи точкам (алгоритм)

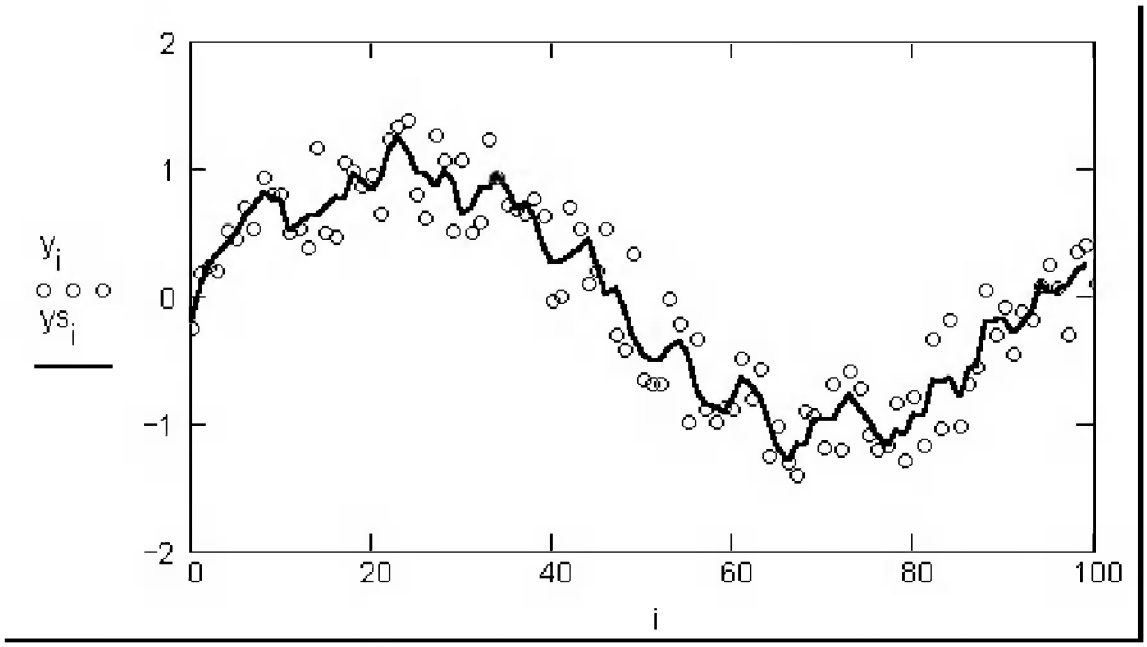


Рис. 7.17. Пример нелинейного сглаживания по семи точкам

7.5. Статистические функции Derive и MuPAD

Малые СКМ Derive и MuPAD также имеют наборы статистических функций, хотя и менее полные, чем у других СКМ, описанные выше.

7.5.1. Функция генерации случайных чисел в Derive

Derive имеет достаточный набор функций для проведения статистических расчетов и типовой статистической обработки данных. Так, в Derive есть функция генерации случайных чисел с равномерным распределением RANDOM (n), которая в зависимости от значения аргумента n выполняет следующие действия:

n>1 – возвращает случайное целое в интервале [0,n];

n=1 – возвращает случайное число в интервале [0,1];

n<1 – устанавливает счетчик генератора случайных чисел в –n;

n=0 – устанавливает счетчик генератора случайных чисел по текущему времени (прошедшему с момента загрузки Derive) в сотых долях секунды.

7.5.2. Функции ошибок в Derive

Функции ошибок в Derive представлены ниже:

ERF (x) – функция ошибок;
ERF (x, y) – функция ошибок общего вида;
ERFC (x) – дополнительная функция ошибок;
NORMAL (x, m, s) – функция нормального распределения со средним m и среднеквадратическим s;
NORMAL (z) – интегральное распределение по z.

Эти функции возвращают единственное значение и имеют аргументы скалярного типа:

```
1: «Вычисление функций ошибки»
2: ERF (0.5)
3: 0.520499
4: ERF (0.5, 2)
5: 0.474822
6: «Вычисление дополнительной функции ошибок»
7: ERFC(0.5)
8: 0.479500
9: «Проверка равенства erf(x)+erfc(x)=1»
10: ERF (X) + ERFC(X)
11: 1
12: «Вычисление функции нормального распределения»
```

```
13: NORMAL (0.5, 1, 1)
14: 0.308537
```

Довольно широкая группа статистических функций обеспечивает вычисление статистических параметров групповых данных. Данные при этом задаются в виде векторов или просто в виде их перечисления. Ниже представлены основные из этих функций.

7.5.3. Основные статистические функции Derive

Статистические функции в Derive представлены небольшим набором:

AVERAGE (z1, …, zn) – арифметическое среднее;
RMS (z1, …, zn) – среднеквадратическое значение;
VAR (z1, …, zn) – дисперсия;
STDEV (z1, …, zn) – среднеквадратическое отклонение;
FIT (m) – обработка матрицы m методом наименьших квадратов;
FIT (v,A) – многомерная линейная регрессия вектора **v**, задающего аргументы и по матрице данных **A**.

В представленном ниже примере иллюстрируется действие большинства из этих функций:

```
1:   "Статистические функции"
2:   "Используйте команду Simplify"
3:   AVERAGE (a, b, c)
      a + b + c
4:    ------
          3
5:   AVERAGE (1, 2, 3)
6:   2
7:   RMS (a, b)
                        2    2
      SQRT (2) SQRT (a  + b )
8:    --------------
                 2
9:   VAR (a, b)
              2
       (a - b)
10:   -----
         4
11:  RMS (1, 1.1, 1.2, 1.3, 1.4)
      SQRT (146)
12:   ------
          10
13:  VAR (1, 1.1, 1.2, 1.3, 1.4)
      1
14:   --
      50
```

Дополнительные вероятностные функции, широко применяемые в специальных статистических расчетах, заданы в файле probabil.mth:

- **POCHHAMMER (a,x)** – функция, вычисляющая GAMMA(a+x)/GAMMA(a);
- **PSI (z)** – пси-функция DIF(GAMMA(z),z)/GAMMA(z) для модуля аргумента z, не превосходящего pi/2;
- **POLYGAMMA (n,z,m)** – полигамма-функция DIF(PSI(z),z,n), где z/=0, -1,-2,..., вычисляемая по разложению в ряд;
- **INCOMPLETE_GAMMA (z,w)** – функция P(z,w)=INT(#e^-t*t^(z-1),t,0,w)/GAMMA(z), E(z)>0;
- **INCOMPLETE_GAMMA_SERIES** – функция P(z,w), вычисляемая по (z,w,m) разложению в ряд;
- **BETA (z,w)** – бета-функция B(z,w)=GAMMA(z)*GAMMA(w)/GAMMA(z+w);
- **INCOMPLETE_BETA (x,z,w)** – функция Bx(z,w) = INT(t^(z-1)*(1-t)^(w-1),t,0,x);
- **POISSON_DENSITY (k,t)** – плотность вероятности распределения Пуассона #e^-t*t^k/k!;
- **POISSON_DISTRIBUTION(k,t)** – распределение Пуассона, то есть функция SUM (#e^-t*t^j/j!,j,0,k);
- **BINOMIAL_DENSITY (k,n,p)** – функция COMB(n,k)*p^k*(1-p)^(n-k);
- **BINOMIAL_DISTRIBUTION(k,n,p)** – биномиальное распределение, то есть функция SUM(COMB(n,j)*p^j*(1-p)^(n -j),j,0,MIN(k,n));
- **HYPERGEOMETRIC_DENSITY (k,n,p)** – гипергеометрическое распределение COMB(m,k)*COMB(j-m,n-k)/COMB(j,n);
- **HYPERGEOMETRIC_DISTRIBUTION (k,n,p)** – функция кумулятивного гипергеометрического распределения;
- **STUDENT (t,v)** – кумулятивная плотность распределения Стьюдента A(t|v);
- **F_DISTRIBUTION (f,v1,v2)** – F-функция кумулятивного распределения P(f|v1,v2);
- **CHI_SQ (u,v)** – функция Chi-квадрат распределения P(u|v), u = Chi^2.

В примерах ниже поясняется применение этих функций:

```
1:   "Вычисление добавочных функций распределения"
2:   POCHHAMMER (2, 0.5)
3:   1.32934
4:   PSI (1)
5:   -0.577215
6:   POLYGAMMA (2, 0.5, 5)
7:   -16.8289
8:   POISSON_DENSITY (k, t)
        k LN (t) - t
      #e
9:    --------
            k!
```

```
10:  POISSON_DENSITY (2, 0.5)
11:  0.0758163
12:  HYPERGEOMETRIC_DENSITY (k, n, m, j)
                n! m! (j − n)! (j − m)!
13:   ----------------------
       (j + k − m − n)! j! k! (m − k)! (n − k)!
14:  HYPERGEOMETRIC_DENSITY (1, 2, 3, 4)
15:  0.5
16:  STUDENT (0.5, 1)
17:  0.295169
18:  F_DISTRIBUTION (0.5, 1, 2)
19:  0.552786
20:  CHI_SQ (1, 1)
21:  0.682690
```

Обширными возможностями по статистической обработке данных – регрессии – обладает функция **FIT**. Она была рассмотрена ранее.

> ***Внимание!*** *Система Derive явно не претендует на применение для серьезных статистических расчетов. Имеющиеся в ней средства носят скорее учебный, чем серьезный практический характер. Видимо, это вполне правомерно, учитывая учебную направленность данной системы.*

7.5.4. Численное дифференцирование со сглаживанием в Derive

Довольно оригинально решена в Derive задача сглаживания. Наиболее распространено решение этой задачи применительно к операции численного дифференцирования данных, содержащих случайные ошибки. Поэтому функции сглаживания и численного дифференцирования в Detive размещены в библиотечном файле numeric.mth. Он задает определения следующих функций:

- **DIF_NUM(y,x,x0,h)** – дает первую частную производную y по x в точке $x0$ при шаге изменения $x - h$;
- **DIF2_NUM(y,x,x0,h)** – дает вторую частную производную y по x;
- **SMOOTH_VECTOR(v)** – дает сглаженную копию вектора **v** (каждый элемент есть среднее между ним и двумя соседними элементами, крайние элементы сохраняются);
- **SMOOTH_COLUMN(A,j)** – дает матрицу **A** со сглаженным столбцом j;
- **DIF_DATA(A)** – дает сглаженную первую производную для двухстолбцовой матрицы данных **A**;
- **DIF2_DATA(A)** – дает сглаженную вторую производную матрицы A;
- **INT_DATA(A)** – дает сглаженную производную матрицы **A** с применением правила трапеций.

Ниже представлено решение задач с данными функциями:

```
1:   "Численное дифференцирование и сглаживание"
2:   DIF_NUM (SIN (x), x, 1, 0.001)
3:   0.540290
4:   DIF2_NUM (SIN (x), x, 1, 0.001)
5:   -0.866002
6:   SMOOTH_VECTOR [0, 1.1, 2.1, 2.8, 4.1, 5.2]
7:   [0, 1.06666, 2, 3, 4.03333, 5.2]
                  //  0  \   \
                  ||     |   |
                  || 1.1 |   |
                  ||     |   |
                  || 2.1 |   |
8:   SMOOTH_COLUMN ||     |, 1|
                  || 2.8 |   |
                  ||     |   |
                  || 4.1 |   |
                  ||     |   |
                  \\ 5.2 /   /
     /    0    \
     |         |
     | 1.06666 |
     |         |
     |    2    |
9:   |         |
     |    3    |
     |         |
     | 4.03333 |
     |         |
     \   5.2   /
               /         / 1  1 \\
               |         |      ||
10:  INT_DATA |DIF_DATA | 2  2 ||
               |         |      ||
               \         \ 3  3 //
     / 1  0 \
     |      |
11:  | 2  1 |
     |      |
     \ 3  2 /
```

7.5.5. Генерация случайных чисел в системе MuPAD

В MuPAD есть несколько форм записи функции генерации случайных чисел:

- **random()** – генерирует случайные целые числа с 12 цифрами;
- **random(m..n)** – генерирует случайные числа в отрезке [m, n];

- **random(n)** – генерирует случайные числа в отрезке [0..n-1].
- **float@random(0..10^10)/10^10** – генерирует вещественные случайные числа в отрезке [0, 1].

В ядре системы MuPAD статистические функции, за исключением функции генерации случайных чисел **random**, вообще отсутствуют. Они включены в библиотеку stats. Ниже показаны примеры вызова некоторых из этих функций:

- **stats::mean(V)** – среднее значение для данных вектора **V**;
- **stats::variance(V)** – дисперсия для данных вектора **V**;
- **stats::stdev(V)** – стандартное отклонение для данных вектора **V**.

Применение этих функций очевидно. С помощью команд **info(stats)** или **?stats** можно просмотреть всю библиотеку **stats**. Помимо указанных, есть функции нормального и хи-квадрат распределения, вычисления медианы и проведения статистических тестов.

> ***Внимание!*** *MuPAD обладает самыми скромными возможностями в части статистических расчетов. Они представлены функциями библиотеки stats. Можно сделать вывод, что средства этой системы в области статистики носят скорее познавательный, чем практический характер. Хотя, используя приличные математические возможности системы, можно самостоятельно задать практически любую статистическую функцию.*

Глава 8

Решение задач линейной алгебры и оптимизации

Задачи линейной алгебры и оптимизации – одни из самых массовых в науке, технике и образовании. Им и посвящена эта глава. В ней даны основные определения линейной алгебры, основы работы с массивами, векторами и матрицами, описаны функции для работы с векторами и матрицами и средства для решения систем линейных уравнений [127–138]. Дано также описание функций оптимизации, основанных на средствах линейной алгебры.

8.1. Основные определения линейной алгебры

8.1.1. Определение векторов, матриц и их характеристик

Прежде чем перейти к рассмотрению решения задач *линейной алгебры*, рассмотрим краткие определения, относящиеся к ней.

Матрица – прямоугольная двумерная таблица, содержащая m строк и n столбцов элементов, каждый из которых может быть представлен числом, константой, переменной, символьным или математическим выражением (расширительная трактовка матрицы).

Блок-матрица – матрица, составленная из меньших по размеру матриц, также можно представить как матрицу, каждый элемент которой – матрица. Частным случаем является блок-диагональная матрица – блок-матрица, элементы-матрицы которой вне диагонали – нуль-матрицы.

Единичная матрица – это квадратная матрица, у которой диагональные элементы равны 1, а остальные элементы равны 0. Ниже представлена единичная матрица размера 4×4:

$$\mathbf{E}=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}.$$

Идемпотентная матрица – матрица, отвечающая условию $\mathbf{P}^2 = \mathbf{P}$.

Комплексно-сопряженная матрица – матрица $\bar{\mathbf{A}}$, полученная из исходной матрицы $\mathbf{A}$ заменой ее элементов на комплексно-сопряженные.

Квадратная матрица – матрица, у которой число строк m равно числу столбцов n. Пример квадратной матрицы размера 3×3:

$$\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{bmatrix}.$$

Кососимметрическая матрица – матрица, отвечающая условию $\mathbf{A}^{\mathrm{T}} = -\mathbf{A}$.

Нуль-матрица – матрица, все элементы которой равны 0.

Обратная матрица – это матрица $\mathbf{M}^{-1}$, которая, будучи умноженной на исходную квадратную матрицу **M**, дает единичную матрицу **E**.

Ортогональная матрица **Q** – это матрица, удовлетворяющая условию $\mathbf{Q}^{\mathrm{T}} \cdot \mathbf{Q} = \mathbf{E}$, где **E** – единичная матрица. Ортогональная матрица удовлетворяет также условию $\mathbf{A}^{\mathrm{T}} = \mathbf{A}^{-1}$.

Симметрическая матрица – матрица, отвечающая условию $\mathbf{A}^{\mathrm{T}} = \mathbf{A}$.

Сингулярная (*вырожденная*) матрица – квадратная матрица, у которой детерминант (определитель) равен 0. Такая матрица обычно не упрощается при символьных вычислениях. Линейные уравнения с *почти сингулярными* матрицами могут давать большие погрешности при решении.

Транспонированная матрица – матрица, у которой столбцы и строки меняются местами, то есть элементы транспонированной матрицы удовлетворяют условию $\mathbf{A}^{\mathrm{T}}_{i,j} = \mathbf{A}_{j,i}$, где i и j – индексы элементов. Приведем простой пример.

Исходная матрица:

$$\mathbf{A} = \begin{bmatrix} a & b & c \\ d & e & f \\ i & k & l \end{bmatrix}.$$

Транспонированная матрица:

$$\mathbf{A}^{\mathrm{T}} = \begin{bmatrix} a & d & i \\ b & e & k \\ c & f & l \end{bmatrix}.$$

Треугольная матрица – это матрица, у которой по одну из сторон диагонали элементы есть нули. Различают верхнюю и нижнюю треугольные матрицы.

Эрмитова матрица – матрица **A**, удовлетворяющая условию $\overline{\mathbf{A}} = \mathbf{A}^{\mathrm{T}}$.

Диагональ матрицы – расположенные диагонально элементы $\mathbf{A}_{i,i}$ матрицы **A**. В приведенной ниже матрице элементы диагонали представлены заглавными буквами:

$$\mathbf{A} = \begin{bmatrix} A & b & c \\ d & E & f \\ i & k & L \end{bmatrix}.$$

Обычно указанную диагональ называют *главной* диагональю – для матрицы **A**, приведенной выше, это диагональ с элементами A, E и L. Иногда вводят понятия *поддиагоналей* (элементы d и k) и *наддиагоналей* (элементы b и f). Матрица, все элементы которой, расположенные кроме как на диагонали, поддиагонали и наддиагонали, равны нулю, называется *ленточной*.

Ранг матрицы – наибольший из порядков, отличных от нуля миноров квадратной матрицы.

Сингулярные значения матрицы **A** – квадратные корни из собственных значений матрицы transpose(**A**)·**A**, где transpose(**A**) – транспонированная матрица **A** (см. ее определение ниже).

След матрицы – сумма диагональных элементов матрицы.

Ступенчатая форма матрицы соответствует условиям, когда первый ненулевой элемент в каждой строке есть 1 и первый ненулевой элемент каждой строки появляется справа от первого ненулевого элемента в предыдущей строке, то есть все элементы ниже первого ненулевого в строке – нули.

Определитель матрицы – это многочлен от элементов квадратной матрицы, каждый член которого является произведением n элементов, взятых по одному из каждой строки и каждого столбца со знаком произведения, заданным четностью перестановок:

$$\det \mathbf{A} = \sum a_{1j} (-1)^{j+1} \mathbf{M}_1^{<j>},$$

где $\mathbf{M}_1^{<j>}$ – определитель матрицы порядка $n - 1$, полученной из матрицы **A** вычеркиванием первой строки и j-го столбца. В таком виде определитель (он же детерминант) легко получить в символьных вычислениях. В численных расчетах мы будем подразумевать под определителем численное значение этого многочлена.

Матрица в целой степени – квадратная матрица в степени n (n – целое неотрицательное число), определяемая следующим образом: $\mathbf{M}^0 = \mathbf{E}$, $\mathbf{M}^1 = \mathbf{M}$, $\mathbf{M}^2 = \mathbf{M}\cdot\mathbf{M}$, ..., $\mathbf{M}^n = \mathbf{M}^{n-1}\cdot\mathbf{M}$.

Собственный вектор квадратной матрицы **A** – любой вектор $\mathbf{x} \in V^n$, $\mathbf{x} \neq \mathbf{0}$, удовлетворяющий уравнению $\mathbf{Ax} = \gamma\mathbf{x}$, где γ – некоторое число, называемое *собственным значением* матрицы **A**.

Характеристический многочлен матрицы – определитель разности этой матрицы и единичной матрицы, умноженный на переменную многочлена – $|\mathbf{A} - \gamma\mathbf{E}|$.

Собственные значения матрицы – корни ее характеристического многочлена.

Норма – обобщенное понятие абсолютной величины числа.

Норма трехмерного вектора $\|\mathbf{x}\|$ – его длина.

Норма матрицы – значение $\sup(\|\mathbf{Ax}\|/\|\mathbf{x}\|)$.

L-норма матрицы **A** – число $\|\mathbf{A}\|_L = \max_j \sum_i |A_{i,j}|$.

Приведенный список определений является неполным. Читатель, серьезно занятый решением задач линейной алгебры, может пополнить его из [127–138]. Это, в частности, относится к определению ряда специальных матриц.

8.1.2. Системы линейных уравнений и их матричная форма

Как известно, обычная *система линейных уравнений* имеет вид:

$$\begin{aligned}
&a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,n}x_n = b_1\\
&a_{2,1}x_1 + a_{2,2}x_2 + \ldots + a_{2,n}x_n = b_2\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&a_{n,1}x_1 + a_{n,2}x_n + \ldots + a_{n,n}x_n = b_n
\end{aligned}$$

Здесь $a_{1,1}, a_{1,2}, ..., a_{n,n}$ – коэффициенты, образующие матрицу **A** и могущие иметь действительные или комплексные значения, $x_1, x_2, ..., x_n$ – неизвестные, образующие вектор **X**, и $b_1, b_2, ..., b_n$ – свободные члены (действительные или комплексные), образующие вектор **B**. Эта система может быть представлена в матричном виде как **AX = B**, где **A** – матрица коэффициентов уравнений, **X** – искомый вектор неизвестных и **B** – вектор свободных членов. Из такого представления системы линейных уравнений вытекают различные способы ее решения: например, **X = B/A** (с применением матричного деления) или $\mathbf{X} = \mathbf{A}^{-1}\mathbf{B}$ (с инвертированием матрицы A) и т. д.

8.1.3. Матричные разложения

В ходе решения задач линейной алгебры часто приходится использовать *матричные разложения*, нередко резко упрощающие решения систем линейных уравнений. Отметим некоторые из наиболее распространенных матричных разложений, которые реализованы в большинстве СКА и СКМ.

LU-разложение, называемое также треугольным разложением, соответствует матричному выражению вида $\mathbf{P} \cdot \mathbf{A} = \mathbf{L} \cdot \mathbf{U}$, где **L** – нижняя и **U** – верхняя треугольные матрицы. Все матрицы в этом выражении квадратные.

QR-разложение имеет вид $\mathbf{A} = \mathbf{Q} \cdot \mathbf{R}$, где **Q** – ортогональная матрица, а **R** – верхняя треугольная матрица. Это разложение часто используется при решении любых систем линейных уравнений, в том числе переопределенных и недоопределенных и с прямоугольной матрицей.

Разложение Холецкого $\mathbf{A} = \mathbf{L} \cdot \mathbf{L}^{\mathrm{T}}$ применяется к симметричной матрице **A**, при этом **L** – треугольная матрица.

Сингулярное разложение матрицы **A** размера $M \times N$ ($M \le N$) определяется выражением $\mathbf{A} = \mathbf{U} \cdot \mathbf{s} \cdot \mathbf{VT}$, где **U** и **V** – ортогональные матрицы размера $N \times N$ и $M \times M$ соответственно, а **s** – диагональная матрица с сингулярными числами матрицы **A** на диагонали.

8.1.4. Элементы векторов и матриц в Maple

Элементы векторов и матриц в Maple являются *индексированными переменными*, то есть место каждого элемента вектора определяется его индексом, а у матрицы – двумя индексами. Обычно их обобщенно обозначают как i (номер строки матрицы или порядковый номер элемента вектора) и j (номер столбца матрицы). Допустимы операции вызова нужного элемента и присваивания ему нового значения:

- `V[i]` – вызов i-го элемента вектора `V`;
- `M[i,j]` – вызов элемента матрицы `M`, расположенного на i-й строке в j-м столбце;
- `V[i]:=x` – присваивание нового значения `x` i-му элементу вектора `V`;
- `M[i,j]:=x` – присваивание нового значения `x` элементу матрицы `M`.

8.1.5. Преобразование списков в векторы и матрицы

Векторы и матрицы хотя и похожи на списки, но не полностью отождествляются с ними. В этом можно убедиться с помощью следующих примеров, в которых функция `type` используется для *контроля типов* множественных объектов (векторов и матриц):

```
> M1:=[1,2,3,4];
```

$$M1 := [1, 2, 3, 4]$$

```
> type(M1,vector);
```

$$false$$

```
> V:=convert(M1,vector);
```

$$V := [1, 2, 3, 4]$$

```
> type(V,vector);
```

$$true$$

```
> M2:=[[1,2],[3,4]];
```

$$M2 := [[1, 2], [3, 4]]$$

```
> type(M2,matrix);
```

$$false$$

```
> M:=convert(M2,matrix);
```

$$M := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
> type(M,matrix);
```

$$true$$

Используя функцию преобразования данных `convert`, можно преобразовывать одномерные списки в векторы, а двумерные – в матрицы. Функция `type` используется в следующих формах:

- `type(V,vector)` – тестирует аргумент `V` и возвращает `true`, если `V` – вектор, и `false` в ином случае;
- `type(M,matrix)` – тестирует аргумент `M` и возвращает `true`, если `M` – матрица, и `false` в ином случае.

Здесь параметры `vector` и `matrix` используются для указания того, какой тип объекта проверяется. Обратите внимание на то, что матрицы отображаются иначе, чем двумерные списки, – без двойных квадратных скобок. Отображение вектора подобно отображению одномерного списка, поэтому здесь особенно важен контроль типов данных.

8.1.6. Операции с векторами в Maple

Важное достоинство систем компьютерной алгебры, к которым относится и Maple, заключается в возможности выполнения аналитических (символьных) *операций* над векторами и матрицами. Перед проведением символьных операций с векторами и матрицами рекомендуется очистить память от предшествующих

определений с помощью команды `restart`. Если какие-то элементы векторов или матриц были ранее определены, это может привести к очень сильным искажениям вида конечных результатов. Очистка памяти устраняет возможность ошибок такого рода.

Приведем примеры операций над векторами:

```
> V:=array(1..4,[1,2,3,4]);
```

$$V := [1, 2, 3, 4]$$

```
> [V[1],V[2],V[4]];
```

$$[1, 2, 4]$$

```
> V[1]:=a:V[3]:=b:
> evalm(V);
```

$$[a, 2, b, 4]$$

```
> evalm(V+2);
```

$$[a + 2, 4, b + 2, 6]$$

```
> evalm(2*V);
```

$$[2a, 4, 2b, 8]$$

```
> evalm(V**V);
```

$$[a, 2, b, 4]^V$$

```
> evalm(a*V);
```

$$[a^2, 2a, ab, 4a]$$

В этих примерах используется функция `evalm(M)`, осуществляющая вычисление матрицы или вектора `M`.

8.1.7. Операции над матрицами с численными элементами

Над матрицами с численными элементами в Maple можно выполнять разнообразные операции. Ниже приведены основные из них:

```
> M:=array(1..2,1..2,[[1,2],[3,4]]);
```

$$M := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
> evalm(2*M);
```

$$\begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}$$

```
> evalm(2+M);
```

$$\begin{bmatrix} 3 & 2 \\ 3 & 6 \end{bmatrix}$$

```
> evalm(M^2);
```

$$\begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}$$

```
> evalm(M^(-1));
```

$$\begin{bmatrix} -2 & 1 \\ \frac{3}{2} & \frac{-1}{2} \end{bmatrix}$$

```
> evalm(M-M);
```

$$0$$

```
> evalm(M+M);
```

$$\begin{bmatrix} 3 & 2 \\ 3 & 6 \end{bmatrix}$$

```
> evalm(M*M);
```

$$\begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}$$

```
> evalm(M/M);
```

$$1$$

```
> evalm(M^0);
```

$$1$$

Рекомендуется внимательно изучить эти примеры и попробовать свои силы в реализации простых матричных операций.

8.1.8. Символьные операции с матрицами в Maple

Одной из привлекательных возможностей СКА является возможность проведения символьных операций с матрицами. Ниже представлены примеры символьных операций, осуществляемых над квадратными матрицами одного размера в системе Maple:

```
> M1:=array(1..2,1..2,[[a1,b1],[c1,d1]]);
```

$$M1 := \begin{bmatrix} a1 & b1 \\ c1 & d1 \end{bmatrix}$$

```
> M2:=array(1..2,1..2,[[a2,b2],[c2,d2]]);
```

$$M2 := \begin{bmatrix} a2 & b2 \\ c2 & d2 \end{bmatrix}$$

```
> evalm(M1+M2);
```

$$\begin{bmatrix} a1+a2 & b1+b2 \\ c1+c2 & d1+d2 \end{bmatrix}$$

```
> evalm(M1-M2);
```

$$\begin{bmatrix} a1-a2 & b1-b2 \\ c1-c2 & d1-d2 \end{bmatrix}$$

```
> evalm(M1&*M2);
```

$$\begin{bmatrix} a1a2+b1c2 & a1b2+b1d2 \\ c1a2+d1c2 & c1b2+d1d2 \end{bmatrix}$$

```
> evalm(M1/M2);
```

$$\begin{bmatrix} \frac{a1d2}{a2d2-b2c2}-\frac{b1c2}{a2d2-b2c2} & -\frac{a1b2}{a2d2-b2c2}+\frac{b1a2}{a2d2-b2c2} \\ \frac{c1d2}{a2d2-b2c2}-\frac{d1c2}{a2d2-b2c2} & -\frac{c1b2}{a2d2-b2c2}+\frac{d1a2}{a2d2-b2c2} \end{bmatrix}$$

```
> evalm(M1&/M2);
```

$$\begin{bmatrix} a1\,\&/\begin{bmatrix} a2 & b2 \\ c2 & d2 \end{bmatrix} & b1\,\&/\begin{bmatrix} a2 & b2 \\ c2 & d2 \end{bmatrix} \\ c1\,\&/\begin{bmatrix} a2 & b2 \\ c2 & d2 \end{bmatrix} & d1\,\&/\begin{bmatrix} a2 & b2 \\ c2 & d2 \end{bmatrix} \end{bmatrix}$$

Приведем еще ряд примеров выполнения символьных операций с одной матрицей:

```
> evalm(M1^2);
```

$$\begin{bmatrix} a1^2+b1c1 & a1b1+b1d1 \\ c1a1+d1c1 & b1c1+d1^2 \end{bmatrix}$$

```
> evalm(sin(M1));
```

$$\begin{bmatrix} \sin(a1) & \sin(b1) \\ \sin(c1) & \sin(d1) \end{bmatrix}$$

```
> evalm(M1*z);
```

$$\begin{bmatrix} za1 & zb1 \\ zc1 & zd1 \end{bmatrix}$$

```
> evalm(M1/z);
```

$$\begin{bmatrix} \frac{a1}{z} & \frac{b1}{z} \\ \frac{c1}{z} & \frac{d1}{z} \end{bmatrix}$$

```
> evalm(M1+z);
```

$$\begin{bmatrix} a1+z & b1 \\ c1 & d1+z \end{bmatrix}$$

```
> evalm(M1-z);
```

$$\begin{bmatrix} a1-z & b1 \\ c1 & d1-z \end{bmatrix}$$

Среди других функций для работы с матрицами полезно обратить внимание на функцию `map`, которая применяет заданную операцию (например, функции дифференцирования `diff` и интегрирования `int`) к каждому элементу матрицы. Примеры такого рода даны ниже:

```
> M:=array(1..2,1..2,[[x,x^2],[x^3,x^4]]);
```

$$M := \begin{bmatrix} x & x^2 \\ x^3 & x^4 \end{bmatrix}$$

```
> map(diff,M,x);
```

$$\begin{bmatrix} 1 & 2x \\ 3x^2 & 4x^3 \end{bmatrix}$$

```
> map(int,%,x);
```

$$\begin{bmatrix} x & x^2 \\ x^3 & x^4 \end{bmatrix}$$

```
> map(sin,M);
```

$$\begin{bmatrix} \sin(x) & \sin(x^2) \\ \sin(x^3) & \sin(x^4) \end{bmatrix}$$

В результате возвращаются матрицы, каждый элемент которых представлен производной или интегралом. Аналогично можно выполнять над матрицами и другие достаточно сложные преобразования.

8.2. Пакет линейной алгебры linalg системы Maple

8.2.1. Состав пакета linalg

В ядро Maple включены скромные и минимально необходимые средства для решения задач линейной алгебры. Упор в реализации решения задач линейной алгебры сделан на подключаемые пакеты. Основным из них, унаследованным от предшествующих реализаций системы, является пакет решения задач линейной алгебры `linalg`. Это один из самых обширных и мощных пакетов в области решения задач линейной алгебры. Для их просмотра достаточно использовать команду

```
> with(linalg);
```

Для большинства пользователей системой Maple набор функций пакета оказывается чрезмерно обширным. Поэтому отметим только с десяток функций пакета linalg:

- `augment` – объединяет две или больше матриц по горизонтали;
- `band` – создает ленточную матрицу;
- `BlockDiagonal` – создает блок-диагональную матрицу;
- `blockmatrix` – создает блок-матрицу;

- `cholesky` – декомпозиция Холесского для квадратной положительно определенной матрицы;
- `charpoly` – возвращает характеристический полином матрицы;
- `cond` – вычисляет число обусловленности матрицы (cond(M) есть величина norm(M)·norm(M^{-1}));
- `diag` – создает блок-диагональную матрицу;
- `eigenvals` – вычисляет собственные значения матрицы;
- `eigenvects` – вычисляет собственные векторы матрицы;
- `leastsqrs` – решение уравнений по методу наименьших квадратов;
- `linsolve` – решение линейных уравнений;
- `toeplitz` – создает матрицу Теплица;
- `trace` – возвращает след матрицы;
- `vandermonde` – создает вандермондову матрицу и т. д.

Назначение многих функций вполне очевидно из их названия. Далее мы рассмотрим более подробно некоторые функции из этого пакета. С деталями синтаксиса функций пакета можно ознакомиться в справочной системе Maple. Для этого достаточно использовать команду `?name;`, где `name` – имя функции.

8.2.2. Интерактивный ввод матриц

Для *интерактивного ввода* матриц можно, определив размерность некоторого массива, использовать функцию `entermatrix`:

```
> c A:=array(1..3,1..3);
```

$$A := \text{array}(1 .. 3, 1 .. 3, [\ ])$$

После исполнения этого фрагмента документа диалог с пользователем имеет следующий вид:

```
> entermatrix(A);
enter element 1,1 > 1;
enter element 1,2 > 2;
enter element 1,3 > 3;
enter element 2,1 > 4;
enter element 2,2 > 5;
enter element 2,3 > 6;
enter element 3,1 > 7;
enter element 3,2 > 8;
enter element 3,3 > 9;
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
> B:=(%);
```

$$B := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
> B[1,1];
```

$$1$$

```
> B[2,2];
```

$$5$$

```
> B[3,3];
```

$$9$$

8.2.3. Основные функции для задания векторов и матриц

В библиотечном файле `linalg` имеются следующие функции для *задания векторов и матриц*:

- `vector(n,list)` – создание вектора с `n` элементами, заданными в списке `list`;
- `matrix(n,m,list)` – создание матрицы с числом строк `n` и столбцов `m` с элементами, заданными списком `list`.

Ниже показано применение этих функций:

```
> V:=vector(3,[12,34,56]);
```

$$V := [12, 34, 56]$$

```
> M:=matrix(2,3,[1,2,3,4]);
```

$$M := \begin{bmatrix} 1 & 2 & 3 \\ 4 & M_{2,2} & M_{2,3} \end{bmatrix}$$

```
> V[2];
```

$$34$$

```
> M[1,3];
```

$$3$$

Обратите внимание на последние примеры – они показывают вызов индексированных переменных вектора и матрицы.

8.2.4. Работа с векторами и матрицами

Для работы с векторами и матрицами Maple имеет множество функций, входящих в пакет `linalg`. Ограничимся приведением краткого описания наиболее распространенных функций этой категории.

Операции со структурой отдельного вектора **V** и матрицы **M**:

- `coldim(M)` – возвращает число столбцов матрицы `M`;
- `rowdim(M)` – возвращает число строк матрицы `M`;
- `vectdim(V)` – возвращает размерность вектора `V`;
- `col(M,i)` – возвращает *i*-й столбец матрицы `M`;
- `row(M,i)` – возвращает *i*-ю строку матрицы `M`;
- `minor(M,i,j)` – возвращает минор матрицы `M` для элемента с индексами `i` и `j`;

- `delcols(M,i..j)` – удаляет столбцы матрицы `M` от *i*-го до *j*-го;
- `delrows(V,i..j)` – удаляет строки матрицы `M` от *i*-й до *j*-й;
- `extend(M,m,n,x)` – расширяет матрицу `M` на `m` строк и `n` столбцов с применением заполнителя `x`.

Основные векторные и матричные операции:

- `dotprod(U,V)` – возвращает скалярное произведение векторов `U` и `V`;
- `crossprod(U,V)` – возвращает векторное произведение векторов `U` и `V`;
- `norm(V)` или `norm(M)` – возвращает норму вектора или матрицы;
- `copyinto(A,B,i,j)` – копирует матрицу `A` в `B` для элементов последовательно от `i` до `j`;
- `concat(M1,M2)` – возвращает объединенную матрицу с горизонтальным слиянием матриц `M1` и `M2`;
- `stack(M1,M2)` – возвращает объединенную матрицу с вертикальным слиянием `M1` и `M2`;
- `matadd(A,B)` и `evalm(A+B)` – возвращает сумму матриц `A` и `B`;
- `multiply(A,B)` и `evalm(A&*B)` – возвращает произведение матриц `A` и `B`;
- `adjoint(M)` или `adj(M)` – возвращает присоединенную матрицу, такую, что M*adj(M) дает диагональную матрицу, определитель которой есть det(M);
- `charpoly(M,lambda)` – возвращает характеристический полином матрицы `M` относительно заданной переменной `lambda`;
- `det(M)` – возвращает детерминант (определитель) матрицы `M`;
- `Eigenvals(M,vector)` – инертная форма функции, возвращающей собственные значения матрицы `M` и (при указании необязательного параметра `vector`) соответствующие им собственные векторы;
- `jordan(M)` – возвращает матрицу `M` в форме Жордана;
- `hermite(M)` – возвращает матрицу `M` в эрмитовой форме;
- `trace(M)` – возвращает след матрицы `M`;
- `rank(M)` – возвращает ранг матрицы `M`;
- `transpose(M)` – возвращает транспонированную матрицу `M`;
- `inverse(M)` или `evalm(1/M)` – возвращает матрицу, обратную к `M`;
- `singularvals(A)` – возвращает сингулярные значения массива или матрицы `A`.

Приведем примеры применения некоторых из этих функций:

```
> M:=matrix(2,2,[a,b,c,d]);
```

$$M := \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

```
> transpose(M);
```

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

```
> inverse(M);
```

$$\begin{bmatrix} -\dfrac{d}{-ad+bc} & \dfrac{b}{-ad+bc} \\ \dfrac{c}{-ad+bc} & -\dfrac{a}{-ad+bc} \end{bmatrix}$$

```
> det(M);
```

$$ad - bc$$

```
> rank(M);
```

$$2$$

```
> trace(M);
```

$$a + d$$

```
> M:=matrix(2,2,[1,2,3,4]);
```

$$M := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
> ev:=evalf(Eigenvals(M,V));
```

$$ev := [-.372281323, 5.372281323]$$

```
> eval(V);
```

$$\begin{bmatrix} -.8245648401 & -.4222291504 \\ .5657674650 & -.9230523142 \end{bmatrix}$$

```
> charpoly(M,p);
```

$$p^2 - 5p - 2$$

```
> jordan(M);
```

$$\begin{bmatrix} \dfrac{5}{2}+\dfrac{1}{2}\sqrt{33} & 0 \\ 0 & \dfrac{5}{2}-\dfrac{1}{2}\sqrt{33} \end{bmatrix}$$

```
> A:= array( [[1,0,1],[1,0,1],[0,1,0]] );
```

$$A := \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

```
> singularvals(A);
```

$$[0, 2, 1]$$

В приведенных примерах полезно обратить внимание на то, что многие матричные функции способны выдавать результаты вычислений в аналитическом виде, что облегчает разбор выполняемых ими операций.

8.2.5. Решение систем линейных уравнений

Одной из самых распространенных задач линейной алгебры является *решение систем линейных уравнений*. Поскольку такое решение численными методами с применением функций, входящих в пакет linalg, тривиально, приведем пример решения матричного уравнения в символьном виде:

```
> A:=matrix(2,2,[a,b,c,d]);
```

$$A := \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

```
> B:=vector(2,[c,d]);
```

$$B := [c, d]$$

```
> X:=linsolve(A,B);
```

$$X := \left[-\frac{d(-b+c)}{bc-da}, \frac{-da+c^2}{bc-da} \right]$$

Следующий пример показывает решение системы линейных уравнений с комплексными коэффициентами:

```
> A:=matrix(2,2,[[10+200*I,-200*I],[-200*I,170*I]]);
```

$$A := \begin{bmatrix} 10+200I & -200I \\ -200I & 170I \end{bmatrix}$$

```
> B:=vector(2,[5,0]);
```

$$B := [5, 0]$$

```
> X:=multiply(inverse(A),B);
```

$$X := \left[\frac{289}{7778} + \frac{510}{3889} I, \frac{170}{3889} + \frac{600}{3889} I \right]$$

```
> Digits:=5: convert(eval(X),float);
```

$$[.37156 + .13114I, .043713 + .15428I]$$

На этот раз решение получено использованием функций умножения матриц и вычисления обратной матрицы в виде $\mathbf{X} = \mathbf{A}^{-1}\,\mathbf{B}$, то есть в матричном виде. В конце примера показано преобразование результатов с целью их получения в обычной форме комплексных чисел с частями, представленными в форме чисел с плавающей точкой.

8.2.6. Визуализация матриц

Графическую визуализацию матриц обеспечивает графическая функция `matrixplot` из пакета `plots`. На рис. 8.1 показано совместное применение этой функции с двумя функциями пакета `linalg`, формирующими две специальные матрицы **A** и **B**.

На рис. 8.1 показана графическая визуализация матрицы, полученной как разность матриц **A** и **B**. Элементы результирующей матрицы представлены столбиками, высота которых задается значениями элементов. Для усиления эффекта восприятия применяется функциональная закраска разными цветами. Для задания цвета введена процедура `F`.

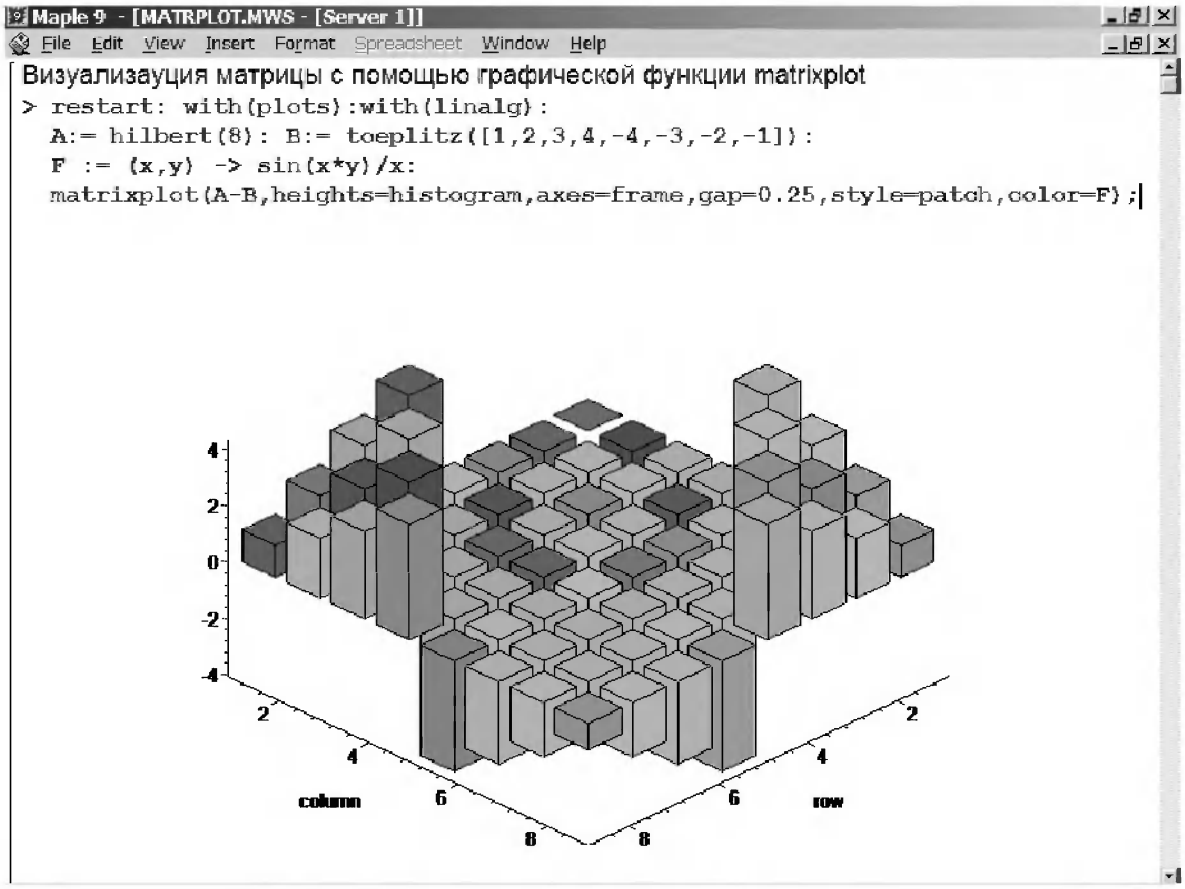


Рис. 8.1. Графическое представление матрицы

8.3. Работа с пакетом LinearAlgebra и алгоритмами NAG

8.3.1. Назначение и загрузка пакета LinearAlgebra

В новых реализациях систем Maple была сделана ставка на использование давно апробированных быстрых алгоритмов линейной алгебры, предложенных создателями Number Algorithm Group (NAG). Они обеспечивают ускорение численных матричных операций от нескольких раз до нескольких десятков раз. В Maple 9.5/10/11 использование алгоритмов NAG реализуется пакетом `LinearAlgebra`. Для его загрузки используются следующие команды:

```
> restart; with(LinearAlgebra):
> infolevel[LinearAlgebra]:=1;
```

$$infolevelLinearAlgebra := 1$$

Многие функции этого пакета (их большой список опущен) повторяет по назначению функции более старого пакета `linalg`, описанного выше. Поэтому мы не будем останавливаться на их повторном описании. Главное – то, что эти функции задействуют возможности быстрых алгоритмов NAG и, в отличие от функций пакета `linalg`, ориентированы на численные расчеты в том формате обработки вещественных чисел, который характерен для применяемой компьютерной платформы. Знающий матричные методы читатель легко поймет назначение функций пакета `LinearAlgebra` по их составным названиям. Например, `DeleteColumn` означает удаление столбца матрицы, `ToeplitzMatrix` означает создание матрицы Теплица, `ZeroMatrix` – создание матрицы с нулевыми элементами и т. д. Все имена функций этого пакета начинаются с заглавной буквы.

8.3.2. Примеры матричных операций с применением пакета LinearAlgebra

Применение алгоритмов NAG особенно эффективно в том случае, когда используется встроенная в современные микропроцессоры арифметика чисел с плавающей запятой. С помощью специального флага такую арифметику можно отключать или включать:

```
> UseHardwareFloats := false;     # use software floats
```

$$UseHardwareFloats := false$$

```
> UseHardwareFloats := true;      # default behaviour
```

$$UseHardwareFloats := true$$

Матрицы в новом пакете линейной алгебры могут задаваться в угловых скобках, как показано ниже:

```
> M1:=<<1|2>,<4|5>>; M2:=<<1|2.>,<4|5>>;
```

$$M1 := \begin{bmatrix} 1 & 2 \\ 4 & 5 \end{bmatrix}$$

$$M2 := \begin{bmatrix} 1 & 2. \\ 4 & 5 \end{bmatrix}$$

После этого можно выполнять с ними типовые матричные операции. Например, можно инвертировать (обращать) матрицы:

```
> M1^(-1); M2^(-1);
```

$$\begin{bmatrix} \frac{-5}{3} & \frac{2}{3} \\ \frac{4}{3} & \frac{-1}{3} \end{bmatrix}$$

```
MatrixInverse:      "calling external function"
MatrixInverse:      "NAG"   hw_f07adf
MatrixInverse:      "NAG"   hw_f07ajf
```

$$\begin{bmatrix} -1.66666666666666652 & .666666666666666630 \\ 1.33333333333333326 & -.333333333333333315 \end{bmatrix}$$

Обратите внимание, что Maple теперь выдает информационные сообщения о новых условиях реализации операции инвертирования матриц с вещественными элементами и, в частности, об использовании алгоритмов NAG и арифметики, встроенной в сопроцессор.

Следующий пример иллюстрирует создание двух случайных матриц `M1` и `M2` и затем их умножение:

```
> M1:=RandomMatrix(2,3); M2:=RandomMatrix(3,3);
Multiply(M1,M2,'inplace'); M1;
```

$$M1 := \begin{bmatrix} -50 & 62 & -71 \\ 30 & -79 & 28 \end{bmatrix}$$

$$M2 := \begin{bmatrix} 20 & -34 & -21 \\ -7 & -62 & -56 \\ 16 & -90 & -8 \end{bmatrix}$$

$$\begin{bmatrix} -2570 & 4246 & -1854 \\ 1601 & 1358 & 3570 \end{bmatrix}$$

$$\begin{bmatrix} -2570 & 4246 & -1854 \\ 1601 & 1358 & 3570 \end{bmatrix}$$

Параметр `inplace` в функции умножения обеспечивает помещение результата умножения матриц на место исходной матрицы `M1` – излюбленный прием создателей быстрых матричных алгоритмов NAG. Поскольку матрицы `M1` и `M2` заданы как случайные, то при повторении этого примера результаты, естественно, будут иными, чем приведенные.

Другой пример иллюстрирует проведение хорошо известной операции LU-разложения над матрицей `M`, созданной функцией `Matrix`:

```
> M:=Matrix([[14,-8,1],[-11,-4,18],[3,12,19]], datatype=float);
LUDecomposition(M,output=['NAG'],inplace);
ipiv:=%[1];
M;
```

$$M := \begin{bmatrix} 14. & -8. & 1. \\ -11. & -4. & 18. \\ 3. & 12. & 19. \end{bmatrix}$$

```
LUDecomposition:    "calling external function"
LUDecomposition:    "NAG"   hw_f07adf
```

$$\begin{bmatrix}1\\3\\3\end{bmatrix}, \begin{bmatrix}14. & -8. & 1.\\ .214285714285714274 & 13.7142857142857136 & 18.7857142857142848\\ -.785714285714285698 & -.750000000000000000 & 32.8750000000000000\end{bmatrix}$$

$$ipiv := \begin{bmatrix}1\\3\\3\end{bmatrix}$$

$$\begin{bmatrix}14. & -8. & 1.\\ .214285714285714274 & 13.7142857142857136 & 18.7857142857142848\\ -.785714285714285698 & -.750000000000000000 & 32.8750000000000000\end{bmatrix}$$

8.3.3. Методы решения систем линейных уравнений средствами пакета LinearAlgebra

Конечной целью большинства матричных операций является решение систем линейных уравнений. Для этого пакет `LinearAlgebra` предлагает ряд методов и средств их реализации. Основными методами решения являются следующие:

- обращение матрицы коэффициентов уравнений и решение вида $\mathbf{X}=\mathbf{A}^{-1}*\mathbf{B}$;
- применение метода LU – декомпозиции (`method='LU'`);
- применение метода QR – декомпозиции (`method='QR'`);
- применение метода декомпозиции Холесски (`method='Cholesky'`);
- метод обратной подстановки (`method='subs'`).

Решение с применением обращения матрицы коэффициентов левой части системы уравнений **A** уже не раз рассматривалось и вполне очевидно. В связи с этим отметим особенности решения систем линейных уравнений другими методами. Любопытно отметить, что указание метода может быть сделано и без его заключения в одинарные кавычки.

8.3.4. Решение системы линейных уравнений методом LU-декомпозиции

Зададим матрицу **A** левой части системы уравнений и вектор свободных членов **B**:

```
> restart; with(LinearAlgebra): UseHardwareFloats := false:
> A:=<<4|.24|-.08>,<.09|3|-.15>,<.04|-.08|4>>; B:=<8,9,20>;
```

$$A := \begin{bmatrix}4 & 0.24 & -0.08\\ 0.09 & 3 & -0.15\\ 0.04 & -0.08 & 4\end{bmatrix}$$

$$B := \begin{bmatrix}8\\9\\20\end{bmatrix}$$

Прямое решение этим методом выполняется одной из двух команд, отличающихся формой записи:

```
> x := LinearSolve(A, B, method='LU');
x := LinearSolve(<A|B>, method='LU');
```

$$x := \begin{bmatrix} 1.909198281 \\ 3.194964417 \\ 5.044807306 \end{bmatrix}$$

$$x := \begin{bmatrix} 1.909198281 \\ 3.194964417 \\ 5.044807306 \end{bmatrix}$$

Проверим решение данной системы уравнений:

```
> A.x-B;
```

$$\begin{bmatrix} 0. \\ 0. \\ 0. \end{bmatrix}$$

В данном случае решение точно (в пределах точности вычислений по умолчанию).

Можно также выполнить решение, проведя отдельно LU-декомпозицию, что делает наглядным алгоритм решения и операции подстановки:

```
> P,L,U:=LUDecomposition(A);
```

$$P, L, U := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1.0 & 0. & 0. \\ 0.02250000000 & 1.0 & 0. \\ 0.01000000000 & -0.02751619582 & 1.0 \end{bmatrix},$$

$$\begin{bmatrix} 4. & 0.24 & -0.08 \\ 0. & 2.994600000 & -0.1482000000 \\ 0. & 0. & 3.996722100 \end{bmatrix}$$

```
> V2:=Transpose(P).B;
```

$$V2 := \begin{bmatrix} 8 \\ 9 \\ 20 \end{bmatrix}$$

```
> V3:=ForwardSubstitute(L,V2);
```

$$V3 := \begin{bmatrix} 8.000000000 \\ 8.820000000 \\ 20.16269285 \end{bmatrix}$$

```
> x:=BackwardSubstitute(U,V3);
```

$$x := \begin{bmatrix} 1.909198281 \\ 3.194964417 \\ 5.044807306 \end{bmatrix}$$

```
> A.x-B;
```

$$\begin{bmatrix} 0. \\ 0. \\ 0. \end{bmatrix}$$

8.3.5. Решение системы линейных уравнений методом QR-декомпозиции

Выполним теперь решение для тех же исходных данных методом QR-декомпозиции, обозначив метод в функции `LinearSolve`:

```
> x := LinearSolve(A, B, method='QR');
```

$$x := \begin{bmatrix} 1.909198281 \\ 3.194964417 \\ 5.044807306 \end{bmatrix}$$

```
> A.x-B;
```

$$\begin{bmatrix} 0. \\ 0. \\ 0. \end{bmatrix}$$

Другой, более явный, но и более громоздкий метод решения представлен ниже:

```
> Q,R := QRDecomposition(A);
```

$$Q, R := \begin{bmatrix} -0.999697013 & 0.02220844667 & -0.01061449860 \\ -0.02249318279 & -0.9993685878 & 0.02750423536 \\ -0.009996970127 & 0.02773465579 & 0.9995653302 \end{bmatrix},$$

$$\begin{bmatrix} -4.001212316 & -0.3066070738 & 0.0433618580 \\ 0. & -2.994994508 & 0.2590672357 \\ 0. & 0. & 3.994984846 \end{bmatrix}$$

```
> V2:=Transpose(Q).B;
```

$$V2 := \begin{bmatrix} -8.399954152 \\ -8.261956601 \\ 20.15392873 \end{bmatrix}$$

```
> x:=BackwardSubstitute(R,V2);
```

$$x := \begin{bmatrix} 1.909198281 \\ 3.194964417 \\ 5.044807306 \end{bmatrix}$$

```
> A.x-B;
```

$$\begin{bmatrix} 0.410^{-8} \\ 0.310^{-8} \\ 0. \end{bmatrix}$$

Тут, пожалуй, любопытно, что погрешность вычислений оказалась несколько выше, чем при использовании функции LinearSolve. Однако погрешность не выходит за рамки допустимой по умолчанию.

8.3.6. Решение системы линейных уравнений методом декомпозиции Холесски

Выполним решение еще и методом декомпозиции Холесски:

```
> x:=LinearSolve(A, B, method='Cholesky');
```

$$x := \begin{bmatrix} 1.880315978 \\ 3.078064069 \\ 5.042758123 \end{bmatrix}$$

Приведем еще один пример решения системы из четырех линейных уравнений с применением метода декомпозиции Холесски:

```
> M_temp := Matrix(4, (i,j)->i+i*j-7, shape=triangular[lower]);
M :=M_temp.Transpose(M_temp);
IsMatrixShape(M,symmetric); IsDefinite(M);
```

$$M\_temp := \begin{bmatrix} -5 & 0 & 0 & 0 \\ -3 & -1 & 0 & 0 \\ -1 & 2 & 5 & 0 \\ 1 & 5 & 9 & 13 \end{bmatrix}$$

$$M := \begin{bmatrix} 25 & 15 & 5 & -5 \\ 15 & 10 & 1 & -8 \\ 5 & 1 & 30 & 54 \\ -5 & -8 & 54 & 276 \end{bmatrix}$$

$$true$$
$$true$$

```
> V := <6,1,3,-2>;
```

$$V := \begin{bmatrix} 6 \\ 1 \\ 3 \\ -2 \end{bmatrix}$$

```
> x:=LinearSolve(M, V, method='Cholesky');
```

$$X := \begin{bmatrix} \frac{42316}{21125} \\ \frac{-12403}{4225} \\ \frac{-229}{4225} \\ \frac{-192}{4225} \end{bmatrix}$$

```
> M.x-V;
```

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

```
> M:=Matrix(3, (i,j)->i+2*j-8, shape=triangular[lower]);
V:=<7,8,1>;
```

$$M := \begin{bmatrix} -5 & 0 & 0 \\ -4 & -2 & 0 \\ -3 & -1 & 1 \end{bmatrix}$$

$$V := \begin{bmatrix} 7 \\ 8 \\ 1 \end{bmatrix}$$

```
> x := ForwardSubstitute(M, V);
x := LinearSolve(M, V);
```

$$x := \begin{bmatrix} \frac{-7}{5} \\ \frac{-6}{5} \\ \frac{-22}{5} \end{bmatrix}$$

$$x := \begin{bmatrix} \frac{-7}{5} \\ \frac{-6}{5} \\ \frac{-22}{5} \end{bmatrix}$$

8.3.7. Одновременное решение нескольких систем уравнений

Мы ограничимся простым примером одновременного решения сразу трех систем уравнений. Дабы не загромождать книгу массивными выражениями, ограничимся решением систем из двух линейных уравнений, матрица коэффициентов у которых одна, а векторы свободных членов разные. Ниже показан пример решения такой системы:

```
> M:=Matrix([[1.,3],[4,5]],datatype=float);
V1:=<1.,2>;
V2:=<7,-11>;
V3:=<-34,-67>;
```

$$M := \begin{bmatrix} 1. & 3. \\ 4. & 5. \end{bmatrix}$$

$$V1 := \begin{bmatrix} 1. \\ 2 \end{bmatrix}$$

$$V2 := \begin{bmatrix} 7 \\ -11 \end{bmatrix}$$

$$V3 := \begin{bmatrix} -34 \\ -67 \end{bmatrix}$$

```
> LinearSolve(M,<V1|V2|V3>);
LinearSolve: "calling external function"
LinearSolve: "NAG"   hw_f07adf
LinearSolve: "NAG"   hw_f07aef
```

$$\begin{bmatrix} .142857142857142905 & -9.71428571428571352 & -4.42857142857143060 \\ .285714285714285698 & 5.57142857142857118 & -9.85714285714285588 \end{bmatrix}$$

```
> M:=Matrix([[1.,3],[4,5]],datatype=float);
ipiv, M := LUDecomposition(M,output=['NAG'],inplace);
LinearSolve([ipiv, M], <V1|V2|V3>);
```

$$M := \begin{bmatrix} 1. & 3. \\ 4. & 5. \end{bmatrix}$$

```
LUDecomposition:    "calling external function"
LUDecomposition:    "NAG"   hw_f07adf
```

$$ipiv, M := \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \begin{bmatrix} 4. & 5. \\ .250000000000000000 & 1.750000000000000000 \end{bmatrix}$$

```
LinearSolve: "calling external function"
LinearSolve: "NAG"   hw_f07aef
```

$$\begin{bmatrix} .142857142857142905 & -9.714285714285713 52 & -4.428571428571430 60 \\ .285714285714285698 & 5.571428571428571 18 & -9.857142857142855 88 \end{bmatrix}$$

8.3.8. Решение системы линейных уравнений с треугольной матрицей

Иногда при решении задач линейной алгебры приходится считаться с особой формой матриц левой части системы. К примеру, она может быть треугольной, как в приведенном ниже примере:

```
> M:=<<-3|1|6>,<0|2|7>,<0|0|-4>>;
V:=<1,3,5>;
```

$$M := \begin{bmatrix} -3 & 1 & 6 \\ 0 & 2 & 7 \\ 0 & 0 & -4 \end{bmatrix}$$

$$V := \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}$$

Ниже представлено решение такой системы уравнений двумя методами – обратной подстановки и с помощью стандартной функции `LinearSolve`:

```
> x := BackwardSubstitute(M,V);
x := LinearSolve(M, V, method=subs);
```

$$x := \begin{bmatrix} \frac{-7}{8} \\ \frac{47}{8} \\ \frac{-5}{4} \end{bmatrix}$$

$$x := \begin{bmatrix} \frac{-7}{8} \\ \frac{47}{8} \\ \frac{-5}{4} \end{bmatrix}$$

Проверка показывает, что решение точно:

```
> M.x-V;
```

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Возможности пакетов `linalg` и `LinearAlgebra` удовлетворят самых требовательных специалистов в этой области математики.

8.4. Интеграция Maple с MATLAB

8.4.1. Краткие сведения о MATLAB

В области линейной алгебры к числу наиболее мощных систем относится система MATLAB, созданная компанией MathWorks, Inc. [98–118]. Ее название происходит именно от слов MATrix LABoratory – матричная лаборатория. MATLAB содержит в своем ядре многие сотни матричных функций и реализует самые современные алгоритмы матричных операций, включая алгоритмы группы NAG. Последней версией системы является MATLAB R2008a.

В то же время нельзя не отметить, что MATLAB – одна из самых громоздких математических систем. Инсталляция ее полной версии занимает до 4 Гбайт дискового пространства. Несмотря на это, интеграция различных математических систем с данной системой, похоже, становится своеобразной модой. Такая возможность предусмотрена и в системе Maple с помощью пакета Matlab (однако не для новых версий MATLAB).

8.4.2. Загрузка пакета расширения MATLAB

Для загрузки пакета Matlab используется команда

```
> with(Matlab);
```

[*chol*, *closelink*, *defined*, *det*, *dimensions*, *eig*, *evalM*, *fft*, *getvar*, *inv*, *lu*, *ode45*, *openlink*, *qr*, *setvar*, *size*, *square*, *transpose*]

Использование этой команды ведет к автоматическому запуску системы MATLAB и установлению необходимой объектной связи между системами Maple и MATLAB – рис. 8.2.

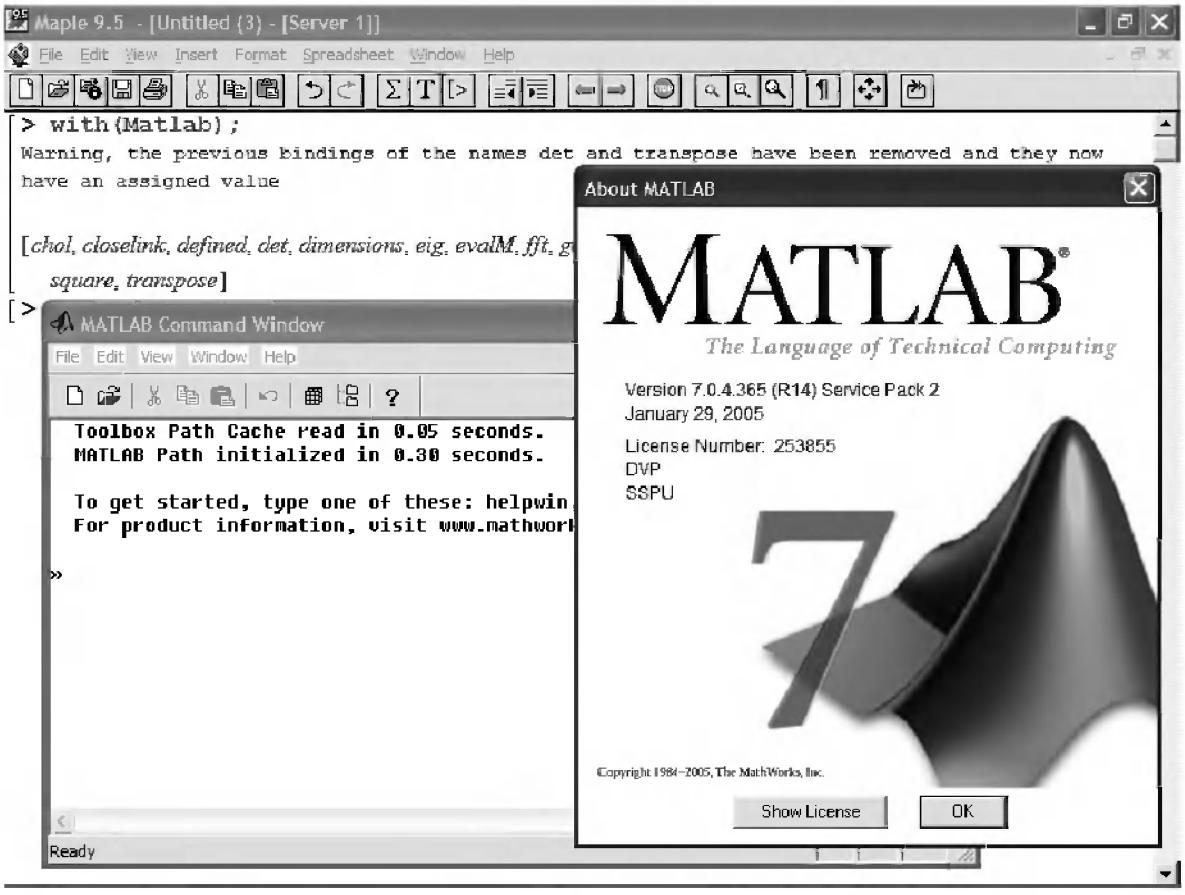


Рис. 8.2. Установление связи между системами Maple и MATLAB

Как нетрудно заметить, данный пакет дает доступ всего к 18 функциям системы MATLAB (из многих сотен, имеющихся только в ядре последней системы). Таким образом, есть все основания полагать, что возможности MATLAB в интеграции с системой Maple используются пока очень слабо и носят рудиментарный характер.

8.4.3. Типовые матричные операции пакета расширения MATLAB

Большинство функций пакета Matlab (не путайте с системой MATLAB) реализуют самые обычные матричные операции, что и иллюстрируют приведенные ниже примеры.

Зададим матрицу `M` в формате Maple:

```
> maplematrix_a:=array(1..3,1..3,[[6,4,2],[7,8,1], [3,7,3]]);
```

$$maplematrix\_a := \begin{bmatrix} 6 & 4 & 2 \\ 7 & 8 & 1 \\ 3 & 7 & 3 \end{bmatrix}$$

Ниже даны примеры транспонирования матрицы, ее инвертирования, вычисления детерминанта и собственных значений матрицы:

```
> Matlab[transpose](maplematrix_a);
```

$$\begin{bmatrix} 6. & 7. & 3. \\ 4. & 8. & 7. \\ 2. & 1. & 3. \end{bmatrix}$$

```
> Matlab[inv](maplematrix_a);
```

$$\begin{bmatrix} .212499999999999967 & .0250000000000000360 & -.150000000000000022 \\ -.224666666666666678 & .149999999999999966 & .100000000000000032 \\ .312500000000000000 & -.374999999999999944 & .249999999999999944 \end{bmatrix}$$

```
> Matlab[det](maplematrix_a);
```

$$80.$$

```
> Matlab[eig](maplematrix_a);
```

$$\begin{bmatrix} 13.8861968635740444 + 0.I \\ 1.55690156821298186 + 1.82689340924767875I \\ 1.55690156821298186 - 1.82689340924767875I \end{bmatrix}$$

Можно проверить, является ли матрица квадратной:

```
> Matlab[square](maplematrix_a);
```

$$true$$

а также вычислить размер матрицы:

```
> Matlab[dimensions](maplematrix_a);
```

$$[3, 3]$$

Можно также проверить, является ли данная матрица матрицей системы MATLAB:

```
> Matlab[defined]('maplematrix_a');
```

$$false$$

Здесь надо иметь в виду, что форматы матриц в системах Maple и MATLAB различны. Выполним LU-преобразование матрицы:

```
> Matlab[lu](maplematrix_a,output='L');
```

$$\begin{bmatrix} 7. & 8. & 1. \\ -.857142857142857094 & 3.57142857142857162 & 2.57142857142857162 \\ .428571428571428548 & 7.99999999999999820 & 3.19999999999999972 \end{bmatrix}$$

Если система MATLAB не установлена на вашем компьютере, то доступ к функциям пакета Matlab будет отсутствовать, а Maple при попытке использования данных функций будет выдавать сообщения об ошибках.

8.5. Операции линейной алгебры СКМ Mathematica

8.5.1. Векторные функции

К векторным функциям системы Mathematica 4 относятся функции точечного и кросс-произведения:

- **a.b.c** или **Dot[a, b, c]** – возвращает точечное произведение для векторов, матриц или тензоров;
- **Cross[a, b]** – возвращает кросс-произведение векторов **a** и **b**.

В системе Mathematica 5 к ним добавились следующие новые функции:

- **Total[list]** – дает общую сумму элементов листа;
- **Total[list, n]** – дает общую сумму элементов листа на уровне **n**;
- **Norm[expr]** – дает норму числа или массива;
- **Norm[expr, p]** – дает **p**-норму.

Применение этих функций достаточно очевидно, так что ограничимся следующими наглядными примерами:

```
Dot[a,b,x]
 a.b.x
{a₁, a₂, a₃}·{b₁, b₂, b₃}
 a₁b₁ + a₂b₂ + a₃b₃
u={1,2,3};
v={4,5,6};
w=u%v
 {-3,6,-3}
Total[{a,b,c}]
 a+b+c
{Total[u],Total[v]}
 {6,15}
vc={1,2+3*i,4+5*i}
 {1,2+3 i,4+5 i}
Total[vc]
 7+8 i
Norm[2+3*i]
 √13
Norm[{x1,x2,x3}]
 √ Abs[x1]^2 + Abs[x2]^2 + Abs[x3]^2
Norm[{1,2,3}]
 √14
```

8.5.2. Функции для операций линейной алгебры

Следующая группа функций системы Mathematica позволяет осуществить основные операции над векторами и матрицами, используемыми в *линейной алгебре*.

- **Cross[v1,v2,v3,...]** – кросс-произведение векторов (может задаваться в виде v1*v2*v3*...).
- **Det[m]** – возвращает детерминант (определитель) квадратной матрицы **m**.
- **DiagonalMatrix[list]** – возвращает диагональную матрицу с главной диагональю, сформированной из элементов списка list , и нулевыми остальными элементами матрицы.
- **Dot[a, b, c]** – возвращает произведения векторов, матриц и тензоров. Операцию произведения можно задавать также и в виде **a.b.c** .
- **Eigensystem[m]** – возвращает список {values,vectors} собственных значений и собственных векторов данной квадратной матрицы **m**.
- **Eigenvalues[m]** – возвращает список собственных значений квадратной матрицы **m**.
- **Eigenvectors[m]** – возвращает список собственных векторов квадратной матрицы **m**.
- **IdentityMatrix[n]** – возвращает единичную матрицу с размером n×n (у нее диагональные элементы имеют значения 1, остальные 0).
- **Inverse[m]** – возвращает обратную матрицу для квадратной матрицы **m**, то есть матрицу $\mathbf{m}^{-1}$, которая, будучи умноженной на исходную матрицу, дает единичную матрицу.
- **MatrixExp[m]** – возвращает экспоненциал матрицы **m**.
- **MatrixPower[m, n]** – возвращает **n**-ую степень матрицы **m**.
- **MatrixQ[expr]** – возвращает True, если expr является списком списков, который может представлять матрицу, иначе возвращает False.
- **MatrixQ[expr, test]** – возвращает True, только если **test** дает True в применении к каждому элементу матрицы в **expr**.
- **Minors[m, k]** – возвращает матрицу, составленную из определителей всех **k×k** субматриц **m**.
- **NullSpace[m]** – возвращает список векторов, которые формируют базис для нулевого пространства матрицы **m**.
- **Pivoting** – опция, относящаяся к функциям декомпозиции матрицы; указывает, что должен выполняться поворот столбца. Результат имеет форму {**Q**,**R**,**P**}, где **P** – матрица перестановок, такая, что имеет место **M.P= =Conjugate[Transpose[Q]].R**, где **M** – начальная (исходная) матрица.
- **PseudoInverse[m]** – ищет псевдообратную квадратной матрице **m**.
- **Tr[list]** – возвращает след матрицы или тензора (функция только у Mathematica 4).
- **Traspose[m]** – возвращает транспонированную матрицу, у которой столбцы и строки меняются местами, в сравнении с матрицей **m**.

- **RowReduce[m]** – возвращает приведенную к строке форму матрицы **m**.
- **ZeroTest** – опция для **LinearSolve** и других линейных алгебраических функций; дает функцию для применения ее к сочетаниям (комбинациям) из матричных элементов с целью определения, следует или нет полагать их равными нулю.

Приведенные ниже примеры иллюстрируют применение основных из этих функций (вывод в текстовом формате):

Ввод (*In*)	**Вывод (*Out*)**
A:=IdentityMatrix[3]	
A	{{1, 0, 0}, {0, 1, 0}, {0, 0, 1}}
MatrixExp[A]	{{E, 0, 0}, {0, E, 0}, {0, 0, E}}
MatrixQ[A]	True
MatrixPower[MatrixExp[A],-1.5]	{{0.22313,0,0},{0,0.22313,0}, {0, 0, 0.22313}}
A+{{1,2,3},{4,5,6},{7,8,9}}	{{2,2,3},{4,6,6},{7,8,10}}
m:={{1,2},{3,7}}	
MatrixForm[m]	1 2 3 7
Det[m]	1
Inverse[m]	{{7, -2}, {-3, 1}}
MatrixQ[m]	True
RowReduce[m]	{{1, 0}, {0, 1}}

Многие матричные функции в системе Mathematica 5 существенно доработаны, алгоритмы их работы улучшены. Введена также новая функция:

- **CharacteristicPolynomial[m, x]** – возвращает характеристический полином матрицы **m** с независимой переменной **x**.

Пример применения этой функции представлен ниже:

$$\mathbf{m} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

```
{{1,2},{3,4}}
```

```
CharacteristicPolynomial[m, x]
```

$$-2 - 5x + x^2$$

Начиная с Mathematica 5.2 в системе обеспечена поддержка новых многоядерных процессоров. Однако она эффективна только при больших матрицах с размером порядка 1000 на 1000 элементов. В этом случае скорость вычислений повышается в несколько раз при использовании процессоров с 8–16 ядрами против скорости вычислений на ПК с одноядерным процессором.

8.5.3. Функции декомпозиции матриц

При реализации ряда матричных методов используются различные виды декомпозиции (разложения матриц). Для числовой матрицы **m** они реализуются следующими функциями:

- **QRDecomposition[m]** – возвращает QR-разложение (декомпозицию);
- **LUDecomposition[m]** – возвращает результат LU-декомпозиции матрицы m;

- **CholeskyDecomposition[m]** – возвращает результат декомпозиции Холески;
- **SchurDecomposition[m]** – возвращает результат декомпозиции Холески;
- **SchurDecomposition[m, a]** – возвращает результат декомпозиции Холески;
- **JordanDecomposition[m]** – возвращает результат декомпозиции Жордана для матрицы m.

Работа с этими функциями аналогична описанной выше для системы Maple.

8.5.4. Решение систем линейных уравнений

Приведем также примеры на решение *систем линейных уравнений* матричными методами. В первом из них решение выполняется в символьном виде на основании формулы $\mathbf{X}=\mathbf{A}^{-1}\mathbf{B}$, где **A** – матрица коэффициентов системы линейных уравнений, **B** – вектор свободных членов. Для перемножения используется функция Dot, а для инвертирования матрицы – функция **Inverse**:

Пример решения системы линейных уравнений в символьном виде:

```
A:={{a,b},{c,d}}
B:={e,f}
X:=Dot[Inverse[A],B]
X
```

$$\left\{\frac{de}{-bc+ad}-\frac{bf}{-bc+ad},-\frac{ce}{-bc+ad}+\frac{af}{-bc+ad}\right\}$$

Для решения систем линейных уравнений существует большое число самых различных методов, в том числе использующих различные типы декомпозиции матрицы коэффициентов системы. Однако на практике обычно применяется специальная функция для решения таких систем уравнений:

- **LinearSolve[m, b]** – возвращает вектор **x** – решение матричного уравнения **m.x==b** , где **m** – матрица коэффициентов левой части системы линейных уравнений, **x** – вектор неизвестных и **b** – вектор свободных членов в правой части системы.

Ниже представлен пример (второй) на ее применение для решения системы из двух уравнений в численном виде:

```
LinearSolve[{{1,2},{3,4}},{7,9}]
 {-5,6}
```

Нередко, например в электротехнических расчетах, встречается необходимость решения *систем линейных уравнений с комплексными элементами*. Все описанные выше функции обеспечивают работу с комплексными числами. Следующий пример иллюстрирует решение системы линейных уравнений с комплексными данными с помощью функции **LinearSolve**:

```
A={{1+2I,2+3I},{3+4I,4+5I}}
 {{(1+2 i)  ,(2+3 i)},{(3+4 i),(4+5 i)}}
B={2I,3}
 {2 i  ,3  }
X=LinearSolve[A,B]
```

$$\left\{\left(\frac{1}{4}-4\,i\right),\frac{11\,i}{4}\right\}$$

Mathematica прекрасно приспособлена для решения больших систем линейных уравнений. Создадим, к примеру, матрицу размера 500×500 случайных чисел

```
A=Table[Random[],{i,1, 500},{j,1,500}];
Do[A[[i,i]]=2,{i,1,500}];
```

и вектор из 500 значений косинуса

```
B=Table[Cos[i*0.01],{i,1,500}];
```

Теперь решим систему из 500 линейных уравнений с оценкой времени решения:

```
X=Timing[LinearSolve[A,B]];
X[[1]]
```

0.062

Итак, система из 500 уравнений решена за время менее одной сотой секунды (использовалась Mathematica 6). Оценим погрешность решения и убедимся в ее малости:

```
Max[Abs[A.X[[2]]-B]]
```

1.69966×10^{-12}

Число матричных функций в системе Mathematica ограничено разумным минимумом, позволяющим реализовать множество других, более сложных матричных функций и преобразований. Их можно найти в пакетах расширения системы, посвященных линейной алгебре.

8.6. Пакет LinearAlgebra СКМ Mathematica 4/5

Пакет расширения линейной алгебры LinearAlgebra системы Mathematica 4/5 добавляет в эту область ее применения ряд новых функций, полезных при решении сложных задач линейной алгебры.

8.6.1. Декомпозиция Холесски

Подпакет Holesky содержит единственную функцию:

- **HoleskyDecomposition[m]** – задает *декомпозицию Холесски* для симметричной положительной матрицы m.

Примеры на декомпозицию Холесски даны ниже:

```
<<LinearAlgebra'Cholesky'
 CholeskyDecomposition::obslt: CholeskyDecomposition is now a
System' function. The package LinearAlgebra'Cholesky' is obsolete.
hil = Table[ 1/(i + j - 1), {i, 1, 4}, {j, 1, 4}]
```

$$\left\{\left\{1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}\right\}, \left\{\frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}\right\}, \left\{\frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}\right\}, \left\{\frac{1}{4}, \frac{1}{5}, \frac{1}{6}, \frac{1}{7}\right\}\right\}$$

```
Eigenvalues[ N[hil] ]
```
$$\{1.50021, 0.169141, 0.00673827, 0.0000967023\}$$
```
u =  CholeskyDecomposition[hil]
```
$$\left\{\left\{1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}\right\}, \left\{0, \frac{1}{2\sqrt{3}}, \frac{1}{2\sqrt{3}}, \frac{3\sqrt{3}}{20}\right\}, \left\{0, 0, \frac{1}{6\sqrt{5}}, \frac{1}{4\sqrt{5}}\right\}, \left\{0, 0, 0, \frac{1}{20\sqrt{7}}\right\}\right\}$$
```
MatrixForm[Transpose[u] . u]
```
$$\begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{pmatrix}$$

8.6.2. Реализация метода исключения Гаусса

Следующие функции обеспечивают реализацию *метода исключения Гаусса* при решении линейного уравнения вида **a·x=b**:

- **LUFactor[m]** – возвращает LU-декомпозицию матрицы **m**;
- **LUSolve[lu, b]** – решает систему линейных уравнений с матрицей коэффициентов lu и вектором свободных членов b методом исключения переменных Гаусса;
- **LU[a,pivots]** – создает объект, используемый в LUSolve.

Применение этих функций поясняют примеры ниже:

```
<<LinearAlgebra'GaussianElimination'
 GaussianElimination::obslt: The package
LinearAlgebra'GaussianElimination' is obsolete. Use the single-
argument form of LinearSolve instead.
MatrixForm[a = {{1, 2, 3}, {4, 5, 6},  {-1, -8, -5}}]
```
$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ -1 & -8 & -5 \end{pmatrix}$$
```
lu = LUFactor[a]
```
$$\mathrm{LU}\left[\left\{\left\{\frac{1}{4}, -\frac{1}{9}, \frac{10}{9}\right\}, \{4, 5, 6\}, \left\{-\frac{1}{4}, -\frac{27}{4}, -\frac{7}{2}\right\}\right\}, \{2, 3, 1\}\right]$$
```
b = {10, -3, 12}
```
$$\{10, -3, 12\}$$
```
LUSolve[lu, b]
```
$$\left\{-\frac{118}{15}, -\frac{109}{15}, \frac{54}{5}\right\}$$

8.6.3. Операции с матрицами

Подпакет MatrixManipulation добавляет к матричным функциям ядра системы Mathematica ряд новых функций. Начнем их описание с функций *объединения матриц*:

- **AppendCollumns[m1,m2,...]** – объединяет по столбцам матрицы **m1**,**m2**,....
- **AppendRows[m1,m2,...]** – объединяет по строкам матрицы **m1**,**m2**,....
- **BlockMatrix[blocks]** – объединяет по строкам и столбцам блоки **blocks**, создавая новую матрицу.

Данные операции с матрицами иллюстрируют следующие примеры:

```
<< LinearAlgebra'MatrixManipulation';
a = {{a11, a12}, {a21, a22}}; MatrixForm[a]
(a11 a12)
(a21 a22)
b = {{b11, b12}, {b21, b22}}; MatrixForm[b]
(b11 b12)
(b21 b22)
MatrixForm[AppendColumns[a, b]]
(a11 a12)
 a21 a22
 b11 b12
(b21 b22)
AppendRows[a, b] //MatrixForm
(a11 a12 b11 b12)
(a21 a22 b21 b22)
BlockMatrix[{{a, b}, {b, {{0, 0}, {0, 0}}}}] //MatrixForm
(a11 a12 b11 b12)
 a21 a22 b21 b22
 b11 b12  0   0
(b21 b22  0   0 )
```

Следующая группа функций вставляет или удаляет столбцы или строки:

- **TakeRows[mat,n]** – вставляет в матрицу **mat** n-ую строку.
- **TakeRows[mat,-n]** – удаляет из матрицы **mat** n-ую строку.
- **TakeRows[mat,{m,n}]** – вставляет в матрицу **mat** строки от m до n.
- **TakeCollumns[mat,n]** – вставляет в матрицу **mat** n-ый столбец.
- **TakeCollumns[mat,-n]** – удаляет из матрицы **mat** n-ый столбец.
- **TakeCollumns[mat,{m,n}]** – вставляет в матрицу **mat** столбцы от m до n.

Действие функции иллюстрируется следующими примерами:

```
mat = Array[m, {3, 4}]; MatrixForm[mat]
(m[1, 1] m[1, 2] m[1, 3] m[1, 4])
 m[2, 1] m[2, 2] m[2, 3] m[2, 4]
(m[3, 1] m[3, 2] m[3, 3] m[3, 4])
TakeRows[mat, -2] //MatrixForm
(m[2, 1] m[2, 2] m[2, 3] m[2, 4])
(m[3, 1] m[3, 2] m[3, 3] m[3, 4])
```

```
TakeColumns[mat, {2,3}] //MatrixForm
```

$$\begin{pmatrix} m[1,2] & m[1,3] \\ m[2,2] & m[2,3] \\ m[3,2] & m[3,3] \end{pmatrix}$$

```
TakeMatrix[mat, {2, 3}, {3, 4}] //MatrixForm
```

$$\begin{pmatrix} m[2,3] & m[2,4] \\ m[3,3] & m[3,4] \end{pmatrix}$$

```
SubMatrix[mat, {2, 3}, {2, 2}] //MatrixForm
```

$$\begin{pmatrix} m[2,3] & m[2,4] \\ m[3,3] & m[3,4] \end{pmatrix}$$

Еще одна группа функций служит для задания матриц специального вида:

- **UpperDiagonalMatrix[f,n]** – формирует наддиагональную матрицу размера n×n;
- **LowerDiagonalMatrix[f,n]** – формирует поддиагональную матрицу размера n×n;
- **ZeroMatrix[n]** – формирует квадратную нулевую матрицу размера n×n;
- **ZeroMatrix[m,n]** – формирует квадратную нулевую матрицу размера m×n;
- **HilbertMatrix[n]** – формирует квадратную матрицу Гильберта размера n×n;
- **HilbertMatrix[m,n]** – формирует матрицу Гильберта размера m×n;
- **HankelMatrix[n]** – формирует квадратную матрицу Ганкеля размера n×n;
- **HankelMatrix[m,n]** – формирует матрицу Ганкеля размера m×n.

Примеры на задание матриц разного типа приведены ниже:

```
UpperDiagonalMatrix[f, 3] //MatrixForm
```

$$\begin{pmatrix} f[1,1] & f[1,2] & f[1,3] \\ 0 & f[2,2] & f[2,3] \\ 0 & 0 & f[3,3] \end{pmatrix}$$

```
LowerDiagonalMatrix[#1 + #2 &, 4] //MatrixForm
```

$$\begin{pmatrix} 2 & 0 & 0 & 0 \\ 3 & 4 & 0 & 0 \\ 4 & 5 & 6 & 0 \\ 5 & 6 & 7 & 8 \end{pmatrix}$$

```
HilbertMatrix[2, 4] //MatrixForm
```

$$\begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ 1 & 1 & 1 & 1 \\ 2 & 3 & 4 & 5 \end{pmatrix}$$

```
HankelMatrix[{w, x, y, z}] //MatrixForm
```

$$\begin{pmatrix} w & x & y & z \\ x & y & z & 0 \\ y & z & 0 & 0 \\ z & 0 & 0 & 0 \end{pmatrix}$$

Наконец, в подпакет входит еще одна функция:

- **LinearEquationsToMatrices[eqns,vars]** – из записи линейного уравнения eqns с переменными vars формирует расширенную матрицу, содержащую матрицу коэффициентов левой части уравнения и вектор свободных членов.

Пример на применение данной функции:

```
LinearEquationsToMatrices[
 {a[1,1]*x + a[1,2]*y == c[1],
  a[2,1]*x + a[2,2]*y == c[2]}, {x, y}]
{{{{a11,a12},{a21,a22}}[1,1],{{a11,a12},{a21,a22}}[1,2]},{{{a11,a12},
{a21,a22}}[2,1],{{a11,a12},{a21,a22}}[2,2]}},{c[1],c[2]}}
```

8.6.4. Ортогонализация и нормализация

В подпакете *ортогонализации* Ortogonalization имеются следующие функции:

- **GramSchmidt[{v1,v2,...}]** – создает ортогональное множество в виде листа векторов v1,v2,....
- **Normalize[vect]** – возвращает нормализованный вектор vect.
- **Projection[vect1,vect2]** – дает ортогональную проекцию вектора v1 на вектор v2.

В этих функциях после аргументов допустимы опции InnerProduct->expr и Normalized->False (отказ от нормализации). Примеры работы с функциями ортогонализации представлены ниже:

```
<<LinearAlgebra'Orthogonalization'
{w1, w2, w3} = GramSchmidt[ {{1,3,2}, {2,4,3}, {2,4,6}}]
```

$$\left\{\left\{\frac{1}{\sqrt{14}}, \frac{3}{\sqrt{14}}, \sqrt{\frac{2}{7}}\right\},\right.$$

$$\left.\left\{\frac{4}{\sqrt{21}}, -\frac{2}{\sqrt{21}}, \frac{1}{\sqrt{21}}\right\}, \left\{-\frac{1}{\sqrt{6}}, -\frac{1}{\sqrt{6}}, \sqrt{\frac{2}{3}}\right\}\right\}$$

```
{ w1 . w2, w2 . w3, w1 . w3, w1 . w1, w2 . w2, w3 . w3}
   {0,0,0,1,1,1}
GramSchmidt[{1, x, x^2, x^3, x^4}, InnerProduct ->
   (Integrate[#1 #2,{x,-1,1}]&)]  //Simplify
```

$$\left\{\frac{1}{\sqrt{2}}, \sqrt{\frac{3}{2}}\, x, \frac{1}{2}\sqrt{\frac{5}{2}}\, (-1+3x^2),\right.$$

$$\left.\frac{1}{2}\sqrt{\frac{7}{2}}\, x\,(-3+5x^2), \frac{3\,(3-30x^2+35x^4)}{8\sqrt{2}}\right\}$$

```
Normalize[LegendreP[2,x], InnerProduct ->
   (Integrate[#1 #2,{x,-1,1}]&)]
```

$$\sqrt{\frac{5}{2}}\left(-\frac{1}{2}+\frac{3x^2}{2}\right)$$

```
{w1, w2} = GramSchmidt[{{3,4,3}, {2,3,6}}, Normalized -> False]
```

$$\left\{\{3, 4, 3\}, \left\{-\frac{20}{17}, -\frac{21}{17}, \frac{48}{17}\right\}\right\}$$

```
{w1 . w1, w1 . w2}
   {34,0}
```

8.6.5. Решение линейных уравнений с трехдиагональной матрицей

При решении линейных уравнений часто встречаются матрицы особой формы – *трехдиагональные*. Подпакет Tridiagonal имеет функцию для решения линейных уравнений с такой матрицей:

- **TridiagonalSolve[a,b,c,r]** – решение системы линейных уравнений с трехдиагональной матрицей **m.x==r** (диагонали представлены векторами a,b и c, вектор свободных членов – r).

Пример применения данной функции:

```
<<LinearAlgebra'Tridiagonal'
 {a, b, c} = {{1, 2, 3}, {4, 5, 6, 7}, {10, 9, 8}}
   {{1,2,3},{4,5,6,7},{10,9,8}}
m = Table[Switch[ j-i, -1, a[[j]], 0, b[[j]], 1,  c[[j-1]], _, 0],
{i, 4}, {j, 4}]//MatrixForm
TridiagonalSolve[a, b, c, {8, 3, 4, 5}]
```

$$\left\{\frac{53}{9}, -\frac{14}{9}, \frac{44}{81}, \frac{13}{27}\right\}$$

С учетом представленных функций и функций ядра набор матричных средств системы Mathematica является одним из наиболее полных. В области решения задач в численном виде он уступает лишь специализированной матричной системе MATLAB. Возможности последней детально рассмотрены в [98–118].

8.7. Средства линейной алгебры СКМ Mathcad

Решение задач линейной алгебры является одной из важных сфер применения различных систем компьютерной математики. Большие возможности в этом представляют и системы Mathcad.

8.7.1. Векторы и матрицы в СКМ Mathcad

В Mathcad массив, как и любая другая переменная, задается именем. Местоположение элемента задается одним индексом для вектора или двумя индексами для матрицы. Нижняя граница индексации определяется системной переменной ORIGIN, которая может принимать значение 0 или 1 (по умолчанию ее значение равно 0). Индексы могут быть только целыми положительными числами (и нулем). Для ввода индекса используется клавиша [– прямая открывающаяся скобка.

Вектор можно задать прямым присваиванием значений некоторым элементам вектора:

$v_0 := 1$ $v_2 := 2$ $v_3 := 3$

В результате будет получен следующий вектор:

$$V = \begin{pmatrix} 1 \\ 2 \\ 0 \\ 3 \end{pmatrix}.$$

Аналогично (то есть без заполнения шаблона) можно создать, например, нулевую матрицу, используя ранжированные переменные:

```
j := 0..2                    i := 0..2
M0i,j := 0
```

В результате будет получена следующая матрица:

$$M0 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

Здесь представлен вариант отображения массива в виде обычной матрицы. Возможен также вариант отображения в виде *электронной таблицы*, которая имеет полосы прокрутки, обеспечивая представление массивов больших размеров.

8.7.2. Векторные и матричные операции в Mathcad

Для работы с векторами и матрицами система Mathcad поддерживает ряд специальных операторов и функций. Ниже представлены основные векторные и матричные операции, решающие простейшие задачи линейной алгебры – прежде всего выполнение арифметических операций над векторами и матрицами.

Запись	Клавиши	Описание
V1+V2	V1 + V2	Сложение двух векторов V1 и V2
V1−V2	V1 − V2	Вычитание двух векторов V1 и V2
−V	− V	Смена знака у элементов вектора V
−M	− M	Смена знака у элементов матрицы M
V−Z	V − Z	Вычитание скаляра Z из всех элементов вектора V
Z·V, V·Z	Z * V, V * Z	Скалярное умножение вектора V на скаляр Z
Z·M, M·Z	Z * M, M * Z	Скалярное умножение матрицы M на скаляр Z
V1·V2	V1 * V2	Скалярное умножение двух векторов V1 и V2
M·V	M * V	Скалярное умножение матрицы M на вектор V
M1·M2	M1 * M2	Скалярное умножение двух матриц M1 и M2
$\frac{V}{Z}$	V / Z	Деление всех элементов вектора V на скаляр Z
$\frac{M}{Z}$	M / Z	Деление матрицы M на скаляр Z

Запись	Клавиши	Описание
M−1	M ^ − 1	Обращение матрицы M
Mn	M ^ n	Возведение матрицы M в степень n
|V|	| V	Вычисление модуля вектора V
|M|	| M	Вычисление определителя матрицы
VT	V Ctrl !	Транспонирование вектора V
MT	M Ctrl !	Транспонирование матрицы M
V1×V2	V1 Ctrl * V2	Векторное умножение двух векторов V1 и V2
$\sum \vec{V}$	Alt $ V	Вычисление суммы элементов вектора V
$\vec{V}$	V Ctrl −	Векторизация вектора V
$\vec{M}$	M Ctrl −	Векторизация матрицы M
M<n>	M Ctrl ^ n	Выделение *n*-го столбца матрицы M
Vn	V [n	Выделение *n*-го элемента вектора V
Mm,n	M [(m,n)	Выделение элемента (m,n) матрицы M
M	M Ctrl T	Вставка рисунка, данные которого хранятся в матрице M
V, M	V ", M "	Получение комплексно-сопряженного вектора или матрицы

Следует отметить, что в некоторых операторах для ввода используется клавиша Ctrl, тогда как в более ранних версиях системы Mathcad использовалась клавиша Alt (в последней версии клавиша Alt предназначена для активизации строки меню). Все представленные выше операторы (кроме последнего) могут вызываться из палитры матричных операций.

8.7.3. Операция векторизации

Под необычным для нашей математической литературы понятием *векторизация* подразумевается одновременное проведение некоторой скалярной операции над всеми элементами вектора или матрицы, помеченными операторами векторизации. Это можно понимать и как возможность записи параллельных вычислений.

Векторизация может изменить смысл математических выражений и даже превратить недопустимое в первых версиях Mathcad выражение во вполне допустимое. Например, если V – вектор, то выражение cos(V) будет недопустимым, поскольку аргументом функции cos может быть только скалярная переменная. Однако с оператором векторизации функция cos(V) возвращает вектор, каждый элемент которого есть косинус соответствующего элемента исходного вектора V.

Начиная с Mathcad 8, в системы класса Mathcad было введено очередное усовершенствование – в качестве аргумента функции можно задавать векторы и матрицы. Таким образом, выражение cos(V), где V – вектор, становится допустимым и без применения оператора векторизации. Система Mathcad 8/2000/2001 стала более «интеллектуальной» – в подобных случаях векторизация выполняет-

ся автоматически. Таким образом, в нашем примере будет возвращен вектор, каждый элемент которого равен косинусу соответствующего элемента вектора V.

Векторизация осуществляется помещением соответствующего выражения под знак длинной стрелки. Если, к примеру, A и B – векторы, то A·B дает скалярное произведение этих векторов. Но то же произведение под знаком векторизации создает новый вектор, каждый j-й элемент которого есть произведение j-х элементов векторов A и B.

8.7.4. Векторные и матричные функции Mathcad

Mathcad поддерживает также ряд встроенных векторных и матричных функций, которые облегчают решение задач линейной алгебры и других сфер приложения векторов и матриц:

- length(V) – возвращает число элементов вектора;
- last(V) – возвращает номер последнего элемента;
- max(V) – возвращает максимальный по значению элемент вектора (или матрицы);
- min(V) – возвращает минимальный по значению элемент вектора (или матрицы);
- Re(V) – возвращает вектор действительных частей вектора с комплексными элементами;
- Im(V) – возвращает вектор мнимых частей вектора с комплексными элементами;
- ε(i,j,k) – единичный полностью антисимметричный тензор третьего ранга, при этом i, j и k должны быть целыми числами от 0 до 2 (или от ORIGIN до ORIGIN+2, если ORIGIN№№0); результат равен 0, если любые два аргумента равны, 1 – если три аргумента являются четной перестановкой (0, 1, 2), и –1, если три аргумента являются нечетной перестановкой (0, 1, 2).

Только для работы с матрицами также существует ряд встроенных функций:

- augment(M1,M2) – объединяет в одну две матрицы M1 и M2, имеющие одинаковое число строк (объединение идет «по горизонтали»);
- identity(n) – создает единичную квадратную матрицу размером n×n;
- stack(M1,M2) – объединяет «по вертикали» две матрицы M1 и M2, имеющие одинаковое число столбцов;
- submatrix(A,ir,jr,ic,jc) – возвращает подматрицу, состоящую из всех элементов, содержащихся в строках с ir по jr и столбцах с ic по jc (ir≤jr и ic≤jc);
- diag(V) – создает диагональную матрицу, элементы главной диагонали которой равны элементам вектора V;
- matrix(m,n,f) – создает матрицу, в которой (i,j)-й элемент равен f(i,j), где i=0, 1, … m и j=0, 1, … n; f(i,j) – некоторая функция;

- `Re(M)` – возвращает матрицу действительных частей матрицы `M` с комплексными элементами;
- `Im(M)` – возвращает матрицу мнимых частей матрицы `M` с комплексными элементами.

8.7.5. Функции, возвращающие специальные характеристики матриц

Следующие функции возвращают специальные характеристики матриц:

- `cols(M)` – возвращает число столбцов матрицы `M`;
- `rows(M)` – возвращает число строк матрицы `M`;
- `rank(M)` – возвращает ранг матрицы `M`;
- `tr(M)` – возвращает след (сумму диагональных элементов) квадратной матрицы `M`;
- `mean(M)` – возвращает среднее значение элементов массива `M`;
- `median(M)` – возвращает медиану элементов массива `M`;
- `cond1(M)` – возвращает число обусловленности матрицы, вычисленное в норме L1;
- `cond2(M)` – возвращает число обусловленности матрицы, вычисленное в норме L2;
- `conde(M)` – возвращает число обусловленности матрицы, вычисленное в норме евклидового пространства;
- `condi(M)` – возвращает число обусловленности матрицы, основанное на бесконечной норме;
- `norm1(M)` – возвращает норму L1 матрицы `M`;
- `norm2(M)` – возвращает норму L2 матрицы `M`;
- `norme(M)` – возвращает евклидову норму матрицы `M`;
- `normi(M)` – возвращает бесконечную норму матрицы `M`.

8.7.6. Примеры работы со средствами линейной алгебры СКМ Mathcad

Рассмотрим примеры использования наиболее распространенных векторных операторов. Они представлены на рис. 8.3.

Mathcad делает работу с векторами и матрицами столь же простой, как и с обычными числами и переменными. Это, безусловно, способствует проникновению векторных и матричных методов математических вычислений в практику научно-технических и иных расчетов. На рис. 8.4 показано начало документа с примерами матричных операций, а на рис. 8.5 – конец этого документа.

Теперь рассмотрим ряд примеров применения наиболее распространенных матричных функций. Примеры таких операций представлены на рис. 8.6 и 8.7.

РАБОТА С ВЕКТОРАМИ

$V := \begin{pmatrix} 1+2 \\ 4 \end{pmatrix}$ $U := V \cdot 3$ $U = \begin{pmatrix} 9 \\ 12 \end{pmatrix}$ Умножение вектора на константу

$V1 := \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$ $V2 := \begin{pmatrix} 3 \\ 4 \\ 5 \end{pmatrix}$ $V3 := \begin{pmatrix} 6 \\ 7 \\ 8 \end{pmatrix}$ $V1 + V2 - V3 = \begin{pmatrix} -2 \\ -1 \\ 0 \end{pmatrix}$ Задание и сложение трех векторов

$V3 := V1 \cdot V2$ $V3 = 26$ Умножение двух векторов

$V4 := V1 \times V2$ $V4 = \begin{pmatrix} -2 \\ 4 \\ -2 \end{pmatrix}$ Кросс-умножение двух трехэлементных векторов

$V := \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$ $VSUM := \sum V$ $VSUM = 6$ Суммирование элементов вектора V

$U := V^T$ $U = (1 \quad 2 \quad 3)$ Транспонирование вектора V

$U := \overrightarrow{\ln(V)}$ $U = \begin{pmatrix} 0 \\ 0.693 \\ 1.099 \end{pmatrix}$ $|V| = 3.742$ Вычисление нормы вектора V

Векторизация вектора V

$U_1 = 0.693$ Выделение элемента вектора U

length(V) = 3 last(V) = 2
max(V) = 3 min(V) = 1 Вычисление встроенных функций вектора V

Рис. 8.3. Примеры операций с векторами в системе Mathcad

ДЕЙСТВИЯ С МАТРИЦАМИ - 1

$A := \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ rows(A) = 2 cols(A) = 3 Задание матрицы A с размерностью 2x3

$B := A^T$ $B = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$ Транспонирование матрицы A

$M := identity(2)$ $M = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ Создание единичной матрицы M

tr(M) = 2 Вычисление следа матрицы

$A := \begin{pmatrix} 7 & 8 \\ 4 & 5 \end{pmatrix}$ $B := A^{-1}$ $B = \begin{pmatrix} 1.667 & -2.667 \\ -1.333 & 2.333 \end{pmatrix}$ Задание и обращение матрицы A

$C := B^{-1}$ $C = \begin{pmatrix} 7 & 8 \\ 4 & 5 \end{pmatrix}$ Повторное обращение восстанавливает исходную матрицу (матрица C = A)

$A := \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ $B := \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$ $A \cdot B = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$ Умножение двух матриц

$A := \begin{pmatrix} 2 & 5 & -3 \\ 1 & 4 & -1 \\ 1 & 3 & 2 \end{pmatrix}$ $D := |A|$ $D = 10$ Задание квадратной матрицы A и вычисление ее детерминанта

Рис. 8.4. Примеры операций с матрицами в системе Mathcad (начало документа)

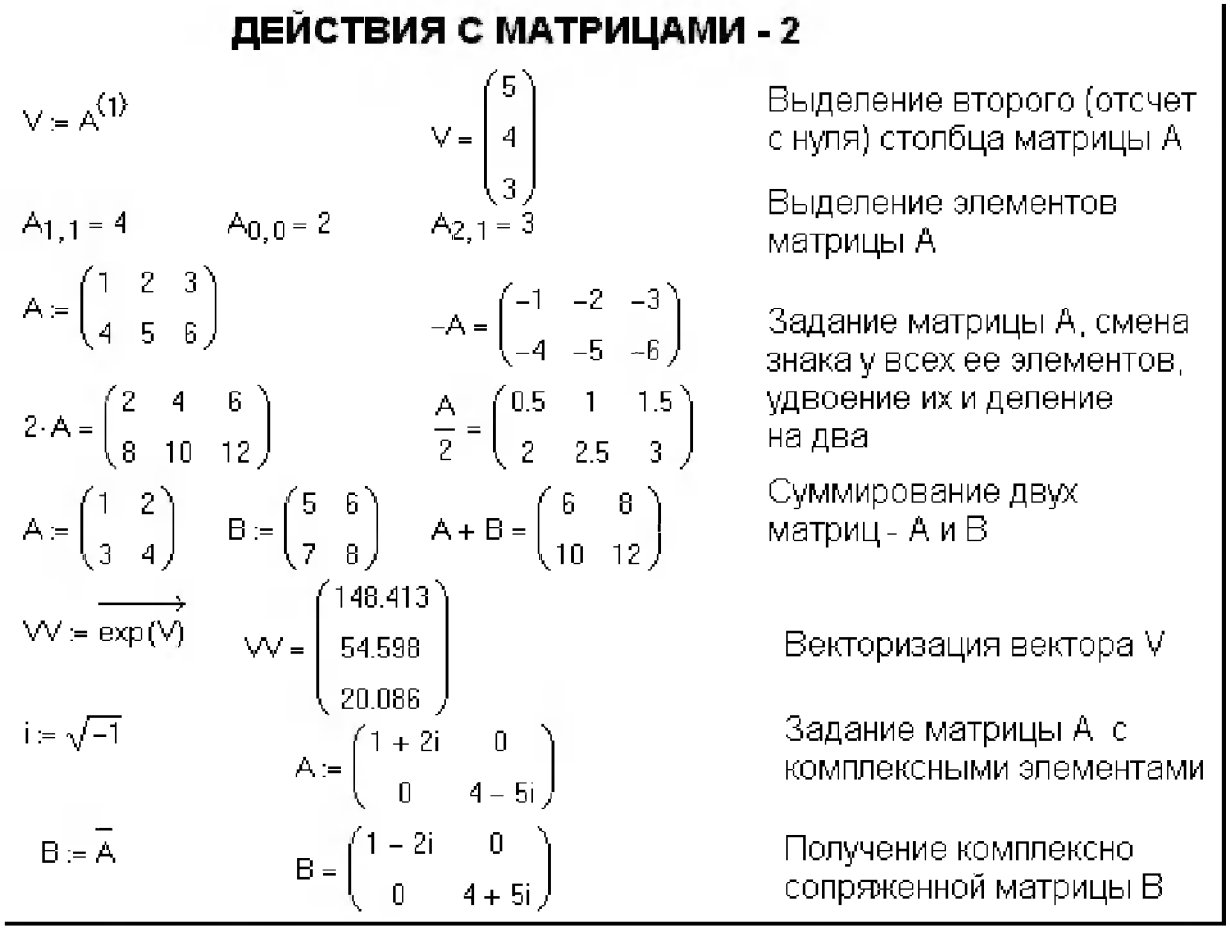


Рис. 8.5. Примеры операций с матрицами в системе Mathcad (конец документа)

Рис. 8.6. Примеры применения матричных функций Mathcad (конец документа)

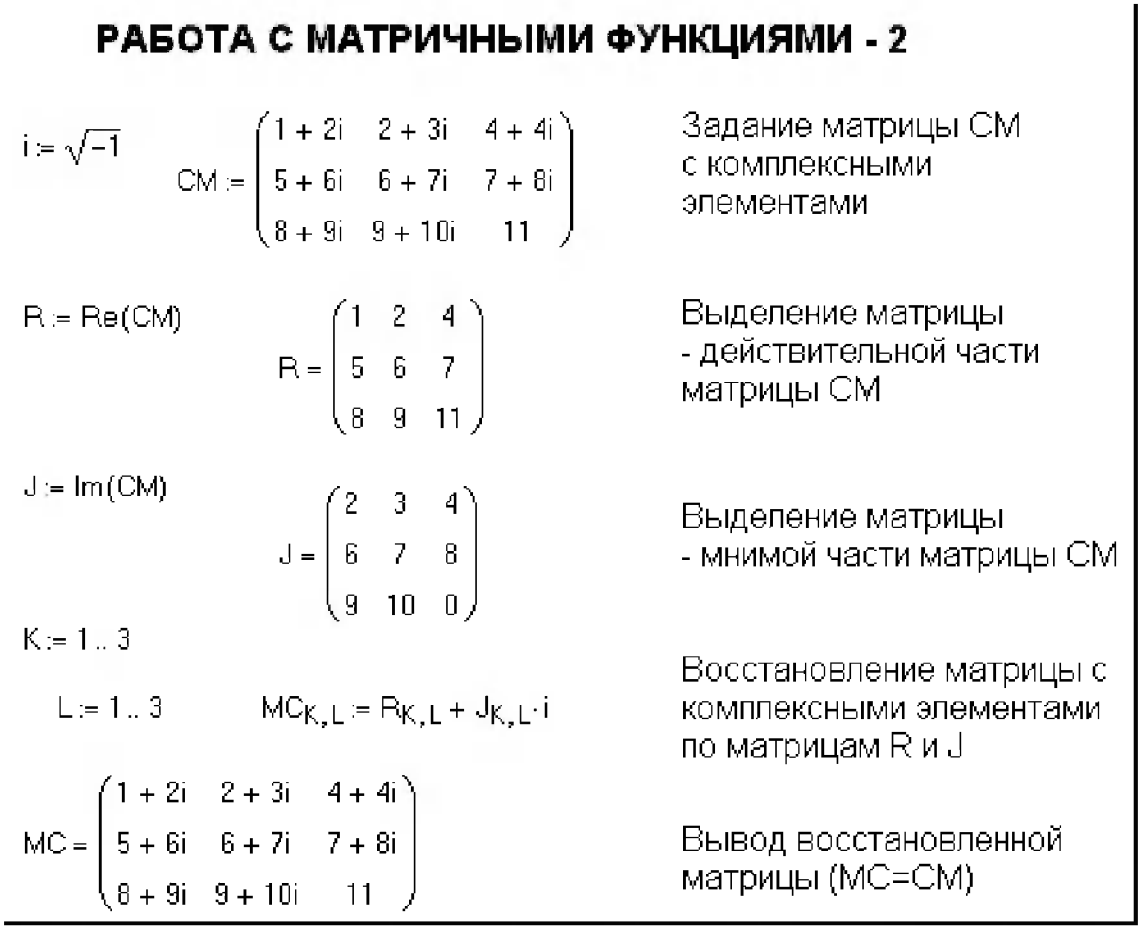


Рис. 8.7. Примеры применения матричных функций Mathcad (конец документа)

8.7.7. Дополнительные матричные функции Mathcad

В последние реализации системы Mathcad включен ряд дополнительных матричных функций. Они перечислены ниже:

- `eigenvals(M)` – возвращает вектор, содержащий собственные значения матрицы `M`;
- `eigenvec(M,Z)` – для указанной матрицы `M` и заданного собственного значения `Z` возвращает соответствующий этому собственному значению вектор;
- `eigenvecs(M)` – возвращает матрицу, столбцами которой являются собственные векторы матрицы `M` (порядок расположения собственных векторов соответствует порядку собственных значений, возвращаемых функцией `eigenvals`);
- `genvals(M,N)` – возвращает вектор обобщенных собственных значений `vi`, соответствующий решению уравнения `M·x = vi·N·x` (матрицы `M` и `N` должны быть вещественными);
- `genvals(M,N)` – возвращает матрицу, столбцы которой содержат нормированные обобщенные собственные векторы;

- `lu(M)` – выполняет треугольное разложение матрицы `M`: `P·M = L·U`, где `L` и `U` – соответственно нижняя и верхняя треугольные матрицы, `P` – матрица перестановки, все четыре матрицы квадратные, одного порядка;
- `qr(A)` – дает разложение матрицы `A`: `A = Q·R`, где `Q` – ортогональная матрица, а `R` – верхняя треугольная матрица;
- `svd(A)` – дает сингулярное разложение матрицы `A` размером `n×m`: `A = U·S·VT`, где `U` и `V` – ортогональные матрицы размером `m×m` и `n×n` соответственно, `S` – диагональная матрица, на диагонали которой расположены сингулярные числа матрицы `A`;
- `svds(A)` – возвращает вектор, содержащий сингулярные числа матрицы `A` размером `m×n`, где `m≥n`;
- `geninv(A)` – левая обратная к матрице `A`: `L·A = E`, где `E` – единичная матрица размером `n×n`, `L` – прямоугольная матрица размером `n×m`, `A` – прямоугольная матрица размером `m×n`.

8.7.8. Функции сортировки для векторов и матриц

Начиная с версии 3.0 для Windows, в системе Mathcad появились некоторые дополнительные функции сортировки – перестановка элементов векторов и матриц:

- `sort(V)` – сортировка элементов вектора в порядке возрастания их значений;
- `csort(M,n)` – перестановка строк матрицы `M` таким образом, чтобы отсортированным оказался *n*-й столбец;
- `rsort(M,n)` – перестановка столбцов матрицы `M` таким образом, чтобы отсортированной оказалась *n*-я строка.

С этими функциями часто используется функция `reverse(V)`, изменяющая порядок расположения элементов вектора на противоположный (начиная с конца).

8.7.9. Примеры применения дополнительных векторных и матричных функций

На рис. 8.8 представлены примеры, иллюстрирующие работу некоторых дополнительных векторных и матричных функций.

В целом видно, что запись матричных операций в системе Mathcad весьма наглядна, и это большое достоинство входного языка данной системы.

Рис. 8.8. Примеры применения дополнительных матричных функций

8.7.10. Решение в Mathcad систем линейных уравнений

Векторные и матричные операторы и функции системы Mathcad позволяют решать широкий круг задач линейной алгебры. К примеру, если заданы матрица `A` и вектор `B` для системы линейных уравнений в матричной форме `A·X = B`, то вектор решения можно получить из очевидного выражения `X = A-1·B`. Ниже приведен пример решения системы линейных уравнений, который иллюстрирует фрагмент документа, показанный на рис. 8.9.

Поскольку решение систем линейных уравнений – довольно распространенная задача, в Mathcad, начиная с шестой версии, введена встроенная функция `lsolve(A,B)`, которая возвращает вектор `X` для системы линейных уравнений `A·X = B` при заданной матрице коэффициентов `A` и векторе свободных членов `B`. Если уравнений `n`, размер вектора `B` должен быть равен `n`, а матрицы `A` – `n×n`. Пример применения этой функции дан в нижней части показанного на рисунке фрагмента документа Mathcad.

РЕШЕНИЕ СИСТЕМЫ ЛИНЕЙНЫХ УРАВНЕНИЙ A*X=B

$i := \sqrt{-1}$

$$A := \begin{pmatrix} 4+i & 0.24 & -0.08 \\ 0.09 & 3 & -0.15 \\ 0.04 & -0.08 & 4+i \end{pmatrix}$$

Матрица комплексных коэффициентов системы линейных уравнений

$$B := \begin{pmatrix} 8 \\ 9 \\ 20 \end{pmatrix}$$

Вектор свободных членов

$X := A^{(-1)} \cdot B$

Решение системы

$$X = \begin{pmatrix} 1.787 - 0.468i \\ 3.184 - 0.045i \\ 4.75 - 1.184i \end{pmatrix}$$

Результаты решения

$X1 := lsolve(A, B)$ $\quad X1 = \begin{pmatrix} 1.787 - 0.468i \\ 3.184 - 0.045i \\ 4.75 - 1.184i \end{pmatrix}$

Решение с применением функции lsolve

Рис. 8.9. Примеры решения систем линейных уравнений

8.8. Средства линейной алгебры СКМ Derive

8.8.1. Векторные функции и операторы Derive

Для задания вектора-столбца в системе Derive используется команда **Vector...** в позиции **Author** главного меню. При этом выводится окно, в котором задается число элементов вектора, то есть его размер – рис. 8.10.

После ввода размера появляется окно для ввода элементов вектора, показанное на рис. 8.11. В поля ввода можно вводить различные значения элементов вектора, например численные или символьные. После ввода они создают вектор в виде его элементов, заключенных в квадратные скобки, например `[1, a, a + b, c·SIN(1)]`. В качестве разделителя элементов используется запятая.

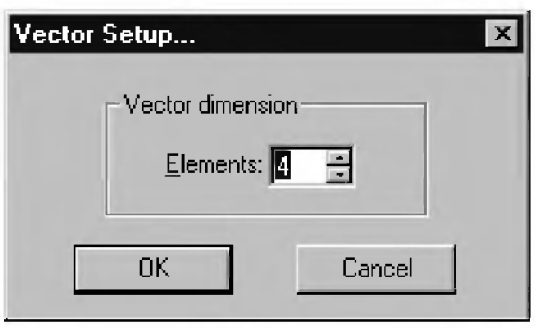


Рис. 8.10. Окно задания размера вектора в системе Derive

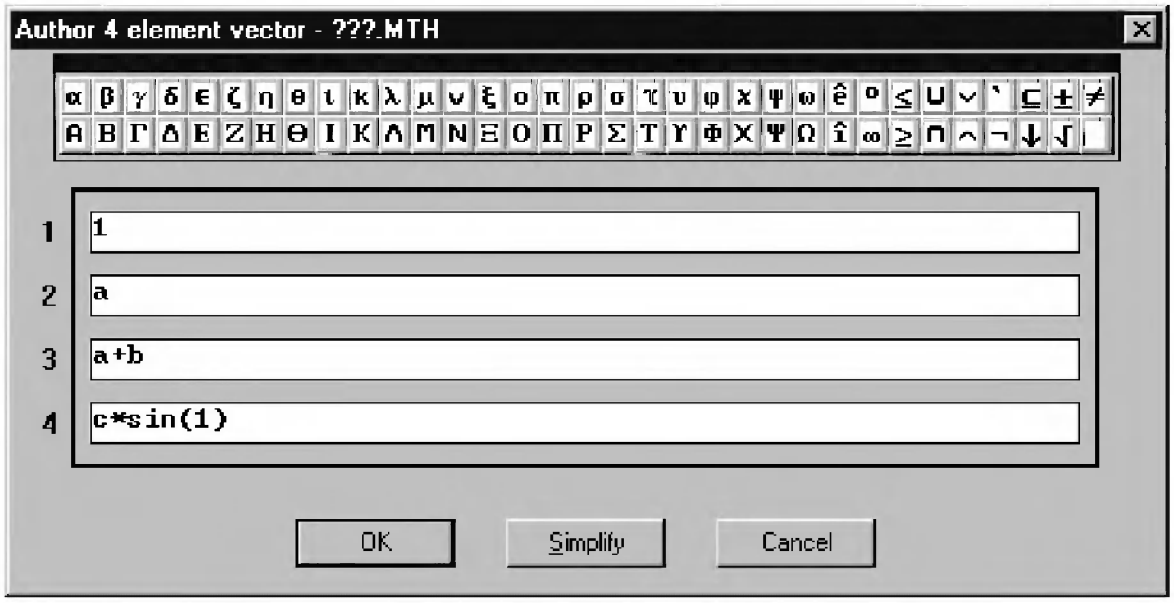


Рис. 8.11. Окно ввода значений элементов вектора

Можно также задавать векторы прямо в окне ввода Author. Векторные операции и функции системы Derive представлены достаточно скромным набором:

- **[x1,x2, . . .,xn]** – задание вектора **x** из n элементов;
- **VECTOR (u,k,n)** – задание вектора значений **u** при k, меняющемся от 1 до n с шагом 1;
- **VECTOR (u,k,m,n)** – задание вектора значений **u** при k, меняющемся от m до n с шагом 1;
- **VECTOR (u,k,m,n,s)** – задание вектора значений **u** при k, меняющемся от m до n с шагом s;
- **ELEMENT (v,n)** – выделение n-го элемента вектора **v**;
- **DIMENSION (v)** – задание размерности n вектора **v**;
- **v.w** – скалярное произведение векторов;
- **CROSS(v, w)** – векторное произведение двух трехэлементных векторов.

Выполнение этих операций проиллюстрируем следующими примерами:

```
1:   "Основные операции с векторами"
2:   "Задание вектора X с пятью элементами"
3:   x := [1, 2, 3, 4, 5]
4:   "Выделение третьего элемента вектора X"
5:   ELEMENT (x, 3)
6:   3
7:   "Определение размерности вектора"
8:   DIMENSION (x)
9:   5
10:  "Действие функций VECTOR"
11:  VECTOR (k y, k, 5)
12:  [y, 2 y, 3 y, 4 y, 5 y]
13:  VECTOR (k x, k, 3)
```

```
     / 1  2  3   4   5 \
     |                 |
14:  | 2  4  6   8  10 |
     |                 |
     \ 3  6  9  12  15 /
15:  VECTOR (k y, k, 3, 6)
16:  [3 y, 4 y, 5 y, 6 y]
17:  VECTOR (k 2, k, 0, 0.5, 0.1)
18:  [0, 0.2, 0.4, 0.6, 0.8, 1]
19:  "Задание вектора w"
20:  w := [6, 7, 8, 9, 10]
21:  "Произведение векторов"
22:  x.w
23:  130
24:  "Кросс-произведение векторов"
25:  CROSS (x, w)
26:  CROSS ([1, 2, 3, 4, 5], [6, 7, 8, 9, 10])
```

Функция **VECTOR**, генерирующая векторы, с позиций программирования частично заменяет циклы – как целочисленные, так и с нецелочисленной управляющей переменной k. Возможны необычные применения функции **VECTOR**, когда она используется для задания векторов и матриц, элементы которых являются производными различного порядка, значениями определенных интегралов с разными верхними пределами интегрирования, суммами рядов с разным числом членов и т. д.

Ниже приведены примеры таких вычислений:

```
1:   "Специальные приемы использования функции VECTOR"
2:   "Создание вектора из трех производных функции"
                  2  3
3:   Z (x, y) := x  y
            //d \k                \
4:   VECTOR ||-|  Z (x, y), k, 3|
            \\dx/                 /
     /     3     3   \
5:   \2 x y , 2 y , 0/
6:   "Создание матрицы из 6 производных функции"
                  4  3
7:   H (x, y) := x  y
            /       //d \m /d \k                \       \
8:   VECTOR |VECTOR ||-|  |-|  H (x,y),k,0,2|,m,1,2|
            \       \\dy/  \dx/                 /       /
     /    4  2      3  2      2  2 \
     | 3 x  y   12 x  y   36 x  y  |
9:   |                             |
     |     4          3          2   |
     \  6 x  y   24 x  y   72 x  y /
10:  "Вычисление определенного интеграла при разных b"
```

```
            /   b                                          \
            | /                                            |
11:  VECTOR | /   SQRT (2 x + 1) dx, b, 0.5, 1.5, 0.5|
            \   0                                          /
12:  [0.609475, 1.39871, 2.33333]
13:  "Вычисление сумм членов ряда с разным их числом"
            / k   1                    \
14:  VECTOR |SUM -, k, 100, 500, 100|
            \n=1  n                    /
15:  [5.18737, 5.87803, 6.28266, 6.56993, 6.79282]
```

Приведенные выше приемы применения функции **VECTOR** широко используются для вычисления ряда специальных математических функций по их разложениям в ряд.

8.8.2. Матричные функции и операторы Derive

Задание матрицы в системе Derive возможно различными способами. Самый простой – использование команды **Matrix...** в позиции **Author** главного меню. При этом вначале появляется окно ввода размера матрицы, показанное на рис. 8.12.

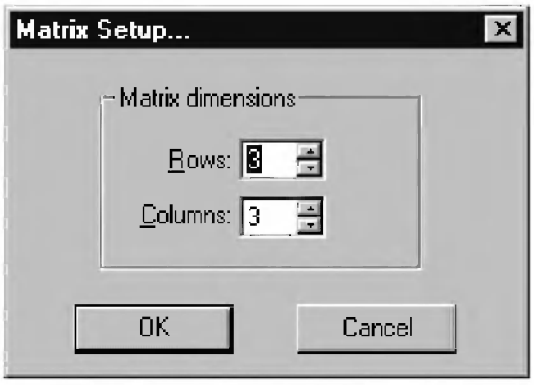


Рис. 8.12. Окно ввода размера матрицы

По умолчанию задается размер матрицы 3×3, но его можно сменить на нужный. После ввода числа строк и столбцов матрицы выводится окно задания ее элементов, показанное на рис. 8.13. В поле каждого элемента вводится его значение – численное или символьное.

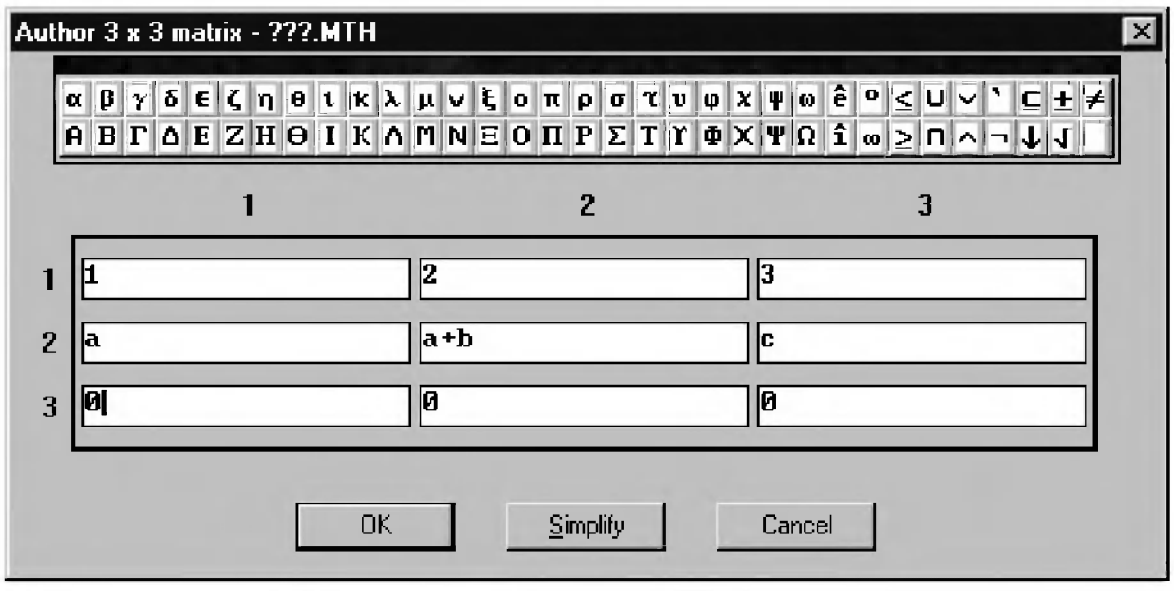


Рис. 8.13. Окно ввода элементов матрицы

Матрицу можно ввести как саму по себе, так и в виде значения, которое присваивается некоторой переменной. Матричные функции и операторы в Derive представлены тем минимумом, который необходим для решения типовых математических задач, связанных с применением матриц:

- **[[a,b],[c,d]]** – задание матрицы с размером 2×2;
- **IDENTITY_MATRIX(n)** – задание единичной матрицы n×n;
- **ELEMENT(A,j,k)** – выделение элемента $\mathbf{A}_{j,k}$ матрицы **A**;
- **A.B** – скалярное произведение матриц **A** и **B**;
- **A`** – транспонирование матрицы **A** (замена строк столбцами);
- **DET(A)** – вычисление детерминанта матрицы **A**;
- **TRACE(A)** – вычисление следа матрицы **A**;
- **A^-1** – инвертирование матрицы **A**;
- **ROW_REDUCE(A)** – ступенчатая форма матрицы **A**;
- **ROW_REDUCE(A,B)** – ступенчатая форма матрицы **A**, расширенная матрицей **B**;
- **CHARPOLY(A,mu)** – вычисление характеристического многочлена матрицы **A** с коэффициентами, заносимыми в вектор **mu**;
- **EIGENVALUES(A,mu)** – вычисление собственных значений матрицы **A** (заносятся в вектор **mu**).

Следующий пример иллюстрирует применение ряда матричных операций:

```
1:   "Основные операции с матрицами"
2:   "Задание матрицы 3*3"
          / 1  2  3 \
          |         |
3:   a := | 4  5  6 |
          |         |
          \ 7  8  1 /
4:   "Выделение элемента матрицы A2,3"
5:   ELEMENT (a, 2, 3)
6:   6
7:   "Вычисление детерминанта матрицы A"
8:   DET (a)
9:   24
10:  "Вычисление следа матрицы A"
11:  TRACE (a)
12:  7
13:  "Транспонирование матрицы A"
14:  a`
     / 1  4  7 \
     |         |
15:  | 2  5  8 |
     |         |
     \ 3  6  1 /
16:  "Инвертирование матрицы A"
      -1
17:  a
```

```
     / -1.79166   0.916666  -0.125 \
     |                             |
18:  |  1.58333  -0.833333   0.25  |
     |                             |
     \  -0.125      0.25    -0.125 /
19:  "Создание единичной матрицы 3*3"
20:  m := IDENTITY_MATRIX (3)
     / 1  0  0 \
     |         |
21:  | 0  1  0 |
     |         |
     \ 0  0  1 /
22:  "Скалярное произведение матриц"
23:  a.m
     / 1  2  3 \
     |         |
24:  | 4  5  6 |
     |         |
     \ 7  8  1 /
25:  "Характеристический полином матрицы a"
26:  CHARPOLY (a, p)
        3      2
27:  - p  + 7 p  + 66 p + 24
28:  "Собственные значения матрицы a"
29:  EIGENVALUES (a, v)
30:  [v = -5.07440]
31:  "Функция ROW_REDUCE"
32:  ROW_REDUCE (a)
     / 1  0  0 \
     |         |
33:  | 0  1  0 |
     |         |
     \ 0  0  1 /
34:  ROW_REDUCE (a, m)
     / 1  0  0  -1.79166   0.916666  -0.125 \
     |                                      |
35:  | 0  1  0   1.58333  -0.833333   0.25  |
     |                                      |
     \ 0  0  1   -0.125      0.25    -0.125 /
```

Для получения ступенчатой формы несингулярной матрицы **A** используется функция **ROW_REDUCE(A)**. Функция **ROW_REDUCE(A,B)** соединяет две матрицы и возвращает ступенчатую форму для расширенной матрицы.

Эти функции можно использовать для решения матричных уравнений вида **A·X=B.**

Например, если **A** и **B** матрицы

```
|1 2|   и   |3 4|
|5 6|       |7 8|,
```

то применение **ROW_REDUCE(A,B)** упрощается к матрице

```
|1  0 -1 -2|
|0  1  2  3|
```

то есть решение есть

```
|-1 -2|
| 2  3|.
```

Таким образом, мы разом решили две системы линейных уравнений с одной матрицей коэффициентов **A**, но с двумя векторами свободных членов, образующих матрицу **B**.

8.8.3. Матричные операции Derive в символьной форме

При упрощении и нормализации выражений, содержащих векторные и матричные операторы, Derive использует следующие соотношения векторной и матричной алгебры:

```
a`` -> a
          1
[1/a]` -> -
          a`
(a+b)` -> a` + b`
(a.b)` -> b` . a`
a.(b+c) -> a.b + a.c
(b+c).a -> b.a + c.a
(a b).c -> a.(b.c)
 1     1   1
- -> - . -
a.b    b   a
               1
DET [1/a] -> ---
             DET a
DET (a.b) -> DET (a) DET (b)
```

Примеры выполнения матричных операций в символьном виде даны ниже:

```
1:   "Символьные операции с матрицами"
2:   "Задание матрица в символьном виде"
          / a  b \
3:   m := |      |
          \ c  d /
4:   "Транспонирование матрицы"
5:   m`
     / a  c \
6:   |      |
     \ b  d /
7:   "Инвертирование матрицы"
```

```
      -1
8:   m
     /       d                b       \
     |  ----------- -  ----------- |
     |   a d - b c       a d - b c  |
9:   |                              |
     |        c             a       |
     | - -----------    ----------- |
     \    a d - b c     a d - b c   /
10:  "Задание вектора"
          / e \
11:  v := |   |
          \ f /
12:  "Решение системы линейных уравнений m*x=v"
13:  "Используйте далее команду Simplify"
           v
14:  x := --
           m
           /       d                b       \
           |   ----------   -  ---------- |
     / e \ |    a d - b c       a d - b c  |
15:  |   || |                              |
     \ f /| |        c             a       |
          | - ----------     ----------- |
          \   a d - b c       a d - b c  /
16:  "Вычисление детерминанта матрицы m"
17:  DET (m)
18:  a d - b c
19:  "Вычисление следа матрицы m"
20:  TRACE (m)
21:  a + d
22:  "Задание матрицы с элементами - выражениями"
          /    1         2 z    \
          | --------  -------- |
          |   2          2      |
          |  z  - 3     z  - 3  |
23:  ms := |                     |
          |     7          6    |
          | --------  -------- |
          \  z - 2      z - 5   /
24:  "Вычисление детерминанта матрицы ms"
25:  DET (ms)
26:  "Используйте команду Simplify"
                2
          2 (7 z  - 38 z + 6)
27:  ----------------------------
               2
          (z  - 3) (z - 2) (z - 5)
```

К сожалению, символьные операции возможны лишь с матрицами малого размера – в противном случае символьные представления результатов оказываются настолько громоздкими, что практическая польза от них становится сомнительной. Для работы с ними их лучше распечатать принтером, чем созерцать на экране дисплея в виде отдельных фрагментов (поскольку целиком большие выражения в окно просмотра не помещаются).

> ***Внимание!*** *Векторные и матричные операции системы Derive дают тот минимум средств линейной алгебры, который достаточен для решения ее типовых задач. Вряд ли стоит рекомендовать эти системы для решения больших и серьезных задач в этой области, поскольку это потребует чрезмерно больших усилий по созданию библиотечных и иных расширений систем.*

8.9. Средства линейной алгебры СКМ MuPAD

8.9.1. Библиотеки линейной алгебры системы MuPAD

Символьные операции линейной алгебры в системе MuPAD реализованы с помощью библиотеки linalg. Для получения информации о включенных в эту библиотеку функциях надо использовать следующую команду:

```
info(linalg);

Library 'linalg': the linear algebra package

-- Interface:
linalg::addCol,              linalg::addRow,
linalg::adjoint,             linalg::angle,
linalg::basis,               linalg::charmat,
linalg::charpoly,            linalg::col,
linalg::companion,           linalg::concatMatrix,
linalg::crossProduct,        linalg::curl,
linalg::delCol,              linalg::delRow,
linalg::det,                 linalg::divergence,
linalg::eigenvalues,         linalg::eigenvectors,
linalg::expr2Matrix,         linalg::factorCholesky,
linalg::factorLU,            linalg::factorQR,
linalg::frobeniusForm,       linalg::gaussElim,
linalg::gaussJordan,         linalg::grad,
linalg::hermiteForm,         linalg::hessenberg,
linalg::hessian,             linalg::hilbert,
```

```
linalg::intBasis,          linalg::inverseLU,
linalg::invhilbert,        linalg::isHermitean,
linalg::isPosDef,          linalg::isUnitary,
linalg::jacobian,          linalg::jordanForm,
linalg::laplacian,         linalg::matdim,
linalg::matlinsolve,       linalg::matlinsolveLU,
linalg::minpoly,           linalg::multCol,
linalg::multRow,           linalg::ncols,
linalg::nonZeros,          linalg::normalize,
linalg::nrows,             linalg::nullspace,
linalg::ogCoordTab,        linalg::orthog,
linalg::permanent,         linalg::pseudoInverse,
linalg::randomMatrix,      linalg::rank,
linalg::row,               linalg::scalarProduct,
linalg::setCol,            linalg::setRow,
linalg::smithForm,         linalg::stackMatrix,
linalg::submatrix,         linalg::substitute,
linalg::sumBasis,          linalg::swapCol,
linalg::swapRow,           linalg::sylvester,
linalg::tr,                linalg::transpose,
linalg::vandermondeSolve,  linalg::vecdim,
linalg::vectorPotential,   linalg::wiedemann
```

Назначение большинства функций очевидно уже из их названий, например: AddCol – добавление столбца, det – вычисление детерминанта матрицы, hermiteForm – представление матрицы в Эрмитовой форме, linearSolve – решение системы линейных уравнений и т. д. Для детального знакомства с той или иной функцией нужно использовать команду

?linalg::name_function,

где name_function – имя функции, по которой ищется справка. Эта команда открывает окно справочной системы для указанной функции.

Для численных расчетов можно использовать библиотеку numeric:

```
info(numeric);
Library 'numeric': functions for numerical mathematics
Interface:
numeric::bairstow,         numeric::bisect,
numeric::butcher,          numeric::det,
numeric::eigenvalues,      numeric::eigenvectors,
numeric::factorCholesky,   numeric::factorLU,
numeric::factorQR,         numeric::fint,
numeric::fsolve,           numeric::gldata,
numeric::gtdata,           numeric::inverse,
numeric::minpoly,          numeric::ncdata,
numeric::newton,           numeric::odesolve,
numeric::quadrature,       numeric::singularvalues,
numeric::singularvectors,  numeric::vonMises
```

Как нетрудно заметить, в ее составе также есть ряд матричных функций.

Внимание! *Библиотека функций линейной алгебры даже системы MuPAD 2.5.2 содержит 74 функции, охватывающие практически все средства, необходимые для решения подавляющего большинства задач линейной алгебры. Тем не менее по этим средствам MuPAD уступает более продвинутым системам – Mathematica, Maple и тем более MATLAB. Для ряда операций в численном виде следует использовать библиотеку numeric, также содержащую ряд функций линейной алгебры.*

8.9.2. Задание векторов и матриц в MuPAD

Для создания векторов и матриц в MuPAD служит определитель – домен **Dom::Matrix()** с различными описаниями в круглых скобках. Примеры его применения даны ниже:

```
A:=Dom::Matrix()([[a,b],[c,d]]):
B:=Dom::Matrix()([[1,2],[3,4]]):
R:=DomMatrix(Dom::Rational)([[1/2,1/3],[1/5,1/7]]):
C:=DomMatrix(Dom::Complex)([[I,1],[2,3+I]]):
S:=DomMatrix(Dom::ExpressionField)([[sin(x),x^2],[exp(x),ln(x)]]):
```

Ввиду очевидности этих форм задания матриц (и векторов) ограничимся их приведением без детального обсуждения. Заметим, что возможно раздельное задание определения матрицы и ее последующего описания, например:

```
M:=Dom::SquareMatrix():
A:=M([[a,b],[c,d]]):
```

В этом случае создается объект типа матрицы – **M**, а затем уже наследующая его свойства матрица **A**. Более подробное описание задания векторов и матриц можно найти в справочной системе MuPAD.

Внимание! *MuPAD имеет обширный набор функций для работы с матрицами и векторами, часть из которых задается как свойства домена – матрицы. Следует отметить, что большая часть символьных операций, например интегрирование, дифференцирование, расширение, вычисление корней и т. д., доступны при их аргументах в виде векторов и матриц. При этом обычно операции выполняются поэлементно.*

8.10. Линейная оптимизация и линейное программирование

8.10.1. Постановка задачи линейного программирования

К числу массовых задач относятся задачи линейного программирования и линейной оптимизации. В общем случае задача линейного программирования может быть сформулирована следующим образом: нужно найти максимум или минимум *целевой функции*

$$f(X)=\sum_{j=1}^{n} c_j \cdot x_j \qquad (11.6)$$

при ограничениях

$$\sum_{j=1}^{n} a_{ij} \cdot x_j = b_i,\ x_j \ge 0,\ b_i \ge 0,\ i=1..m,\ j=1..n. \qquad (11.7)$$

Среди универсальных методов решения задач линейного программирования наиболее распространен итерационный *симплексный метод*. Симплекс – это многоугольник, перемещаемый в пространстве решения таким образом, чтобы, постепенно сужаясь, он мог охватить точку искомого решения, отвечающую решению задачи с заданной погрешностью.

8.10.2. Обзор средств пакета simplex СКМ Maple

В пакете simplex системы Maple имеется небольшой, но достаточно представительный набор функций и определений для решения задач линейной оптимизации и программирования. Он вызывается командой

```
> with(simplex);
```

и выводит список функции пакета. Они описаны ниже.

8.10.3. Переопределенные функции maximize и minimize

Главными из этих функций являются `maximize` и `minimize`, оптимизирующие задачу *симплекс-методом*. Они записываются в следующих формах:

```
maximize(f, C)                    minimize(f, C)
maximize(f , C, vartype)          minimize(f , C, vartype)
maximize(f , C, vartype,          'NewC', 'transform')
minimize(f , C, vartype,          'NewC', 'transform')
```

Здесь `f` – линейное выражение, `C` – множество или список условий, `vartype` – необязательно задаваемый тип переменных `NONNEGATIVE` или `UNRESTRICTED`, `NewC` и `transform` – имена переменных, которым присваиваются соответственно оптимальное описание и переменные преобразования. Ниже даны примеры применения этих функций:

```
> minimize( x-y, {4*x+2*y <= 10, 3*x+4*y <= 16},
NONNEGATIVE,'NC','vt' );
```

$$\{x=0, y=4\}$$

```
> NC;vt;
```

$$\{y=-\frac{1}{4}\_SL2+4-\frac{3}{4}x, \_SL1=2-\frac{5}{2}x+\frac{1}{2}\_SL2\}$$

$$\{\ \}$$

```
> maximize( x+y, {4*x+2*y <= 10, 3*x+4*y <= 16}, NONNEGATIVE );
```

$$\{x = \frac{4}{5}, y = \frac{17}{5}\}$$

```
> maximize( x+y, {3*x+2*y <= 5, 2*x+4*y <= 4 } );
```

$$\{x = \frac{3}{2}, y = \frac{1}{4}\}$$

```
> z :=  2*x1 - x2 + 3*x3;
```

$$z := 2x1 - x2 + 3x3$$

```
> cnts1 :=[ x2+2*x3 <= 1, 2*x1 - 4*x2 + 6*x3 <= 3, -x1 + 3*x2 +
4*x3 <= 12 ];
```

$$cnts1 := [x2 + 2x3 \le 1, 2x1 - 4x2 + 6x3 \le 3, -x1 + 3x2 + 4x3 \le 12]$$

```
> sol1 := maximize(z,cnts1,NONNEGATIVE);
```

$$sol1 := \{x3 = 0, x2 = 1, x1 = \frac{7}{2}\}$$

При использовании функций `minimize` и `maximize` надо не забывать, что это переопределенные функции – аналогичные по названию функции есть в ядре, но они реализуют иные методы вычислений. Для возврата к исходному определению функций надо выполнить команду `restart`.

8.10.4. Прочие функции пакета simplex СКМ Maple

Функция `basis(C)` возвращает *базис* для системы линейных уравнений `C`. Например:

```
> basis( [ x = 2*z+w , z = 2*y - w ] );
```

$$[x, z]$$

Функция `convexhull(ps)` возвращает выпуклую оболочку множества точек `ps`. Например:

```
> convexhull({[0,0],[1,1],[2,-1],[1,1/3],[1,1/2]});
```

$$[[0, 0], [2, -1], [1, 1]]$$

Для определения констант для системы линейных уравнений или неравенств служит функция `cterm(C)`:

```
> cterm([2*x+y<=6,7*y-z-3=4]);
```

$$[6, 7]$$

Функция `define_zero(C)` возвращает ближайшее ненулевое значение, зависящее от установки переменной `Digits`:

```
> define_zero();
```

$$.1000000000\ 10^{-7}$$

```
> Digits:=40;
```

$$Digits := 40$$

```
> define_zero();
```

$$.1000000000000000000000000000000000000000\ \ 10^{-37}$$

```
> define_zero(1*10^(-10));
```

$$.1000000000000000000000000000000000000000 \quad 10^{-9}$$

Функция `display©` имеет еще и форму `display(C,[x, y, z])`. Она задает вывод линейных уравнений и неравенств в матричной форме:

```
> display({2*x+5*y-z<= 0, 2*w-4*y-z<=2});
```

$$\begin{bmatrix} 0 & -4 & 2 & -1 \\ 2 & 5 & 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \\ z \end{bmatrix} \le \begin{bmatrix} 2 \\ 0 \end{bmatrix}$$

Функция `dual(f, C, y)` имеет следующие параметры: `f` – линейное выражение, `C` – множество неравенств и `y` – имя. Эта функция возвращает сопряженное с `f` выражение:

```
> dual( x-y,{2*x+3*y<=5,3*x+6*y<=15},z);
```

$$15z1 + 5z2, \{1 \le 3z1 + 2z2, -1 \le 6z1 + 3z2\}$$

Функция `feasible` может быть задана в трех формах:

```
feasible(C)          feasible(C,vartype)
feasible(C,vartype,'NewC','Transform')
```

Здесь параметр `vartype` может иметь значения `NONNEGATIVE` или `UNRESTRICTED`. Эта функция определяет систему как осуществимую или нет:

```
> feasible({2*x+3*y<=5, 3*x+6*y<=15}, NONNEGATIVE);
```

$$true$$

```
> feasible({2*x+3*y<=5, 3*x+6*y<=-15}, NONNEGATIVE);
```

$$false$$

Если функция возвращает логическое значение `true`, то заданная система осуществима, а если `false` – неосуществима, то есть ни при каких значениях переменных не способна удовлетворить записанным неравенствам и равенствам.

Функция `pivot(C, x, eqn)` конструирует новую систему с заданным главным элементом:

```
> pivot({_SL1=5-4*x-3*y,_SL2=4-3*x-4*y},x,[_SL1=5-4*x-3*y]);
```

$$\{x = -\frac{1}{4}\_SL1 + \frac{5}{4} - \frac{3}{4}y, \_SL2 = \frac{1}{4} + \frac{3}{4}\_SL1 - \frac{7}{4}y\}$$

Функция `pivoteqn(C, var)` возвращает подсистему для заданного диагонального элемента `C`:

```
> pivoteqn( {_SL1 = 5-3*x-2*y, _SL2 = 4-2*x-2*y}, x );
```

$$[\_SL1 = 5 - 3x - 2y]$$

Функция `pivotvar(f, List)` или `pivotvar(f)` возвращает список переменных, имеющих положительные коэффициенты в выражении для целевой функции:

```
> pivotvar( x1-2*x2+3*x3-x4);
```

$$x1$$

```
> pivotvar( x1 + 2*x3 – 3*x4 , [x4,x3,x1] );
```

$$x3$$

Функция `ratio(C, x)` возвращает список отношений, задающих наиболее жесткие ограничения:

```
> ratio( [_SL1=10-3*x-2*y,_SL2=8-2*x-4*y],x );
```

$$\left[\frac{10}{3}, 4\right]$$

Функция `setup` может иметь три формы:

```
setup(C) setup(C, NONNEGATIVE) setup(C, NONNEGATIVE, 't')
```

Она обеспечивает конструирование множества уравнений с переменными в левой части:

```
> setup( {2*x+3*y<=5,3*x+5*y=15});
```

$$\{\_SL1 = -5 + \frac{1}{3}y, x = -\frac{5}{3}y + 5\}$$

Последняя функция – `standartize(C)` – конвертирует список уравнений (неравенств) в неравенства типа «меньше или равно»:

```
> standardize({2*x+3*y<=5,3*x+5*y=15});
```

$$\{2x + 3y \le 5, 3x + 5y \le 15, -3x - 5y \le -15\}$$

8.11. Средства оптимизации в СКМ Mathematica 4/5

8.11.1. Поиск глобального максимума и минимума

В системах Mathematica 4/5 имеются следующие две функции для поиска глобального максимума и минимума с ограничениями для аналитически заданной функции:

- **ConstrainedMax[f, {inequalities}, {x, y, ...}]** – ищет глобальный максимум функции f в области, определяемой неравенствами inequalities. Полагается, что все переменные x, y, ... неотрицательны.
- **ConstrainedMin[f, {inequalities}, {x, y,...}]** – ищет глобальный минимум функции f в области, определяемой неравенствами inequalities. Все переменные x, y, ... полагаются неотрицательными.

После имени функции указывается целевая функция, затем указываются все ограничения и, наконец, список искомых переменных. Результатом вычислений являются стоимость продукции и количество изделий, представленные списком.

Рассмотрим типичный пример на линейное программирование. Пусть цех малого предприятия должен изготовить 100 изделий трех типов, причем каждое не менее 20 штук. На изготовление этих изделий уходит соответственно 4, 3.4 и 2 кг

металла при его общем весе 700 кг. Спрашивается, сколько изделий x1, x2 и x3 каждого типа надо выпустить для обеспечения максимальной стоимости продукции, если цена каждого из изделий равна соответственно 4, 3 и 2 рубля. Ниже представлено решение этой задачи с помощью функции **ConstrainedMax**:

```
ConstrainedMax[
                  4*x1+3*x2+2*x3,
                  {x1>=20,x2>=20,x3>=20,
                  4*x1+3.4*x2+2*x3<=340,
                  4.75*x1+11*x2+2*x3<=700,
                  x1+x2+x3==100},
                  {x1,x2,x3}]
  {332.,{x1→56.,x2→20.,x3→24.}}
```

8.11.2. Функции линейного программирования

Две описанные выше функции решают типовые задачи линейного программирования. В дополнение к ним может использоваться функция, специально созданная для решения таких задач:

- **LinearProgramming[c, m, b]** – ищет вектор **x**, минимизирующий величину **c.x** в соответствии с условиями **m.x >= b** и **x >= 0**.
- **LinearProgramming[c, m, aaa, aa, aa, aa, j a]** – ищет вектор **x**, минимизирующий величину **c.x** в соответствии с линейными ограничениями, заданными в матрице m, и парами **aa**, **aa**. Для каждой строки m задается соответствие условию ограничения в виде **a if a**, или **a == a**, если **a == 0**, или **a**, если **a**.
- **LinearProgramming[c, m, b, l]** – минимизирует **c.x** с ограничениями, заданными в **m** и **b** и **a**.
- **LinearProgramming[c, m, b, aa, a, j a]** – минимизирует c.x с ограничениями, заданными в **m** и **b** и **a**.
- **LinearProgramming [c, m, b, aaa, aa, aa, aa, j a]** – минимизирует c.x с ограничениями, заданными в **m** и **b** и **a**.

Ниже даны примеры решения задачи линейного программирования функциями **ConstrainedMin** и **LinearProgramming**:

```
ConstrainedMin[2x-3y, {x+y<12, x-y>1, x+2y==14, x>1},
 {x, y}]
```

$$\left\{-\frac{7}{3}, \left\{x \to \frac{16}{3}, y \to \frac{13}{3}\right\}\right\}$$

```
LinearProgramming[{2, -3}, {{1, 1}, {1, -1}, {1, 2}, {1, 0}}, {{12, -1}, {1, 1}, {14, 0}, {1, 1}}]
```

$$\left\{\frac{16}{3}, \frac{13}{3}\right\}$$

Задачи минимизации традиционно относятся к сложным задачам программирования – особенно при поиске глобального минимума. Наличие в ядре системы Mathematica их реализаций делает систему привлекательной для решения задач этого класса.

8.11.3. Новые функции оптимизации в Mathematica 5

В версии Mathematica 5 средства оптимизации были существенно переработаны и дополнены. К функции **FindMinimum** добавлена новая функция **FindMaximum** с аналогичным **FindMinimum** синтаксисом, которая обеспечивает поиск локального максимума функции f одной или ряда переменных. Рекомендуется просмотреть по справке возможные опции этих функций.

В Mathematica 5 включены новые функции для символьного и численного поиска максимумов и минимумов функций ряда переменных:

- **Maximize[f, {x, y, j }]** – ищет **x, y, j** для максимума функции **f**;
- **Maximize[{f, cons}, {x, y, j }]** – ищет максимум функции **f** с ограничениями **cons**;
- **Minimize[f, {x, y, j }]** – ищет **x, y, j** для минимума функции **f**;
- **Minimize[{f, cons}, {x, y, j }]** – минимизирует функцию **f** с ограничениями **cons**.

Примеры применения этих функций представлены ниже:

```
Minimize[(Sqrt[5 + x^2])/2 + (5 - x)/4, x]
```

$$\left\{\frac{1}{4}\left(5+\sqrt{15}\right),\ \left\{x\to\sqrt{\frac{5}{3}}\right\}\right\}$$

```
Minimize[x^2 - y^3, x^4 + y^4 <= 2, {x, y}]
```

```
{Root[-8 + #1^4 &, 1], {y → Root[-2 + #1^4 &, 2], x → 0}}
```

```
Minimize[-2a+7b+c+9d,{6a-
 b+cJ12,a+5bJ3,a+5b+d==5,ai0,bi0,ci0,di0},{a,b,c,d}]
```

$$\left\{\frac{474}{31},\ \left\{a\to\frac{63}{31},\ b\to\frac{6}{31},\ c\to 0,\ d\to 2\right\}\right\}$$

```
Minimize[e^x + 1.2/x, 1 <= x <= 2, x]
```

```
{3.91828,{x→1.}}
```

```
Maximize[x*e^-x, x]
```

$$\left\{\frac{1}{e},\ \{x\to 1\}\right\}$$

```
Maximize[Sin[x],0<x<2,x]
```

$$\left\{1,\ \left\{x\to\frac{\pi}{2}\right\}\right\}$$

Для численного вычисления максимумов и минимумов в Mathematica 5 введены также функции:

- **NMaximize[f, {x, y, j }]** – ищет **x, y, j** для максимума функции **f**;
- **NMaximize[{f, cons}, {x, y, j }]** – ищет максимум функции **f** с ограничениями **cons**;
- **NMinimize[f, {x, y, j }]** – ищет **x, y, j** для минимума функции **f**;
- **NMinimize[{f, cons}, {x, y, j }]** – минимизирует функцию **f** с ограничениями **cons**.

Ниже представлены примеры применения этих функций:

```
NMinimize[Sqrt[5+x^2]/2 + (5-x)/4, x]
  {2.21825,{x→1.29099}}
NMinimize[{x^2-y^3, x^4+y^4<=2}, {x, y}]
  {-1.68179,{x→0.,y→1.18921}}
NMinimize[{-2a+7b+c+9d,6a-b+cJ12,a+5bJ3,
a+5b+d==5,ai0,bi0,ci0,di0},{a,b,c,d}]
 {15.2903,{a→2.03226,b→0.193548,c→0.,d→2.}}
NMinimize[{E^x + 1.2/x, 1<=x<=2}, x]
 {3.91828,{x→1.}}
NMaximize[x*E^-x, x]
 {0.367879,{x→1.}}
NMaximize[{Sin[x],0<x<2},x]
 {1.,{x→1.5708}}
```

Функции **ConstrainedMax** и **ConstrainedMax** в Mathematica 5 удалены из списка функций оптимизации. Точнее говоря, они скрыты. Ниже представлено решение рассмотренной выше задачи на максимизацию выпуска продукции малым предприятием с помощью функции **ConstrainedMax** в системе Mathematica 5:

```
ConstrainedMax[
                  4*x1+3*x2+2*x3,
                  {x1>=20,x2>=20,x3>=20,
                  4*x1+3.4*x2+2*x3<=340,
                  4.75*x1+11*x2+2*x3<=700,
                  x1+x2+x3==100},
                  {x1,x2,x3}]
ConstrainedMax::deprec:
 ConstrainedMax is deprecated and will not be
   supported in future versions of Mathematica.
   Use NMaximize or Maximize instead. More...
  {332.,{x1→56.,x2→20.,x3→24.}}
```

В приведенном примере Mathematica 5 выдает сообщение о том, что функция **ConstrainedMax** отнесена к неперспективным и в последующих версиях может не поддерживаться. Вместо нее рекомендуется применять функцию **NMaximize** или

Maximize. Тем не менее в Mathematica 5 функция ConstrainedMax работает и решает представленную задачу верно.

Примеры решения данной задачи с помощью более предпочтительных для Mathematica 5 функций **Maximize** и **NMaximize** даны ниже:

```
Maximize[
                    4*x1+3*x2+2*x3,
                    {x1>=20,x2>=20,x3>=20,
                    4*x1+3.4*x2+2*x3<=340,
                    4.75*x1+11*x2+2*x3<=700,
                    x1+x2+x3==100},
                    {x1,x2,x3}]
  {332.,{x1→56.,x2→20.,x3→24.}}
NMaximize[
                    {4*x1+3*x2+2*x3,
                    x1>=20,x2>=20,x3>=20,
                    4*x1+3.4*x2+2*x3<=340,
                    4.75*x1+11*x2+2*x3<=700,
                    x1+x2+x3==100},
                    {x1,x2,x3}]
  {332.,{x1→56.,x2→20.,x3→24.}}
```

8.12. Оптимизация и линейное программирование в системе Mathcad

8.12.1. Решение задач линейного программирования

Функции Maximize и Minimize системы Mathcad могут успешно применяться при решении задач линейного программирования, которые широко используются в экономических и производственных расчетах [138]. Рассмотрим такую задачу. Фрагмент документа Mathcad, дающий постановку задачи и реализующий ее решение, представлен на рис. 8.14.

Очевидно, что введение в Mathcad (начиная с версии 8.0) новых функций оптимизации `Maximize` и `Minimize` расширяет круг решаемых системой задач при минимальных затратах времени на подготовку средств их решения. Функция `Find` также может использоваться для решения задач линейного программирования, пример чему приведен на рис. 8.15.

Здесь интересно отметить, что результат получен в форме массива (вектора), вложенного в другой массив. Это указывает на множественность (в нашем случае тройственность) решения.

Решение типовой задачи линейного программирования

Пусть цех малого предприятия должен изготовить 100 изделий трех типов. Каждого изделия нужно сделать не менее 20 штук. На изделия уходят соответственно 4, 3.4 и 2 кг метапла при его общем запасе 340 кг, а также по 4.75, 11 и 2 кг пластмассы при ее общем запасе 700 кг. Сколько изделий каждого типа x1, x2 и x3 надо выпустить для получения максимального объема выпуска в денежном выражении, если цена изделий составляет по калькуляции 4, 3 и 2 рубля? Итак, задача сводится к вычислению максимума функции

$$f(x1,x2,x3) := 4\cdot x1 + 3\cdot x2 + 2\cdot x3$$

Решение данной задачи имеет следующий вид

$x1 := 1 \quad x2 := 1 \quad x3 := 1$ — Произвольные начальные значения

Given — Блок решения Given

$x1 \geq 20 \quad x2 \geq 20 \quad x2 \geq 20$

$4\cdot x1 + 3.4\cdot x2 + 2\cdot x3 \leq 340$ — Ограничивающие условия

$4.75\cdot x1 + 11\cdot x2 + 2\cdot x3 \leq 700$

$x1 + x2 + x3 = 100$

$R := \text{Maximize}(f,x1,x2,x3) \quad R = \begin{pmatrix} 56 \\ 20 \\ 24 \end{pmatrix}$ — Полученное решение x1=56, x2=20 и x3=24

Рис. 8.14. Пример решения типовой задачи линейного программирования

$$A := \begin{pmatrix} -1 & 2 & 3 \\ 4 & 8 & 1 \\ 7 & -4 & 0 \end{pmatrix} \qquad b := \begin{pmatrix} 2 \\ -1 \\ 5 \end{pmatrix}$$

Given

$$A\cdot x = b \qquad y^3 = 5 + 7\cdot y$$

$$\text{Find}(x,y) = \begin{bmatrix} \begin{pmatrix} 0.437 \\ -0.485 \\ 1.136 \end{pmatrix} \\ -0.783 \end{bmatrix}$$

Рис. 8.15. Пример применения функции Find для решения задачи линейного программирования

8.12.2. Оптимальные экономико-математические модели

Современная экономика широко использует математические методы и самые разнообразные математические модели. Большую группу в моделировании экономических процессов составляют задачи, относящиеся к методам принятия оптимальных решений, исследованию операций. В повседневной практике хозяйствования требуется выбрать производственную программу, поставщиков, распределение ресурсов, маршрут транспортировки и т. д. [172–175]. Требование оптимальности в планировании и управлении приводит к задачам *оптимального (математического) программирования* – разделу прикладной математики, занимающемуся условной оптимизацией.

Необходимо найти такое управленческое решение $X = \{x_1, x_2, \ldots x_n\}$, которое в некоторой области допустимых решений D обеспечивало бы наилучшее значение некоторого критерия оптимальности – экономического показателя. Такими экономическими показателями чаще всего являются «максимум прибыли», «минимум затрат», «максимум рентабельности» и т. д. Задача условной оптимизации в общем виде может быть записана так: найти максимум или минимум функции

$$f(X) = f\{x_1, x_2, \ldots x_n\} \quad (8.1)$$

при ограничениях

$$\varphi_i(x_1, x_2, \ldots x_n)(\leq, =, \geq)b_i, \text{ где } i = 1 \ldots m \quad (8.2)$$

$$x_j \geq 0, j = 1 \ldots n. \quad (8.3)$$

Условия (8.3) может и не быть, но чаще всего переменные в экономическом моделировании должны быть неотрицательными. Выбор оптимального управленческого решения в конкретной производственной ситуации требует решения задачи оптимального программирования. Если функция и ограничения (8.1)–(8.3) линейные, то проблема сводится к задаче линейного программирования. К математическим задачам линейного программирования приводят различные производственные и хозяйственные ситуации, которые требуют оптимального использования ограниченных ресурсов (задачи о планировании выпуска продукции, о смесях, транспортная задача и т. д.).

8.12.3. Решение задач максимизации объема продукции

Рассмотрим вполне типовую для малого бизнеса задачу на максимизацию объема выпуска изделий некоторым малым предприятием в денежном эквиваленте. Постановка задачи и ее решение представлены в документе рис. 8.16.

В задаче ищется максимальный объем выпуска тканей – также в денежном эквиваленте.

ЗАДАЧА ПЛАНИРОВАНИЯ ВЫПУСКА ТКАНЕЙ

Фабрика выпускает три вида тканей, причём суточное плановое задание составляет не менее 90 м тканей первого вида, 70 м - второго и 60 м - третьего. Суточные ресурсы 780 единиц производственного оборудования, 850 единиц сырья, 750 единиц электроэнергии. Определить, сколько метров ткани каждого вида следует выпустить, чтобы общая стоимость продукции была максимальной.

ORIGIN := 1

$f(x) := 80 \cdot x_1 + 70 \cdot x_2 + 60 \cdot x_3$ Целевая функция

$x_1 := 10 \quad x_2 := 10 \quad x_3 := 10$

Given

$2 \cdot x_1 + 3 \cdot x_2 + 4 \cdot x_3 \le 780$

$x_1 + 4 \cdot x_2 + 5 \cdot x_3 \le 850$ Ограничения по суточным ресурсам

$3 \cdot x_1 + 4 \cdot x_2 + 2 \cdot x_3 \le 790$

$x_1 \ge 90 \quad x_2 \ge 70 \quad x_3 \ge 60$ Плановое задание

$R := \text{Maximize}(f, x) \quad R = \begin{pmatrix} 112.5 \\ 70 \\ 86.25 \end{pmatrix}$ Оптимальный план выпуска ткани

$f(R) = 1.9075 \times 10^4$

Рис. 8.16. Решение задачи планирования выпуска тканей

8.12.4. Решение задач минимизации ресурсов

Подобным описанному образом решаются и задачи на минимизацию ресурсов производства. Пример решения такой задачи на минимизацию числа автомобильных перевозок дан на рис. 8.17.

Другой такой задачей является задача минимизации стоимости смеси, например бензина. Стандартом требуется, что октановое число бензина А-76 должно быть не ниже 76, а содержание серы – не более 0.3%. Для изготовления бензина используется смесь из четырех компонентов. Данные о компонентах приведены в таблице:

Характеристика	Компонент бензина			
	1	2	3	4
Октановое число	68	72	80	90
Содержание серы, %	0,35	0,35	0,3	0,2
Ресурсы, т	700	600	500	300
Себестоимость, ден.ед./т	40	45	60	90

ЗАДАЧА МИНИМИЗАЦИИ ЧИСЛА АВТОМОБИЛЬНЫХ СМЕН

На участок строящейся дороги необходимо вывезти 20 000 м³ каменных материалов. В районе строительства есть три карьера с запасами 8000 м³, 9000 м³, 10000 м³. Для погрузки материалов используются экскаваторы, имеющие производительность 250 м³ в смену на карьерах 1 и 2 и 500 м³ в смену в карьере 3. На погрузку материалов для рассматриваемого участка для экскаваторов выделен общий лимит 60 смен. Для перевозки 10000 м³ материалов с карьера 1 требуется 1000 смен автомобилей, из карьера 2 - 1350, из карьера 3 - 1700 смен автомобилей. Найти оптимальный план перевозок, обеспечивающий минимальное количество автомобильных смен. Объем материалов нормируем относительно 1=10 000 м³ для уменьшения вычислительных погрешностей.

$ORIGIN := 1 \qquad f(x) := 1000 \cdot x_1 + 1350 \cdot x_2 + 1700 \cdot x_3$ — Целевая функция

$x_1 := 10 \qquad x_2 := 10 \qquad x_3 := 10$

Given

$x_1 + x_2 + x_3 = 2.0$ — Потребность в материалах

$40 \cdot x_1 + 40 \cdot x_2 + 20 \cdot x_3 \le 60$ — Ограничение по сменам экскаваторов

$0 \le x_1 \le 0.8 \qquad 0 \le x_2 \le 0.9 \qquad 0 \le x_3 \le 1.0$ — Запасы материалов в карьерах

$R := \text{Minimize}(f, x) \qquad R = \begin{pmatrix} 0.8 \\ 0.2 \\ 1 \end{pmatrix}$ — Количество вывозимых матералов с карьеров в 10000 м³

$f(R) = 2.77 \times 10^3$ — Число автомобильных смен

Рис. 8.17. Решение задачи на минимизацию автомобильных перевозок

Требуется определить, сколько тонн каждого компонента следует использовать для получения 1000 т автомобильного бензина А-76, чтобы его себестоимость была минимальной. Решение задачи представлено на рис. 8.18.

8.12.5. Решение транспортной задачи

На практике часто встречается так называемая транспортная задача. В n пунктах – складах поставщиков находится определенное количество S_i ($i = 1...n$) единиц некоторого однородного продукта. Этот продукт потребляется m потребителями в определенном объеме B_j ($j = 1...m$). Известны расходы на перевозку единицы продукта из i-го склада j-му потребителю, которые равны C_{ij} и приведены в таблице транспортных расходов. Требуется составить такой план перевозок, при котором полностью удовлетворяются заказы потребителей с минимальными транспортными затратами.

Пусть на трех складах хранится 310, 260 и 280 единиц груза соответственно. Требуется его доставить пяти потребителям, заказы которых равны 180, 80, 200, 160, 220 единиц. Стоимости перевозки единицы груза со склада потребителю указаны в транспортной таблице.

Разработать модель, описывающую затраты при перевозке грузов со складов потребителям и позволяющую оптимизировать затраты на транспортировку. Исходные данные к задаче следующие:

ЗАДАЧА О СМЕСЯХ

Стандартом требуется, что октановое число бензина А-76 должно быть не ниже 76, а содержание серы - не более 0.3%. Для изготовления бензина используется смесь из четырех компонентов. Требуется определить, сколько тонн каждого компонента следует использовать для получения 1000 т автомобильного бензина А-76, чтобы его себестоимость была минимальной.

$f(x) := 40 \cdot x_1 + 45 \cdot x_2 + 60 \cdot x_3 + 90 \cdot x_4$ Целевая функция

$x_1 := 10 \quad x_2 := 10 \quad x_3 := 10 \quad x_4 := 10$

Given

$x_1 + x_2 + x_3 + x_4 = 1000$ Условиие получения заданного количества бензина

$68 \cdot x_1 + 72 \cdot x_2 + 80 \cdot x_3 + 90 \cdot x_4 \geq 76 \cdot 1000$ Ограничение по октановому числу бензина

$0.35 \cdot x_1 + 0.35 \cdot x_2 + 0.3 \cdot x_3 + 0.2 \cdot x_4 \leq 0.3 \cdot 1000$ Ограничение по содержанию серы

$0 \leq x_1 \leq 700 \quad 0 \leq x_2 \leq 600 \quad 0 \leq x_3 \leq 500 \quad 0 \leq x_4 \leq 300$ Ограничения по числу компонентов

$R := \text{Minimize}(f, x)$

$$R = \begin{pmatrix} 0 \\ 571.429 \\ -2.842 \times 10^{-14} \\ 142.857 \\ 285.714 \end{pmatrix}$$

$f(R) = 5.7143 \times 10^4$ Минимальная себестоимость

Рис. 8.18. Решение задачи о смесях

- груз на всех складах одинаковый;
- количество груза на каждом i-м складе;
- заказ каждого j-го потребителя;
- стоимость перевозки груза с i-го склада j-му потребителю;

Примем следующие гипотезы:

- считаем, что пропускная способность дороги от каждого склада не ограничена;
- длительность перевозки от склада к потребителю не учитывается при выборе предпочтительного плана перевозок;
- общее количество грузов на складах всегда больше или равно заказу потребителей.

Введем обозначения:

- n – количество поставщиков;
- m – количество потребителей;
- A_i – поставки от i-го поставщика всем потребителям, ограниченные S_i – количеством груза на складе;
- B_j – заказ j-го потребителя – поставки ему от всех поставщиков;
- X_{ij} – перевозки от i-го поставщика j-му потребителю;
- C_{ij} – цена поставки единицы груза от i-го поставщики j-му потребителю.

Требуется обеспечить полное выполнение всех заказов B_j при минимальных затратах на перевозку грузов. Общие затраты на перевозку равны

$$F(X)=\sum_{i=1}^{n}\sum_{j=1}^{m}C_{i,j}\cdot x_{i,j},$$

где $F(X)$ – минимизируемая функция, зависящая от $n\times m$ переменных X_{ij}. Очевидно, что при этом должны выполняться ограничения:

$A_i=\sum_{j=1}^{m}x_{i,j}\le S_i$ – нельзя поставить груза больше, чем есть на складе;

$B_j=\sum_{i=1}^{n}x_{i,j}$ – потребитель должен получить точное количество заказанного груза;

$x_{i,j}\ge 0$ – товаропоток не может быть отрицательным.

Математическая постановка соответствует решению задачи линейного программирования – условия оптимизации описываются системами линейных уравнений и неравенств. Решения задачи будем проводить в среде пакета Mathcad, используя универсальную встроенную функцию `Minimize`. Документ, решающий данную задачу, представлен на рис. 8.19.

РЕШЕНИЕ ТРАНСПОРТНОЙ ЗАДАЧИ

ORIGIN := 1

$S:=\begin{pmatrix}310\\260\\280\end{pmatrix}$ Запасы продукции на складах

$B:=\begin{pmatrix}180\\80\\200\\160\\220\end{pmatrix}$ Заказы потребителей

$C:=\begin{pmatrix}10&8&6&5&4\\6&5&4&3&6\\3&4&5&5&9\end{pmatrix}$ Матрица транспортных расходов

$f(x):=\sum_{i=1}^{3}\sum_{j=1}^{5}C_{i,j}\cdot x_{i,j}$ Целевая функция - общая сумма транспортных затрат

$x:=\begin{pmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{pmatrix}$ Матрица начальных приближений

Given

$\sum_{j=1}^{5}x_{1,j}\le S_1\quad\sum_{j=1}^{5}x_{2,j}\le S_2\quad\sum_{j=1}^{5}x_{3,j}\le S_3$ Ограничения по запасам продукции на складах

$\sum_{i=1}^{3}x_{i,1}=B_1\quad\sum_{i=1}^{3}x_{i,2}=B_2\quad\sum_{i=1}^{3}x_{i,3}=B_3\quad\sum_{i=1}^{3}x_{i,4}=B_4\quad\sum_{i=1}^{3}x_{i,5}=B_5$ Требование выполнения заказа потребителя

$x\ge 0$ Объем перевозок не может быть отрицательным

R := Minimize(f,x)

$R=\begin{pmatrix}0&0&0&80&220\\0&0&180&80&0\\180&80&20&0&0\end{pmatrix}$ Оптимальный план перевозок, строка соответствует складу, столбец - потребителю, элемент матрицы - объему перевозок.

$f(R)=3.2\times 10^3$ Затраты на транспортировку

Рис. 8.19. Решение транспортной задачи

8.12.6. Задачи целочисленного программирования с булевыми переменными

Если искомые параметры (переменные) должны иметь только целые значения, то для их нахождения надо применять методы решения задач целочисленного программирования. Однако в некоторых практических задачах целочисленного программирования искомые переменные могут принимать не любые целые значения, а лишь значения 0 – ответ «нет» и 1 – ответ «да». Такие переменные называют логическими, или булевыми.

Одной из задач с такими переменными является *задача о назначениях*. В подобных случаях требуется решить, как распределить рабочих по различным рабочим местам, чтобы общая выработка была наибольшей или затраты на зарплату – наименьшими, и как выбрать из нескольких возможных вариантов инвестиционных проектов (например, закупки станков для модернизации цеха) наиболее эффективный проект.

В общем случае можно подобные задачи сформулировать следующим образом.

Предлагается n управленческих решений x_i, каждое из которых позволяет получить эффект от его реализации $C_1, C_2, \ldots C_n$. В наличии имеется m видов ресурсов в количестве S_j. Управленческое решение требует для своей реализации объем ресурсов B_{ij}, где $i = 1...n, j = 1...m$. Необходимо так распорядиться имеющимися ресурсами, чтобы максимизировать эффект от принимаемых решений.

Оптимизируемая переменная может принимать только два значения:

$$x_i = \begin{cases} 1, \text{ если решение следует принять.} \\ 0, \text{ если решение нецелесообразно.} \end{cases}$$

Тогда задача оптимизации будет следующей:

Найти максимум целевой функции

$F(X) = \sum_{i=1}^{n} C_i \cdot x_i$ – экономическая эффективность, при ограничениях;

$\sum_{i=1}^{n} B_{i,j} \cdot x_i \le S_j, \ j = 1...m$ – по ресурсам;

$x_i \in \{0, 1\}, \ i = 1...n$ – задание булевого характера переменных.

Рассмотрим пример на выбор инвестиционного проекта. Имеются четыре инвестиционных проекта, каждый из которых требует затрат материальных и трудовых ресурсов. Количество ресурсов ограничено и не позволяет реализовать все проекты сразу. Выбрать для реализации оптимальные по суммарному экономическому эффекту проекты. Конкретные числовые данные приведены в таблице, представленной ниже.

Показатели	Варианты инвестиционных проектов				Запасы
	1	2	3	4	
Материальные ресурсы	200	180	240	250	800
Трудовые ресурсы	10	15	22	28	50
Прибыль	65	80	90	210	

Будем решать задачу *методом полного, или сплошного, перебора*. Метод заключается в переборе всех возможных вариантов сочетаний допустимых значений переменных, проверке выполнения для каждого ограничения и вычислении в удовлетворительных случаях соответствующих значений целевой функции. Из полученного множества значений выбирается максимальное (или минимальное), а набор значений переменных для него и будет решением задачи. В общем случае число вычислительных процедур при полном переборе быстро растет и равняется $N = 2^n \cdot (m + 1)$, где n – число переменных, m – число ограничений. Метод имеет простой алгоритм и может быть легко реализован с использованием средств программирования пакета Mathcad – рис. 8.20.

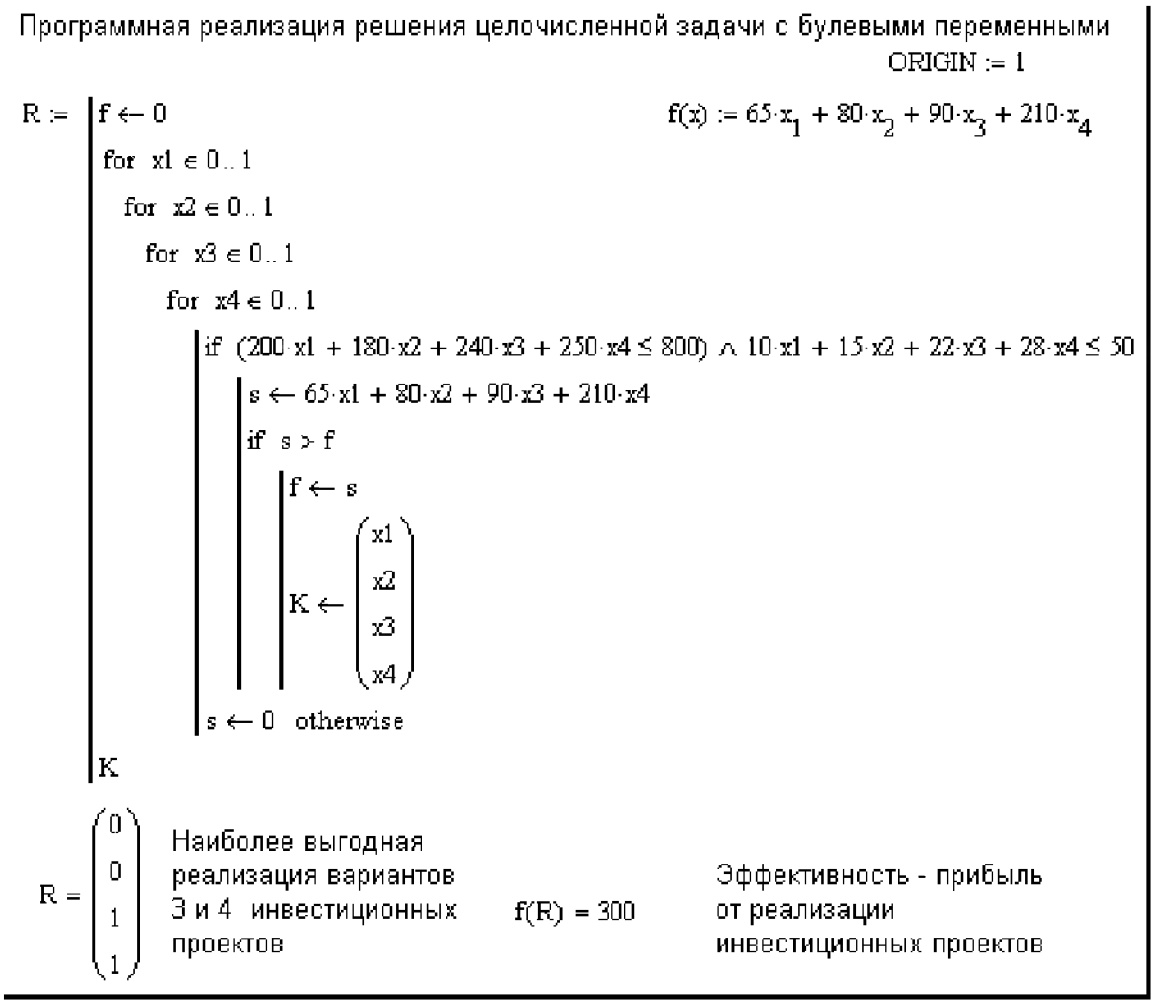


Рис. 8.20. Пример решения задачи целочисленного программирования с булевыми переменными методом прямого перебора

8.12.7. Задача поиска кратчайшего пути

Для представления различных технических объектов, описания процессов и функционирования систем часто используются модели, построенные на основе графов. К модели в виде графа можно свести и многие практические экономические задачи. К таким проблемам относятся задача поиска кратчайшего пути в заданной транспортной системе, задачи о распределении потока в сети, сетевые модели планирования последовательности работ, задача коммивояжера и др.

В общем виде задача формулируется следующим образом. Имеется некоторое количество пунктов, соединенных определенным образом одно- или двунаправленными связями. Каждая связь имеет определенный вес – длину. Требуется найти кратчайший путь из пункта i в пункт j.

При составлении математической модели задачи необходимо учитывать, что маршрут должен быть непрерывным, а каждый промежуточный пункт на пути следования может быть посещен только один раз. Транспортная система в задаче является ориентированным графом – двухполюсной сетью, где N_1 – вход, N_n – выход, весовые коэффициенты c_{ij} ребер δ_{ij} являются длинами пути между пунктами i и j, требуется определить кратчайший путь из N_1 в N_n. Сопоставим каждому ребру графа булеву переменную, то есть $\delta_{ij} \in \{0, 1\}$. Если ребро входит в маршрут, то $\delta_{ij} = 1$, иначе $\delta_{ij} = 0$. Тогда целевая функция, которая минимизируется при поиске кратчайшего пути, имеет вид:

$$F = \sum_i \sum_j c_{ij} \cdot \delta_{ij}.$$

Все пункты маршрута можно разделить на начальный, промежуточный и конечный. Очевидно, что в каждом промежуточном пункте должно быть по одному входящему и исходящему ребру, а для начального и конечного пунктов может быть только одно исходящее или входящее ребро соответственно. Математически эти ограничения могут быть записаны следующим образом:

- для перечисления всех k, входящих в i-й пункт маршрута ребер
  $\sum_k \delta_{ki} = 1,\ i = 2, \ldots n;$
- для перечисления всех j, исходящих из i-го пункта ребер
  $\sum_j \delta_{ij} = 1,\ i = 1, \ldots n-1.$

Если же i-й пункт не входит в кратчайший маршрут, то соответствующая сумма как для входящих, так и исходящих из вершины графа ребер должна быть равна нулю. Тогда для любого пункта сети, кроме начального и конечного, должно выполнять условие

$$\sum_k \delta_{ki} - \sum_j \delta_{ij} = 0.$$

В начальном пункте

$$\sum_j \delta_{1j} = 1,$$

в конечном

$$\sum_k \delta_{kn} = 1 \text{ и } \delta_{ij} \geq 0 \text{ для всех } i \text{ и } j.$$

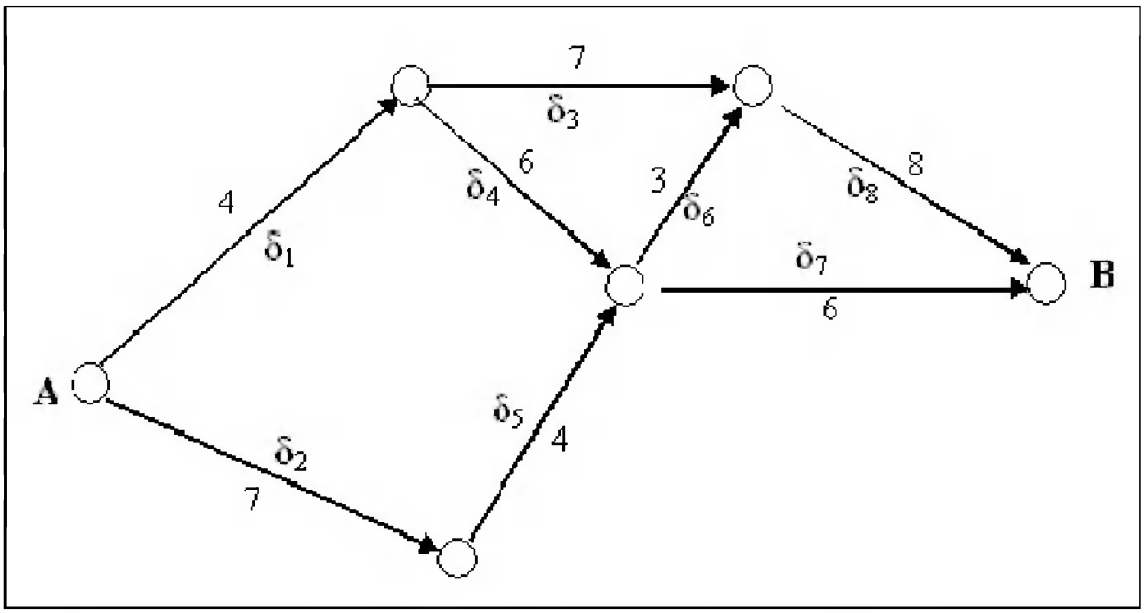
От переменных δ_{ij} достаточно потребовать только неотрицательности. Из-за ограничений в решении могут быть получены лишь значения нуля либо единицы. Таким образом, мы получили обычную задачу линейного программирования, которую можно решить без наложения требований целочисленности.

Очевидно, что к подобной формулировке, а точнее соответствующей математической модели можно свести самые разнообразные задачи, в том числе планирование последовательности выполнения технологических процессов и работ. Вес ребер графа при этом может иметь самый различный смысл: продолжительность, трудоемкость, стоимость и т. д.

Пусть требуется найти кратчайший маршрут из пункта А в пункт В, если схема движения и расстояния между объектами заданы рис. 8.21 [24]. Там же внизу дано решение задачи – номера ребер, дающих кратчайший мартшрут.

ЗАДАЧА ПОИСКА КРАТЧАЙШЕГО ПУТИ В ТРАНСПОРТНОЙ СЕТИ

Пусть требуется найти кратчайший маршрут из пункта **А** в пункт **В**, если схема движения и расстояния между объектами заданы на приведенном ниже рисунке.



ORIGIN := 1 $\quad f(\delta) := 4\cdot\delta_1 + 7\cdot\delta_2 + 7\cdot\delta_3 + 6\cdot\delta_4 + 4\cdot\delta_5 + 3\cdot\delta_6 + 6\cdot\delta_7 + 8\cdot\delta_8$

$\delta_1 := 1 \quad \delta_2 := 1 \quad \delta_3 := 1 \quad \delta_4 := 1 \quad \delta_5 := 1 \quad \delta_6 := 1 \quad \delta_7 := 1 \quad \delta_8 := 1$

Given

$\delta_1 \geq 0 \quad \delta_2 \geq 0 \quad \delta_3 \geq 0 \quad \delta_4 \geq 0 \quad \delta_5 \geq 0 \quad \delta_6 \geq 0 \quad \delta_7 \geq 0 \quad \delta_8 \geq 0$

$\delta_1 + \delta_2 = 1 \quad \delta_1 = \delta_3 + \delta_4 \quad \delta_2 = \delta_5 \quad \delta_4 + \delta_5 = \delta_6 + \delta_7 \quad \delta_3 + \delta_6 = \delta_8 \quad \delta_7 + \delta_8 = 1$

$R := \text{Minimize}(f,\delta) \quad R^T = (1\ 0\ 0\ 1\ 0\ 0\ 1\ 0)$ Кратчайший маршрут - ребра $\delta_1, \delta_4, \delta_7$

Рис. 8.21. Решение задачи на поиск кратчайшего маршрута

8.12.8. Задача о распределении потоков в сетях

В задачах подобного типа требуется найти оптимальный вариант транспортировки продукта по сети определенной конфигурации. В этом случае элементы сети имеют следующие характеристики: c_{ij} – стоимость транспортировки единицы продукции для ребра сети между вершинами i и j, D_{ij} – пропускная способность этого ребра, в общем случае ограниченная в пределах $0 \le D_{ij} \le \times$ (если ребро между данными вершинами i и j графа отсутствует, то пропускная способность равна нулю, если поток ничем не ограничен – то бесконечности). Очевидно, что в этом случае должно выполняться требование *сохранения потока* – суммарного потока, то есть входящий и выходящий из узла потоки должны быть равны.

Пусть x_{ij} – поток в ребре графа, тогда для промежуточной вершины сети

$$\sum_k x_{ki} - \sum_j x_{ij} = 0,$$

где k – перечисление всех входящих, j – всех исходящих ребер для вершины i.

Для потока в любом ребре требуется, чтобы

$$0 \le x_{ij} \le D_{ij}.$$

Для начальной и конечной вершин очевидно необходимо выполнение условия

$$\sum_j x_{1j} = A_1,$$

где A_1 – максимальный выходной поток, создаваемый исходной вершиной сети, необходимо, чтобы он был меньше, чем суммарная пропускная способность всех исходящих из вершины ребер:

$$\sum_k x_{kn} = B_n,$$

где B_n – максимальный поток, потребляемый конечной вершиной сети, он также не должен превышать пропускной способности входящих ребер.

Возможны различные постановки задачи оптимизации – минимизации стоимости транспортировки и максимизации потока. Получаем соответственно две формулировки математической модели задачи.

1. Минимизация стоимости:

   $F = \sum_i \sum_j c_{ij} \cdot x_{ij}$ – минимизируемая целевая функция – общая стоимость транспортировки.

   Ограничения:

   $A_1 = B_n$ – поток не может накапливаться в промежуточных вершинах, то есть $\sum_j x_{1j} = \sum_k x_{kn}$;

   $0 \le x_{ij} \le D_{ij}$ – по пропускной способности;

   $\sum_k x_{ki} - \sum_j x_{ij} = 0$ – сохранение непрерывности потока.

2. Максимизация потока:

$F = \sum_k x_{kn}$ – максимизируемая целевая функция – суммарный поток, входящий в конечный узел;

$\sum_i \sum_j c_{ij} \cdot x_{ij} \le C_s$ – суммарные затраты не должны превысить величины имеющихся средств C_s.

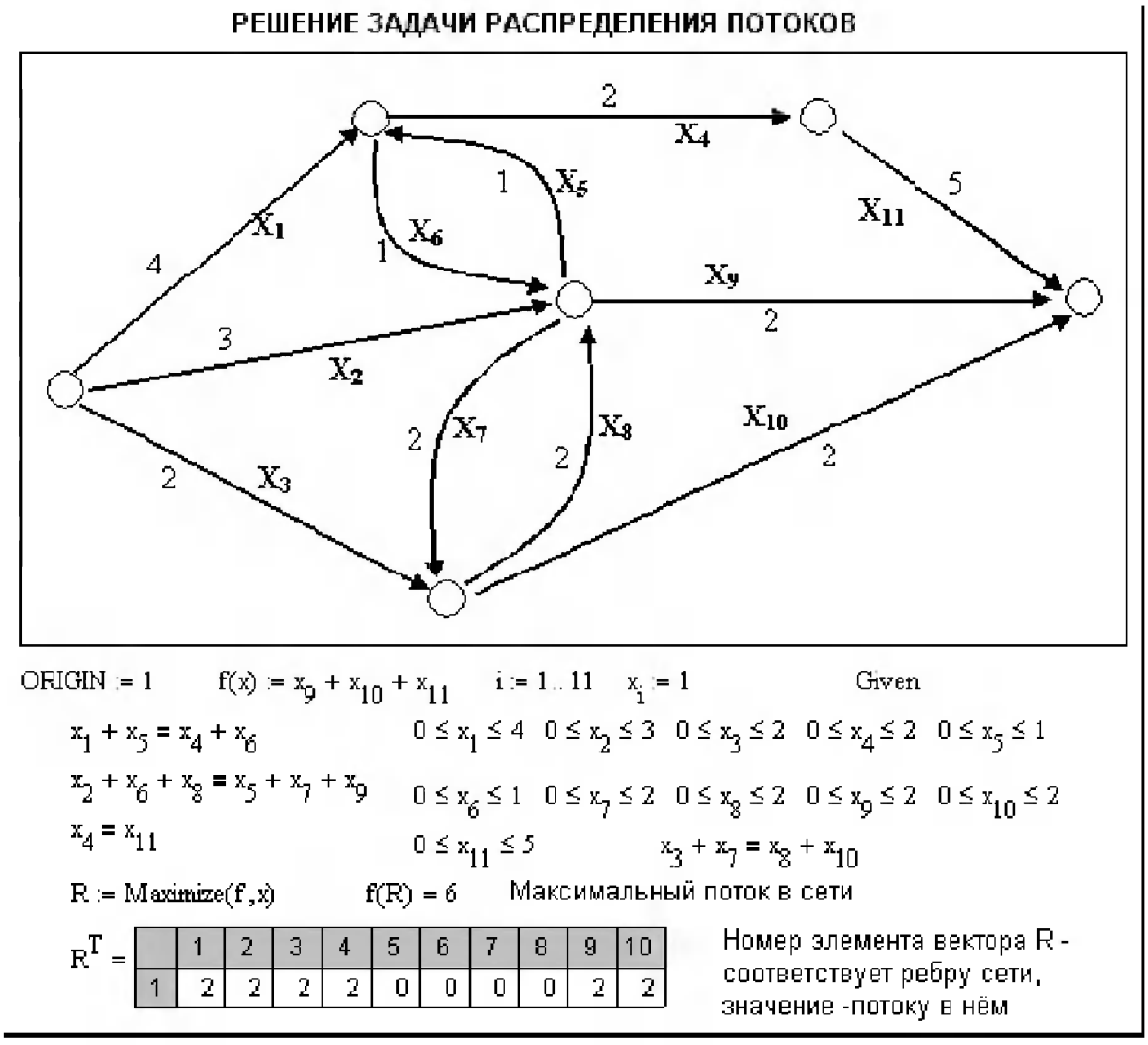
Ограничение:

$A_1 = B_n$ (поток не может накапливаться в промежуточных вершинах), означает также, что:

$\sum_j x_{1j} = \sum_k x_{kn}$ и $0 \le x_{ij} \le D_{ij}$ – на пропускную способность;

$\sum_k x_{ki} - \sum_j x_{ij} = 0$ – на сохранение непрерывности потока.

Рассмотрим задачу на поиск максимального потока для системы автодорог, представленных на документе рис. 8.22, где цифрами обозначена максимальная пропускная способность участков транспортной сети, тысяч машин в день.



	1	2	3	4	5	6	7	8	9	10
1	2	2	2	2	0	0	0	0	2	2

Рис. 8.22. Решение задачи на поиск максимального потока для системы автодорог

Заданный граф частично ориентирован. Для того чтобы прийти к математической модели, необходимо преобразовать граф в ориентированную сеть. Это можно сделать, заменив каждое неориентированной ребро – дорогу с двусторонним движением – двумя ориентированными – односторонними полосами движения, каждая с исходной пропускной способностью. Дороги x_4 и x_5 стали односторонними, так как возможность противоположного направления движения в данной задаче для них несущественна.

Задача может быть решена геометрически: согласно теореме, максимальная пропускная способность сети равна минимальной пропускной способности сечений сети. Аналитическое решение сводится к методам линейного программирования. Кроме того, тогда возможно определить соответствующие ему потоки в каждом ребре сети.

Сравнение максимально возможного потока, исходящего из начального узла сети, с результатом решения ($9 > 6$) показывает, что данная транспортная сеть требует дополнительного расширения для его пропуска.

8.13. Новый пакет оптимизации Optimization в Maple 9.5

В систему Maple 9.5 был добавлен новый пакет оптимизации Optimization, основанный на новейших, существенно улучшенных алгоритмах оптимизации. С его помощью можно решать не только задачи линейного, но и квадратичного и нелинейного программирований с повышенной степенью визуализации.

8.13.1. Доступ к пакету Optimization и его назначение

Пакет оптимизации Optimization вызывается как обычно:

```
> with(Optimization);
```

[*ImportMPS*, *Interactive*, *LPSolve*, *LSSolve*, *Maximize*, *Minimize*, *NLPSolve*, *QPSolve*]

```
Warning, the name changecoords has been redefined
```

Для получения справки по пакету надо исполнить команду

```
> help(Optimization);
```

Пакет использует при вычислениях алгоритмы группы NAG, которые считаются наиболее эффективными при реализации численных методов вычислений, в частности реализующих алгоритмы оптимизации. Пакет вводит 8 функций. Две из них – это переопределенные функции вычисления максимума `Maximize` и минимума `Minimize`. Кроме того, пакет имеет 4 решателя уравнений с заданными ограничениями, реализующих следующие методы:

- `LPSolve` – линейное программирование;
- `LSSolve` – улучшенная реализация метода наименьших квадратов;

- `QPSolve` – квадратичное программирование;
- `NLPSolve` – нелинейное программирование.

Функция `ImportMPC` обеспечивает ввод данных для оптимизации из файла, а функция `Interactive` позволяет работать с интерактивным Maplet-окном для оптимизации.

С пакетом Optimization можно познакомиться по его справке. В ее разделе Examples есть довольно обширный документ с примерами применения пакета – дополнительными к тем, которые даются к функциям пакета в справке. Начало этого документа дано на рис. 8.23. В нем представлены основные задачи, решаемые пакетом Optimization, – линейное, квадратичное и нелинейное программирование, а также приближение данных и функциональных зависимостей методом наименьших квадратов (нелинейная регрессия).

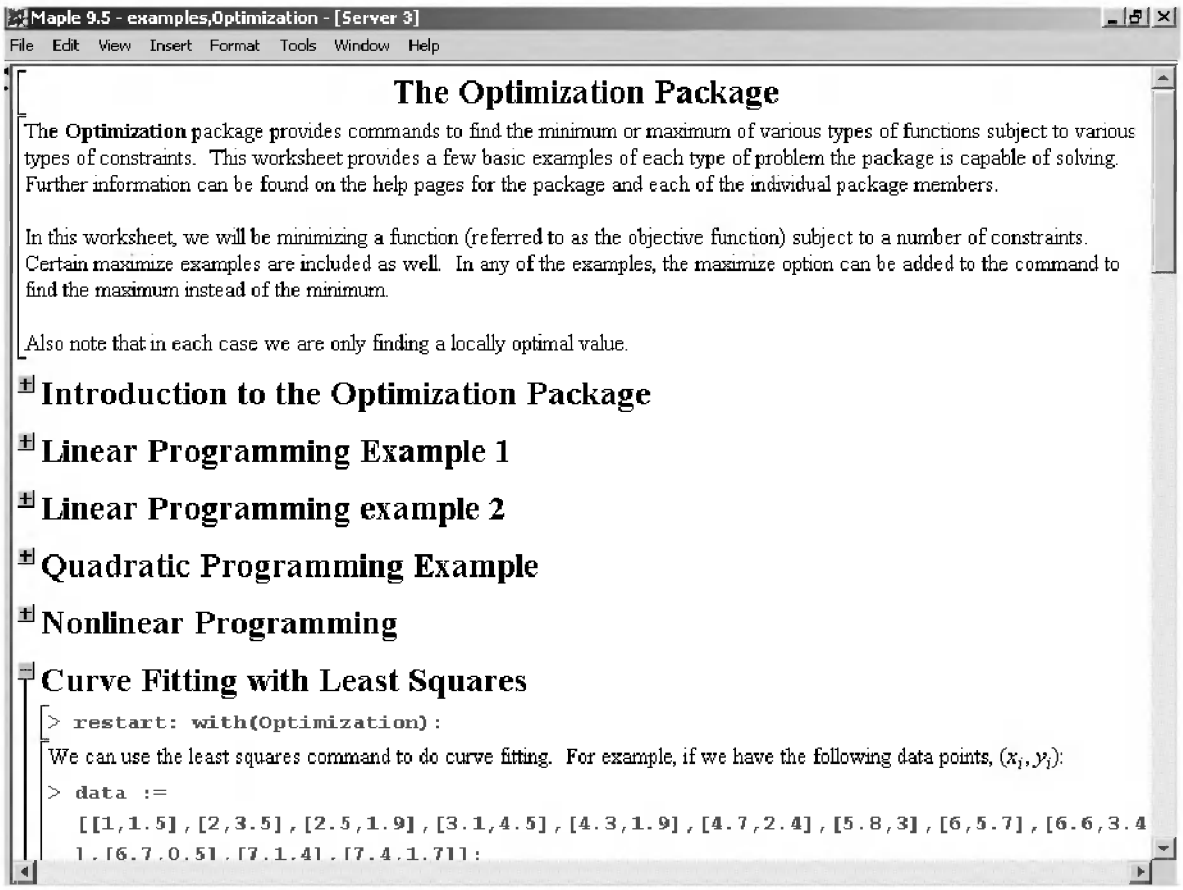


Рис. 8.23. Начало документа с примерами применения пакета Optomization

8.13.2. Работа с функциями Minimize и Maximize

Функции `Minimize` и `Maximize` служат для поиска минимумов и максимумов математических выражений с учетом ограничений самыми современными численными методами. Функции записываются в виде:

```
Minimize(obj [, constr, bd, opts])
Minimize(opfobj [, ineqcon, eqcon, opfbd, opts])
Maximize(obj [, constr, bd, opts])
Maximize(opfobj [, ineqcon, eqcon, opfbd, opts])
```

Параметры функций следующие:

- `obj` – алгебраический объект, целевая функция;
- `constr` – список с ограничивающими условиями;
- `bd` – последовательность вида name=range, задающая границы для одной или более переменных;
- `opts` – равенство или равенства вида option=value, где option – одна из опций feasibilitytolerance, infinitebound, initialpoint, iterationlimit или optimalitytolerance, специфицированных в команде Minimize или Maximize;
- `opfobj` – процедура, целевая функция;
- `ineqcon` – множество или список процедур с ограничениями типа неравенств;
- `eqcon` – множество или список процедур с ограничениями типа равенств;
- `opfbd` – последовательность пределов; границы для всех переменных.

Примеры применения этих функций представлены ниже:

```
> Maximize(sin(x)/x);
```

$$[1., [x = 2.9384741186727256710^{-11}]]$$

```
> Minimize(x^2+y^2);
```

$$[0., [x = 0., y = 0.]]$$

```
> Minimize(sin(x)/x, initialpoint={x=5});
```

$$[-0.21723365821122l636, [x = 4.4934094579236472O]]$$

```
> Maximize(sin(x*y*z));
```

$$[1., [x = 1.162447351509623694, z = 1.16244735150962364, y = 1.16244735150962364]]$$

```
> Minimize(2*x+3*y, {3*x-y<=9, x+y>=2}, assume=nonnegative);
```

$$[4., [x = 2., y = 0.]]$$

Из этих примеров видно, что результаты вычислений представляются в виде чисел с плавающей точкой с так называемой двойной точностью (правильнее было бы сказать с двойной длиной или разрядностью). При вычислениях используются алгоритмы группы NAG и решатели, описанные ниже.

8.13.3. Линейное программирование – LPSolve

Для решения задач линейного программирования в пакете Optimization введена функция:

```
LPSolve(obj [, constr, bd, opts])
```

Она имеет следующие параметры:

- `obj` – алгебраическое выражение, целевая функция;
- `constr` – множество или список линейных ограничений;

- `bd` – последовательность вида name=range, задающая границы одной или многих переменных;
- `opts` – равенство или равенства в форме option=value, где option – одна из опций `assume`, `feasibilitytolerance`, `infinitebound`, `initialpoint`, `iterationlimit` или `maximize`, специализированных для команды LPSolve.

Пример на решение задачи линейного программирования дан на рис. 8.24. Здесь оптимизируется целевая функция $-3x - 2y$, которая линейно зависит от переменных x и y. В этом примере интересна техника графической визуализации решения.

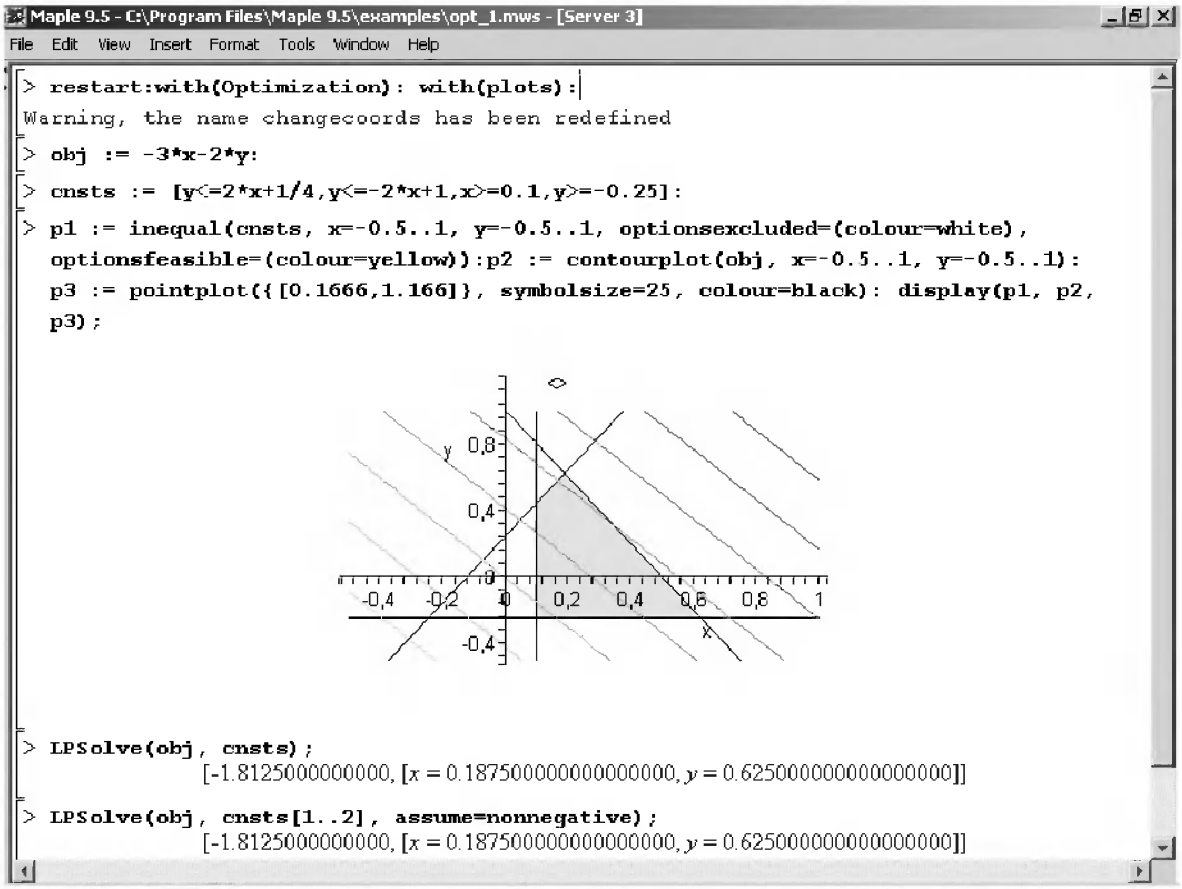


Рис. 8.24. Пример решения задачи линейного программирования

Эта функция может задаваться также в матричной форме:

```
LPSolve(c [, lc, bd, opts])
```

Здесь `c` – вектор, задающий целевую функцию, остальные параметры были определены выше. Пример применения функции LPSolve в матричном виде представлен ниже:

```
> c := Vector([-1,4,-2], datatype=float):
bl := Vector([2,3,1], datatype=float):
bu := Vector([5,8,2.5], datatype=float):
LPSolve(c, [], [bl, bu]);
```

```
[     [                          ]]
|     |                 5.       ||
|     |                          ||
| 2., |                 3.       ||
|     |                          ||
|     |                          ||
[     [ 2.5000000000000000       ]]
```

Ряд других подобных примеров применения функции `LPSolve` можно найти в справке по этой функции.

8.13.4. Квадратичное программирование – QPSolve

Для реализации квадратичного программирования служит функция

`QPSolve(obj, constr, bd, opts)`

с параметрами, описанными выше для функции `LPSolve`. Пример реализации квадратичного программирования представлен на рис. 8.25. Здесь оптимизируется выражение $-3x^2 - 2y^2$, которое квадратично зависит от переменных x и y. Здесь также интересна техника визуализации квадратичного программирования.

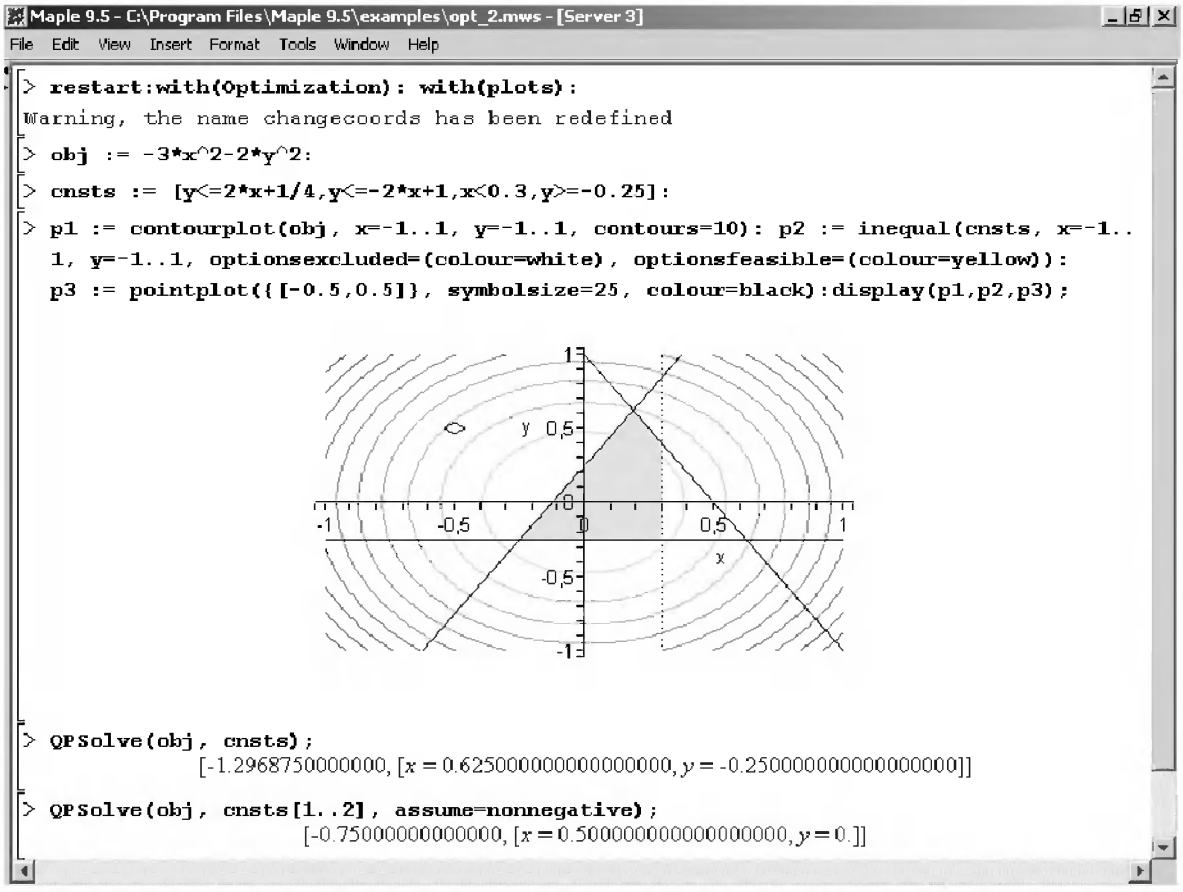


Рис. 8.25. Пример квадратичного программирования

Эта функция также может быть записана в матричной форме:

```
QPSolve(obj, lc, bd, opts)
```

Пример применения этой функции дан ниже:

```
> c := Vector([2,5], datatype=float):
H := Matrix([[6,3],[3,4]], datatype=float):
A := Matrix([[-1,1]], datatype=float):
b := Vector([-2], datatype=float):
QPSolve([c, H], [A, b]);
```

```
          [                      [0.466666666666656]]
          | -3.5333333333333,    |                  ||
          |                      |-1.6000000000000030||
          [                      [                  ]]
```

Ряд подобных примеров можно найти в справке по данной функции.

8.13.5. Нелинейное программирование – NLPSolve

Нелинейное программирование позволяет решать задачи оптимизации при нелинейных зависимостях целевой функции от ее аргументов. Для этого в пакете Optimization имеется функция:

```
NLPSolve(obj, constr, bd, opts)
NLPSolve(opfobj, ineqcon, eqcon, opfbd, opts)
```

Ее параметры те же, что и у ранее описанных функций. В связи с этим ограничимся парой примеров ее применения при целевых функциях одной и двух переменных:

```
> NLPSolve(x*exp(-x), x=0..6, maximize);
```

$$[0.367879441171442278, [x = 0.99999998943752966]]$$

```
> NLPSolve(x*y*exp(-x)*exp(-y), x=0..6, y=0..6,maximize);
```

$$[0.135335283236612674, [x = 0.99999999994630706, y = 1.00000000003513966]]$$

В оптимизируемых функциях этих примеров присутствует экспоненциальная зависимость, что и указывает на решение задачи нелинейного программирования. Однако следует отметить, что ограничения должны быть линейными – в противном случае возвращается сообщение об ошибке с указанием на необходимость обеспечения линейности ограничивающих условий.

Возможна и матричная форма функции:

```
NLPSolve(n, p, nc, nlc, lc, bd, opts)   NLPSolve(n, p, lc, bd, opts)
```

Примеры на ее применение можно найти в справке по функции NLPSolve.

8.13.6. Работа с функцией импорта данных из файлов – ImportMPC

Для импорта данных из файлов служит функция:

```
ImportMPS(filename [, maxm, maxn, lowbnd, upbnd, opts])
```

В ней используются следующие параметры:

- filename – имя файла для MPS(X) в виде строки;
- maxm – максимальное число линейных ограничений;
- maxn – максимальное число переменных;
- lowbnd – значение нижней границы для переменных;
- upbnd – значение верхней границы для переменных;
- opts – выражения в виде опций, записываемых в форме option=value, где option – один из объектов rhsname, rangename или boundsname, заданный дляImportMPS команд.

С деталями применения этой функции можно ознакомиться по справке по ней.

8.13.7. Нелинейная регрессия

Для решения задач нелинейной регрессии (метод наименьших квадратов) служит функция:

```
LSSolve(obj, constr, bd, opts)
LSSolve(opfobj, ineqcon, eqcon, opfbd, opts)
```

Назначение ее параметров уже описывалось. Напомним, что параметр `opfobj` – это список процедур для остатков (разностей) метода наименьших квадратов. Пример применения этой функции для приближения облака заданных точек data нелинейной зависимостью с именем p дан на рис. 8.26.

8.13.8. Оптимизация в Maplet-окне с помощью функции Interactive

Функция Interactive служит для организации интерактивной оптимизации в Maplet-окне. Эта функция может задаваться в виде:

```
Interactive()      Interactive(obj, constr)
```

В первом случае открывается «пустое» Maplet-окно, а во втором – окно с введенной целевой функцией obj и ограничивающими условиями constr. Вид окна с примером квадратичной оптимизации представлен на рис. 8.27.

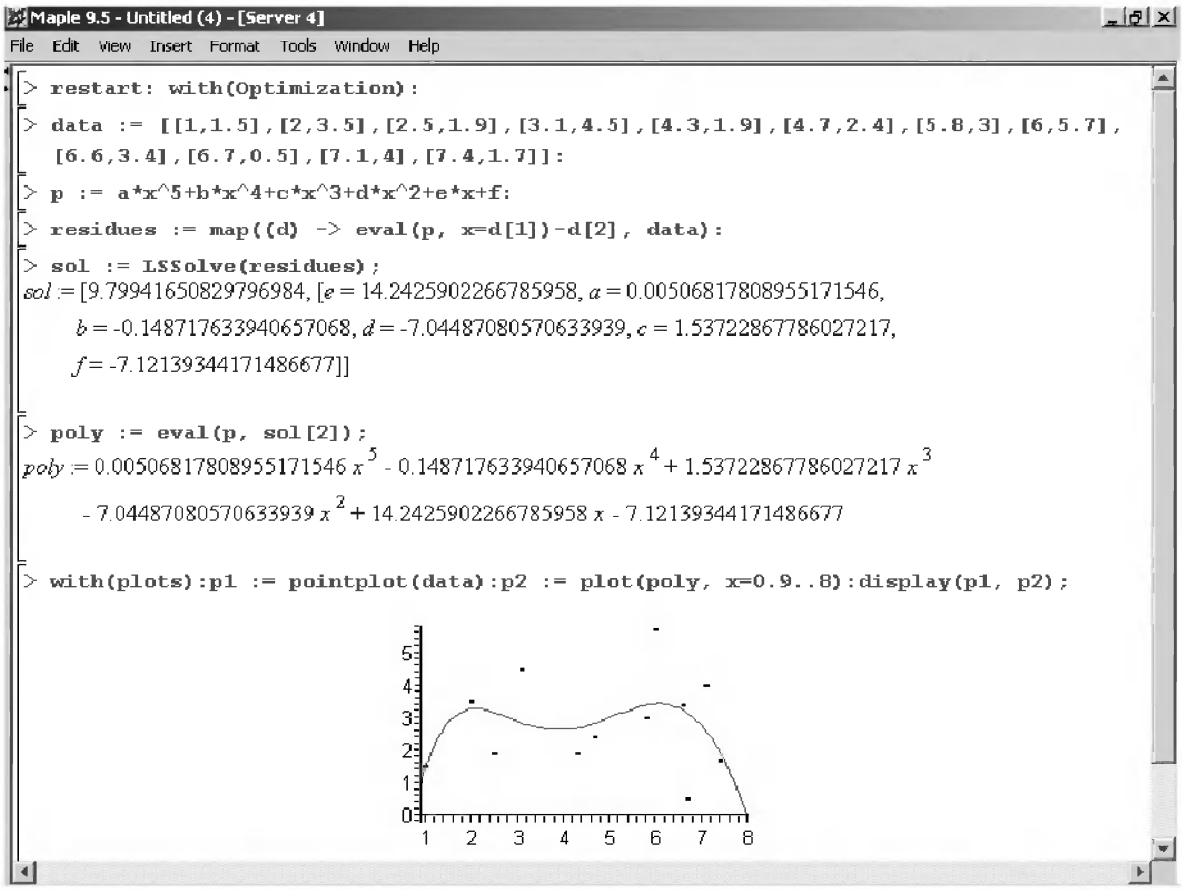


Рис. 8.26. Пример нелинейной регрессии с помощью функции LSSolve

В левом верхнем углу окна имеется список классов задач оптимизации. Справа расположены панели для ввода оптимизируемого выражения и ограничивающих условий. Кнопки Edit позволяют вызывать простые окна для редактирования их, а кнопка Solve запускает вычисления, результат которых появляется в окошке Solution. Остальные элементы интерфейса Maplet-окна в особых пояснениях не нуждаются.

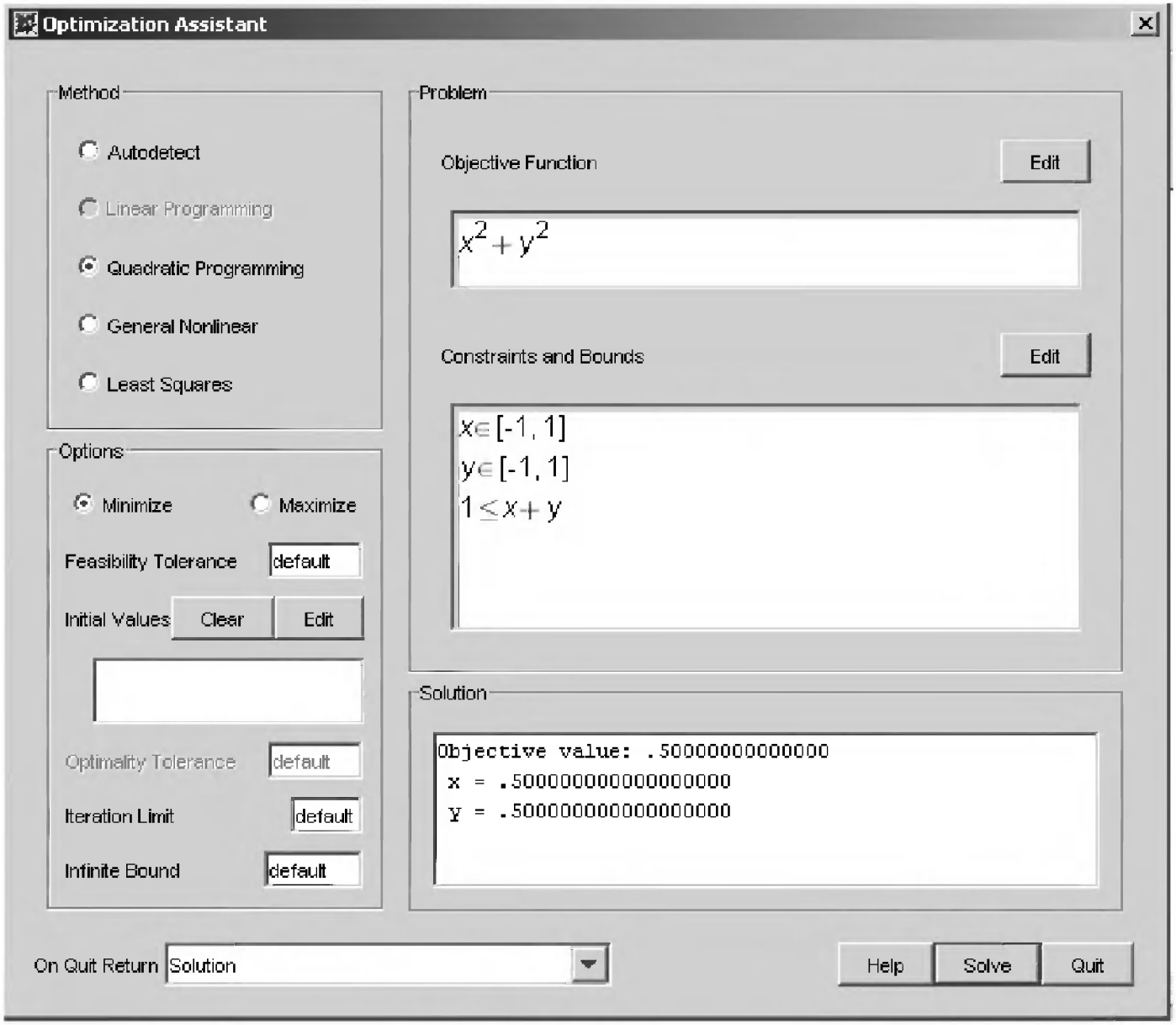


Рис. 8.27. Пример квадратичной оптимизации в Maplet-окне

Глава 9

Решение дифференциальных уравнений

Дифференциальные уравнения лежат в основе математического моделирования различных, в том числе физических, систем и устройств [1, 126–137]. Решению таких уравнений посвящена эта глава. В ней рассмотрено как аналитическое, так и численное решение дифференциальных уравнений различного вида – линейных и нелинейных, классических и специальных, например в частных производных и с учетом двухсторонних граничных условий. Описание сопровождается множеством наглядных примеров, реализованных в различных СКМ. Примеры, приведенные в этой главе для системы Maple 11, пригодны без ограничений для предшествующих версий Maple 9/9.5/10 и для недавно появившейся новой версии Maple 12.

9.1. Введение в решение дифференциальных уравнений

9.1.1. Дифференциальные уравнения первого порядка

Дифференциальные уравнения (*ДУ*) – это уравнения, при записи которых встречаются производные того или иного порядка. Простейшее *ДУ первого порядка*

$$y' = \frac{dy}{dx} = f(x,y) \tag{9.1}$$

в общем случае имеет решение в виде зависимости $y(x)$, которое зависит от вида функции $f(x, y)$ и начальных значений x_0 и y_0. Это решение может быть аналитическим, конечно-разностным или численным.

В качестве примера аналитического решения ДУ первого порядка в среде СКМ Maple 9.5 запишем *ДУ радиоактивного распада атомов* (N – число атомов в момент времени t, $g = 1/c$):

```
> restart: deq:=diff(N(t),t)=-g*N(t);
```

$$deq := \frac{d}{dt}\mathrm{N}(t) = -g\mathrm{N}(t)$$

Используя функцию `dsolve`, которая более подробно будет описана чуть позже, получим его общее аналитическое решение:

```
> dsolve(deq,N(t));
```

$$N(t) = \_C1\,\mathbf{e}^{(-gt)}$$

В решении присутствует *произвольная постоянная* $\_C1$. Но ее можно заменить на постоянную $\mathrm{N}(0) = \mathrm{N}_0$, означающую начальное число атомов в момент $t = 0$:

```
> dsolve({deq,N(0)=No},N(t));
```

$$N(t) = No\,\mathbf{e}^{(-gt)}$$

Если конкретно N_0=100 и g=4, то получим:

```
> No := 100;g:=3;
```

$$No := 100$$

$$\mathrm{g} := 3$$

```
> s:=dsolve({deq,N(0)=No},N(t));assign(s);
```

$$s := \mathrm{N}(t) = 100\mathrm{e}^{(-3t)}$$

Теперь мы можем воспользоваться полученной зависимостью $N(t)$ и построить ее график:

```
> plot(N(t),t=0..3,color=black);
```

Этот график, показанный на рис. 9.1 (вместе с представленными выше вычислениями), описывает хорошо известный апериодический экспоненциальный закон уменьшения числа атомов вещества в ходе его радиоактивного распада.

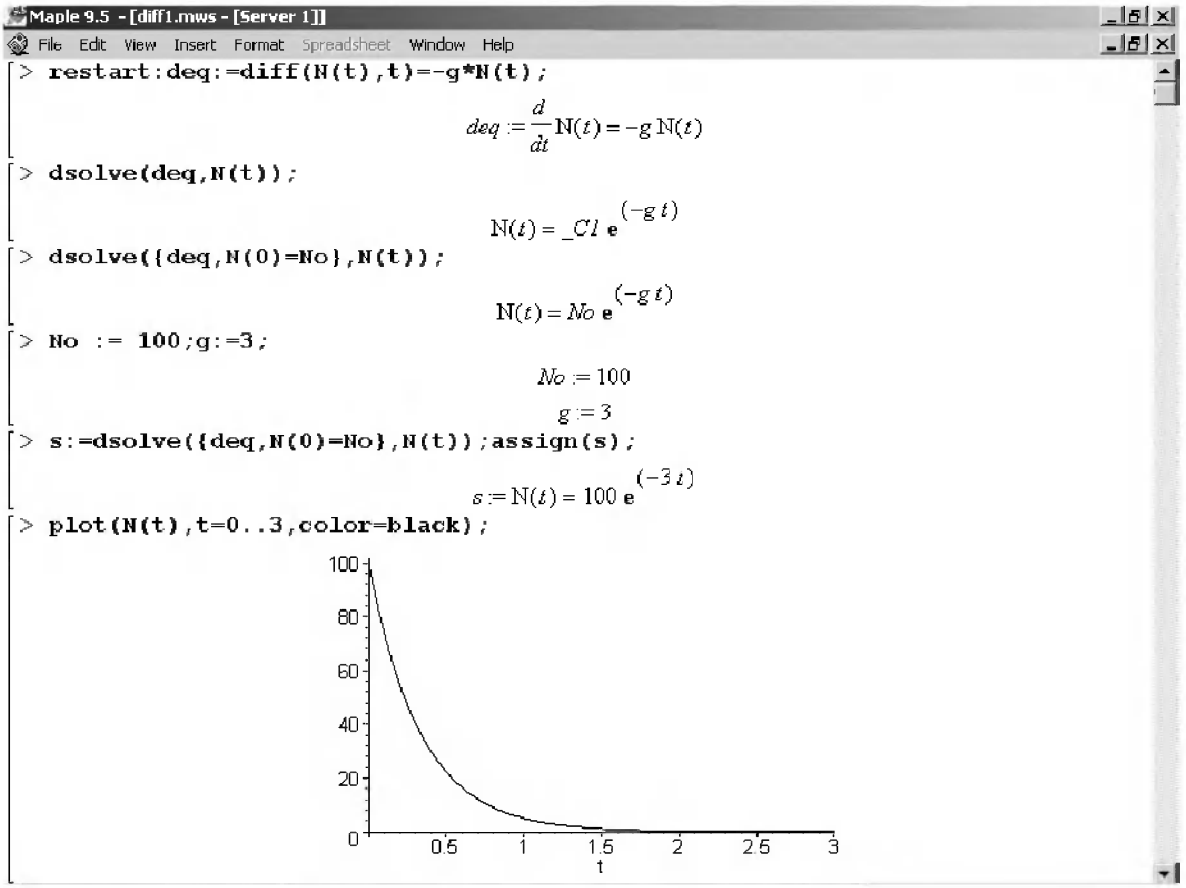


Рис. 9.1. Пример решения дифференциального уравнения радиоактивного распада

Подобные зависимости, кстати, характерны для напряжения на конденсаторе C при его разряде через резистор R, для тока в LR-цепи и для многих простых физических явлений, описывающихся дифференциальным уравнением первого порядка.

9.1.2. Системы дифференциальных уравнений

Встроенные в математические системы функции обычно решают *систему* из обыкновенных дифференциальных уравнений (ОДУ), представленную в *форме Коши*:

$$\begin{cases} y_1(x_0) = y_{0,1} \\ y_2(x_0) = y_{0,2} \\ \ldots\ldots\ldots\ldots \\ y_n(x_0) = y_{0,n} \end{cases}, \quad \begin{cases} y_1' = f_1(x, y_1, y_2, \ldots, y_n) \\ y_2' = f_2(x, y_1, y_2, \ldots, y_n) \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ y_n' = f_n(x, y_1, y_2, \ldots, y_n) \end{cases}.$$

Здесь левая система задает начальные условия, а вторая представляет систему ОДУ.

9.1.3. Сведение ДУ высокого порядка к системам ОДУ первого порядка

Часто встречаются *ДУ высокого порядка*:

$$y^{(n)} = f(x, y, y', y'', \ldots y^{(n-1)}),$$

где

$$y(x_0) = y_0,\ y'(x_0) = y_{0,1},\ y''(x_0) = y_{0,2},\ \ldots,\ y^{(n-1)}(x_0) = y_{0,n-1}.$$

Обозначив

$$y_1(x) = y(x),\ y_2(x) = y'(x),\ \ldots,\ y_n(x) = y^{(n-1)}(x)$$

и

$$y_{0,0} = y(x_0),\ y_{0,1} = y'(x_0),\ \ldots,\ y_{0,n-1} = y^{(n-1)}(x_0),$$

теперь решение этого уравнения можно свести к решению *системы ОДУ*:

$$\begin{cases} y_1(x_0) = y_{0,0} \\ y_2(x_0) = y_{0,1} \\ \ldots\ldots\ldots\ldots \\ y_n(x_0) = y_{0,n-1} \end{cases} \quad \begin{cases} y_1' = y_2 \\ y_2' = y_3 \\ \ldots\ldots\ldots \\ y_{n-1}' = y_n \\ y_n' = f(x, y_1, y_2, \ldots, y_{n-1}) \end{cases}$$

В таком виде ДУ n-го порядка может решаться стандартными средствами решения систем ОДУ, входящими в большинство математических систем.

9.1.4. Решение задачи на полет камня

В качестве примера аналитического решения системы дифференциальных уравнений рассмотрим постановку типичной задачи моделирования «Бросок камня», позволяющую описать полет камня-точки с массой m, брошенного под углом α_0 к горизонту. Пренебрегая сопротивлением воздуха и зная начальные координаты камня x_0 и y_0 и его начальную скорость v_0, получим следующую модель в виде дифференциальных уравнений:

$$m \cdot \frac{d^2x}{dt^2} = 0, \quad m \cdot \frac{d^2y}{dt^2} = -m \cdot g, \quad v_x = \frac{dx}{dt}, \quad v_y = \frac{dy}{dt} \tag{9.2}$$

при следующих начальных условиях:

$x(0) = x_0, \ y(0) = y_0, \ v_x(0) = v_0 \cdot \cos\alpha_0, \ v_y(0) = v_0 \cdot \sin\alpha_0.$

Надо найти зависимости $x(t), y(t), v_x(t), v_y(t)$.

Решение этой задачи есть в любом учебнике физики. Тем не менее выполним его. Из (9.2) запишем систему ОДУ первого порядка:

$$\frac{dv_x}{dt} = 0, \quad \frac{dv_y}{dt} = -g, \quad v_x = \frac{dx}{dt}, \quad v_y = \frac{dy}{dt}. \tag{9.3}$$

После интегрирования получим:

$$v_x(t) = C_1, \quad v_y(t) = C_2 - g \cdot t, \quad x(t) = C_3 + C_1 \cdot t, \quad y(t) = C_4 + C_2 \cdot t - \frac{g \cdot t^2}{2}. \tag{9.4}$$

Определив константы интегрирования из начальных условий, окончательно запишем:

$$x(t) = x_0 + v_0 \cdot \cos a_0 \cdot t \qquad y(t) = y_0 + v_0 \cdot \sin a_0 \cdot t - \frac{g \cdot t^2}{2}$$

$$v_x(t) = v_0 \cdot \cos a_0 \qquad v_y(t) = v_0 \cdot \sin \alpha_0 - g \cdot t$$

Из аналитического решения вытекает, что полет камня при отсутствии сопротивления воздуха происходит строго по параболической траектории, причем она на участках полета камня вверх и вниз симметрична.

9.1.5. Классификация дифференциальных уравнений в Maple

Дифференциальные уравнения могут быть самого разного вида. На рис. 9.2 представлен раздел справки Maple 9.5 с классификацией дифференциальных уравнений. В ней представлены:

- 20 дифференциальных уравнений первого порядка;
- 25 дифференциальных уравнений второго порядка;
- 6 типов дифференциальных уравнений высшего порядка;
- основные функции решения дифференциальных уравнений.

В качестве примера работы с классификатором выберем решение дифференциального уравнения Бернулли. Для этого активизируем на рис. 9.2 гиперссылку с его именем – Bernoulli. Появится окно справки по этому уравнению, показанное на рис. 9.3 с открытой позицией меню **Edit**.

С помощью команды **Copy Examples** в позиции **Edit** меню можно перенести примеры решения с окна справки в буфер Clipboard операционной системы Windows. После этого командой **Paste** в меню **Edit** окна документа можно пере-

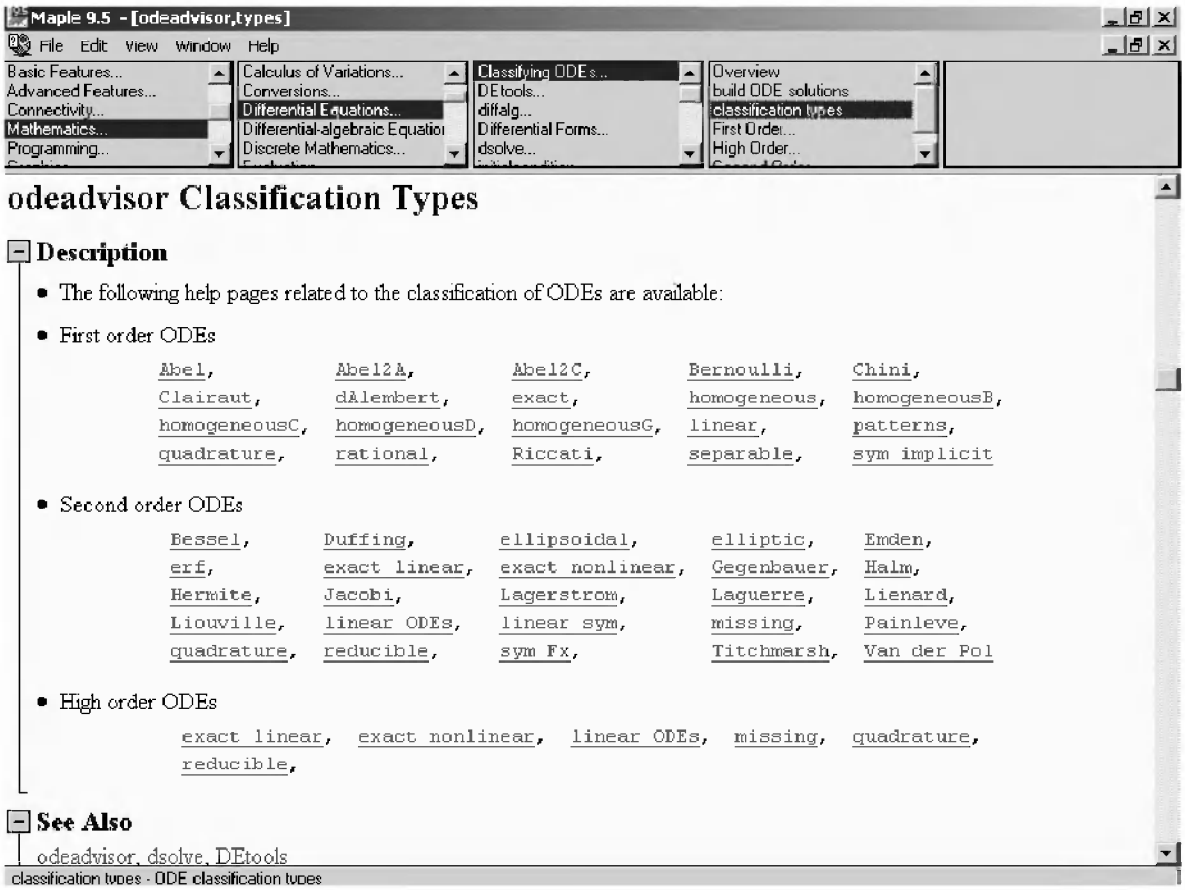


Рис. 9.2. Классификация дифференциальных уравнений

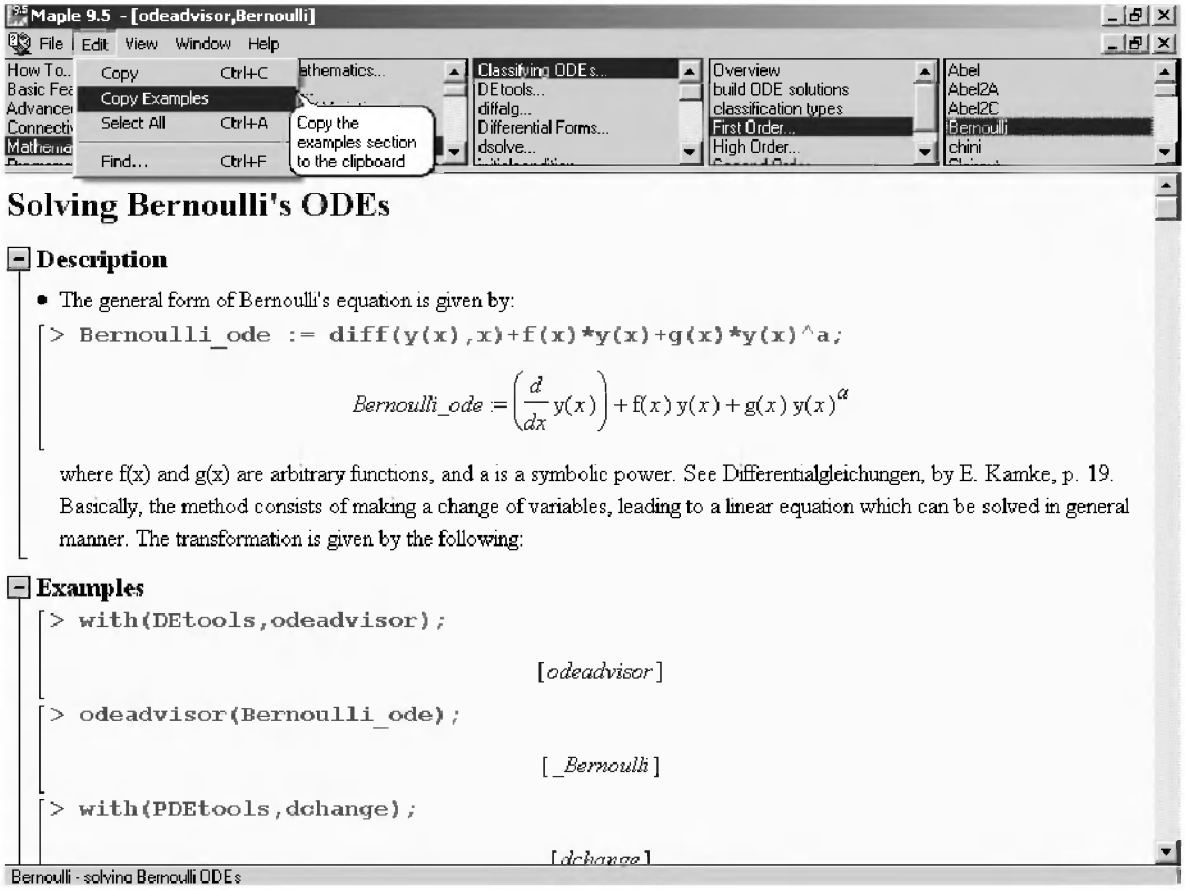


Рис. 9.3. Окно справки по решению дифференциального уравнения Бернулли

нести примеры в текущий документ – желательно (но не обязательно) новый. Теперь можно наблюдать решение выбранного дифференциального уравнения – рис. 9.4.

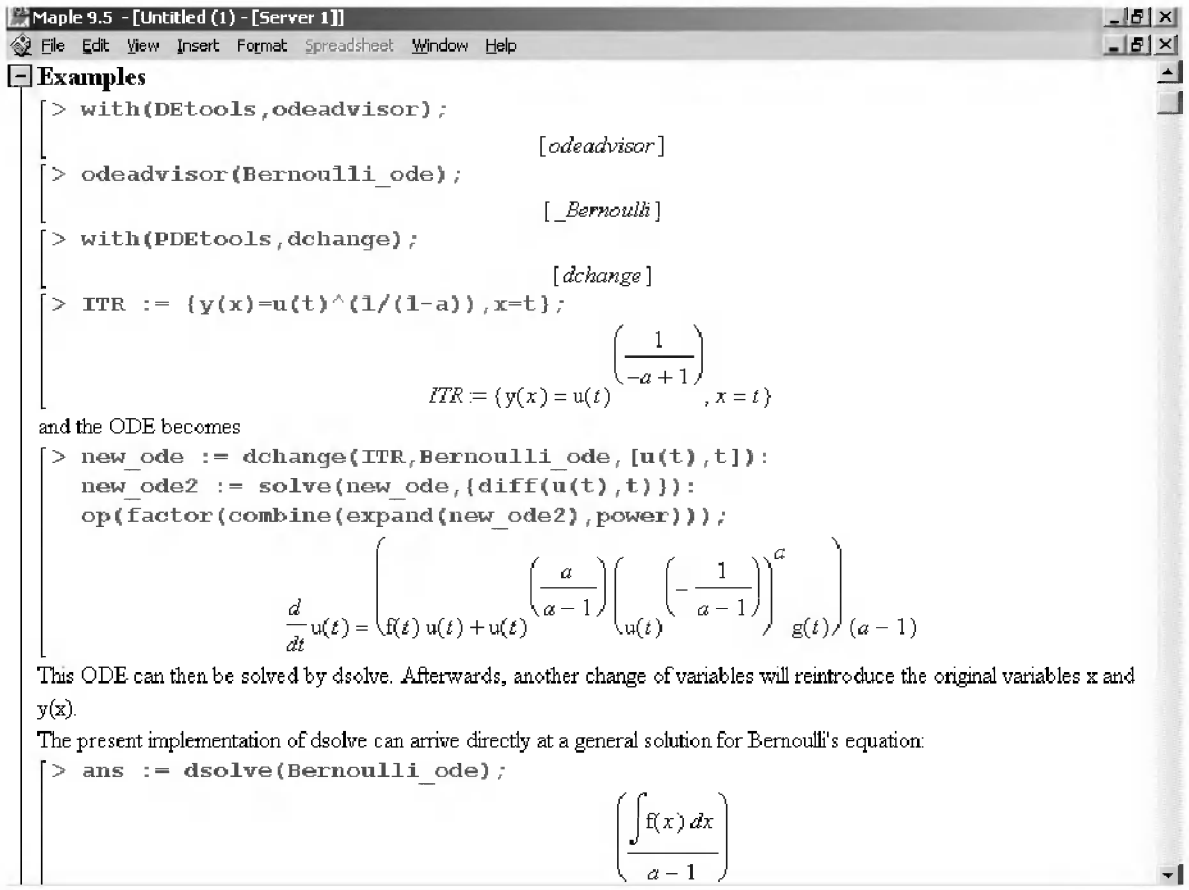


Рис. 9.4. Пример решения дифференциального уравнения Бернулли из справки

Уже в Maple 9.5 и в последующих версиях средства решения дифференциальных уравнений подверглись переработке. В систему введены новые методы решения для дифференциальных уравнений Абеля, Риккати и Матье и инициализации и решения уравнений с кусочными функциями, улучшены алгоритмы решения численными методами. Детальное описание этих новинок можно найти в справке по разделу What's New....

9.1.6. Функция решения ДУ dsolve

Maple позволяет решать одиночные дифференциальные уравнения и системы дифференциальных уравнений как аналитически, так и в численном виде. Для решения системы простых дифференциальных уравнений (*задача Коши*) используется функция `dsolve` в разных формах записи:

```
dsolve(ODE)
dsolve(ODE, y(x), extra_args)
dsolve({ODE, ICs}, y(x), extra_args)
dsolve({sysODE, ICs}, {funcs}, extra_args)
```

Здесь `ODE` – одно обыкновенное дифференциальное уравнение или система из дифференциальных уравнений первого порядка с указанием начальных условий, `y(x)` – функция одной переменной, `Ics` – выражение, задающее начальные условия, `{sysODE}` – множество дифференциальных уравнений, `{funcs}` – множество неопределенных функций, `extra_argument` – опция, задающая тип решения.

Параметр `extra_argument` задает класс решаемых уравнений. Отметим основные значения этого параметра:

- `exact` – аналитическое решение (принято по умолчанию);
- `explicit` – решение в явном виде;
- `system` – решение системы дифференциальных уравнений;
- `ICs` – решение системы дифференциальных уравнений с заданными начальными условиями;
- `formal series` – решение в форме степенного многочлена;
- `integral transform` – решение на основе интегральных преобразований Лапласа, Фурье и др.;
- `series` – решение в виде ряда с порядком, указываемым значением переменной `Order`;
- `numeric` – решение в численном виде.

Возможны и другие опции, подробное описание которых выходит за рамки данной книги. Его можно найти в справке по этой функции, вызываемой командой `?dsolve`.

Для решения задачи Коши в параметры `dsolve` надо включать начальные условия, а при решении краевых задач – краевые условия. Если Maple способна найти решение при числе начальных или краевых условий меньше порядка системы, то в решении будут появляться неопределенные константы вида `_C1`, `_C2` и т. д. Они же могут быть при аналитическом решении системы, когда начальные условия не заданы. Если решение найдено в неявном виде, то в нем появится параметр `_T`. По умолчанию функция `dsolve` автоматически выбирает наиболее подходящий метод решения дифференциальных уравнений. Однако в параметрах функции `dsolve` в квадратных скобках можно указать предпочтительный метод решения дифференциальных уравнений. Допустимы следующие методы:

```
> 'dsolve/methods'[1];
```

[*quadrature*, *linear*, *Bernoulli*, *separable*, *inverse_linear*, *homogeneous*,
Chini, *lin_sym*, *exact*, *Abel*, *pot_sym*]

Более полную информацию о каждом методе можно получить, используя команду `?dsolve,method` и указав в ней конкретный метод. Например, команда `?dsolve,linear` вызовет появление страницы справочной системы с подробным описанием линейного метода решения дифференциальных уравнений.

9.1.7. Уровни решения дифференциальных уравнений в Maple

Решение дифференциальных уравнений может сопровождаться различными комментариями. Команда

```
infolevel[dsolve] := n:,
```

где n – целое число от 0 до 5, управляет детальностью вывода. По умолчанию задано n = 0. Значение n = 5 дает максимально детальный вывод.

Производные при записи дифференциальных уравнений могут задаваться функцией `diff` или оператором дифференцирования `D`. Выражение `sysODE` должно иметь структуру множества и содержать, помимо самой системы уравнений, их начальные условия.

Читателю, всерьез интересующемуся проблематикой решения линейных дифференциальных уравнений, стоит внимательно просмотреть разделы справки по ним и ознакомиться с демонстрационным файлом linearoade.mws, содержащим примеры решения таких уравнений в закрытой форме.

9.2. Примеры решения дифференциальных уравнений

9.2.1. Примеры аналитического решения ОДУ первого порядка

Отвлекшись от физики, приведем несколько примеров на составление и решение дифференциальных уравнений первого порядка в аналитическом виде:

```
> dsolve(diff(y(x),x)-a*x=0,y(x));
```

$$y(x)=\frac{ax^2}{2}+\_C1$$

```
> dsolve(diff(y(x),x)-y(x)=exp(-x),y(x));
```

$$y(x)=\left(-\frac{1}{2}\mathbf{e}^{(-2x)}+\_C1\right)\mathbf{e}^{x}$$

```
> dsolve(diff(y(x),x)-y(x)=sin(x)*x,y(x));
```

$$y(x)=-\frac{1}{2}\cos(x)x-\frac{1}{2}\cos(x)-\frac{1}{2}\sin(x)x+\mathbf{e}^{x}\,\_C1$$

```
> infolevel[dsolve] := 3:
> dsolve(diff(y(x),x)-y(x)=sin(x)*x,y(x));
```

```
Methods for first order ODEs:
- Trying classification methods --
trying a quadrature
trying 1st order linear
<- 1st order linear successful
```

$$y(x) = -\frac{1}{2}\cos(x)x - \frac{1}{2}\cos(x) - \frac{1}{2}\sin(x)x + e^{x}\_C1$$

Обратите внимание на вывод в последнем примере. Он дан при уровне вывода n=3.

Следующие примеры иллюстрируют возможность решения одного и того же дифференциального уравнения `ode_L` разными методами:

```
> ode_L := sin(x)*diff(y(x),x)-cos(x)*y(x)=0;
```

$$ode\_L := \sin(x)\left(\frac{\partial}{\partial x}y(x)\right) - \cos(x)y(x) = 0$$

```
> dsolve(ode_L, [linear], useInt);
```

$$y(x) = \_C1 e^{\left(\int \frac{\cos(x)}{\sin(x)} dx\right)}$$

```
> value(%);
```

$$y(x) = \_C1 \sin(x)$$

```
> dsolve(ode_L, [separable], useInt);
```

$$\int \frac{\cos(x)}{\sin(x)} dx - \int^{y(x)} \frac{1}{\_a} d\_a + \_C1 = 0$$

```
> value(%);
```

$$\ln(\sin(x)) - \ln(y(x)) + \_C1 = 0$$

```
> mu := intfactor(ode_L);
```

$$\mu := \mathrm{intfactor}\left(\sin(x)\left(\frac{\partial}{\partial x}y(x)\right) - \cos(x)y(x) = 0\right)$$

```
> dsolve( mu*ode_L, [exact], useInt);
```

$$y(x) = \frac{\_C1}{\frac{1}{2}\tan\left(\frac{1}{2}x\right) + \frac{\frac{1}{2}}{\tan\left(\frac{1}{2}x\right)}}$$

Разумеется, приведенными примерами далеко не исчерпываются возможности аналитического решения дифференциальных уравнений.

9.2.2. Полет тела, брошенного вверх

Из приведенных выше примеров видно, что для задания производной используется ранее рассмотренная функция `diff`. С помощью символа `$` в ней можно задать производную более высокого порядка.

В соответствии со вторым законом Ньютона многие физические явления, связанные с движением объектов, описываются дифференциальными уравнениями второго порядка. Ниже дан пример задания и решения такого уравнения, описывающего движение тела, брошенного вверх на высоте h_0 со скоростью v_0 при ускорении свободного падения g:

```
> restart; eq2:=diff(h(t),t$2) = -g;
```

$$eq2 := \frac{d^2}{dt^2}\mathrm{h}(t) = -g$$

```
> dsolve({eq2,h(0)=h[0],D(h)(0)=v[0]},h(t));assign(s2);
```

$$\mathrm{h}(t) = -\frac{g\,t^2}{2} + v_0 t + h_0$$

Итак, получено общее уравнение для временной зависимости высоты тела $h(t)$. Разумеется, ее можно конкретизировать, например для случая, когда g=9.8, h_0=10 и v_0=100:

```
> g:=9.8:
> s2:=dsolve({eq2,h(0)=10,D(h)(0)=100},h(t));assign(s2);
```

$$s2 := \mathrm{h}(t) = -\frac{49}{10}t^2 + 100t + 10$$

```
> plot(h(t),t=0..20,color=black);
```

Зависимость высоты тела от времени $h(t)$ представлена на рис. 9.5. Нетрудно заметить, что высота полета тела вначале растет и, достигнув максимума, начинает снижаться. Оговоримся, что сопротивление воздуха в данном примере не учитывается, что позволяет считать задачу линейной. Полученное с помощью Maple решение совпадает с полученным в примере, описанном в разделе 9.1.3.

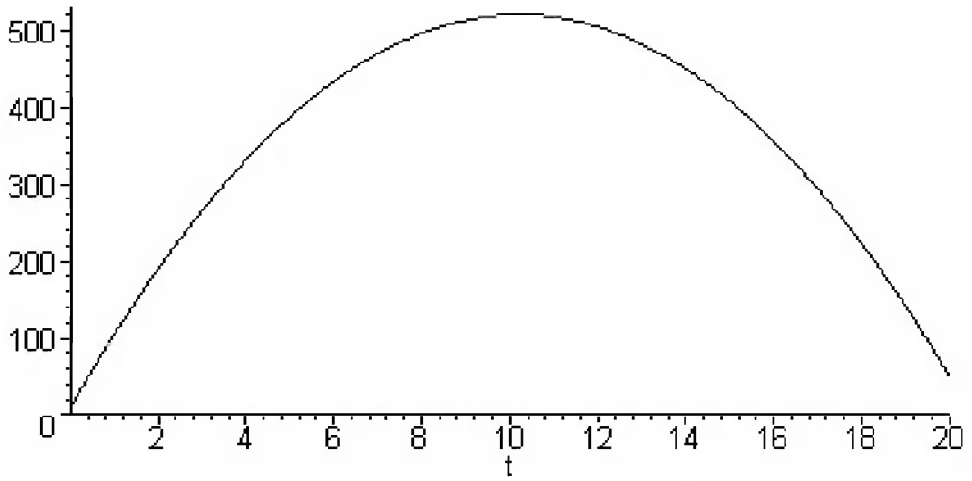


Рис. 9.5. Зависимость высоты полета тела от времени h(t)

9.2.3. Поведение идеального гармонического осциллятора

Еще одним классическим применением дифференциальных уравнений второго порядка является решение уравнения идеального гармонического осциллятора:

```
> restart:eq3:=diff(y(t),t$2)=-omega^2*y(t);
```

$$eq3 := \frac{d^2}{dt^2}y(t) = -\omega^2 y(t)$$

```
> dsolve(eq3,y(t));
```

$$y(t) = \_C1 \sin(\omega t) + \_C2 \cos(\omega t)$$

```
> s:=dsolve({eq3,y(0)=-1,D(y)(0)=1},y(t));
```

$$s := y(t) = \frac{\sin(\omega t)}{\omega} - \cos(\omega t)$$

```
> assign(s);omega:=2;
```

$$\omega := 2$$

```
> plot(y(t),t=0..20,color=black):
```

График решения этого уравнения (рис. 9.6) представляет хорошо известную синусоидальную функцию. Интересно, что амплитуда колебаний в общем случае

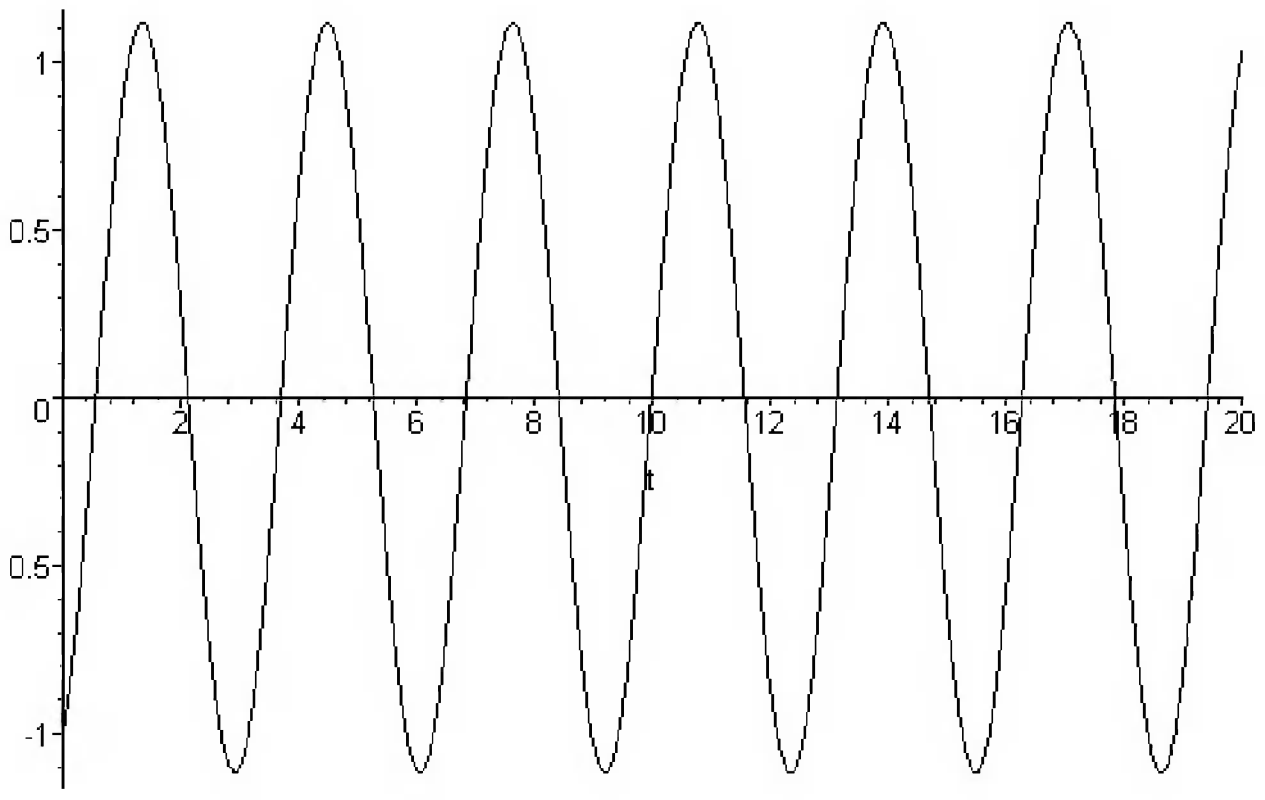


Рис. 9.6. Решение уравнения гармонического осциллятора

отлична от 1 и зависит от значения y(0) – при y(0)=0 она равна 1 (в нашем случае синусоида начинается со значения y(0)=–1). Подобным осциллятором может быть LC-контур или механический маятник без потерь.

9.2.4. Дополнительные примеры решения дифференциальных уравнений второго порядка

Ниже представлено решение еще двух дифференциальных уравнений второго порядка в аналитическом виде:

```
> dsolve(diff(y(x),x$2)-diff(y(x),x)=sin(x),y(x));
```

$$y(x)=-\frac{1}{2}\sin(x)+\frac{1}{2}\cos(x)+\mathbf{e}^{x}\_C1+\_C2$$

```
> de:=m*diff(y(x),x$2)-k*diff(y(x),x);
```

$$de:=m\left(\frac{\partial^2}{\partial x^2}y(x)\right)-k\left(\frac{\partial}{\partial x}y(x)\right)$$

```
> yx0:=y(0)=0,y(1)=1;
```

$$yx0:=y(0)=0,\ y(1)=1$$

```
> dsolve({de,yx0},y(x));
```

$$y(x)=-\frac{1}{-1+\mathbf{e}^{\left(\frac{k}{m}\right)}}+\frac{\mathbf{e}^{\left(\frac{kx}{m}\right)}}{-1+\mathbf{e}^{\left(\frac{k}{m}\right)}}$$

Ряд примеров на применение дифференциальных уравнений второго порядка при решении практических математических и физических задач вы найдете в главе 15.

9.2.5. Решение систем дифференциальных уравнений

На рис. 9.7 представлено решение системы из двух дифференциальных уравнений различными методами – в явном виде, в виде разложения в ряд и с использованием преобразования Лапласа. Здесь следует отметить, что решение в виде ряда является приближенным. Поэтому полученные в данном случае аналитические выражения отличаются от явного решения и решения с применением преобразования Лапласа.

Следует отметить, что, несмотря на обширные возможности Maple в области аналитического решения дифференциальных уравнений, оно возможно далеко не всегда. Поэтому, если не удается получить такое решение, полезно попытаться найти решение в численном виде.

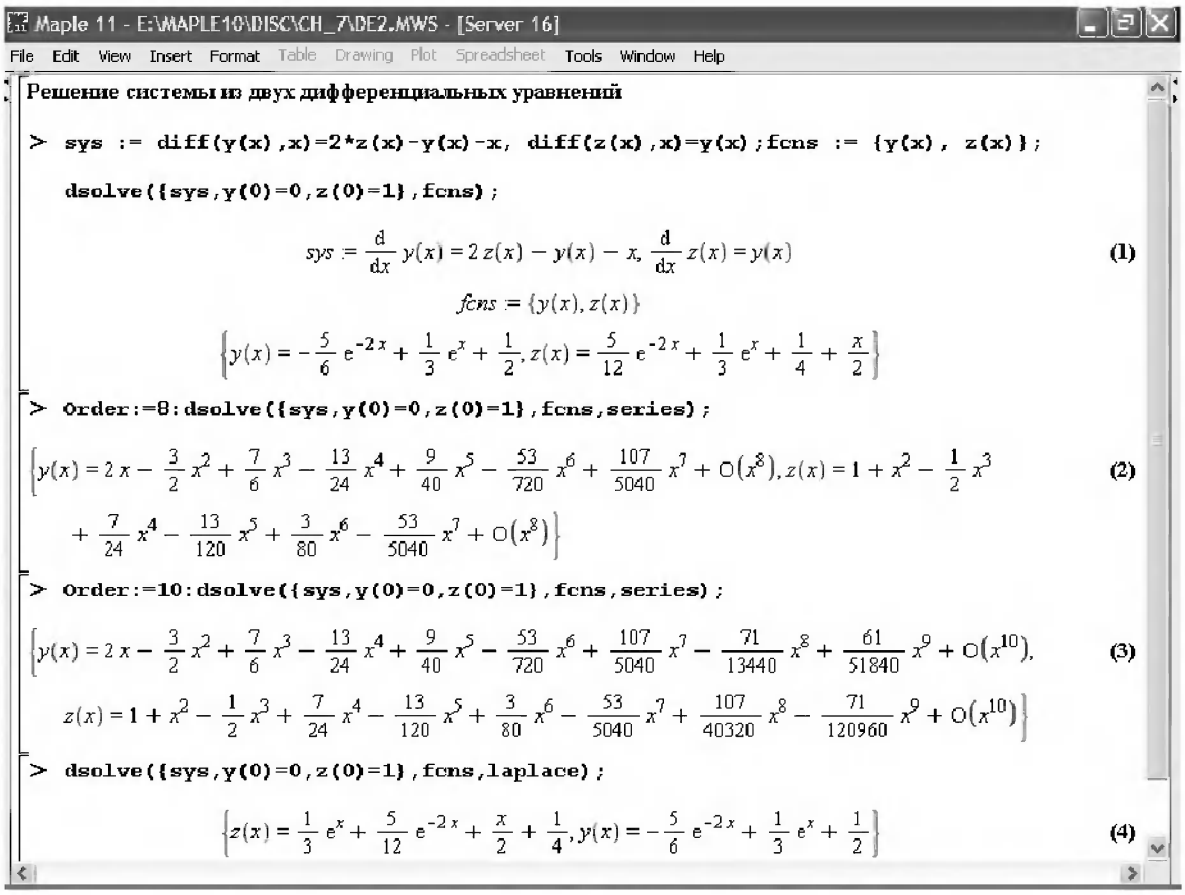


Рис. 9.7. Решение системы из двух дифференциальных уравнений различными методами

9.3. Специальные средства решения дифференциальных уравнений

9.3.1. Численное решение дифференциальных уравнений

Для решения дифференциальных уравнений в численном виде в Maple используется функция `dsolve` с параметром `numeric` или `type=numeric`. При этом решение возвращается в виде специальной процедуры, по умолчанию реализующей широко известный метод решения дифференциальных уравнений Рунге-Кутта–Фелберга порядков 4 и 5 (в зависимости от условий адаптации решения к скорости его изменения). Эта процедура называется `rkf45` и символически выводится (без тела) при попытке решения заданной системы дифференциальных уравнений. Последнее достаточно наглядно иллюстрирует рис. 9.8. Решение показано в среде Maple 11.

Указанная процедура возвращает особый тип данных, позволяющих найти решение в любой точке или построить график решения (или решений). Для графического отображения Maple предлагает ряд возможностей, и одна из них пред-

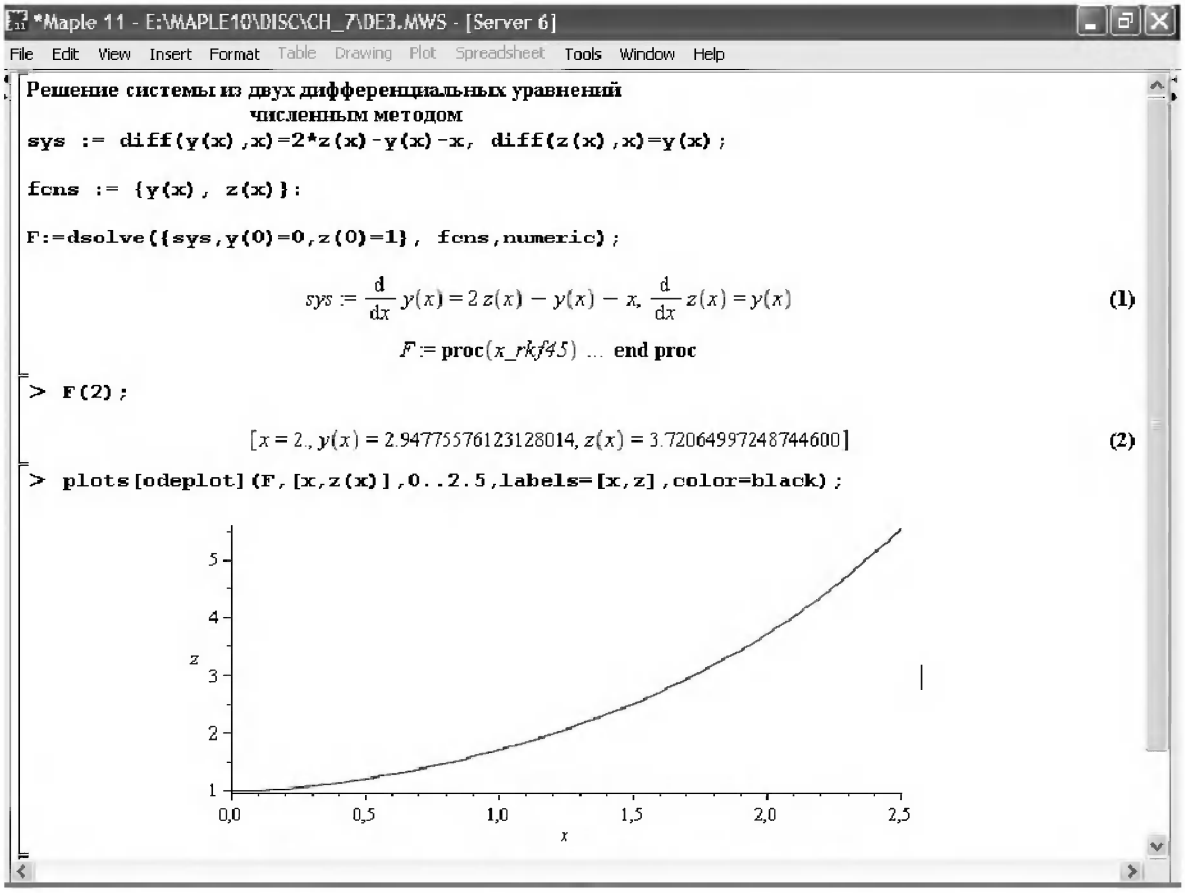


Рис. 9.8. Решение системы дифференциальных уравнений численным методом rkf45 с выводом графика решения

ставлена на рис. 9.8 – см. последнюю строку ввода. При этом используется функция `plot[odeplot]` из пакета `odeplot`, предназначенного для визуализации решений дифференциальных уравнений.

В список параметров функции `dsolve` можно явным образом включить указание на метод решения, например опция `mathod=dverk78` задает решение непрерывным методом Рунге-Кутта порядка 7 или 8. Вообще говоря, численное решение дифференциальных уравнений можно производить одним из следующих методов:

- `classical` – одна из восьми версий классического метода, используемого по умолчанию;
- `rkf45` – метод Рунге-Кутта 4 или 5 порядка, модифицированный Фелбергом;
- `dverk78` – непрерывный метод Рунге-Кутта порядка 7 или 8;
- `gear` – одна из двух версий одношагового экстраполяционного метода Гира;
- `mgear` – одна из трех версий многошагового экстраполяционного метода Гира;
- `lsode` – одна из восьми версий Ливенморского решателя жестких дифференциальных уравнений;
- `taylorseries` – метод разложения в ряд Тейлора.

Обилие используемых методов расширяет возможности решения дифференциальных уравнений в численном виде. Большинство пользователей Maple вполне устроит автоматический выбор метода решения по умолчанию. Однако в сложных случаях возможна прямая установка одного из указанных выше методов. С деталями реализации методов можно ознакомиться по справочной системе (см. также примеры на решение дифференциальных уравнений в системе Mathcad в этой главе).

С помощью параметра `'abserr'=aerr` можно задать величину абсолютной погрешности решения, а с помощью `'minerr'=mine` – минимальную величину погрешности. В большинстве случаев эти величины, заданные по умолчанию, оказываются приемлемыми для расчетов.

Maple реализует адаптируемые к ходу решения методы, при которых шаг решения `h` автоматически меняется, подстраиваясь под условия решения. Так, если прогнозируемая погрешность решения становится больше заданной, шаг решения автоматически уменьшается. Более того, система Maple способна автоматически выбирать наиболее подходящий для решаемой задачи метод решения.

Еще один пример решения системы дифференциальных уравнений представлен на рис. 9.9 (решение выполнено в среде Maple 11). Здесь на одном графике даны зависимости $y(x)$ и $z(x)$, представляющие полное решение заданной системы. При этом процедура имеет особый вид `listprocedure` и для преобразования списка выходных данных в векторы решения `Y` и `Z` используется функция `subs`. Результаты решений аналогичны предшествующему примеру (рис. 9.8).

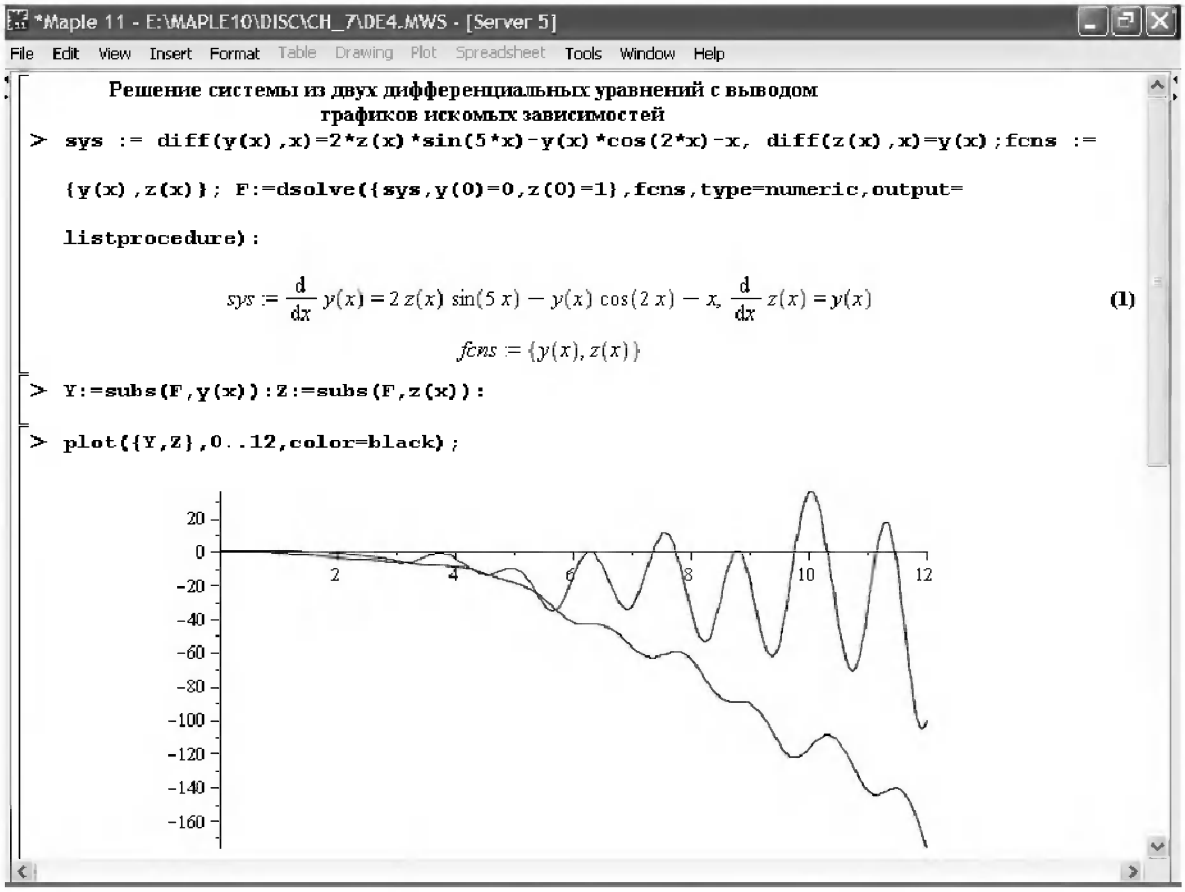


Рис. 9.9. Решение системы дифференциальных уравнений численным методом с выводом всех графиков искомых зависимостей

Для решения достаточно сложных задач полезны специальная структура `DESol` для решения дифференциальных уравнений и инструментальный пакет `SEtools`, содержащий самые изысканные средства для графической визуализации результатов решения дифференциальных уравнений. Эти средства мы более подробно рассмотрим в дальнейшем.

При решении некоторых задач физики и радиоэлектроники выбираемый по умолчанию шаг изменения аргумента x или $t - h$ может привести к неустойчивости решения. Неустойчивости можно избежать рядом способов. Можно, например, нормировать уравнения, избегая необходимости использования малого шага. А можно задать заведомо малый шаг. Например, при `method=classical` для этого служит параметр `stepsize=h`.

9.3.2. Дифференциальные уравнения с кусочными функциями

Функции кусочного типа широко используются при математическом моделировании различных физических объектов и систем. В основе такого моделирования обычно лежит решение дифференциальных уравнений, описывающих поведение объектов и систем. Покажем возможность применения кусочных функций для решения дифференциальных уравнений.

Ниже представлено задание дифференциального уравнения первого порядка, содержащего кусочную функцию:

```
> restart;
> eq := diff(y(x), x)+ piecewise(x<x^2-3, exp(x/2))*y(x);
```

$$eq := \left(\frac{d}{dx}\mathrm{y}(x)\right) + \left(\begin{cases} \mathbf{e}^{\left(\frac{x}{2}\right)} & x < x^{2-3} \\ 0 & otherwise \end{cases}\right)\mathrm{y}(x)$$

Используя функцию `dsolve`, выполним решение этого дифференциального уравнения:

```
> dsolve(eq, y(x));
```

$$y(x) = \begin{cases} \_C1\mathbf{e}^{\left(-2\mathbf{e}^{\left(\frac{x}{2}\right)}\right)} & x < \frac{1}{2} - \frac{\sqrt{13}}{2} \\ \_C1\mathbf{e}^{\left(-2\mathbf{e}^{\left(1/4-\frac{\sqrt{13}}{4}\right)}\right)} & x < \frac{1}{2} + \frac{\sqrt{13}}{2} \\ \_C1\mathbf{e}^{\left(-2\mathbf{e}^{\left(1/4-\frac{\sqrt{13}}{4}\right)} - 2\mathbf{e}^{\left(\frac{x}{2}\right)} + 2\mathbf{e}^{\left(1/4+\frac{\sqrt{13}}{4}\right)}\right)} & \frac{1}{2} + \frac{\sqrt{13}}{2} \le x \end{cases}$$

Нетрудно заметить, что результат получен также в форме кусочной функции, полностью определяющей решение на трех интервалах изменения x.

Приведем пример решения дифференциального уравнения второго порядка с кусочной функцией:

```
> eq := diff(y(x), x$2) + x*diff(y(x), x) + y(x) = piecewise(x > 0, 1);
```

$$eq := \left(\frac{\partial^2}{\partial x^2} y(x)\right) + x\left(\frac{\partial}{\partial x} y(x)\right) + y(x) = \begin{cases} 1 & 0 < x \\ 0 & otherwise \end{cases}$$

```
> dsolve(eq, y(x));
```

$$y(x) = \begin{cases} \_C1\,\mathbf{e}^{\left(-2\mathbf{e}^{\left(\frac{x}{2}\right)}\right)} & x < \frac{1}{2} - \frac{\sqrt{13}}{2} \\ \_C1\,\mathbf{e}^{\left(-2\mathbf{e}^{\left(1/4-\frac{\sqrt{13}}{4}\right)}\right)} & x < \frac{1}{2} + \frac{\sqrt{13}}{2} \\ \_C1\,\mathbf{e}^{\left(-2\mathbf{e}^{\left(1/4-\frac{\sqrt{13}}{4}\right)} - 2\mathbf{e}^{\left(\frac{x}{2}\right)} + 2\mathbf{e}^{\left(1/4+\frac{\sqrt{13}}{4}\right)}\right)} & \frac{1}{2} + \frac{\sqrt{13}}{2} \le x \end{cases}$$

Любопытно отметить, что результат выполнения этого примера в предшествующей версии Maple был иным. Это говорит о постоянной работе разработчика системы над ядром символьных операций и необходимости перепроверки результатов символьных операций.

В заключение этого раздела приведем пример решения нелинейного дифференциального уравнения Риккати с кусочной функцией:

```
> eq := diff(y(x), x)=piecewise(x>0, x)*y(x)^2;
```

$$eq := \frac{\partial}{\partial x} y(x) = \begin{pmatrix} \{ & x & 0 < x \\ & 0 & otherwise \end{pmatrix} y(x)^2$$

```
> dsolve({ y(0) = 1, eq }, y(x));
```

$$y(x) = \begin{cases} 1 & x < 0 \\ -\frac{2}{x^2 - 2} & 0 \ge x \end{cases}$$

В ряде случаев желательна проверка решения дифференциальных уравнений. Ниже показано, как она делается для последнего уравнения:

```
> simplify( eval(subs(%, eq)));
```

$$\begin{cases} 0 & x \le 0 \\ \frac{4x}{(x^2-2)^2} & 0 < x \end{cases} = \begin{cases} 0 & x \le 0 \\ \frac{4x}{(x^2-2)^2} & 0 < x \end{cases}$$

Как видно из приведенных достаточно простых и наглядных примеров, результаты решения дифференциальных уравнений с кусочными функциями могут быть довольно громоздкими. Это, однако, не мешает эффективному применению функций этого класса.

9.3.3. Структура неявного представления дифференциальных уравнений – DESol

В ряде случаев иметь явное представление дифференциальных уравнений нецелесообразно. Для неявного их представления в Maple введена специальная структура

```
DESol(expr,vars),
```

где `exprs` – выражение для исходной системы дифференциальных уравнений, `vars` – заданный в виде опции список переменных (или одна переменная).

Структура `DESol` образует некоторый объект, дающий представление о дифференциальных уравнениях, чем-то напоминающее `RootOf`. С этим объектом можно обращаться как с функцией, то есть его можно интегрировать, дифференцировать, получать разложение в ряд и вычислять численными методами.

На рис. 9.10 показаны примеры применения структуры `DESol`.

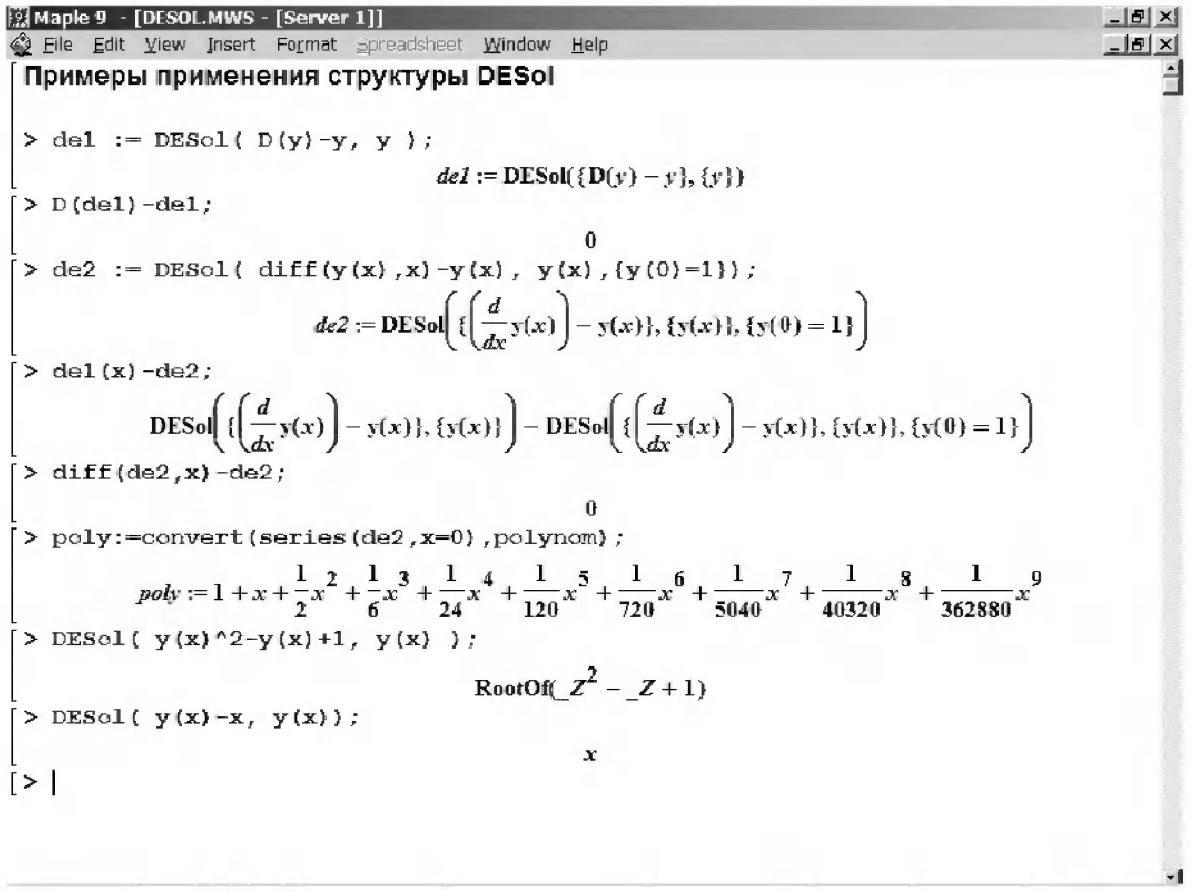


Рис. 9.10. Примеры применения структуры DESol (Maple 9)

Обратите внимание на последний пример – в нем структура DESol использована для получения решения дифференциального уравнения в виде степенного ряда.

9.4. Инструментальный пакет решения дифференциальных уравнений DEtools

9.4.1. Средства пакета DEtools

Решение дифференциальных уравнений самых различных типов – одно из достоинств системы Maple. Пакет DEtools предоставляет ряд полезных функций для решения дифференциальных уравнений и систем с такими уравнениями. Для загрузки пакета используется команда

```
> with(DEtools):
```

Этот пакет дает самые изысканные средства для аналитического и численного решений дифференциальных уравнений и систем с ними. По сравнению с версией Maple V R5 число функций данного пакета в Maple 9.5 возросло в несколько раз. Многие графические функции пакета DEtools были уже описаны. Ниже приводятся полные наименования тех функций, которые есть во всех реализациях системы Maple:

- DEnormal – возвращает нормализованную форму дифференциальных уравнений;
- DEplot – строит графики решения дифференциальных уравнений;
- DEplot3d – строит трехмерные графики для решения систем дифференциальных уравнений;
- Dchangevar – изменение переменных в дифференциальных уравнениях;
- PDEchangecoords – изменение координатных систем для дифференциальных уравнений в частных производных;
- PDEplot – построение графиков решения дифференциальных уравнений в частных производных;
- autonomous – тестирует дифференциальные уравнения на автономность;
- convertAlg – возвращает список коэффициентов для дифференциальных уравнений;
- convertsys – преобразует систему дифференциальных уравнений в систему одиночных уравнений;
- dfieldplot – строит график решения дифференциальных уравнений в виде векторного поля;
- indicialeq – преобразует дифференциальные уравнения в полиномиальные;
- phaseportrait – строит график решения дифференциальных уравнений в форме фазового портрета;
- reduceOrder – понижает порядок дифференциальных уравнений;

- `regularsp` – вычисляет регулярные особые точки для дифференциальных уравнений второго порядка;
- `translate` – преобразует дифференциальные уравнения в список операторов;
- `untranslate` – преобразует список операторов в дифференциальные уравнения;
- `varparam` – находит общее решение дифференциальных уравнений методом вариации параметров.

Применение этих функций гарантирует совместимость документов старых реализаций Maple.

9.4.2. Консультант по дифференциальным уравнениям

Для выявления свойств дифференциальных уравнений в Maple 9.5 в составе пакета DEtools имеется консультант (адвизор), вводимый следующей функцией:

```
odeadvisor(ODE)    odeadvisor(ODE, y(x), [type1, type2, ...], help)
```

Здесь ODE – одиночное дифференциальное уравнение, y(x) – неопределенная (определяемая функция), type1, type2, ... – опционально заданное множество типов, которые классифицируются, и help – опционально заданное указание на вывод страницы справки по методу решения.

Примеры работы с классификатором представлены ниже:

```
> with(DEtools): ODE := x*diff(y(x),x)+a*y(x)+b*x^2;
```

$$ODE := x\left(\frac{d}{dx}\mathrm{y}(x)\right)+a\mathrm{y}(x)+bx^2$$

```
> odeadvisor(ODE);
```

$$[\_linear]$$

```
> ODE1 := x*diff(y(x)^2,x)+a*y(x)+b*x^2;
```

$$ODE1 := 2x\mathrm{y}(x)\left(\frac{d}{dx}\mathrm{y}(x)\right)+a\mathrm{y}(x)+bx^2$$

```
> odeadvisor(ODE1);
```

$$[\_rational, [\_Abel, 2nd\ type, class\ B]]$$

```
> ODE2 := diff(y(x),x,x,x)+D(g)(y(x))*diff(y(x),x)^3
  + 2*g(y(x))*diff(y(x),x)*diff(y(x),x,x)
  + diff(f(x),x)*diff(y(x),x) + f(x)*diff(y(x),x,x) = 0;
```

$$ODE2 := \left(\frac{d^3}{dx^3}\mathrm{y}(x)\right)+D(g)(\mathrm{y}(x))\left(\frac{d}{dx}\mathrm{y}(x)\right)^3+2g(\mathrm{y}(x))\left(\frac{d}{dx}\mathrm{y}(x)\right)\left(\frac{d^2}{dx^2}\mathrm{y}(x)\right)$$
$$+\left(\frac{d}{dx}\mathrm{f}(x)\right)\left(\frac{d}{dx}\mathrm{y}(x)\right)+\mathrm{f}(x)\left(\frac{d^2}{dx^2}\mathrm{y}(x)\right)=0$$

```
> odeadvisor(ODE2,y(x));
```

$$[[\_3rd\_order, \_exact, \_nonlinear], [3rd\_order, \_reducible, \_mu\_y2]]$$

9.4.3. Основные функции пакета DEtools

Рассмотрим наиболее важные функции этого пакета. Функция

```
autonomous(des,vars,ivar)
```

тестирует дифференциальное уравнение (или систему) `des`. Ее параметрами, помимо `des`, являются независимая переменная `ivar` и зависимая переменная `dvar`. Следующие примеры поясняют применение этой функции:

```
> autonomous(sin(z(t)-z(t)^2)*(D@@4)(z)(t)-cos(z(t))-5,z,t);
```

true

```
> DE:=diff(x(s),s)-x(s)*cos(arctan(x(s)))=arctan(s):
> autonomous(DE,{x},s);
```

false

В разделе 9.4.4 описание этой функции будет продолжено. Функция `Dchangevar` используется для обеспечения замен (подстановок) в дифференциальных уравнениях:

```
Dchangevar(trans, deqns, c_ivar, n_ivar)
Dchangevar(tran1, tran2, …, tranN, deqns, c_ivar, n_ivar)
```

В первом случае `trans` – список или множество уравнений, которые подставляются в дифференциальное уравнение, список или множество дифференциальных уравнений `deqns`. При этом `c_ivar` – имя текущей переменной, `n_ivar` – имя новой переменной (его задавать необязательно). Во второй форме для подстановки используются уравнения `tran1`, `tran2`, ...

Ниже представлены примеры применения функции `Dchangevar`:

```
# Преобразование 1-го типа
> Dchangevar(m(x)=l(x)*sin(x),n(x)=k(x),[D(m)(x)=m(x),
(D@@2)(n)(x)= n(x)^2],x);
```

$$[\mathrm{D}(l)(x)\sin(x) + 1(x)\cos(x) = 1(x)\sin(x), (\mathrm{d}^{(2)})(k)(x) = \mathrm{k}(x)^2]$$

```
> Dchangevar(c=d,e=sin(f),{D(c),(D@@2)(e)},dummy);
```

$$[\mathrm{D}(d), \{\mathrm{D}^{(2)})(\sin(f))]$$

```
# Преобразование 2-го типа
> Dchangevar(t=arctan(tau),diff(x(t),t)=sin(t),t,tau);
```

$$\mathrm{D}(x)(\mathrm{arctab}(\tau)) = \sin(\arctan(\tau))$$

```
> Dchangevar(x=sin(cos(t)),diff(y(x),x,x,x),x,t);
```

$$(\mathrm{D}^{(3)})(y)(\sin(\cos(t)))$$

```
# Преобразование 3-го типа
> Dchangevar(x(t)=L*y(phi),diff(x(t),t$3) = tan(t),t,phi);
```

$$\frac{\partial^2}{\partial\phi^3} L\mathrm{y}(\phi) = \tan(\phi)$$

```
# Дополнительные примеры
> Dchangevar({t=T*phi,x(t)=L*y(phi)},diff(x(t),
t$3)=tan(t),t,phi);
```

$$\frac{L\left(\frac{\partial^3}{\partial\phi^3}\mathrm{y}(\phi)\right)}{T^3} = \tan(T\phi)$$

```
> de := diff(y(x),x$2) = y(x)*diff(y(x),x)/x;
```

$$de := \frac{\partial^2}{\partial x^2} y(x) = \frac{y(x)\left(\frac{\partial}{\partial x} y(x)\right)}{x}$$

```
> Dchangevar({x=exp(t),y(x)=Y(t)},de,x,t);
```

$$\frac{-\frac{\frac{\partial}{\partial t} Y(t)}{e^t} + \frac{\frac{\partial^2}{\partial t^2} Y(t)}{e^t}}{e^t} = \frac{Y(t)\left(\frac{\partial}{\partial t} Y(t)\right)}{(e^t)^2}$$

Следует отметить, что подстановки являются мощным средством решения дифференциальных уравнений. Нередки случаи, когда дифференциальное уравнение не решается без их применения.

Функция нормализации ОДУ DEnormal синтаксически записывается в виде

DEnormal(des,ivar,dvar),

где `des` – система дифференциальных уравнений, `ivar` – независимая переменная и `dvar` – зависимая переменная. Применение этой функции поясняют следующие примеры:

```
> DE := x^3*y(x)+x^2*(x-1)*D(y)(x)+50*x^3*(D@@2)(y)(x)=x*sin(x);
```

$$DE := x^3 y(x) + x^2(x-1)D(y)(x) + 50x^3(D^{(2)})(y)(x) = x\sin(x)$$

```
> DE2 := convertAlg(DE,y(x));
```

$$DE2 := [[x^3, x^3 - x^2, 50x^3], x\sin(x)]$$

```
> DEnormal(DE,x,y(x));
```

$$x\, y(x) + (x-1)\left(\frac{\partial}{\partial x} y(x)\right) + 50x\left(\frac{\partial^2}{\partial x^2} y(x)\right) = \frac{\sin(x)}{x}$$

```
> DEnormal(DE2,x);
```

$$\left[[x, x-1, 50], \frac{\sin(x)}{x}\right]$$

Функция `convertAlg(des,dvar)` возвращает список коэффициентов формы системы дифференциальных уравнений `des` с зависимыми переменными `dvar`. Это поясняют следующие примеры:

```
> A := diff(y(x),x)*sin(x)-diff(y(x),x)-tan(x)*y(x)=5;
```

$$A := \left(\frac{\partial}{\partial x} y(x)\right)\sin(x) - \left(\frac{\partial}{\partial x} y(x)\right) - \tan(x) y(x) = 5$$

```
> convertAlg(A,y(x));
```

$$[[-\tan(x), \sin(x) - 1], 5]$$

```
> B := (D@@2)(y)(x)*cos(x) + (D@@2)(y)(x)*5*x^2;
```

$$B := (D^{(2)})(y)(x)\cos(x) + 5(D^{(2)})(y)(x)x^2$$

```
> convertAlg(B,y(x));
```

$$[[0, 0, \cos(x) + 5x^2], 0]$$

Для изменения переменных в системах дифференциальных уравнений используется функция convertsys:

```
convertsys(deqns, inits, vars, ivar, yvec, ypvec)
```

Здесь deqns – одно дифференциальное уравнение или список (множество), представляющие систему дифференциальных уравнений первого порядка, inits – множество или список начальных условий, vars – зависимые переменные, ivar – независимые переменные, yvec – вектор решений и ypvec – вектор производных.

Функция

```
indicialeq(des,ivar,alpha,dvar)
```

обеспечивает полиномиальное представление для линейного однородного дифференциального уравнения второго порядка des. Параметр alpha намечает точку сингулярности.

```
> Y := (2*x^2+5*x^3)*diff(y(x),x,x)+(5*x-
x^2)*diff(y(x),x)+(1+x)*y(x)=0:
> Y := convertAlg( Y, y(x) );
```

$$Y := [[1 + x, 5x - x^2, 2x^2 + 5x^3], 0]$$

```
> indicialeq( Y, x, -2/5, y(x) );
```

$$x^2 - \frac{37}{10}x = 0$$

```
> indicialeq( Y, x, 0, y(x) );
```

$$x^2 + \frac{3}{2}x + \frac{1}{2} = 0$$

```
> indicialeq( Y, x, 1, y(x) );
```

$$x^2 - x = 0$$

Функция

```
reduceOrder(des,dvar,partsol, solutionForm)
```

обеспечивает понижение порядка дифференциального уравнения des (или системы уравнений, представленных списком или множеством) при зависимых переменных dvar, частном решении partsol (или списке частных решений) и флаге solutionForm, показывающем, что решение происходит явным методом (explicitly).

Для демонстрации действия этой функции воспользуемся примером из ее справочной страницы:

```
> de := diff(y(x),x$3) - 6*diff(y(x),x$2) + 11*diff(y(x),x) -
6*y(x);
```

$$de := \left(\frac{\partial^3}{\partial x^3}\mathrm{y}(x)\right) - 6\left(\frac{\partial^2}{\partial x^2}\mathrm{y}(x)\right) + 11\left(\frac{\partial}{\partial x}\mathrm{y}(x)\right) - 6\mathrm{y}(x)$$

```
> sol := exp(x);
```

$$sol := \mathbf{e}^x$$

```
> reduceOrder( de, y(x), sol);
```

$$\left(\frac{\partial^2}{\partial x^2}\mathrm{y}(x)\right)-3\left(\frac{\partial}{\partial x}\mathrm{y}(x)\right)+2\mathrm{y}(x)$$

```
> reduceOrder( de, y(x), sol, basis);
```

$$\left[\mathbf{e}^{x}, \mathbf{e}^{(2x)}, \frac{1}{2}\mathbf{e}^{(3x)}\right]$$

Функция

regularsp(des,ivar,dvar)

вычисляет регулярные особые (сингулярные) точки для дифференциального уравнения второго порядка или системы дифференциальных уравнений `des`. Следующий пример поясняет применение данной функции:

```
> coefs := [21*(x^2 – x + 1),0,100*x^2*(x-1)^2]:
> regularsp(coefs, x);
```

$$[0, 1]$$

Еще две функции пакета `DEtools`

translate(des,ivar,pt,dvar) untranslate(des,ivar,pt,dvar)

выполняют особую операцию трансляции дифференциального уравнения (или списка дифференциальных уравнений) из центрированного относительно 0 в центрированное относительно 1 и наоборот. С деталями этого специфического процесса заинтересованный читатель может познакомиться в справочной базе данных.

И еще одна полезная функция пакета

varparam(sols,v,ivar)

находит общее решение дифференциального уравнения (или системы уравнений) `sols` методом вариации параметров. Параметр `v` задает правую часть уравнения; если он равен `0`, ищется только частичное решение:

```
> varparam( [u1(x), u2(x)], g(x), x);
```

$$\_C_1\mathrm{u1}(x)+\_C_2\mathrm{u2}(x)+\int-\frac{\mathrm{u2}(x)g(x)}{\mathrm{u1}(x)\left(\frac{\partial}{\partial x}\mathrm{u2}(x)\right)-\mathrm{u2}(x)\left(\frac{\partial}{\partial x}\mathrm{u1}(x)\right)}dx\,\mathrm{u1}(x)$$

$$+\int\frac{\mathrm{u1}(x)g(x)}{\mathrm{u1}(x)\left(\frac{\partial}{\partial x}\mathrm{u2}(x)\right)-\mathrm{u2}(x)\left(\frac{\partial}{\partial x}\mathrm{u1}(x)\right)}dx\,\mathrm{u2}(x)$$

Более подробную информацию об этих функциях читатель найдет в их справочных страницах, а также в информационном документе `DEtools.mws`, содержащем систематизированное описание пакета `DEtools` с многочисленными примерами его применения.

9.4.4. Дифференциальные операторы и их применение

Средствами пакета DEtools предусмотрена работа с *дифференциальными операторами* DF, которые дают компактное представление производных, например:

```
> restart; with(DEtools):
> df := x^2*DF^2 - x*DF + (x^2 - 1);
```

$$df := x^2 DF^2 - xDF + x^2 - 1$$

Данное выражение представляет собой дифференциальное уравнение второго порядка, записанное через дифференциальные операторы. С помощью функции diffop2de это уравнение можно преобразовать в обычное дифференциальное уравнение:

```
> diffop2de(df,y(x),[DF,x]);
```

$$(x^2-1)\mathrm{y}(x)-x\left(\frac{d}{dx}\mathrm{y}(x)\right)+x^2\left(\frac{d^2}{dx^2}\mathrm{y}(x)\right)$$

Теперь это уравнение можно решить с помощью функции dsolve:

```
> dsolve(%,y(x));
```

$$\mathrm{y}(x)=\_C1x\mathrm{BesselJ}(\sqrt{2},x)+\_C2x\mathrm{BesselY}(\sqrt{2},x)$$

Уравнения с дифференциальными операторами имеет вид степенного многочлена. Поэтому с ним можно выполнять множество операций, характерных для полиномов, например факторизацию, комплектование по степеням и др. В практике инженерных и научных расчетов дифференциальные операторы применяются довольно редко. Множество примеров с ними дано в файле примеров diffop.mws.

9.5. Графическая визуализация решений дифференциальных уравнений

9.5.1. Применение функции odeplot пакета plots

Для обычного графического представления результатов решения дифференциальных уравнений может использоваться функция odeplot из описанного выше пакета plots. Эта функция используется в следующем виде:

```
odeplot(s,vars,r,o),
```

где s – запись (в выходной форме) дифференциального уравнения или системы дифференциальных уравнений, решаемых численно функцией dsolve, vars –

переменные, `r` – параметр, задающий пределы решения (например, `a..b`), и `o` – необязательные дополнительные опции.

На рис. 9.11 представлен пример решения одиночного дифференциального уравнения с выводом решения $y(x)$ с помощью функции `odeplot`.

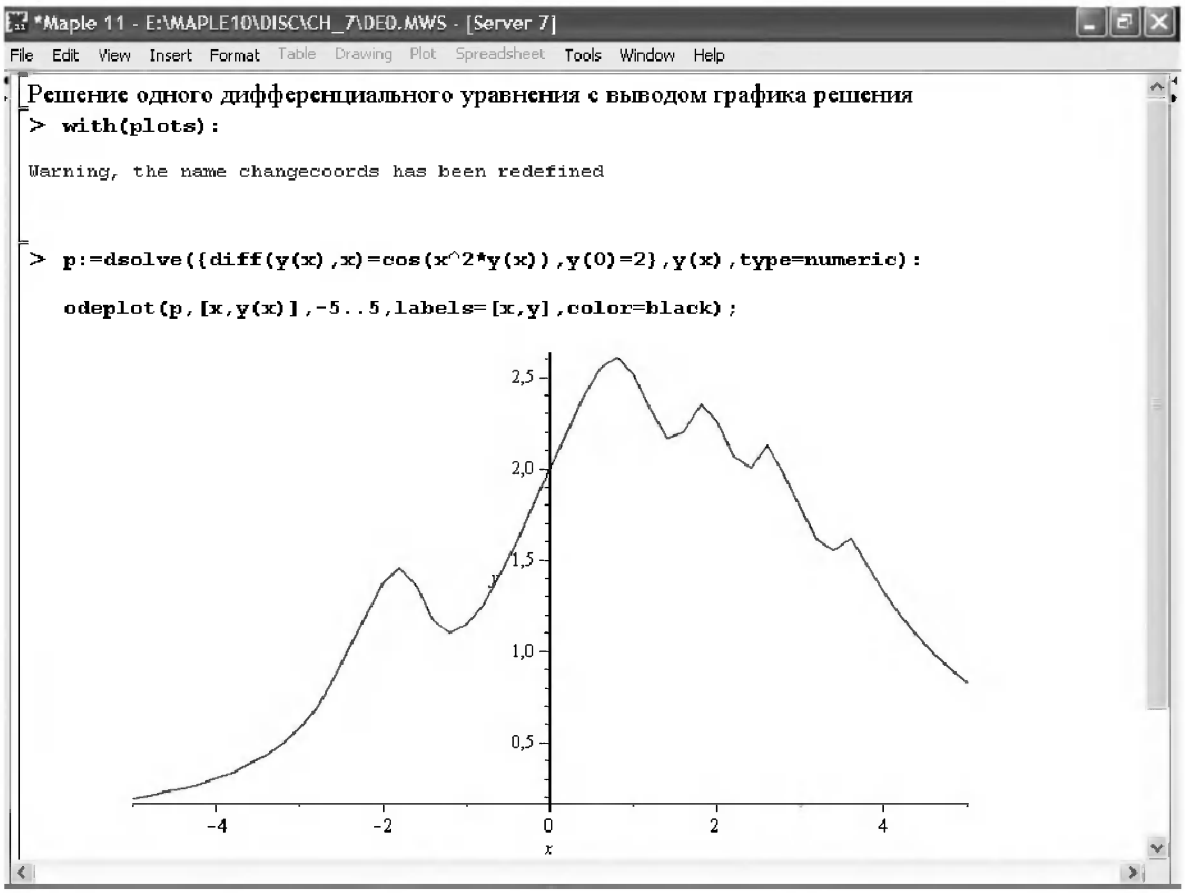


Рис. 9.11. Пример решения одиночного дифференциального уравнения (Maple 11)

В этом примере решается дифференциальное уравнение

$y'(x) = \cos(x^2 y(x))$

при $y(0) = 2$ и x, меняющемся от -5 до 5. Левая часть уравнения записана с помощью функции вычисления производной `diff`. Результатом построения является график решения $y(x)$.

В другом примере (рис. 9.12) представлено решение системы из двух нелинейных дифференциальных уравнений. Здесь с помощью функции `odeplot` строятся графики двух функций – $y(x)$ и $z(x)$.

В этом примере решается система:

$y'(x) = z(x)$,
$z'(x) = 3\sin(y(x))$

при начальных условиях $y(0) = 0$, $z(0) = 1$ и x, меняющемся от -4 до 4 при числе точек решения, равном 100.

Иногда решение системы из двух дифференциальных уравнений (или одного дифференциального уравнения второго порядка) представляется в виде фазового

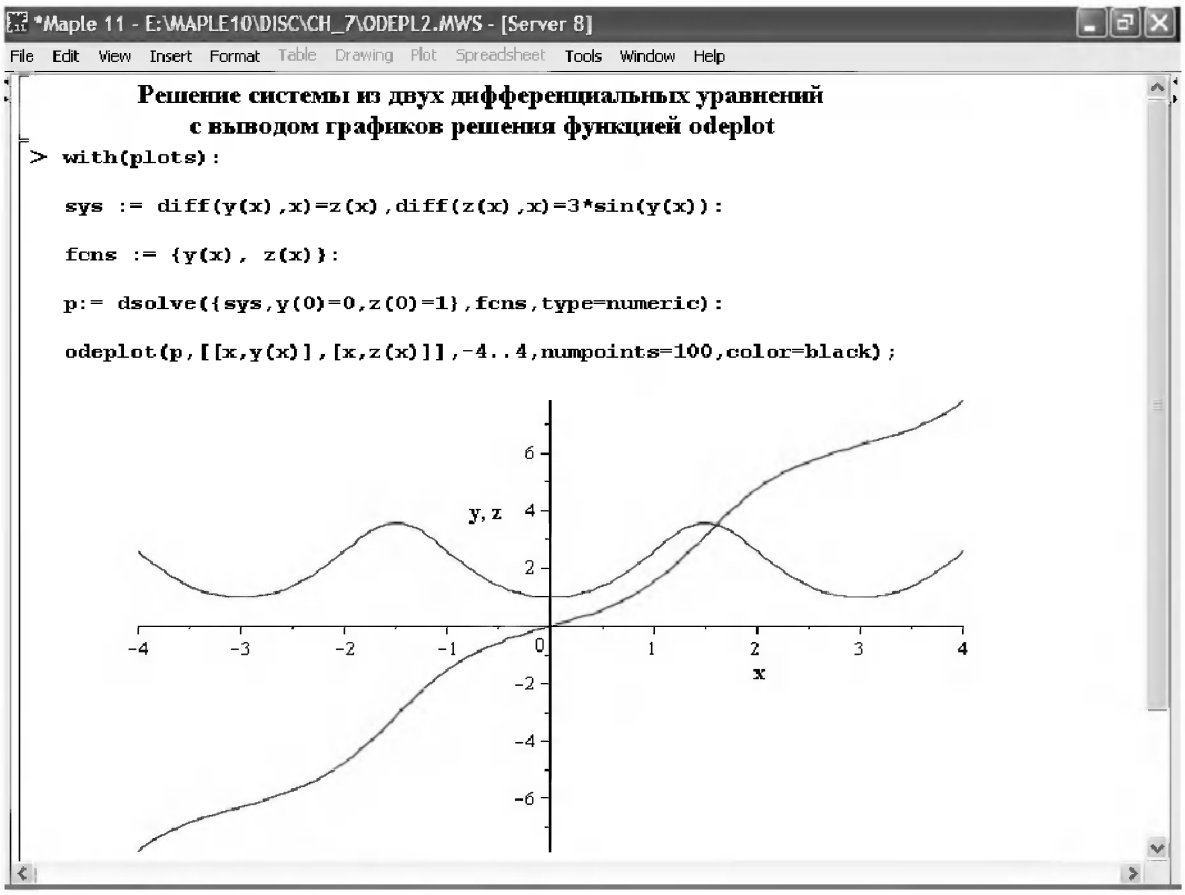


Рис. 9.12. Пример решения системы из двух дифференциальных уравнений (Maple 11)

портрета – при этом по осям графика откладываются значения $y(x)$ и $z(x)$ при изменении x в определенных пределах. Рисунок 9.13 демонстрирует построение фазового портрета для системы, представленной выше.

Обычное решение, как правило, более наглядно, чем фазовый портрет решения. Однако для специалистов (например, в теории колебаний) фазовый портрет порой дает больше информации, чем обычное решение. Он более трудоемок для построения, поэтому возможность Maple быстро строить фазовые портреты трудно переоценить.

9.5.2. Функция DEplot из пакета DEtools

Специально для решения и визуализации решений дифференциальных уравнений и систем с дифференциальными уравнениями служит инструментальный пакет `DEtools`. В него входит ряд функций для построения наиболее сложных и изысканных графиков решения дифференциальных уравнений. Основной из этих функций является функция `DEplot`.

Функция `DEplot` может записываться в нескольких формах:

```
DEplot(deqns, vars, trange, eqns)
DEplot(deqns, vars, trange, inits, eqns)
DEplot(deqns, vars, trange, yrange, xrange, eqns)
DEplot(deqns, vars, trange, inits, xrange, yrange, eqns)
```

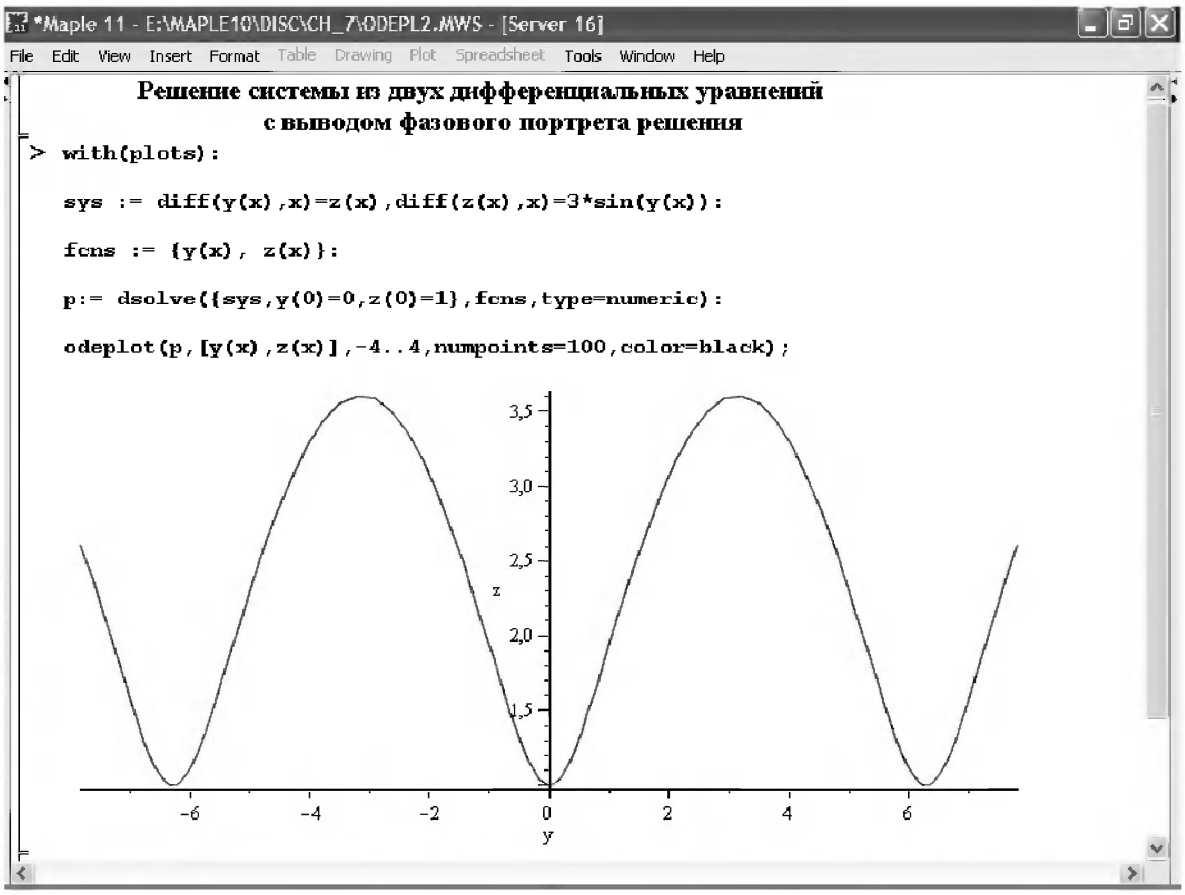


Рис. 9.13. Представление решения системы дифференциальных уравнений в виде фазового портрета

Здесь `deqns` – список или множество, содержащее систему дифференциальных уравнений первого порядка или одиночное уравнение любого порядка; `vars` – зависимая переменная или список либо множество зависимых переменных; `trange` – область изменения независимой переменной `t`; `inits` – начальные условия для решения; `yrange` – область изменения для первой зависимой переменной; `xrange` – область изменения для второй зависимой переменной; `eqns` – опция, записываемая в виде `keyword=value`. Замена имен переменных другими в данном случае недопустима.

Эта функция обеспечивает численное решение дифференциальных уравнений или их систем при одной независимой переменной `t` и строит графики решения. Для автономных систем эти графики строятся в виде векторного поля направлений, а для неавтономных систем – только в виде кривых решения. По умолчанию реализуется метод Рунге-Кутта 4-го порядка, что соответствует опции `method=classical[rk4]`.

С функцией `DEplot` могут использоваться следующие параметры:

- `arrows = type` – тип стрелки векторного поля (`'SMALL'`, `'MEDIUM'`, `'LARGE'`, `'LINE'` или `'NONE'`);
- `colour, color = arrowcolour` – цвет стрелок (задается 7 способами);
- `dirgrid=[integer,integer]` – число линий сетки (по умолчанию [20, 20]);
- `iterations = integer` – количество итераций, представленное целым числом;

- `linecolor, linecolor = line_info` – цвет линии (задается 5 способами);
- `method='rk4'` – задает метод решения (`'euler'`, `'backeuler'`, `'impeuler'` или `'rk4'`);
- `obsrange = TRUE,FALSE` – задает (при `TRUE`) прерывание вычислений, если кривая решения выходит из области обзора;
- `scene = [name,name]` – задает имена зависимых переменных, для которых строится график;
- `stepsize = h` – шаг решения, по умолчанию равный `abs((b-a))/20` и представленный вещественным значением.

На рис. 9.14 показано решение системы дифференциальных уравнений

$$x'(t) = x(t)\,(1 - y(t)),$$
$$y'(t) = 0{,}3\,y(t)\,(x(t) - 1),$$

описывающих *модель Лотки–Вольтерра* (изменение популяции в биологической среде хищник–жертва) при заданных в документе изменениях t, $x(t)$ и $y(t)$. Решение представлено в виде векторного поля, стрелки которого являются касательными к кривым решения (сами эти кривые не строятся). Обратите внимание на функциональную закраску стрелок векторного поля, делающую решение особенно наглядным (правда, лишь на экране цветного дисплея, а не на страницах книги).

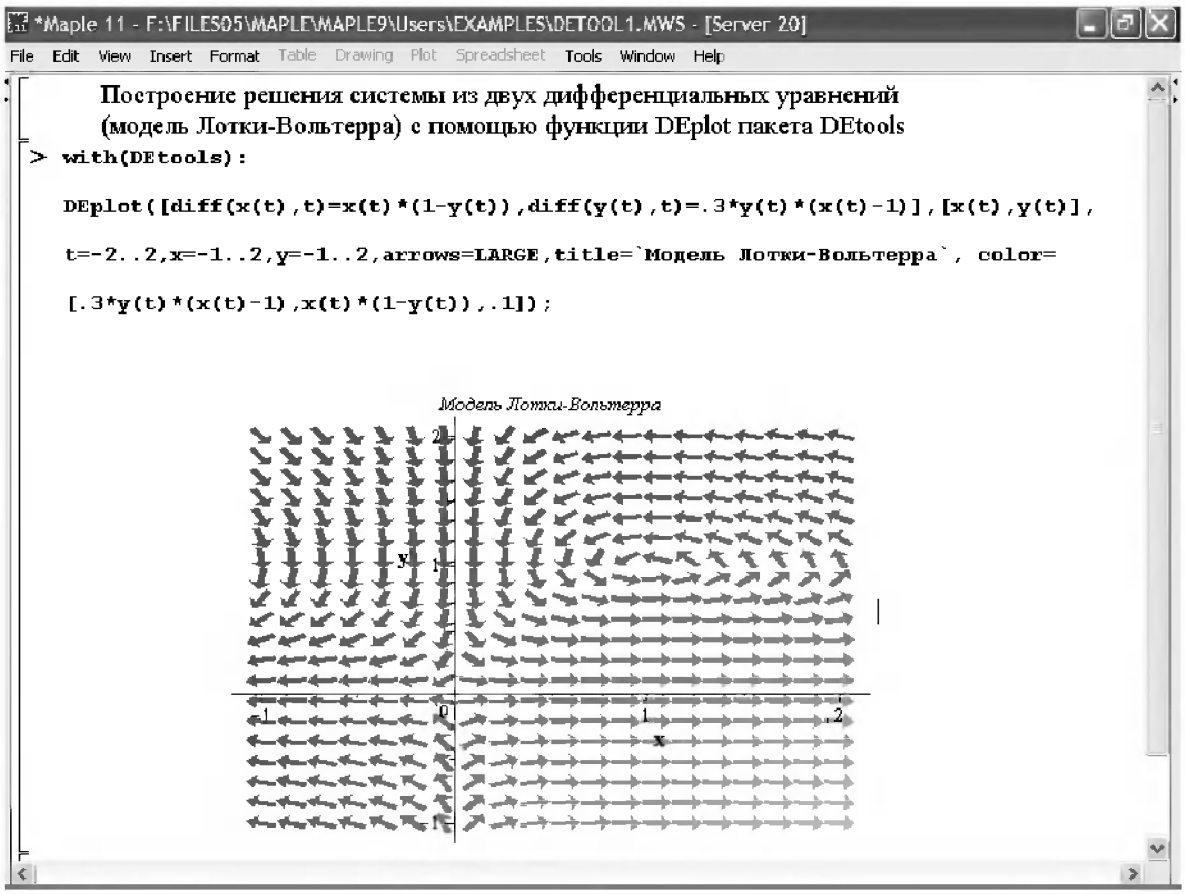


Рис. 9.14. Решение системы дифференциальных уравнений Лотки–Вольтерра с выводом в виде графика векторного поля (Maple 11)

Еще интересней вариант графиков, представленный на рис. 9.15. Здесь, помимо векторного поля несколько иного стиля, построены фазовые портреты решения с использованием функциональной закраски их линий. Фазовые портреты построены для двух наборов начальных условий: $x(0) = y(0) = 1{,}2$ и $x(0) = 1$ и $y(0) = 0{,}9$.

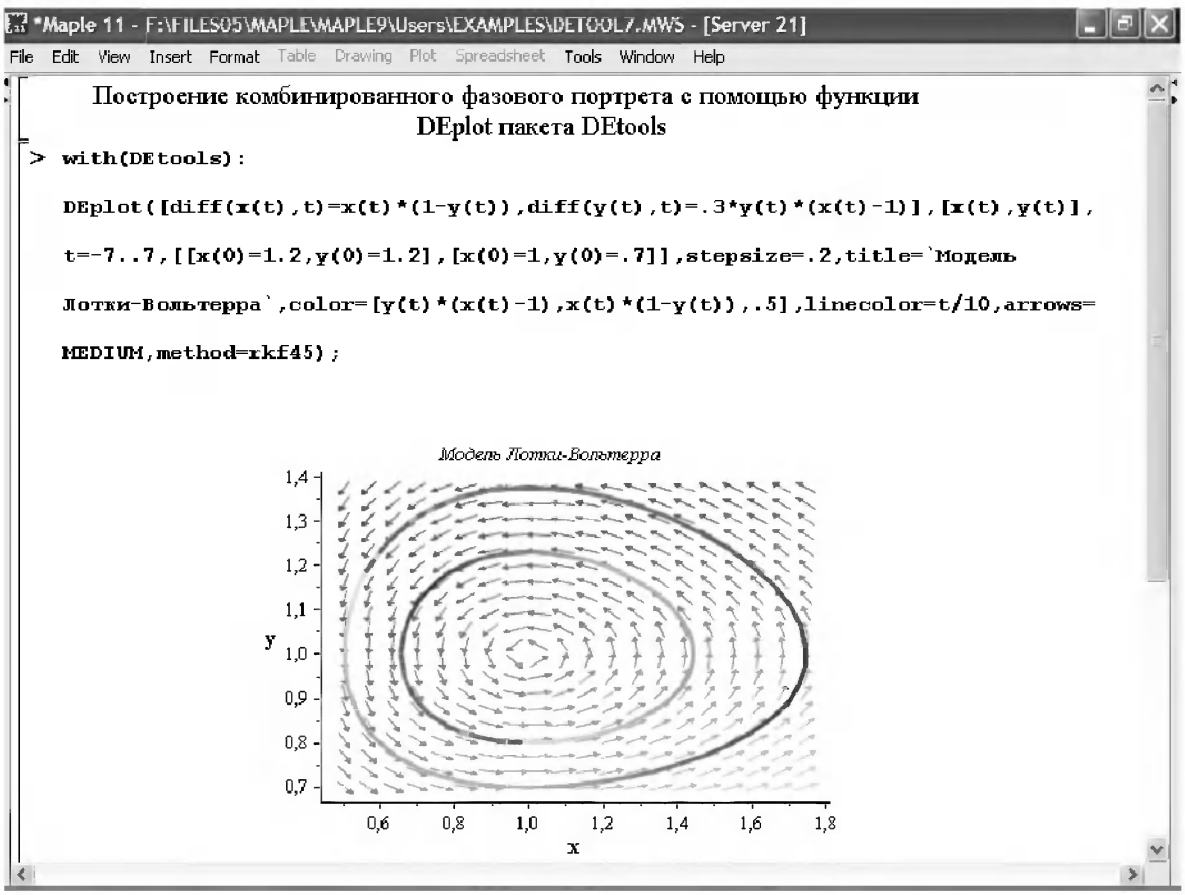


Рис. 9.15. Пример построения двух фазовых портретов на фоне векторного поля

Следует отметить, что функция `DEplot` может обращаться к другим функциям пакета `SEtools` для обеспечения специальных графических возможностей, таких как построение векторного поля или фазового портрета решения. В файле deplot.mws можно найти множество дополнительных примеров на применение функции Deplot.

9.5.3. Функция DEplot3d из пакета DEtools

В ряде случаев решение систем дифференциальных уравнений удобно представлять в виде *пространственных кривых* – например, линий равного уровня, или просто в виде кривых в пространстве. Для этого служит функция `DEplot3d`:

```
DEplot3d(deqns, vars, trange, initset, o)
DEplot3d(deqns, vars, trange, yrange, xrange, initset, o)
```

Назначение параметров этой функции аналогично указанному для функции `DEplot`.

Рисунок 9.16 поясняет применение функции `DEPlot3d` для решения системы из двух дифференциальных уравнений с выводом фазового портрета колебаний в виде параметрически заданной зависимости $x(t), y(t)$. В данном случае фазовый портрет строится на плоскости по типу построения графиков линий равной высоты (контурных графиков).

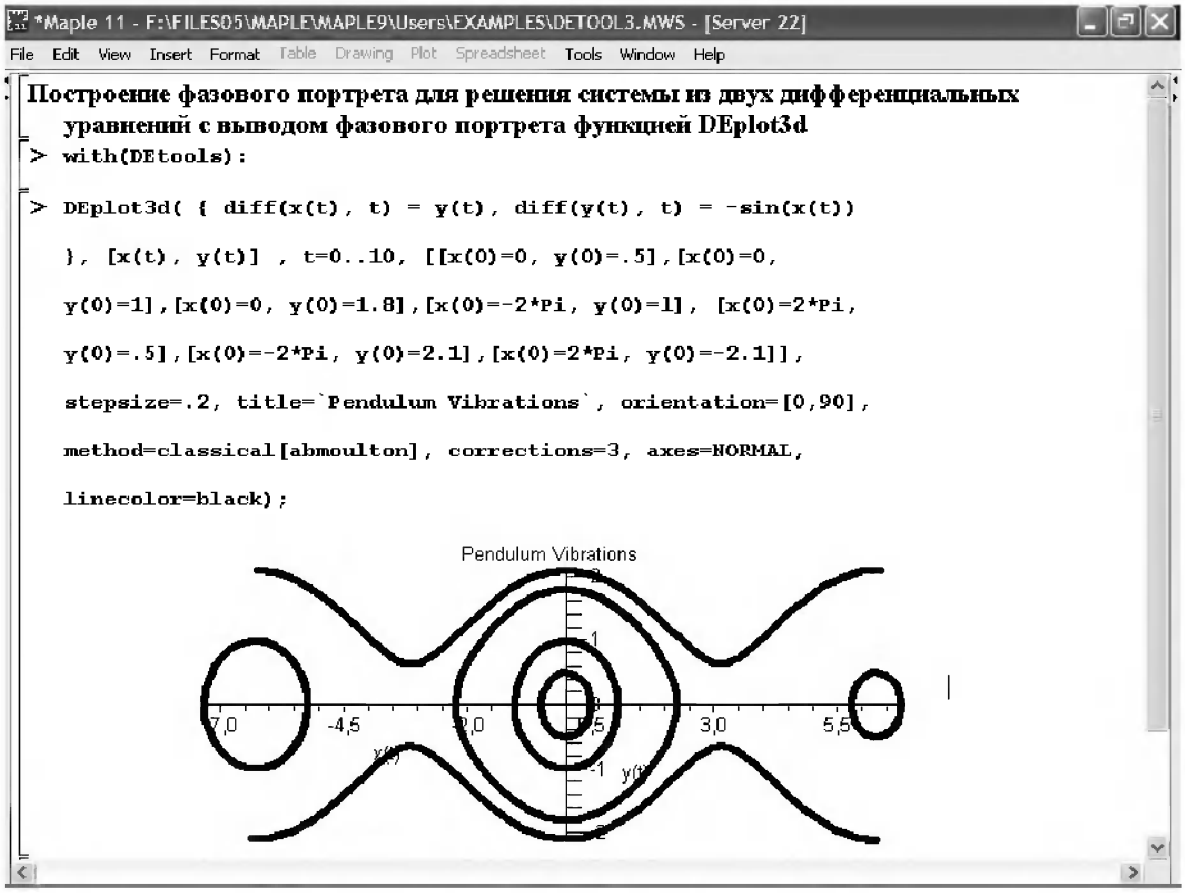


Рис. 9.16. Пример решения системы из двух дифференциальных уравнений с помощью функции DEplot3d

Другой пример (рис. 9.17) показывает решение системы из двух дифференциальных уравнений с построением объемного фазового портрета. В этом случае используется трехмерная координатная система и графические построения соответствуют параметрическим зависимостям $x(t), y(t)$ и $z(t)$. Вид фазового портрета напоминает разворачивающуюся в пространстве объемную спираль. Функциональная окраска делает график пикантным, что, увы, теряется при черно-белом воспроизведении графика.

Возможности функции `DEplot3d` позволяют решать системы, состоящие более чем из двух дифференциальных уравнений. Однако в этом случае число решений, представляемых графически, выходит за пределы возможного для трехмерной графики. При этом от пользователя зависит, какие из зависимостей опускаются при построении, а какие строятся. В файле deplot3d.mws есть ряд дополнительных примеров на применение функции Deeplot3d.

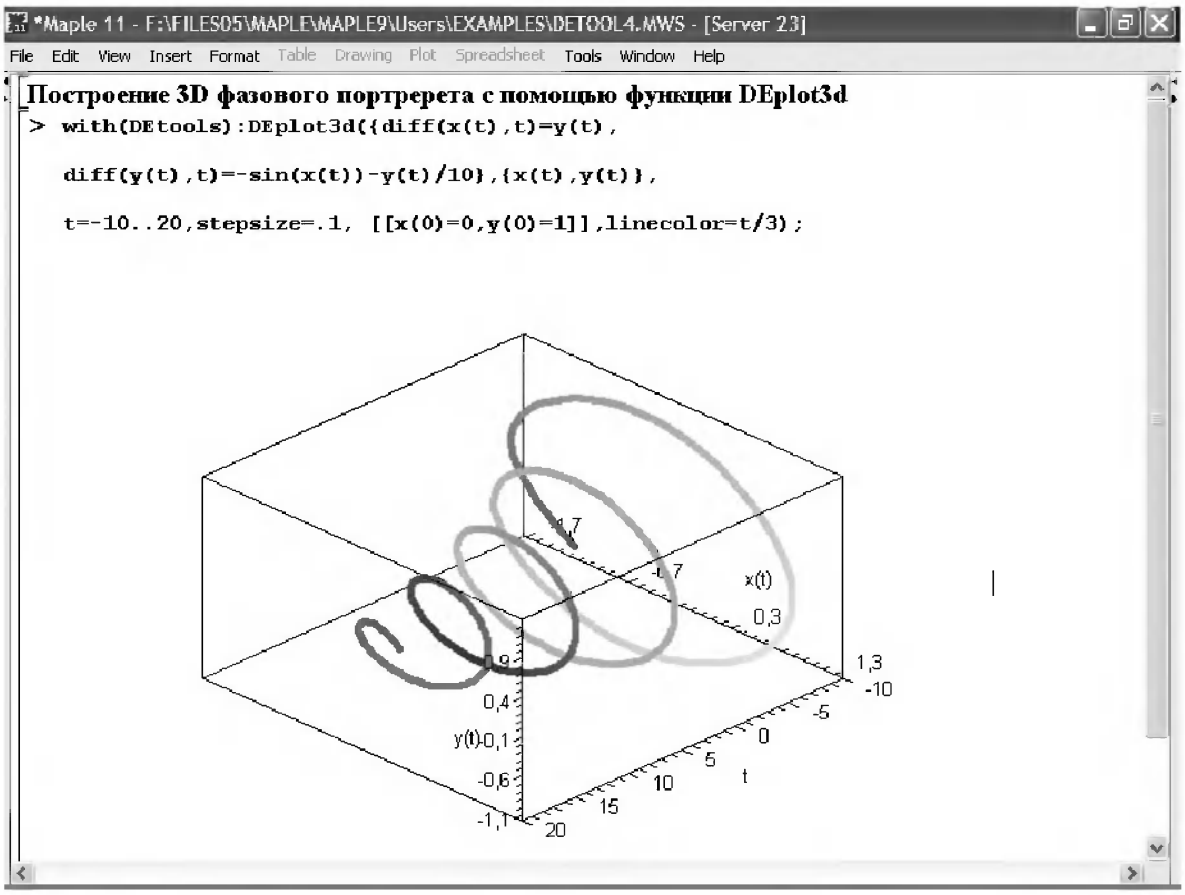


Рис. 9.17. Пример решения системы из двух дифференциальных уравнений с построением трехмерного фазового портрета

9.5.4. Графическая функция dfieldplot

Графическая функция `dfieldplot` служит для построения поля направления с помощью векторов по результатам решения дифференциальных уравнений. Фактически эта функция как бы входит в функцию `DEplot` и при необходимости вызывается последней. Но она может использоваться и самостоятельно, что демонстрирует рис. 9.18, на котором показан пример решения следующей системы дифференциальных уравнений: $x'(t) = x(t)(1 - y(t))$, $y'(t) = 0{,}3\,y(t)(x(t) - 1)$.

Обратите внимание на использование опций в этом примере – в частности, на вывод надписи на русском языке. В целом список параметров функции `phaseportrait` аналогичен таковому для функции `DEplot` (отсутствует лишь задание начальных условий).

9.5.5. Графическая функция phaseportrait

Графическая функция `phaseportrait` служит для построения фазовых портретов по результатам решения одного дифференциального уравнения или системы дифференциальных уравнений `deqns`. Она задается в следующем виде:

```
phaseportrait(deqns,vars,trange,inits,o)
```

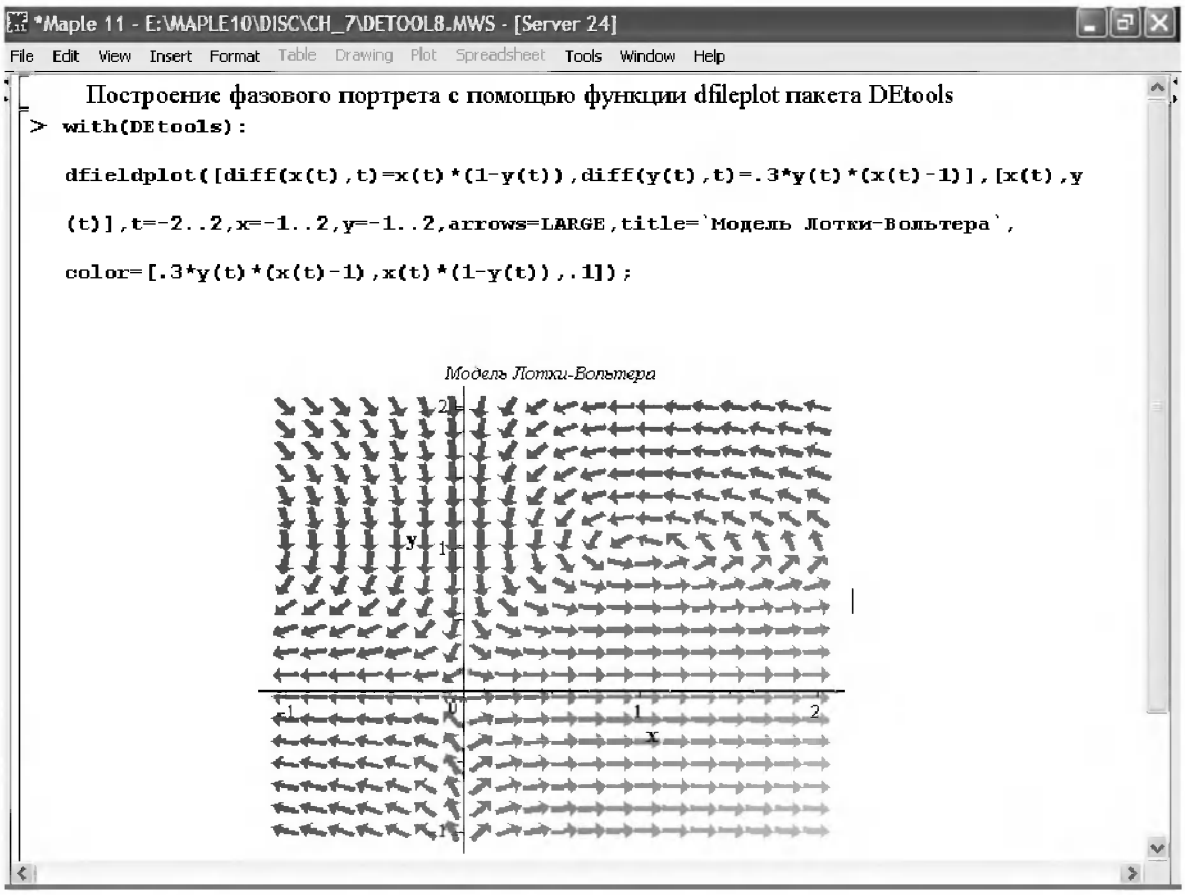


Рис. 9.18. Построение фазового портрета в виде графика векторного поля

При задании уравнений достаточно указать их правые части. На рис. 9.19 представлен пример применения функции `phaseportrait` для решения системы из трех дифференциальных уравнений первого порядка.

В этом примере система дифференциальных уравнений задана с помощью оператора дифференцирования `D`. Функциональная окраска линии фазового портрета достигается использованием параметра `linecolor`, в правой части которого задана формула для цвета.

Еще более интересный пример решения дифференциального уравнения представлен на рис. 9.20. Здесь построены фазовые портреты для асимптотических решений. Хотя пример дан для системы Maple 11, он работает и с предшествующими версиями – например, Maple 9.

В целом надо отметить, что возможности визуализации решений дифференциальных уравнений с помощью систем Maple весьма велики, и приведенные выше примеры лишь приоткрывают ларец с возможностями этих систем в решении данного класса задач.

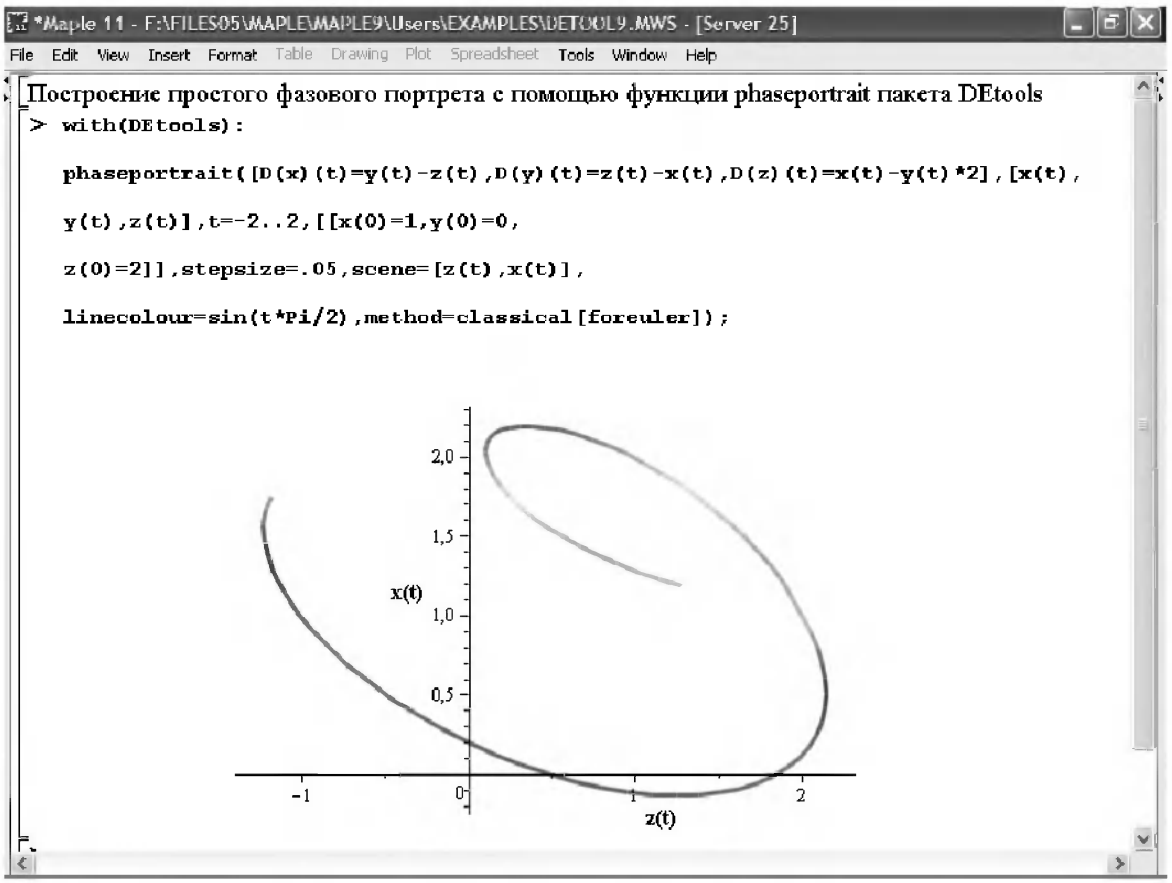


Рис. 9.19. Построение фазового портрета с помощью функции phaserportrait

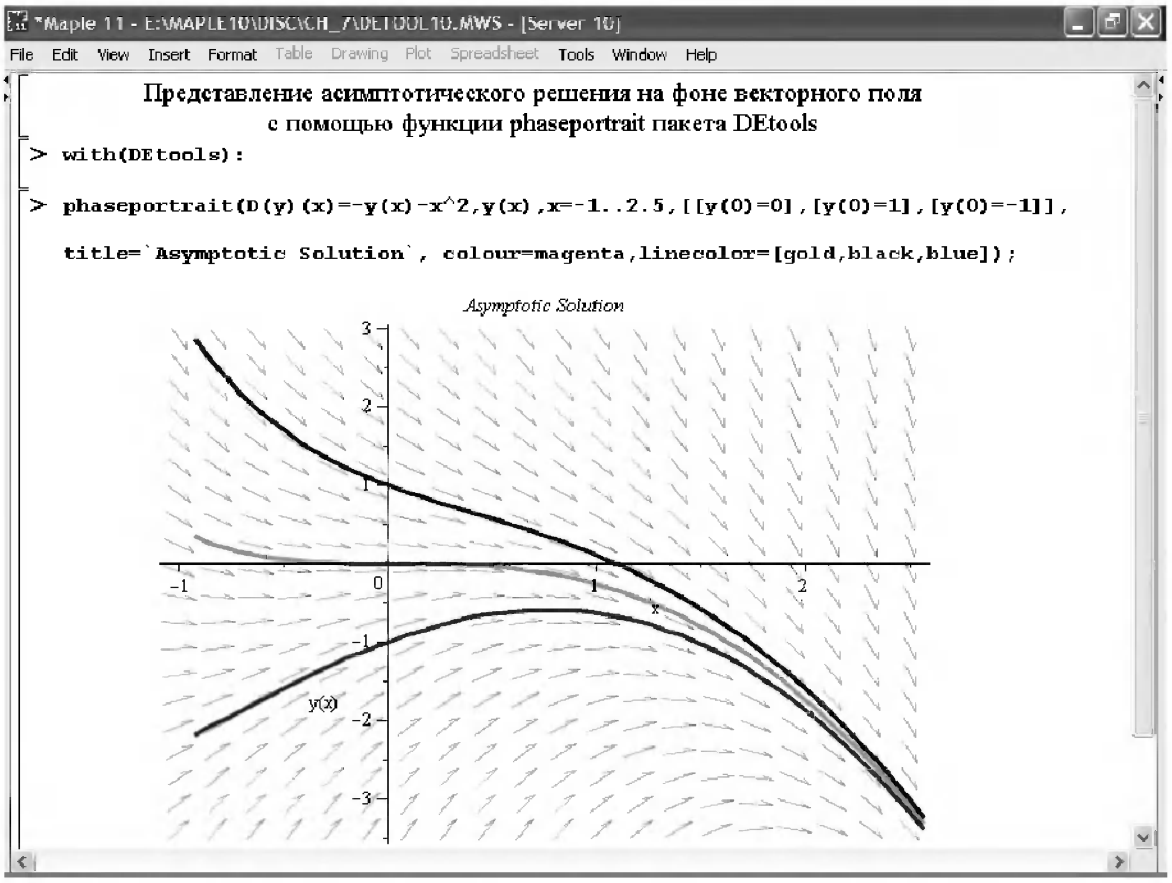


Рис. 9.20. Построение асимптотического решения на фоне графика векторного поля (Maple 11)

9.6. Углубленный анализ дифференциальных уравнений

9.6.1. Задачи углубленного анализа ДУ

Уже Maple 9.5 существенно доработана по части решения дифференциальных уравнений (ДУ) и систем с ДУ. Эта доработка прежде всего направлена на получение верных решений как можно большего числа ДУ разных классов и систем с ДУ. В частности, расширен круг нелинейных дифференциальных уравнений, для которых система Maple способна дать аналитические решения.

Весь арсенал средств решения ДУ и методика их применения вполне заслуживают отражения в отдельной большой книге. Мы ограничимся описанием только трех средств системы Maple – проверки ДУ на автономность, углубленным анализом решения с помощью контроля уровня выхода и получением приближенного полиномиального аналитического решения.

9.6.2. Проверка ДУ на автономность

Одиночное дифференциальное уравнение или система дифференциальных уравнений называется автономной, если их правая часть явно не зависит от независимой переменной. Для автономных дифференциальных уравнений или систем при построении графиков решений функцией `DEplot` не обязательно задавать начальные условия, но нужно указывать диапазон изменения искомых переменных.

Для проверки уравнений (или систем) на автономность используется функция

`autonomous(des,vars,ivar)`,

где `des` – заданное дифференциальное уравнение или (в виде списка) система дифференциальных уравнений, `vars` – зависимые переменные и `ivar` – независимая переменная. Если система автономна, то эта функция возвращает `true`, в противном случае `false`.

Примеры:

```
> dif1:=diff(x(t),t)=x(t)*(1-y(t));
dif2:=diff(y(t),t)=.3*y(t)*(x(t)-1);
```

$$dif1 := \frac{\partial}{\partial t}\mathrm{x}(t) = \mathrm{x}(t)(1-\mathrm{y}(t))$$

$$dif2 := \frac{\partial}{\partial t}\mathrm{y}(t) = .3\mathrm{y}(t)(\mathrm{x}(t)-1)$$

```
> autonomous({dif1,dif2},[x(t),y(t)],t);
```

$$true$$

```
> autonomous(diff(x(t),t)=sin(t),x,t);
```

$$false$$

В первом случае система дифференциальных уравнений (модель Лотки-Вольтерра) автономна, а во втором случае дифференциальное уравнение неавтономно.

9.6.3. Контроль уровня вывода решения ДУ

Для углубленного анализа аналитического решения ДУ (или системы ДУ) можно использовать специальную возможность управления уровнем вывода решения с помощью системной переменной `infilevel(dsolve):=level`. Значение `level=all` дает обычный вывод решения без комментариев, уровень 1 зарезервирован для информации, которую может сообщить пользователь, уровень 2 или 3 дает более детальный вывод (включая сообщения об использованном алгоритме и технике решения), и, наконец, уровни 4 и 5 дают наиболее детальную информацию (если таковая есть в дополнение к той информации, которую дает уровень 2 или 3).

Приведем пример аналитического решения ДУ третьего порядка с контролем уровня вывода решения:

```
> myDE := x^2*diff(y(x),x,x,x)-
2*(n+1)*x*diff(y(x),x,x)+(a*x^2+6*n)*diff(y(x),x)-
2*a*x*y(x)=0;
```

$$myDE := x^2\left(\frac{d^3}{dx^3}\mathrm{y}(x)\right)-2(n+1)x\left(\frac{d^2}{dx^2}\mathrm{y}(x)\right)+(ax^2+6n)\left(\frac{d}{dx}\mathrm{y}(x)\right)$$
$$-2ax\mathrm{y}(x)=0$$

```
> infolevel[dsolve] := all: dsolve(myDE);
```

$$\mathrm{y}(x)=\_C1x^{(n+1/2)}\mathrm{BesselJ}\left(-n-\frac{1}{2},\sqrt{a}x\right)$$
$$+\_C2x^{(n+1/2)}\mathrm{BesselY}\left(-n-\frac{1}{2},\sqrt{a}x\right)+\_C3(4n-2+ax^2)$$

```
> infolevel[dsolve] := 1:dsolve(myDE);
<- No Liouvillian solutions exists
```

$$\mathrm{y}(x)=\_C1x^{(n+1/2)}\mathrm{BesselJ}\left(-n-\frac{1}{2},\sqrt{a}x\right)$$
$$+\_C2x^{(n+1/2)}\mathrm{BesselY}\left(-n-\frac{1}{2},\sqrt{a}x\right)+\_C3(4n-2+ax^2)$$

```
> infolevel[dsolve] := 3:dsolve(myDE);
Methods for third order ODEs:
- Trying classification methods --
trying a quadrature
checking if the LODE has constant coefficients
checking if the LODE is of Euler type
trying high order exact linear fully integrable
trying to convert to a linear ODE with constant coefficients
```

```
Equation is the LCLM of -2*x/(2*(2*n-1)/a+x^2)*y(x)+diff(y(x),x),
a*y(x)-2*n/x*diff(y(x),x)+diff(diff(y(x),x),x)
checking if the LODE is of Euler type
-> Attemtping a differential factorization
   trying exponential solutions
   checking if the LODE is of Euler type
   1, exponential solutions found
   exponential solutions successful
<- differential factorization successful
   -> Tackling the linear ODE «as given»:
      trying a quadrature
      checking if the LODE has constant coefficients
      checking if the LODE is of Euler type
      trying a symmetry of the form [xi=0, eta=F(x)]
      checking if the LODE is missing "y"
      -> Trying a Liouvillian solution using Kovacic's algorithm
      <- No Liouvillian solutions exists
      -> Trying a solution in terms of special functions:
         -> Bessel
         <- Bessel successful
      <- special function solution successful
   <- successful solving of the linear ODE «as given»
<- solving the LCLM ode successful
```

$$y(x) = \_C1x^{(n+1/2)}\mathrm{BesselJ}\left(-n-\frac{1}{2}, \sqrt{a}x\right) + \_C2x^{(n+1/2)}\mathrm{BesselY}\left(-n-\frac{1}{2}, \sqrt{a}x\right) + \_C3(4n-2+ax^2)$$

В данном случае повышение уровня вывода до 4 или 5 бесполезно, поскольку вся информация о решении сообщается уже при уровне 2 (или 3).

9.6.4. Приближенное полиномиальное решение ДУ

Во многих случаях аналитические решения даже простых ДУ оказываются весьма сложными, например содержат специальные математические функции. При этом нередко полезна подмена такого решения другим, тоже аналитическим, но приближенным решением. Наиболее распространенным приближенным решением в этом случае может быть полиномиальное решение, то есть замена реального решения полиномом той или иной степени. При этом порядок полинома задается значением системной переменной `Order`, а для получения такого решения функция `dsolve` должна иметь параметр `series`.

На рис. 9.21 представлено решение ДУ третьего порядка различными методами: точное аналитическое и приближенное в виде полинома с максимальным заданным порядком 10 и 60. График дает сравнение этих решений для зависимости y(t).

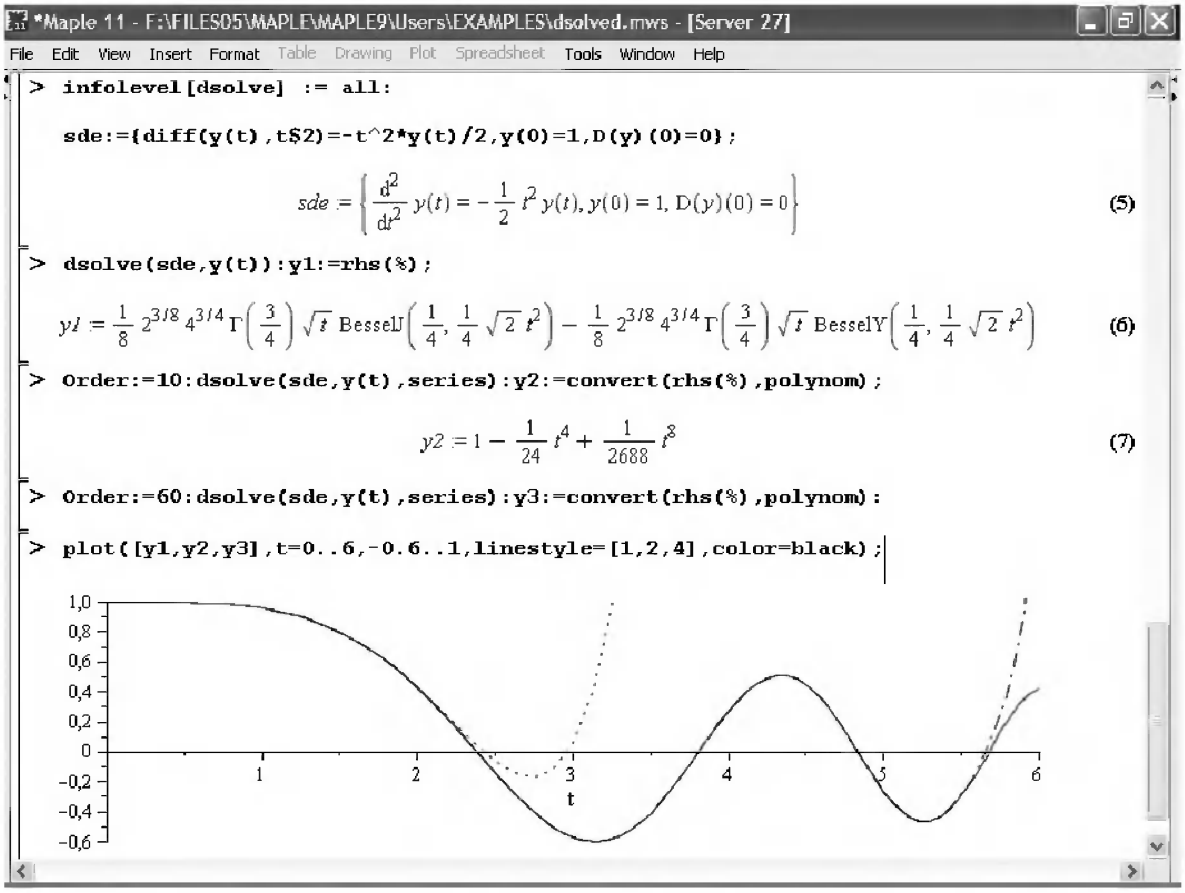


Рис. 9.21. Примеры решения ДУ третьего порядка (Maple 9)

Дадим небольшой комментарий. Нетрудно заметить, что точное аналитическое решение весьма сложно и содержит специальные функции Бесселя и гамма-функции. При порядке полинома 8 (он несколько меньше заданного максимального) решение практически совпадает с точным до значений $t < 2$, а при максимальном заданном порядке 60 область совпадения расширяется до значений $t < 5{,}5$. Затем приближенное решение резко отходит от точного.

Этот пример, с одной стороны, иллюстрирует хорошо известный факт – быстрое нарастание погрешности полиномиального приближения за пределами области хорошего совпадения решений. С другой – он показывает, что степень полинома более 60 (и даже выше) вовсе не так уж бесполезна, как это утверждается во многих статьях и книгах по полиномиальному приближению.

9.7. Решение дифференциальных уравнений с частными производными

9.7.1. Функция pdsolve

В Maple 9.5 имеется функция `pdsolve` для решения дифференциальных уравнений с частными производными. Она может использоваться в следующих формах записи:

```
pdsolve(PDE, f, HINT, INTEGRATE, build)
pdsolve(PDE_system, funcs, HINT, other_options)
pdsolve(PDE_system, conds, numeric, other_options)
pdsolve(PDE_system, conds, type=numeric, other_options)
```

Эта функция введена вместо устаревшей функции `pdesolve`. В функции `pdsolve` используются следующие параметры:

- `PDE` – одиночное дифференциальное уравнение с частными производными;
- `PDE_system` – система дифференциальных уравнений с частными производными;
- `conds` – начальные или граничные условия;
- `f` – неопределенная функция или имя;
- `funcs` – (опция) множество или список с неопределенными функциями или именами;
- `HINT` – (опция) равенство в форме HINT = argument, где аргумент может быть символом '+', '*' любым алгебраическим выражением или строкой 'strip';
- `INTEGRATE` – (опция) задает автоматическое интегрирование для множества ODEs (если PDE решается при разделении переменных);
- `build` – опция, задающая попытку построения явного выражения для неопределенной функции, независимо от общности найденного решения;
- `numeric` – ключевое слово, задающее решение в численном виде;
- `other_options` – другие опции.

9.7.2. Инструментальный пакет расширения PDEtool

Для решения дифференциальных уравнений с частными производными и его визуализации в Maple 9.5 служит специальный инструментальный пакет PDEtool:

```
> with(PDEtools);
```

[*RDEplot, build, casesplit, charstrip, dchange, dcorffs, declare, difforder, dpolyform, dsubs, mapde, separability, splitstrip, splitsys, undeclare*]

Ввиду небольшого числа функций этого пакета приведем их определения:

- `build(sol)` – конструирует улучшенную форму решения, полученного функцией pdsolve;
- `casesplit(sys, o1, 02,...)` – преобразует форму дифференциального уравнения;
- `charstrip(PDE,f)` – находит характеристическую последовательность, дающую дифференциальное уравнение первого порядка;
- `dchange(tr,expr,o1,02,...)` – выполняет замену переменных в математических выражениях или функциях;
- `dcoeff(expr,y(x))` – возвращает коэффициенты полиномиала дифференциального уравнения;
- `declare(expr)` и др. – задает функцию для компактного ее отображения;

- `differorder(a,x)` – возвращает порядок дифференциала в алгебраическом выражении a;
- `dpolyform(sys,no_Fn,opts)` – возвращает полиномиальную форму для заданной системы sys неполиномиальных дифференциальных уравнений;
- `dsubs(deriv1=a,…,expr)` – выполняет дифференциальные подстановки в выражение expr;
- `mapde(PDE,into,f)` – создает карту PDE в различных форматах into с опциональным заданием имени неизвестной функции f;
- `separability(PDE, F(x,y,...), '*')` – определяет условия разделения для сумм или произведений PDE;
- `splitstrip(PDE,f)` – разделяет характеристическую последовательность на несоединенные поднаборы;
- `splitsys(sys,funcs)` – разделяет наборы уравнений (алгебраические и дифференциальные) на несоединенные поднаборы;
- `undeclare(f(x),…)` и др. – отменяет задание функции для компактного ее отображения.

9.7.3. Примеры решения дифференциальных уравнений с частными производными

Примеры решения дифференциальных уравнений и систем с частными производными представлены ниже:

```
> with(PDEtools):
> PDE := x*diff(f(x,y),y)-diff(f(x,y),x)=f(x,y)^2*g(x)/h(y);
```

$$PDE := x\left(\frac{\partial}{\partial y}\mathrm{f}(x,y)\right)-\left(\frac{\partial}{\partial x}\mathrm{f}(x,y)\right)=\frac{\mathrm{f}(x,y)^2 g(x)}{\mathrm{h}(y)}$$

```
> ans := pdsolve(PDE);
```

$$ans := \mathrm{f}(x,y)=\frac{1}{\int^{x}\frac{g(\_a)}{\mathrm{h}\left(-\frac{a^2}{2}+y+\frac{x^2}{2}\right)}d\_a+\_\mathrm{F1}\left(y+\frac{x^2}{2}\right)}$$

```
> PDE := S(x,y)*diff(S(x,y),y,x) +
  diff(S(x,y),x)*diff(S(x,y),y) = 1;
```

$$PDE := \mathrm{S}(x,y)\left(\frac{\partial^2}{\partial y\,\partial x}\mathrm{S}(x,y)\right)+\left(\frac{\partial}{\partial x}\mathrm{S}(x,y)\right)\left(\frac{\partial}{\partial y}\mathrm{S}(x,y)\right)=1$$

```
> struc := pdsolve(PDE,HINT=f(x)*g(y));
```

$$struc := (\mathrm{S}(x,y)=\mathrm{f}(x)\mathrm{g}(y))\&\mathrm{where}$$

$$\left[\left\{\frac{d}{dx}\mathrm{f}(x)=\frac{\_c_1}{f(x)},\frac{d}{dy}\mathrm{g}(y)=\frac{1}{2}\frac{1}{\mathrm{g}(y)\_c_1}\right\}\right]$$

```
> build(struc);
```

$$\mathrm{S}(x,y)=-\frac{\sqrt{2\_c_1}x+\_C1\sqrt{\_c_1}y+\_C2\_c_1}{\_c_1}$$

```
> pdsolve(PDE,HINT=P(x,y)^(1/2));
```

$$\mathrm{S}(x,y)=\sqrt{\_\mathrm{F2}(x)+\_\mathrm{F1}(y)+2xy}$$

```
> PDE := diff(f(x,y,z),x) + diff(f(x,y,z),y)^2 = f(x,y,z)+z;
```

$$PDE:=\left(\frac{\partial}{\partial x}\mathrm{f}(x,y,z)\right)+\left(\frac{\partial}{\partial y}\mathrm{f}(x,y,z)\right)^2=\mathrm{f}(x,y,z)+z$$

```
> pdsolve(PDE,HINT=strip);
```

$$\left(\left(\frac{\partial}{\partial x}\mathrm{f}(x,y,z)\right)+\left(\frac{\partial}{\partial y}\mathrm{f}(x,y,z)\right)^2-\mathrm{f}(x,y,z)-z=0\right)\&\,\mathrm{where}\left[\{\{\right.$$

$$\_p1(\_s)=\_C3\mathbf{e}^{-s},\ \_p2(\_s)=\_C4\mathbf{e}^{-s},\ \mathrm{z}(\_s)=\_C5,\ \mathrm{x}(\_s)=\_s+\_C6,$$
$$\mathrm{y}(\_s)=2\_C4\mathbf{e}^{-s}+\_C2,\ \mathrm{f}(\_s)=\_C3\mathbf{e}^{-s}-\_C5+\mathbf{e}^{(2-s)}\_C1\}\},$$

$$\&\mathrm{and}\left(\left\{\_p_1=\frac{\partial}{\partial x}\mathrm{f}(x,y,z),\_p_2=\frac{\partial}{\partial y}\mathrm{f}(x,y,z)\right\}\right)\Bigg]$$

```
> myPDEsystem := [-y*diff(f(x,y,z,t),x) +
  z^2*diff(f(x,y,z,t),z) + 3*t*z*diff(f(x,y,z,t),t) - 3*t^2-
  4*f(x,y,z,t)*z = 0,
  -y*diff(f(x,y,z,t),y) - z*diff(f(x,y,z,t),z) -
  t*diff(f(x,y,z,t),t) + f(x,y,z,t) = 0,
  -x*diff(f(x,y,z,t),y) - diff(f(x,y,z,t),z) = 0]:

  for _eq in myPDEsystem do
     _eq;
  od;
```

$$-y\left(\frac{\partial}{\partial x}\mathrm{f}(x,y,z,t)\right)+z^2\left(\frac{\partial}{\partial z}\mathrm{f}(x,y,z,t)\right)+3tz\left(\frac{\partial}{\partial t}\mathrm{f}(x,y,z,t)\right)-3t^2-4\mathrm{f}(x,y,z,t)z=0$$

$$-y\left(\frac{\partial}{\partial y}\mathrm{f}(x,y,z,t)\right)-z\left(\frac{\partial}{\partial z}\mathrm{f}(x,y,z,t)\right)-t\left(\frac{\partial}{\partial t}\mathrm{f}(x,y,z,t)\right)+\mathrm{f}(x,y,z,t)=0$$

$$-x\left(\frac{\partial}{\partial y}\mathrm{f}(x,y,z,t)\right)-\left(\frac{\partial}{\partial z}\mathrm{f}(x,y,z,t)\right)=0$$

```
> sol := pdsolve(myPDEsystem);
```

$$sol:=\left\{\mathrm{f}(x,y,z,t)=\frac{\_C1(-xz+y)t^{(3/2)}-3t^2x\sqrt{-xz+y}}{(-xz+y)^{(3/2)}}\right\}$$

Обратите внимание на то, что в последнем примере из справки решена система дифференциальных уравнений в частных производных.

9.7.4. Функция PDEplot пакета DEtools

Одна из важнейших функций пакета `DEtools` – `DEtools[PDEplot]` – служит для построения графиков решения систем с квазилинейными дифференциальными уравнениями первого порядка в частных производных. Эта функция используется в следующем виде:

```
PDEplot(pdiffeq, var, i_curve, srange, o)
PDEplot(pdiffeq, var, i_curve, srange, xrange, yrange, urange, o)
```

Здесь, помимо упоминавшихся ранее параметров, используются следующие: `pdiffeq` – квазилинейные дифференциальные уравнения первого порядка (PDE), `vars` – независимая переменная и `i_curve` – начальные условия для параметрических кривых трехмерной поверхности. Помимо опций, указанных для функции `DEplot`, здесь могут использоваться следующие опции:

- `animate = true, false` – включение (`true`) или выключение (`false`) режима анимации графиков;
- `basechar = true, false, ONLY` – устанавливает показ начального условия на плоскости (x,y);
- `basecolor = b_color` – устанавливает цвет базовых характеристик;
- `ic_assumptions` – задание (в виде равенств или неравенств) ограничений на начальные условия для первых производных;
- `initcolor = i_color` – инициализация цвета кривой начальных условий;
- `numchar = integer` – задает число отрезков кривых, которое не должно быть меньше 4 (по умолчанию 20);
- `numsteps = [integer1,integer2]` – задает число шагов интегрирования (по умолчанию [10,10]);
- `obsrange = true, false` – прекращение интегрирования (true) при выходе отображаемой переменной за заданные пределы или продолжение интегрирования (`false`) в любом случае;
- `scene=[x,y,u(x,y)]` – вывод обозначений координатных осей.

С помощью параметров и опций можно задать множество возможностей для наглядной визуализации довольно сложных решений систем дифференциальных уравнений с частными производными. Следует отметить, что неправильное задание параметров ведет просто к выводу функции в строке вывода без построения графиков и нередко без сообщений об ошибках. Поэтому полезно внимательно просмотреть примеры применения этой функции – как приведенные ниже, так и в справке.

9.7.5. Примеры применения функции PDEplot

Рисунок 9.22 демонстрирует применение функции PDEplot. Этот пример из справки показывает, насколько необычным может быть решение даже простой системы дифференциальных уравнений в частных производных.

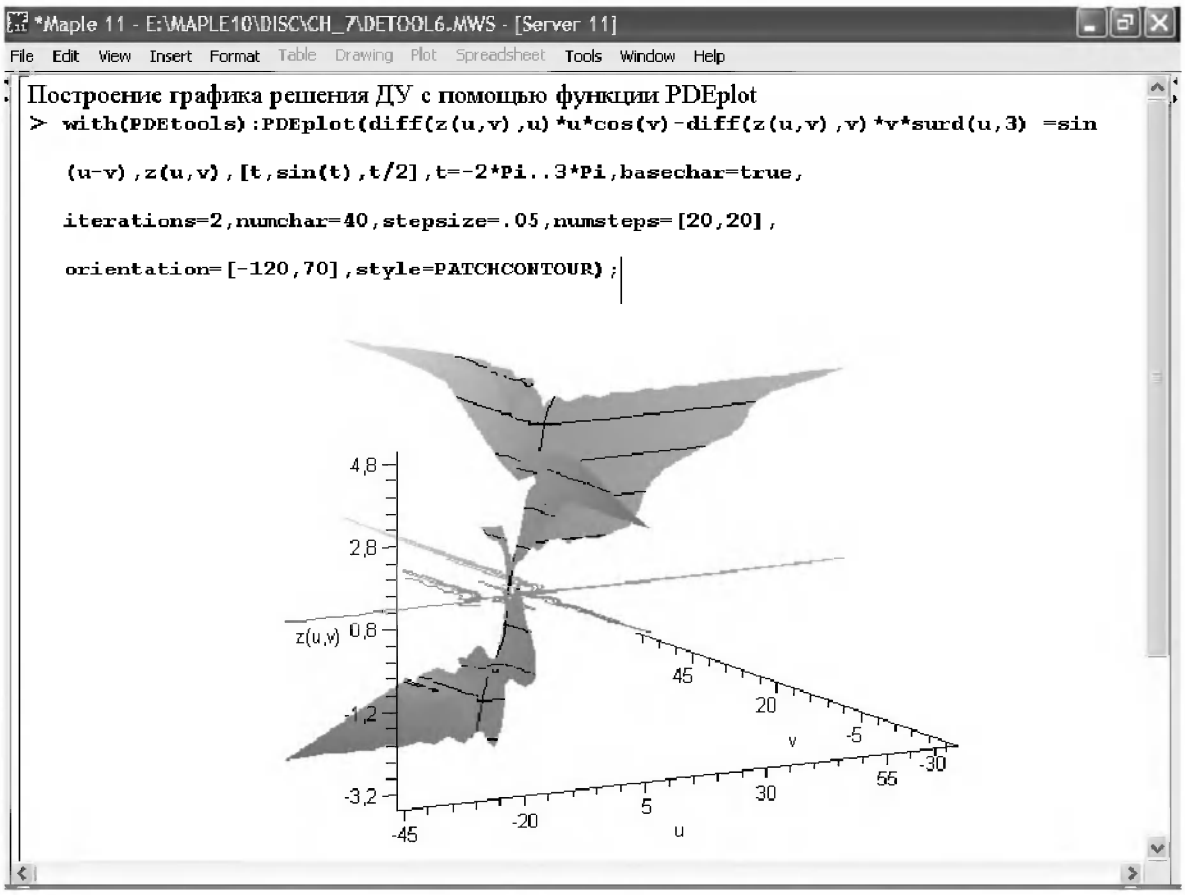


Рис. 9.22. Пример применения функции PDEplot (Maple 11)

В данном случае решение представлено трехмерной фигурой весьма нерегулярного вида. Этот пример работает в среде систем Maple 9/9.5/10/11.

Другой пример использования функции `PDEplot` показан на рис. 9.23. Он иллюстрирует комбинированное построение графиков решения разного типа с применением функциональной закраски, реализуемой по заданной формуле с помощью опции `initcolor`.

Еще раз отметим, что, к сожалению, рисунки в данной книге не дают представления о цвете выводимых системой Maple графиков. Поэтому наглядность решений, видимых на экране монитора, существенно выше.

9.8. Сложные колебания в нелинейных системах и средах

9.8.1. Пример нелинейной системы и моделирование колебаний в ней

Многие системы (например, нелинейные оптические резонаторы, лазерные устройства и др.) описываются системами из более чем двух нелинейных дифференциальных уравнений. Колебания в таких системах нередко носят сложный неста-

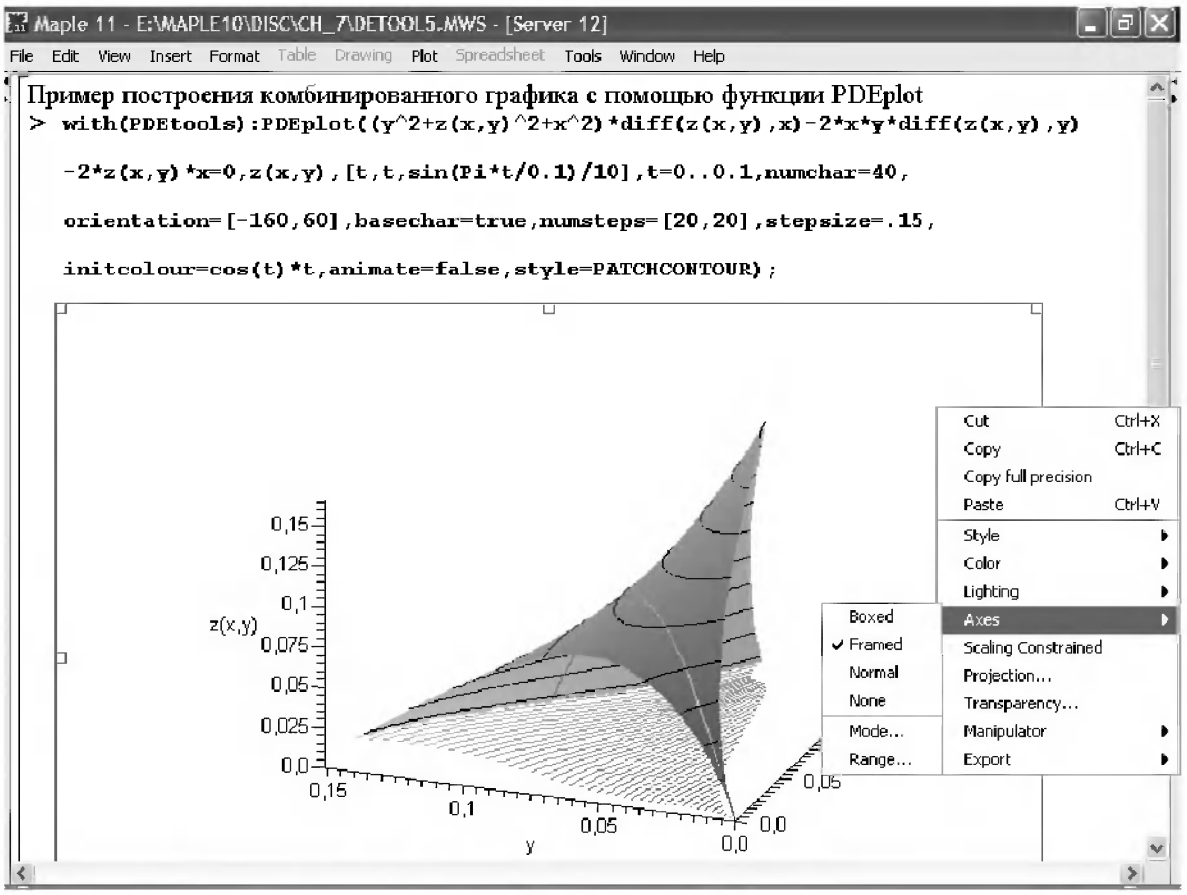


Рис. 9.23. Построение комбинированного графика с помощью функции PDEplot

ционарный, а порой даже хаотический характер. Примером этого может служить анализ переходных процессов в системе, описываемой тремя дифференциальными уравнениями и представленной на рис. 9.24.

Поведение системы описывается тремя постоянными sigma, b и r, меняя которые можно получить самый различный вид временных зависимостей $x(t)$, $y(t)$ и $z(t)$. Даже на ограниченном промежутке времени эти зависимости имеют весьма сложный и почти непредсказуемый характер и далеки от периодических колебаний. Нередко в них проглядывает фрактальный характер.

9.8.2. Фазовый портрет на плоскости

Функция odeplot позволяет получать не только графики временных зависимостей, но и фазовые портреты колебаний. Рисунок 9.25 показывает построение фазового портрета в плоскости (x,y).

Нетрудно заметить, что фазовый портрет отчетливо выделяет два фокуса, которые соответствуют слабым осцилляциям нарастающих, почти гармонических колебаний, время от времени повторяющимся. В целом же фазовый портрет колебаний оказывается довольно запутанным и хорошо иллюстрирует развитие нестационарных компонент колебаний.

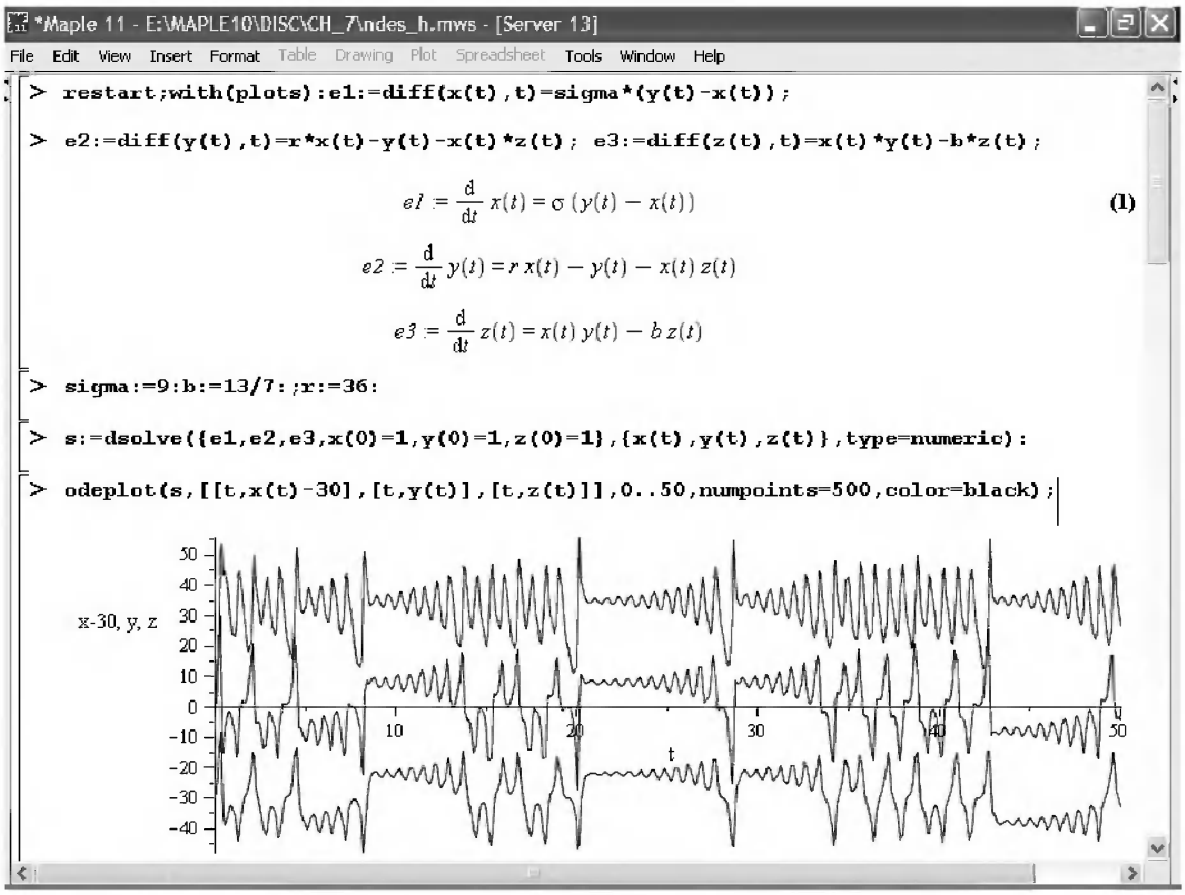


Рис. 9.24. Пример решения системы из трех нелинейных дифференциальных уравнений, создающей колебания сложной формы

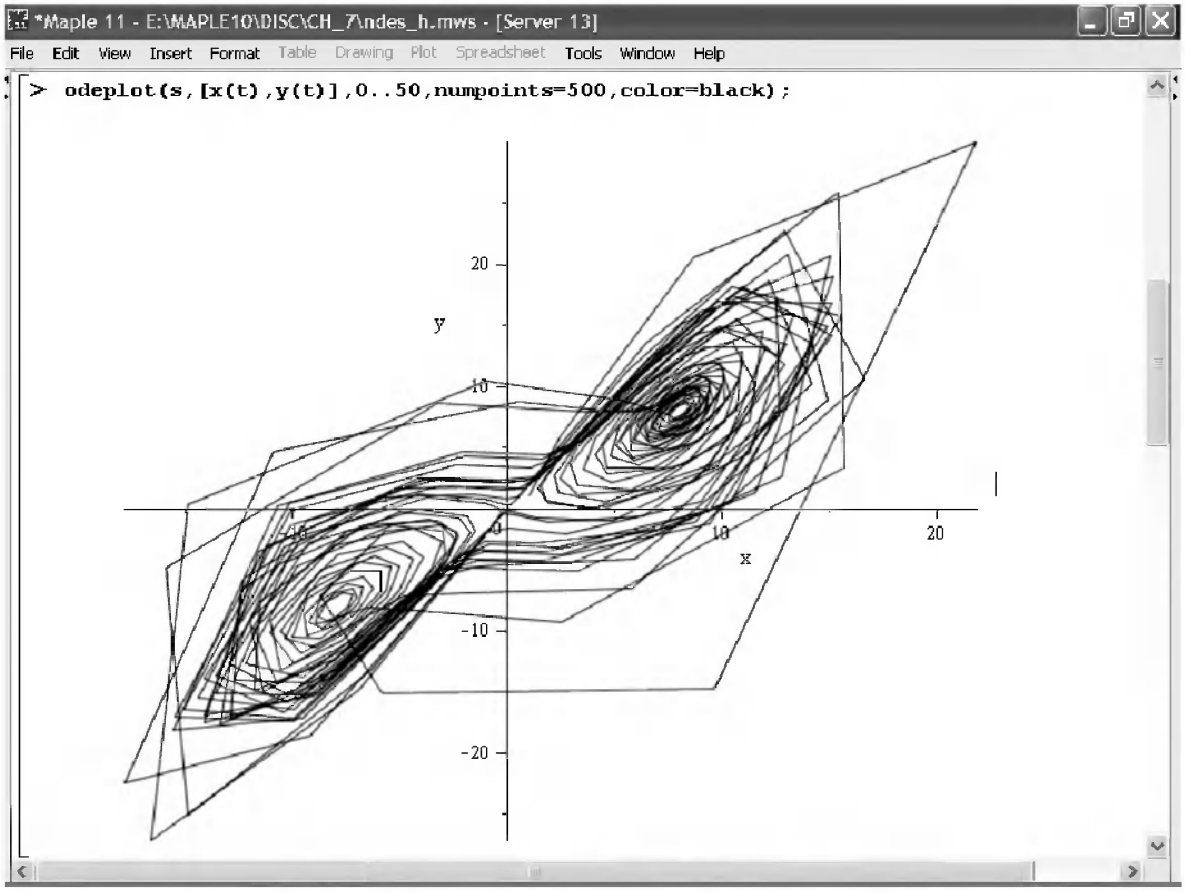


Рис. 9.25. Фазовый портрет колебаний на плоскости (x,y)

9.8.3. Фазовые портреты в пространстве

Можно разнообразить представления о колебаниях, перейдя к построению трехмерных (пространственных) фазовых портретов. Они делают такое представление более полным. На рис. 9.26 представлен фазовый портрет в пространстве при параметрическом задании семейства функций $[x(t), y(t), z(t)]$.

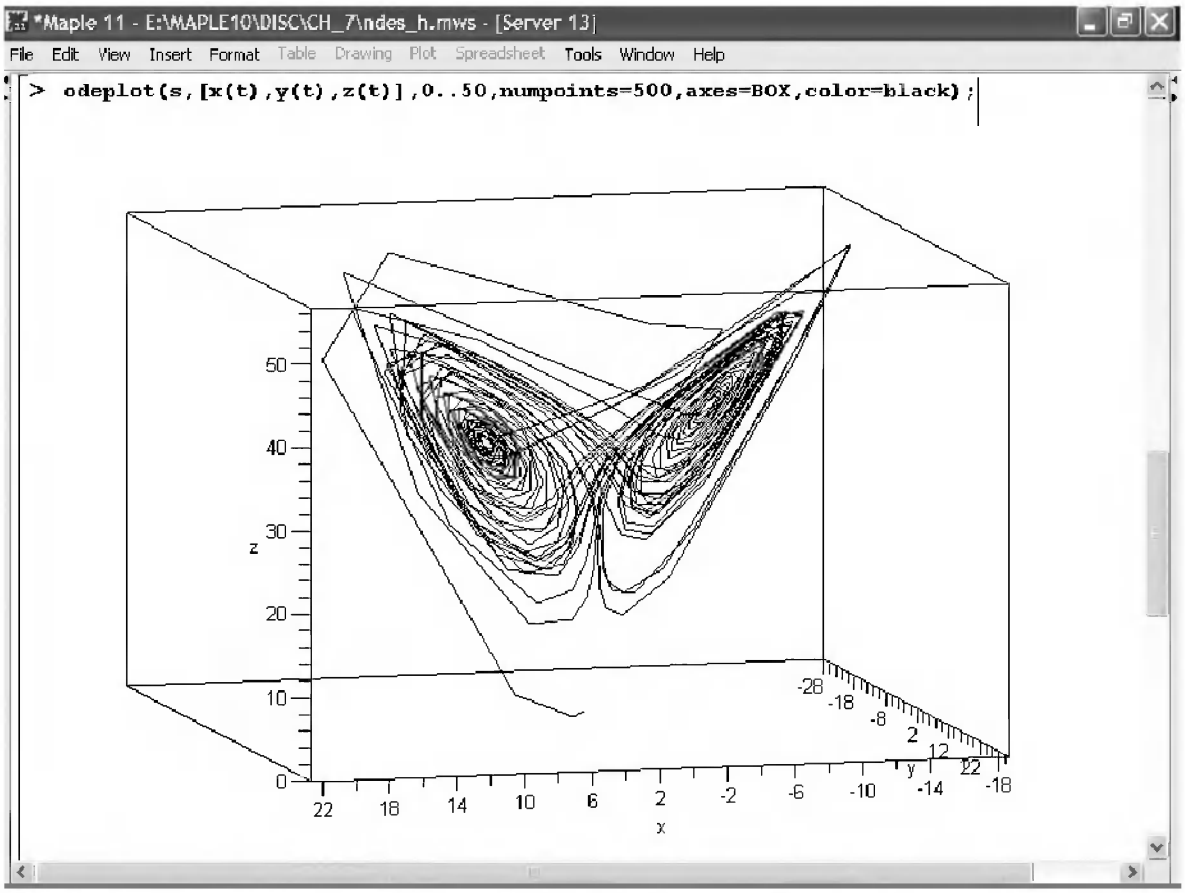


Рис. 9.26. Фазовый портрет колебаний в пространстве

Фазовый портрет отчетливо выявляет, что большая часть колебаний развивается в двух плоскостях пространства, причем в каждой из них имеется свой фокус.

Еще один вариант пространственного фазового портрета показан на рис. 9.27. Он хорошо представляет динамику развития колебаний в плоскости (y, z) при изменении времени t. Фазовый портрет весьма любопытен – хорошо видны две «трубки», в которых развиваются переходные процессы. В них можно выделить характерные раскручивающиеся спирали.

Остается отметить, что для повышения наглядности переходных процессов в графиках рис. 9.26 и 9.27 используются вывод осей координат в виде «ящика» (опция axes=BOX) и поворот изображения с помощью мыши.

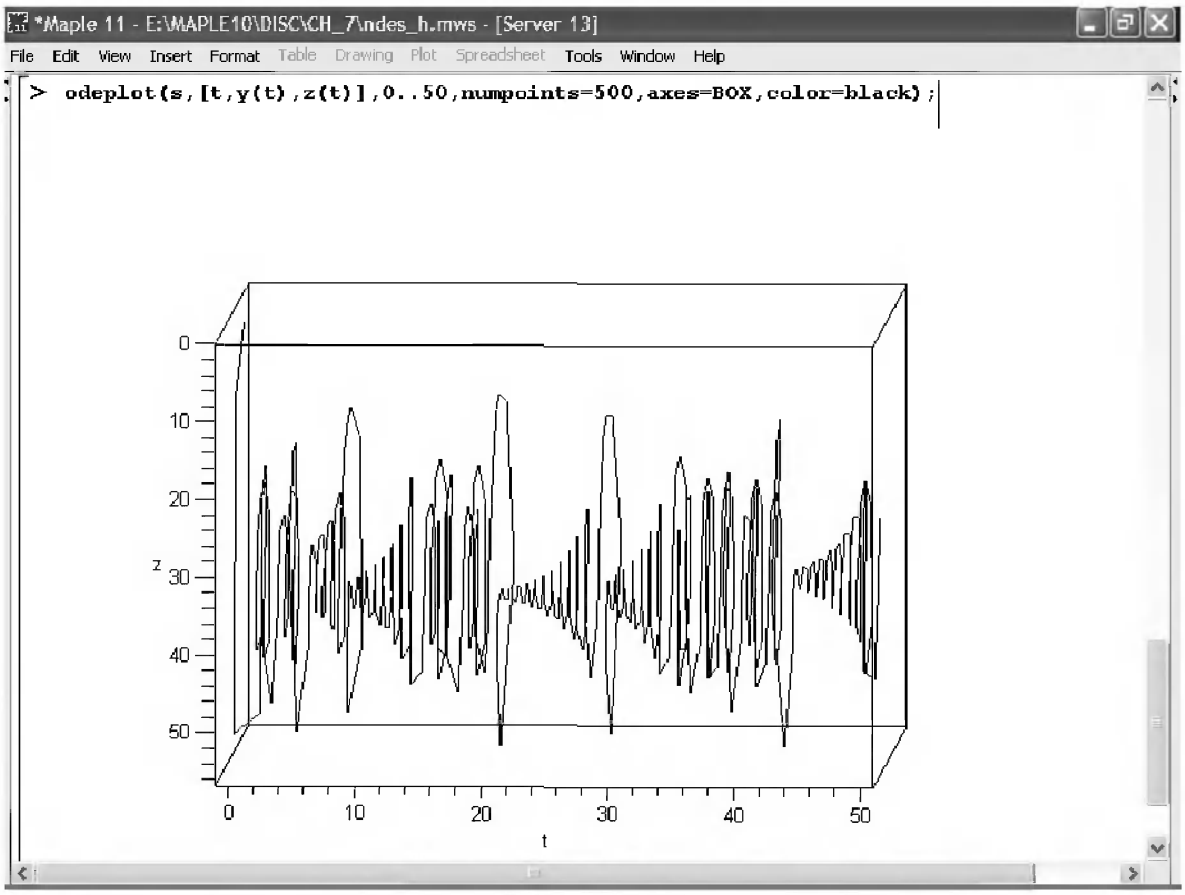


Рис. 9.27. Фазовый портрет колебаний в пространстве [t, y(t), z(t)]

9.8.4. Распространение волн в нелинейной среде

Многие наяву или в кино видели, как большие волны воды в море или океане теряют свой гармонический характер. Их гребни явно движутся быстрее, чем впадины, в результате во времени гребень достигает предшествующей ему впадины и может гипотетически даже перегнать ее. Радиотехники давно научились использовать распространение волн в нелинейных средах для получения очень коротких перепадов напряжения или тока.

Моделирование этого сложного явления (обострения фронта волн и потеря ими устойчивости) достаточно просто осуществляется волновым дифференциальным уравнением в частных производных Бюргерса. Рисунок 9.28 показывает пример задания и решения этого уравнения.

Здесь поначалу задана синусоидальная волна, которая хорошо видна на переднем плане рисунка для малых времен t. Представление результата моделирования в трехмерном пространстве позволяет наглядно представить, как меняется форма волны во времени. Нетрудно заметить, что фронт волны и впрямь обостряется и может даже приобрести отрицательный наклон. Сваливание гребня волны во впадину в этой модели не учитывается.

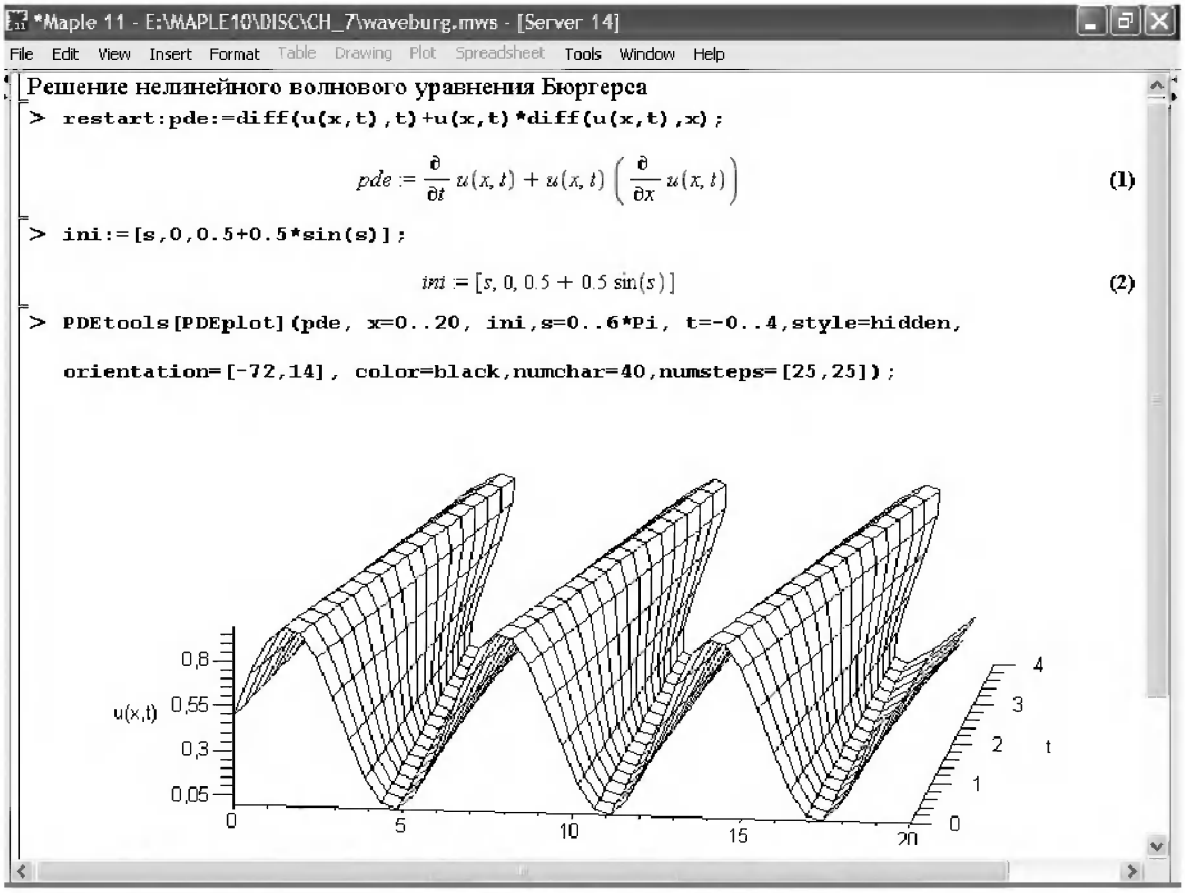


Рис. 9.28. Моделирование процесса распространения волн в нелинейной среде

9.9. Интерактивное решение дифференциальных уравнений в Maple

9.9.1. Новые средства интерактивного решения дифференциальных уравнений

Поскольку Maple – университетская система, разработчики Maple предприняли большие усилия в повышении степени визуализации всех стадий решения дифференциальных уравнений. В частности, были введены новые средства решения дифференциальных уравнений в интерактивном режиме, при котором каждая стадия решения наглядно отображается в соответствующем окне. Это едва ли нужно инженерам и научным работникам, понимающим суть и стадии решения, но, безусловно, полезно преподавателям и студентам высших учебных заведений.

Новые средства решения дифференциальных уравнений представляют собой ряд окон, созданных средствами пакета расширения Maplets. Каждое окно содержит поля для представления уравнений или параметров, там, где это надо, – поля для представления графиков и кнопки управления. Довольно подробное описа-

ние процесса интерактивного решения дифференциальных уравнений дано в разделе ODE Analyzer справки. Доступ к нему представлен на рис. 9.29.

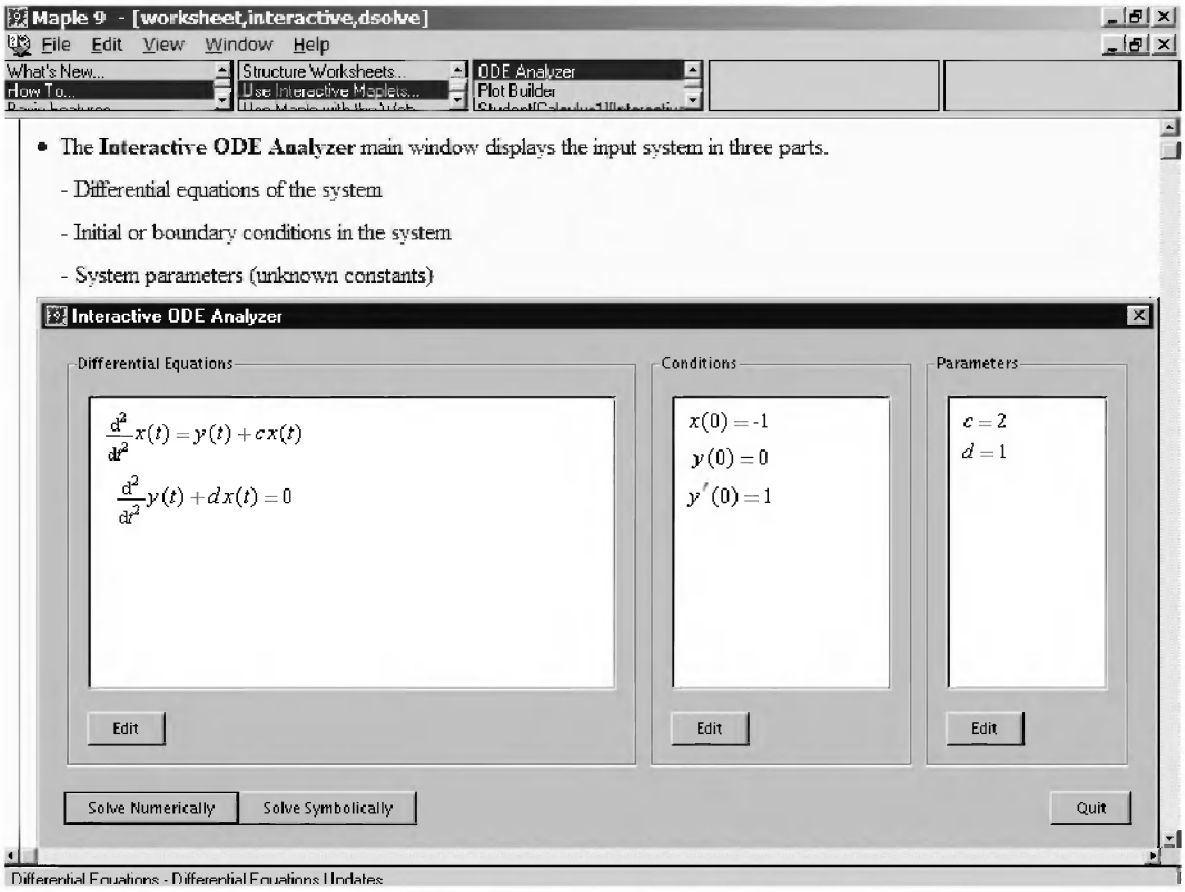


Рис. 9.29. Окно справки по разделу ODE Analyzer (Maple 9)

9.9.2. Примеры интерактивного решения дифференциальных уравнений

Для интерактивного решения дифференциальных уравнений используется функция dsolve в следующей записи:

```
dsolve[interactive](odesys, options)
```

Здесь указание [interactive] задает вывод первого окна с записью дифференциальных уравнений, представленных параметром odesys и необходимыми опциями options. Примеры применения функции dsolve уже неоднократно приводились.

На рис. 9.29 в центре видно окно с записью дифференциальных уравнений, начальных условий для его решения и значениями параметров. Предусмотрено

редактирование как самой системы, так и начальных условий и параметров. Для этого достаточно активизировать кнопку Edit в соответствующей части окна. Примеры редактирования можно найти в справке, они просты и наглядны, а потому детали описания редактирования опускаются.

Внизу окна имеются две кнопки:

- `Solve Numerically` – решение численное;
- `Solve Symbolically` – решение символьное (аналитическое).

На рис. 9.30 представлен пример численного решения. Для этого случая возможно задание одного из пяти методов решения, задание (если нужно) граничных условий и построение графика решения.

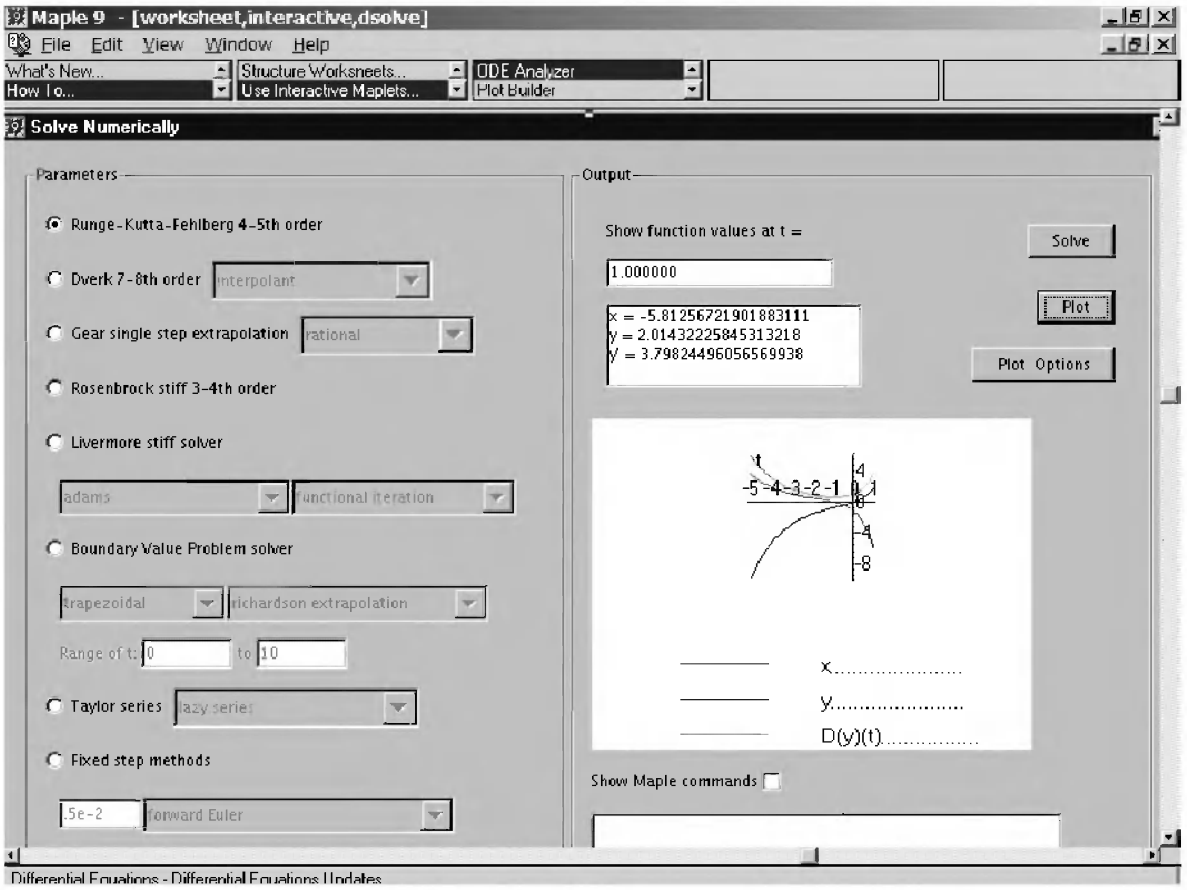


Рис. 9.30. Пример решения системы ОДУ численным методом

Другой случай решения аналитически представлен на рис. 9.31. Решение появляется в подокне при нажатии клавиши Solve. В данном случае оно поместилось в подокне, но если решение слишком громоздко, то, активизировав кнопку Lagre Display, можно вывести решение в отдельное большое окно.

Для изменения параметров графиков служит отдельное окно. С его работой и другими деталями интерактивного решения можно познакомиться по справке.

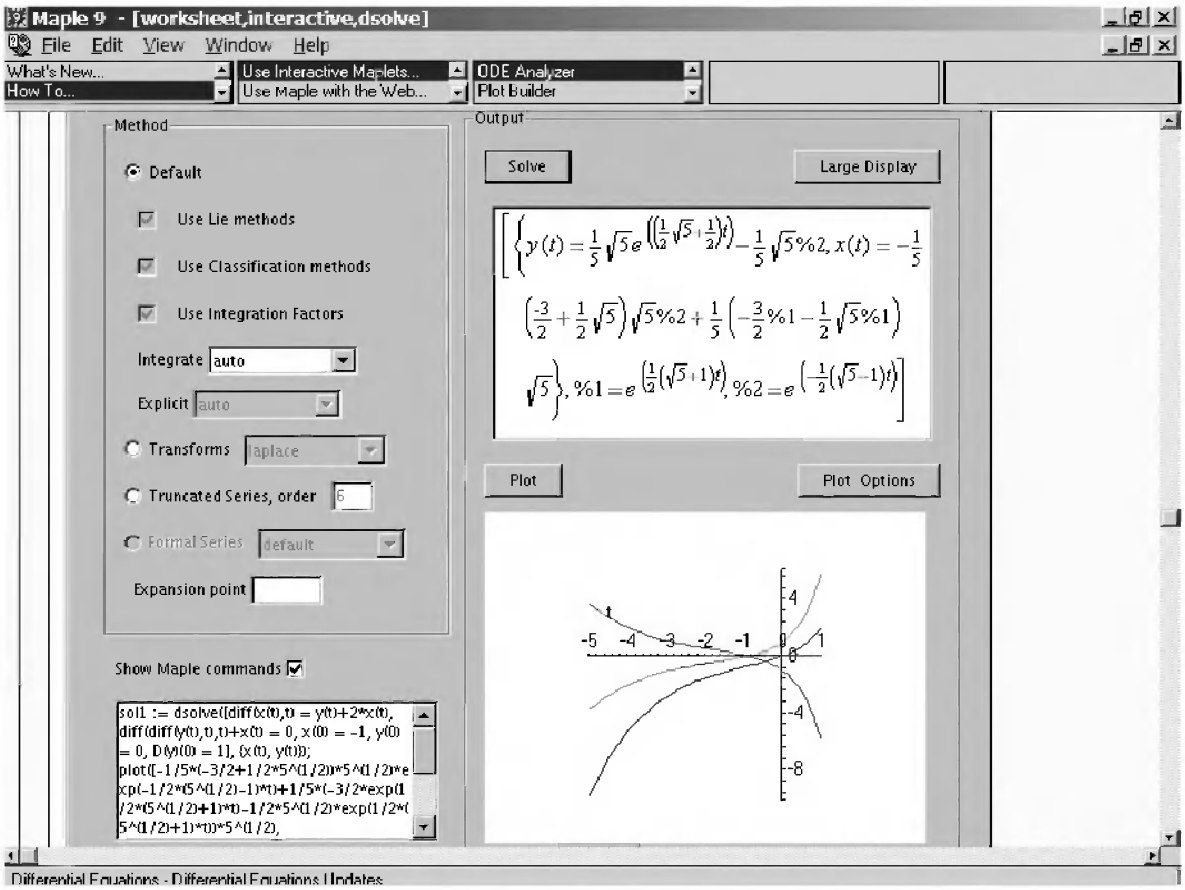


Рис. 9.31. Пример решения системы ОДУ символьным методом

9.10. Анализ линейных функциональных систем

Пакет расширения LinearFunctionalSystems системы Maple содержит специальные средства для решения дифференциальных уравнений, описывающих линейные функциональные системы.

9.10.1. Назначение пакета LinearFunctionalSystems СКМ Maple

Пакет LinearFunctionalSystems содержит набор функций для решения задач, связанных с анализом *линейных функциональных систем*. Обычно такие системы описываются линейными дифференциальными уравнениями, имеющими то или иное решение. Пакет LinearFunctionalSystems позволяет провести тестирование подготовленной системы, оценить ряд ее параметров и получить решение одним из ряда методов.

Вызов всех функций пакета осуществляется командой:

```
> with(LinearFunctionalSystems);
```

[*AreSameSolution*, *CanonicalSystem*, *ExtendSeries*, *HomogeneousSystem*, *IsSolution*,

MatrixTriangularization, PolynomialSolution, Properties, RationalSolution, SeriesSolution, UniversalDenominator]

9.10.2. Тестовые функции пакета LinearFunctionalSystems

Прежде чем рассматривать основные функции пакета, рассмотрим две тестовые функции. Они представлены следующими формами записи:

```
IsSolution(sol, sys, vars)   IsSolution(sol, A, b, x, case)
IsSolution(sol, A, x, case)  AreSameSolution(sol, sol1)
```

В них: `sol` – тестируемое решение, `sys` – система функциональных уравнений, `x` – независимая переменная решения, `A` и `b` – матрица и вектор с рациональными элементами, `case` – имя метода решения (`'differential'`, `'difference'` или `'qdifference'`).

9.10.3. Функции решения линейных функциональных систем

Группа основных функций пакета `LinearFunctionalSystems` имеет идентичный синтаксис и записывается в виде:

```
name(sys,vars,[method])   или   name(A[,b],x,case,[method])
```

Здесь `name` – одно из следующих имен:

- `PolynomialSolution` – решение в форме полинома;
- `RationalSolution` – решение в форме рационального выражения;
- `SeriesSolution` – решение в виде ряда;
- `UniversalDenominator` – решение с универсальным знаменателем (и числителем, равным 1).

Система функциональных уравнений задается либо в виде полной системы `sys` со списком переменных `vars`, либо в матричном виде с заданием матрицы коэффициентов системы **A** и вектора свободных членов **b** (может отсутствовать) с указанием независимой переменной `x` и параметра `case`, имеющего значения `'differential'`, `'difference'` или `'qdifference'`. Параметр `method`, задающий метод EG-исключения, может иметь значения `'quasimodular'` или `'ordinary'`.

9.10.4. Вспомогательные функции

Несколько вспомогательных функций пакета `LinearFunctionalSystems` представлены ниже:

- `MatrixTriangularization(mat, m, n, x, lt)` – триангуляция матрицы `mat` размера `m`ґ`n` с указанием типа `lt` (`'lead'` или `'trail'`);
- `CanonicalSystem(shift, sys, vars)` или `CanonicalSystem(shift, A[, b], x, case)` – возвращает систему

в каноническом виде (параметр `shift` задается как `'difference'` или `'q-difference'`, назначение других параметров соответствует указанным выше для других функций);

- `ExtendSeries(sol, deg)` – расширяет ряд решения `sol` до расширенного ряда степени `deg`;
- `HomogeneousSystem(homo, sys, vars)` или `HomogeneousSystem (homo, A[, b], x, case)` – преобразует исходную систему в гомогенную с именем `homo`;
- `Properties(sys, vars)` или `Properties(A[, b], x, case)` – возвращает основные свойства системы.

9.10.5. Примеры применения пакета LinearFunctionalSystems

Ниже представлен ряд примеров применения пакета LinearFunctionalSystems, иллюстрирующих его возможности:

```
> with(LinearFunctionalSystems):
sys := [diff(y1(x), x) - y2(x),
diff(y2(x), x) - y3(x) - y4(x),
diff(y3(x), x) - y5(x),
diff(y4(x), x) - 2*y1(x) - 2*x*y2(x) - y5(x),
diff(y5(x), x) - x^2*y1(x) - 2*x*y3(x) - y6(x),
diff(y6(x), x) - x^2*y2(x) + 2*y3(x)]:
vars := [y1(x), y2(x), y3(x), y4(x), y5(x), y6(x)]:
sol:=PolynomialSolution(sys, vars);
```

$$sol := \left[-\_c_2 - \frac{1}{2}x\_c_3, -\frac{1}{2}\_c_3, \_c_1 + x\_c_2 + \frac{1}{2}x^2\_c_3, -\_c_1 - x\_c_2 - \frac{1}{2}x^2\_c_3, \_c_2 + x\_c_3, \right.$$
$$\left. \_c_3 - 2x\_c_1 - x^2\_c_2 - \frac{1}{2}x^3\_c_3 \right]$$

```
> IsSolution(sol, sys, vars);
```

$$true$$

```
> sol1 := [-_c[1]+x*_c[3], _c[3], -_c[2]+x*_c[1]-x^2*_c[3], _c[2]-
x*_c[1]+x^2*_c[3], _c[1]-2*x*_c[3], -2*_c[3]+2*x*_c[2]-
x^2*_c[1]+x^3*_c[3]];
```

$$sol1 := [-\_c_1 + x\_c_3, \_c_3, -\_c_2 + x\_c_1 - x^2\_c_3, \_c_2 - x\_c_1 + x^2\_c_3, \_c_1 - 2x\_c_3,$$
$$-2\_c_3 + 2x\_c_2 - x^2\_c_1 + x^3\_c_3]$$

```
> IsSolution(sol1, sys, vars);
```

$$true$$

```
> AreSameSolution(sol,sol1);
```

$$true$$

```
> sol[1] := 1;
```

$$sol_1 := 1$$

```
> IsSolution(sol, sys, vars);
```

$$[0, -\infty, -\infty, 1, 3, -\infty]$$

```
> sol2 := eval(sol1, _c[1]=0);
```

$$sol2 := [x\_c_3, \_c_3, -\_c_2 - x^2\_c_3, \_c_2 + x^2\_c_3, -2x\_c_3, -2\_c_3 + 2x\_c_2 + x^3\_c_3]$$

```
> IsSolution(sol2, sys, vars);
```

$$true$$

```
> AreSameSolution(sol1,sol2);
```

$$false$$

```
> sys := [-x^2*y2(x) + 2*x^2*y1(x) + 4*y1(x)*x - y1(x + 1)*x^2 -
4*y1(x + 1)*x + 2*y1(x) - 4*y1(x + 1), y2(x + 1) - y1(x)]:
vars:= [y1(x), y2(x)]:
UniversalDenominator(sys, vars);
```

$$\frac{1}{(x+1)^2 x^2}$$

```
> Properties(sys,vars);
```

$$res$$

Множество дополнительных примеров на анализ и решение линейных функциональных систем можно найти в справке по функциям данного пакета.

9.11. Решение дифференциальных уравнений СКМ Mathematica

9.11.1. Решение дифференциальных уравнений в символьном виде

Система Mathematica 4/5 имеет минимум средств как для символьного, так и для численного решения дифференциальных уравнений и систем дифференциальных уравнений. Прежде всего это функция:

- **DSolve[eqn, y[x], x]** – решает дифференциальное уравнение относительно функций y[x] с независимой переменной x.
- **DSolve[{eqn1, eqn2, ...}, {y1[x1, ...], ...}, {x1, ...}]** – решает систему дифференциальных уравнений.
- **DSolveConstants** – опция к **DSolve**, определяющая постоянные интегрирования, которые могут быть возвращены.
- **StartingStepSize** – опция к **NDSolve**, определяющая величину начального шага.

Приведем примеры аналитического решения дифференциальных уравнений:

```
DSolve[Derivative[1][y][x] == 2*a*x^3, y[x], x]
```

$$\{\{y[x] \to 1.5x^4 + C[1]\}\}$$

```
DSolve[{y1'[x] == 2x^2, y2'[x] == 3x}, {y1[x], y2[x]}, x]
```

$$\left\{\left\{y1[x] \to \frac{2x^3}{3} + C[1],\ y2[x] \to \frac{3x^2}{2} + C[2]\right\}\right\}$$

```
DSolve[y''[x] - y'[x] - 6y[x] == 0, y[x], x]
{{y[x] → e^(-2x) C[1] + e^(3x) C[2]}}
DSolve[y''[x] + 4y'[x] == 10 Sin[2x], y[x], x]
{{y[x] → -1/4 e^(-4x) C[1] + C[2] - Cos[2x] - 1/2 Sin[2x]}}
DSolve[y'[x] == Sin[E^x], y[x], x]
{{y[x] → C[1] + SinIntegral[e^x]}}
DSolve[z^2 w''[z] + z w'[z] - (z^2 + 1) w[z] == 0, w[z], z]

{{w[z]→BesselJ[1,-i z] C[1]+BesselY[1,-i z] C[2]}}
DSolve[y''[x] - (a - 2qCosh[2x]) y[x] == 0, y[x], x]
{{y[x] → C[1] MathieuC[a, q, -i x] + C[2] MathieuS[a, q, -i x]}}
```

Как нетрудно заметить, аналитические решения дифференциальных уравнений могут содержать не только элементарные, но и специальные математические функции, что заметно расширяет возможности применения системы Mathematica в решении задач динамического моделирования.

В решении дифференциальных уравнений встречаются постоянные интегрирования. В общем случае они обозначаются как C[i]. Однако с помощью опции **GeneratedParameters** можно сменить обозначения постоянных интегрирования, что иллюстрируют следующие примеры:

```
DSolve[y''[x]==y'[x]+y[x]+a,y,x]
{{y → Function[{x}, -a + e^((1/2 - Sqrt[5]/2) x) C[1] + e^((1/2 + Sqrt[5]/2) x) C[2]]}}
DSolve[y''[x]==y'[x]+y[x]+a,y,x,GeneratedParameters®K]
{{y → Function[{x}, -a + e^((1/2 - Sqrt[5]/2) x) K[1] + e^((1/2 + Sqrt[5]/2) x) K[2]]}}
```

В записи дифференциальных уравнений можно ввести граничные условия, которые должны учитываться при решении. Пример этого (и проверки) решения представлен ниже:

```
DSolve[{y''[x]==a*y'[x]+y[x],y[0]==1,y'[0]==0},y,x]
{{y → Function[{x}, 1/(2 Sqrt[4 + a^2]) (a e^(1/2 (a - Sqrt[4+a^2]) x) +
    Sqrt[4 + a^2] e^(1/2 (a - Sqrt[4+a^2]) x) - a e^(1/2 (a + Sqrt[4+a^2]) x) + Sqrt[4 + a^2] e^(1/2 (a + Sqrt[4+a^2]) x))]}}
{y''[x]==ay'[x]+y[x],y[0]==1,y'[0]==0}/.%//Simplify
{{True,True,True}}
```

В следующем примере решение задается при граничном условии $y(1)=1$:

```
DSolve[{y'[x] == (-1 + x)^(-1+x) (1 + Log[-1 + x]), y[1] == 1}, y[x], x]
{{y[x] → (-1 + x)^(-1+x)}}
```

В справке по функции **DSolve** системы Mathematica 5 можно найти символьные решения ряда дифференциальных уравнений специального типа, например Абеля, Риккати и Матье. Ниже представлен пример на решение дифференциального уравнения Абеля:

```
DSolve[y'[x] + y[x]^3 == 1, y[x], x]
Solve::tdep : The equations appear to involve the variables
   to be solved for in an essentially non-algebraic way. More...
{{y[x] → InverseFunction[
      -ArcTan[(1+2 #1)/Sqrt[3]]/Sqrt[3] + 1/3 Log[-1 + #1] - 1/6 Log[1 + #1 + #1^2] &][-x + C[1]]}}
```

Mathematica способна также решать системы дифференциально-алгебраических уравнений, например вида $F(t, x, x') = \text{expr}$. Ниже представлен пример решения системы дифференциально-алгебраических уравнений с проверкой решения:

```
eqns = {x'[t] - y[t]==1, x[t] + y[t] == 2};
sol = DSolve[eqns, {x,y}, t]
Out[6]= {{x → Function[{t}, 1/4 (12 + E^-t C[1])], y → Function[{t}, 2 + 1/4 (-12 - E^-t C[1])]}}
eqns/.sol//Simplify
   {{True, True}}
```

Обратите внимание на то, что ответ получен через чистые функции. Они описаны в главе 13 и представляют собой функции без конкретного имени (но с обобщенным – **Function**).

9.11.2. Решение дифференциальных уравнений в частных производных

Для решения таких уравнений в системе Mathematica предусмотрена функция **DSolve**, параметры которой были уже описаны. В справке по системе и во встроенной книге Вольфрама можно найти множество примеров на решение дифференциальных уравнений и систем дифференциальных уравнений в частных производных. В связи с этим ограничимся приведением нескольких примеров на решение таких уравнений:

```
DSolve[D[y[x1, x2], x1] + D[y[x1, x2], x2] == a*x1/x2,
    y[x1, x2], {x1, x2}]
 {{y[x1,x2]→a x1+a x1 Log[x2]-a x2 Log[x2]+C[1][-x1+x2]}}
DSolve[D[y[x1, x2], x1] + D[y[x1, x2], x2] == a*x1/x2,
    y, {x1, x2}]
 {{y®Function[{x1,x2},a x1+a x1 Log[x2]-a x2 Log[x2]+C[1][-x1+x2]]}}
(c^2 D[#, x, x] - D[#, t, t])& [y[x, t]] == 0
 -y^(0,2)[x, t] + c^2 y^(2,0)[x, t] ==0
DSolve[x1 D[y[x1, x2], x1] + x2 D[y[x1, x2], x2]
                == Exp[x1/x2], y[x1, x2], {x1, x2}]
```

$$\left\{\left\{y[x1, x2] \to e^{x1/x2}\,\text{Log}[x1] + C[1]\left[\frac{x2}{x1}\right]\right\}\right\}$$

```
DSolve[x1 D[y[x1, x2], x1] + x2 D[y[x1, x2], x2]
                == Exp[x1 x2], y[x1, x2], {x1, x2}]
```

$$\left\{\left\{y[x1, x2] \to \frac{1}{2}\left(\text{ExpIntegralEi}[x1\,x2] + 2\,C[1]\left[\frac{x2}{x1}\right]\right)\right\}\right\}$$

```
DSolve[D[y[x1, x2], x1] D[y[x1, x2], x2] == w^2, y[x1, x2], {x1, x2}]
 DSolve[y^(0,1)[x1,x2]y^(1,0)[x1,x2] == w^2, y[x1,x2], {x1,x2}]
```

Из этих примеров хорошо видны формы задания дифференциальных уравнений в частных производных и формы вывода их решений.

9.11.3. Решение дифференциальных уравнений в численном виде

Для численного решения систем дифференциальных уравнений в Mathematica 4/5 используется функция:

- **NDSolve[eqns, y, {x, xmin, xmax}]** – ищет численное решение дифференциальных уравнений eqns относительно функции y независимой переменной x в интервале от xmin до xmax.
- **NDSolve[eqns, {y1, y2, ...}, {x, xmin, xmax}]** – ищет численные решения относительно функций yi.
- **MaxSteps** – опция к **NDSolve**, которая определяет максимальное количество шагов.

Часто весьма желательно выводить результаты решения дифференциальных уравнений в графической форме. Рисунок 9.32 поясняет, как это делается при решении системы нелинейных дифференциальных уравнений, описывающих достаточно сложный колебательный процесс.

Нередко решение предпочитают представить на фазовой плоскости. Рисунок 9.33 иллюстрирует такую возможность. Более того, учитывая, что решается система из трех дифференциальных уравнений, фазовая траектория решения находится в трехмерном пространстве.

Простота задания решения и вывода его результатов в графической форме открывает широкие возможности в применении системы для математического моделирования сложных явлений. При этом, в отличие от такого решения с помощью обычных языков высокого уровня (например, Фортрана, Бейсика, Паскаля или Си), не требуется составления каких-либо программ по реализации численных методов решения систем дифференциальных уравнений, скажем таких, как метод Рунге-Кутта. Они представлены в виде уже готовых функций.

Несмотря на сказанное, представляется, что степень визуализации решений дифференциальных уравнений в версии системы Mathematica 5 заметно уступает таковой у конкурирующей версии системы Maple 11. Впрочем, это различие сведено к минимуму у новой версии Mathematica 6. В ней заметно расширен круг решаемых дифференциальных уравнений и улучшена техника их визуализации.

```
NDSolve[{x'[t] == -3 (x[t] - y[t]), y'[t] == -x[t] z[t] + 27 x[t] - y[t],
  z'[t] == x[t] y[t] - z[t], x[0] == z[0] == 0, y[0] == 1}, {x, y, z},
 {t, 0, 20}, MaxSteps → 3000];

Plot[Evaluate[{x[t], y[t], z[t]} /. %], {t, 0, 20}];
```

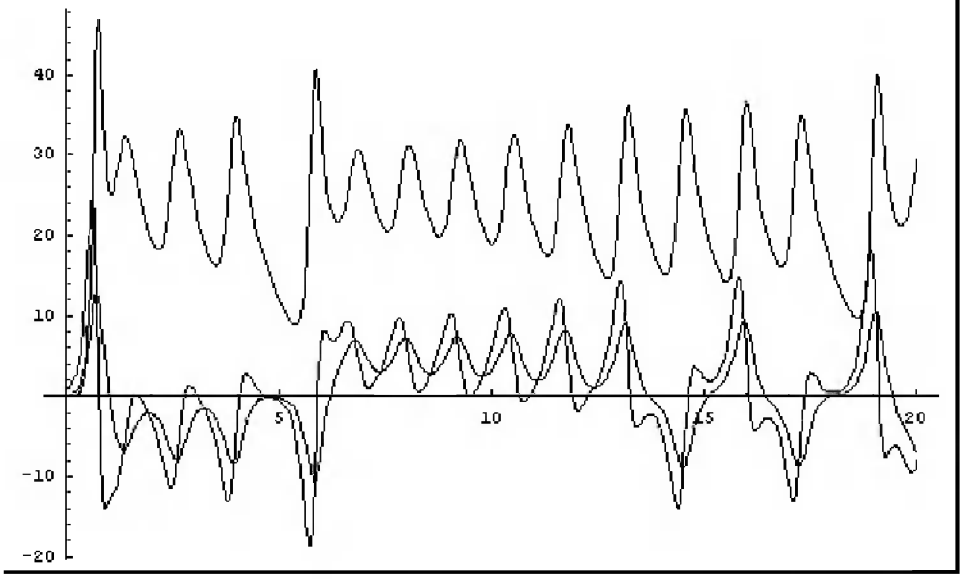


Рис. 9.32. Решение системы дифференциальных уравнений с выводом решения в виде графиков временных зависимостей

```
NDSolve[{x'[t] == -3 (x[t] - y[t]), y'[t] == -x[t] z[t] + 27 x[t] - y[t], z'[t] == x[t] y[t
  x[0] == z[0] == 0, y[0] == 1}, {x, y, z}, {t, 0, 20}, MaxSteps → 3000];

ParametricPlot3D[Evaluate[{x[t], y[t], z[t]} /. %], {t, 0, 20}, PlotPoints → 1000];
```

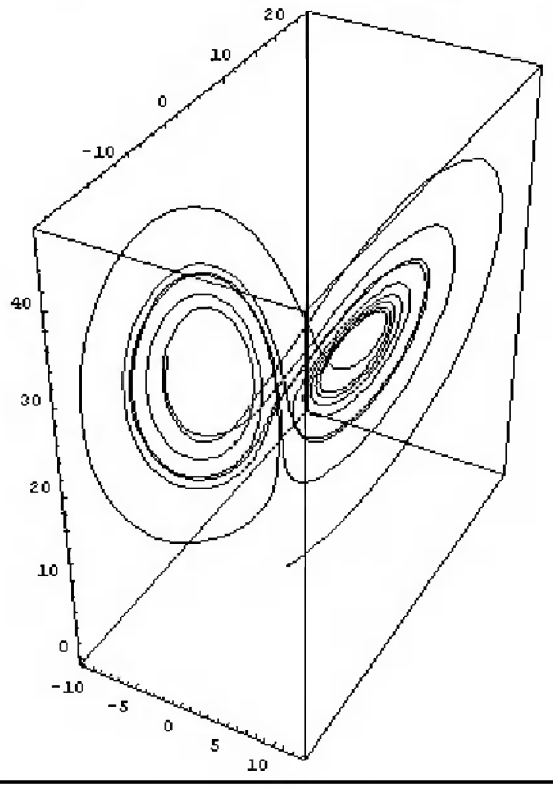


Рис. 9.33. Решение системы дифференциальных уравнений с выводом решения в форме кривых на фазовых плоскостях

9.12. Численное решение ДУ в системе Mathcad

В отличие от систем Maple и Mathematica, системы Mathcad ориентированы прежде всего на решение дифференциальных уравнений в численном виде. Для этого они предлагают довольно изысканные и вполне законченные средства, явно ориентированные на практиков, а не теоретиков. С них мы и рассмотрим возможности системы Mathcad 12 в решении дифференциальных уравнений и систем с ними. Все эти возможности есть и в последних реализациях системы – Mathcad 13 и Mathcad 14. В детали небольших различий между ними в контексте данной книги вникать не стоит.

9.12.1. Решение систем ОДУ

В Mathcad системы ОДУ можно представить в векторном виде:

```
Y(x0) = Y0      Y' = F(x, Y)
```

Отсюда следует важный вывод: решение системы ОДУ в форме Коши осуществляется аналогично решению одиночного ДУ, но должно быть организовано в *векторной форме*. При этом добавление очередного уравнения не увеличивает число уравнений в векторной их записи. ДУ `n`-го порядка может решаться стандартными средствами решения систем ОДУ, входящими в большинство математических систем после преобразования в систему ОДУ.

Для решения задач такого класса в Mathcad введен ряд функций. Вначале остановимся на функциях, дающих решения для систем обыкновенных дифференциальных уравнений, представленных в обычной форме Коши:

- `rkadapt(y,x1,x2,acc,n,F,k,s)` – возвращает матрицу, содержащую таблицу значений решения задачи Коши на интервале от `x1` до `x2` для системы обыкновенных дифференциальных уравнений, вычисленную методом Рунге-Кутта с переменным шагом и начальными условиями в векторе `y` (правые части системы записаны в векторе `F`, `n` – число шагов, `k` – максимальное число промежуточных точек решения и `s` – минимально допустимый интервал между точками, он же шаг интегрирования);
- `Rkadapt(y,x1,x2,n,F)` – возвращает матрицу решений методом Рунге-Кутта с переменным шагом для системы обыкновенных дифференциальных уравнений с начальными условиями в векторе `y`, правые части которых записаны в символьном векторе `F` на интервале от `x1` до `x2` при фиксированном числе шагов `n`;
- `rkfixed(y,x1,x2,n,F)` – возвращает матрицу решений методом Рунге-Кутта системы обыкновенных дифференциальных уравнений с начальными условиями в векторе `y`, правые части которых записаны в символьном векторе `F` на интервале от `x1` до `x2` при фиксированном числе шагов `n`.

Создаваемая этими функциями матрица содержит ряд столбцов, число которых на 1 больше числа уравнений. Первый столбец содержит значения перемен-

ной x на равных интервалах решения, а другие столбцы – значения искомых переменных. Если в процессе решения ищутся временные зависимости, то параметр x означает время t, то есть $x = t$.

Фрагмент документа Mathcad, представленный на рис. 9.34, иллюстрирует технику решения системы из двух дифференциальных уравнений Ван-дер-Поля и представление решения в виде фазового портрета колебаний, которые описывает рассматриваемая система уравнений, а также временных зависимостей решения. Система уравнений Ван-дер-Поля описывает в обобщенном виде широкий класс систем и устройств, относящихся к автогенераторам (впервые эта система была предложена для описания автогенераторов на электронных лампах, но вполне применима и к автогенераторам на современных полевых транзисторах [41]).

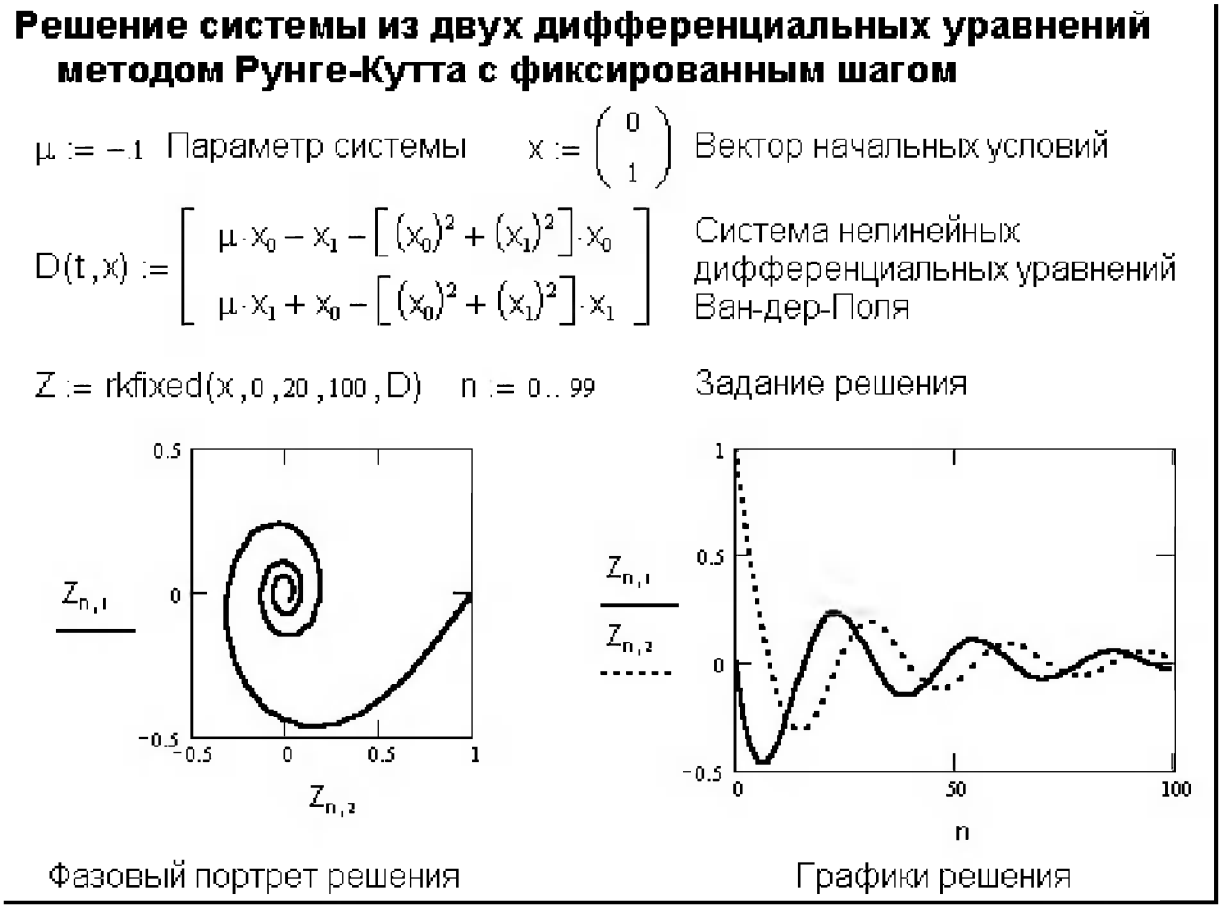


Рис. 9.34. Решение системы дифференциальных уравнений с применением функции rkfixed

Фрагмент документа Mathcad, представленный на рис. 9.35, иллюстрирует решение той же системы с применением функции `Rkadapt`. Эта функция благодаря автоматическому изменению шага решения дает более точный результат. Естественно, по скорости вычислений она проигрывает функции `rkfixed`, хотя и не всегда – если решение меняется медленно, это может привести к заметному уменьшению числа шагов. Таким образом, функция `Rkadapt` более привлекательна для решения систем дифференциальных уравнений, имеющих решения как с медленными, так и с быстрыми участками изменения. Несмотря на автома-

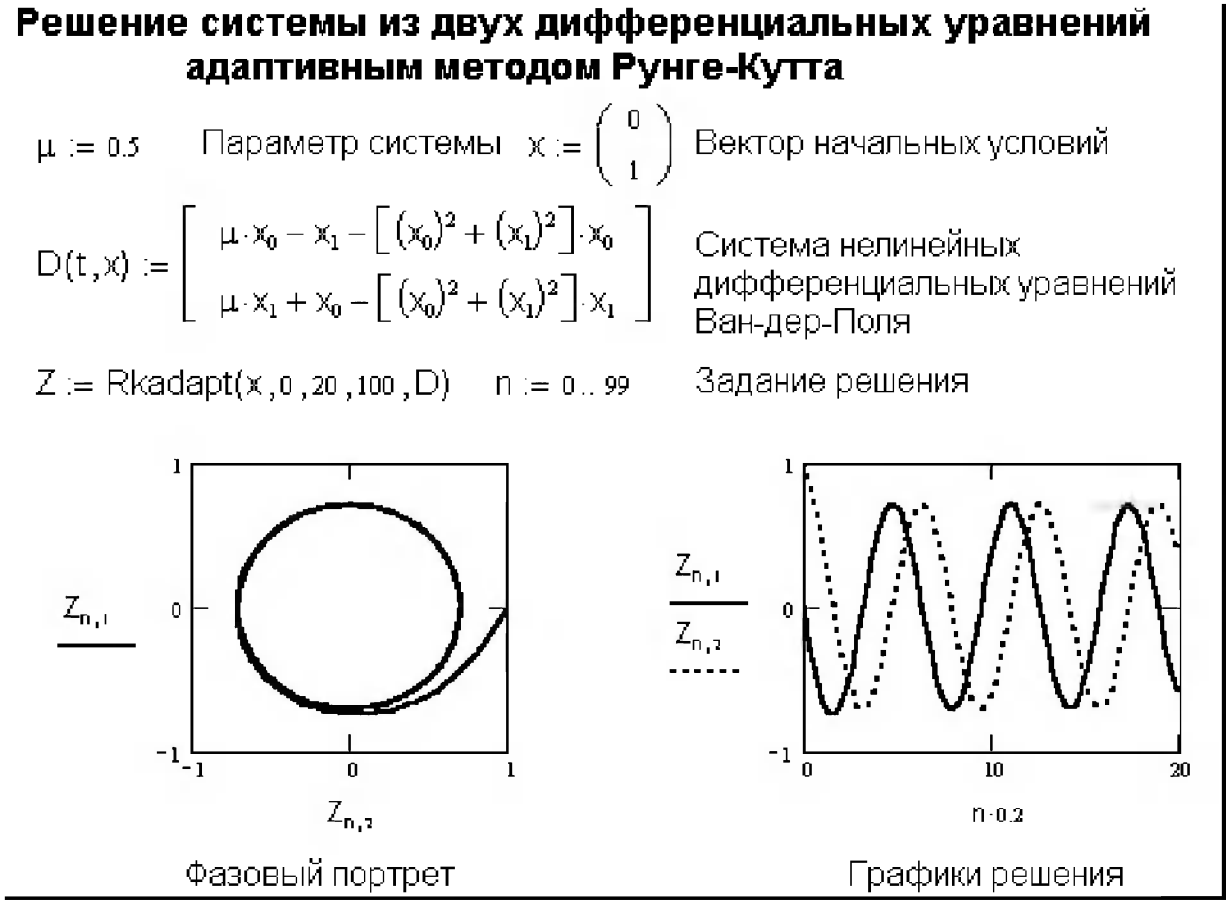


Рис. 9.35. Решение системы из двух дифференциальных уравнений с применением функции Rkadapt

тическое изменение шага, вывод решения дается при равномерном расположении точек решения.

В последнем примере, как и в предыдущем, решение представлено в виде временных зависимостей. Много других примеров решения дифференциальных уравнений можно найти в электронных книгах системы.

Если решение системы дифференциальных уравнений имеет вид гладких функций, то вместо описанной ранее функции `rkfixed` целесообразно применять новую функцию `Bulstoer (y,x1,x2,n,F)`. Она возвращает матрицу решения системы обыкновенных дифференциальных уравнений, правая часть которых (в виде первых производных неизвестных функций) записана в векторе `F(x,y)` при заданных в векторе `y` начальных условиях и при решении на интервале от `x1` до `x2` для `n` точек решения, не считая начальной точки. Вы можете заменить функцию `Rkadapt` во фрагменте документа Mathcad, показанном рис. 9.35, на функцию `Bulstoer` и опробовать ее в работе. Она реализует метод Булирша-Штера (Bulirsch-Stoer).

9.12.2. Решение ДУ с помощью функции odesolve

Решение дифференциальных уравнений в Mathcad 8.0 и в более ранних версиях выглядело несколько бессистемно и сложно. Поэтому, начиная с Mathcad 2000, была введена новая функция для решения одиночных дифференциальных урав-

нений `odesolve(x,b[,число_шагов])`, которая возвращает решение дифференциальных уравнений, описанных в блоке `Given`, при заданных начальных условиях и конце интервала интегрирования `b`.

Эта функция имеет ряд особенностей. Если указан параметр `число_шагов`, то решение выполняется с фиксированным шагом, иначе – адаптивным методом. Хотя аналитическое выражение для этой функции не выводится, с ней можно выполнять математические преобразования, например дифференцировать. Фрагмент документа Mathcad с примером применения функции `odesolve` показан на рис. 9.36.

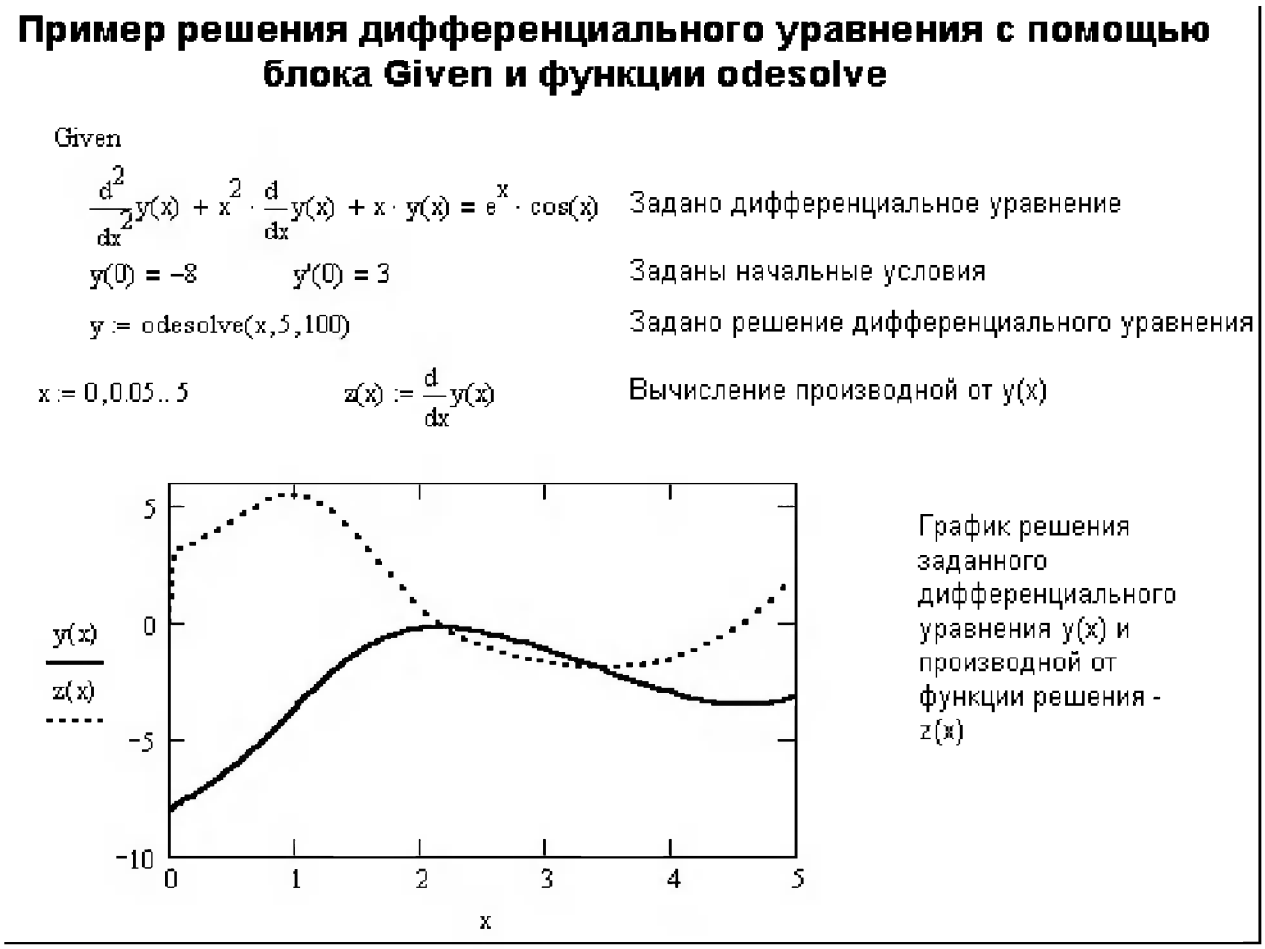


Рис. 9.36. Решение дифференциального уравнения второго порядка с помощью функции odesolve

Как показано выше, функция `odesolve` используется в составе вычислительного блока, открывающегося директивой `Given`. В нем перед функцией задаются само уравнение и начальные условия. В этом случае решение дифференциальных уравнений выглядит более логично и привычно – так же как в блоках, решающих нелинейные уравнения.

9.12.3. Решения жестких систем дифференциальных уравнений

Система дифференциальных уравнений, записанная в матричной форме y = A·x, где A – *почти вырожденная* матрица, называется *жесткой*. Впервые такие системы оказались нужны при решении задач химической кинетики. Оказалось, что при этом явные методы решения (например, методы Рунге-Кутта) не только давали низкую точность решения, но порой явно оказывались неустойчивыми. Та же ситуация имеет место при решении систем дифференциальных уравнений, описывающих работу электронных цепей с сильно различающимися постоянными времени и при решении резко нелинейных дифференциальных уравнений.

Решение таких систем характерно резко различной скоростью изменения значений переменных и требует очень малого шага, выбираемого исходя из наивысшей скорости изменения значений переменных. Оно подчас просто невозможно указанными выше явными методами. Для решения жестких дифференциальных уравнений в Mathcad введен ряд функций:

- `bulstoer(y,x1,x2,acc,n,F,k,s)` – возвращает матрицу решения системы обыкновенных дифференциальных уравнений на интервале от `x1` до `x2`, правая часть которых записана в символьном векторе `F` с заданными в векторе `y` начальными условиями (используется метод решения Булирша-Штера с переменным шагом, параметры `k` и `s` задают максимальное число промежуточных точек, на которых ищется решение, и минимально допустимый интервал между ними);
- `Stiffb(y,x1,x2,n,F,J)` – возвращает матрицу решений жесткого дифференциального уравнения, записанного в векторе `F`, и функции Якобиана `J`, `y` – вектор начальных значений на интервале `[x1,x2]` (для решения используется метод Булирша-Штера);
- `stiffb(y,x1,x2,acc,n,F,J,k,s)` – возвращает матрицу решений только в конечной точке жесткого дифференциального уравнения, записанного в векторе `F`, и функции Якобиана `J`, `y` – вектор начальных значений на интервале `[x1,x2]` (для решения используется метод Булирша-Штера с переменным шагом);
- `Stiffr(y,x1,x2,n,F,J)` – возвращает матрицу решений дифференциального уравнения, записанного в векторе `F`, и функции Якобиана `J`, `y` – вектор начальных значений на интервале `[x1,x2]` (для решения используется метод Розенброка);
- `stiffr(y,x1,x2,acc,n,F,J,k,s)` – матрица решений только в конечной форме жесткого дифференциального уравнения, записанного в векторе `F`, и функции Якобиана `J`, `y` – вектор начальных значений на интервале `[x1,x2]` (для решения используется метод Розенброка с переменным шагом).

В приведенных функциях: `acc` – погрешность решения (рекомендуется порядка 0.001), `k` – максимальное число промежуточных точек, `s` – минимально до-

пустимый интервал между точками, в которых ищется решение (шаг интегрирования). Обратите внимание, что функции, начинающиеся с малой буквы, дают решения только для конечной точки. Различаются функции также и методом решения. Если решение расходится, что может привести к переполнению разрядной сетки чисел, то прежде всего надо попытаться уменьшить шаг интегрирования.

Матрица-функция Якоби `J`, фигурирующая в этих функциях, имеет размер `n × (n+1)` и представляется в виде:

$$J(x,y)=\begin{bmatrix} \frac{\partial f_1(x,y)}{\partial x} & \frac{\partial f_1(x,y)}{\partial y_1} & \cdots & \frac{\partial f_1(x,y)}{\partial y_n} \\ \frac{\partial f_2(x,y)}{\partial x} & \frac{\partial f_2(x,y)}{\partial y_1} & \cdots & \frac{\partial f_2(x,y)}{\partial y_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_n(x,y)}{\partial x} & \frac{\partial f_n(x,y)}{\partial y_1} & \cdots & \frac{\partial f_n(x,y)}{\partial y_n} \end{bmatrix}.$$

Чаще всего эту матрицу несложно вычислить вручную или с помощью символьного дифференцирования.

Фрагмент документа Mathcad с простым примером решения жесткой системы из двух дифференциальных уравнений показан на рис. 9.37. Для решения системы характерно наличие двух резко отличающихся стадий – крутой и пологой (или быстрой и медленной, если под независимой переменной подразумевать время).

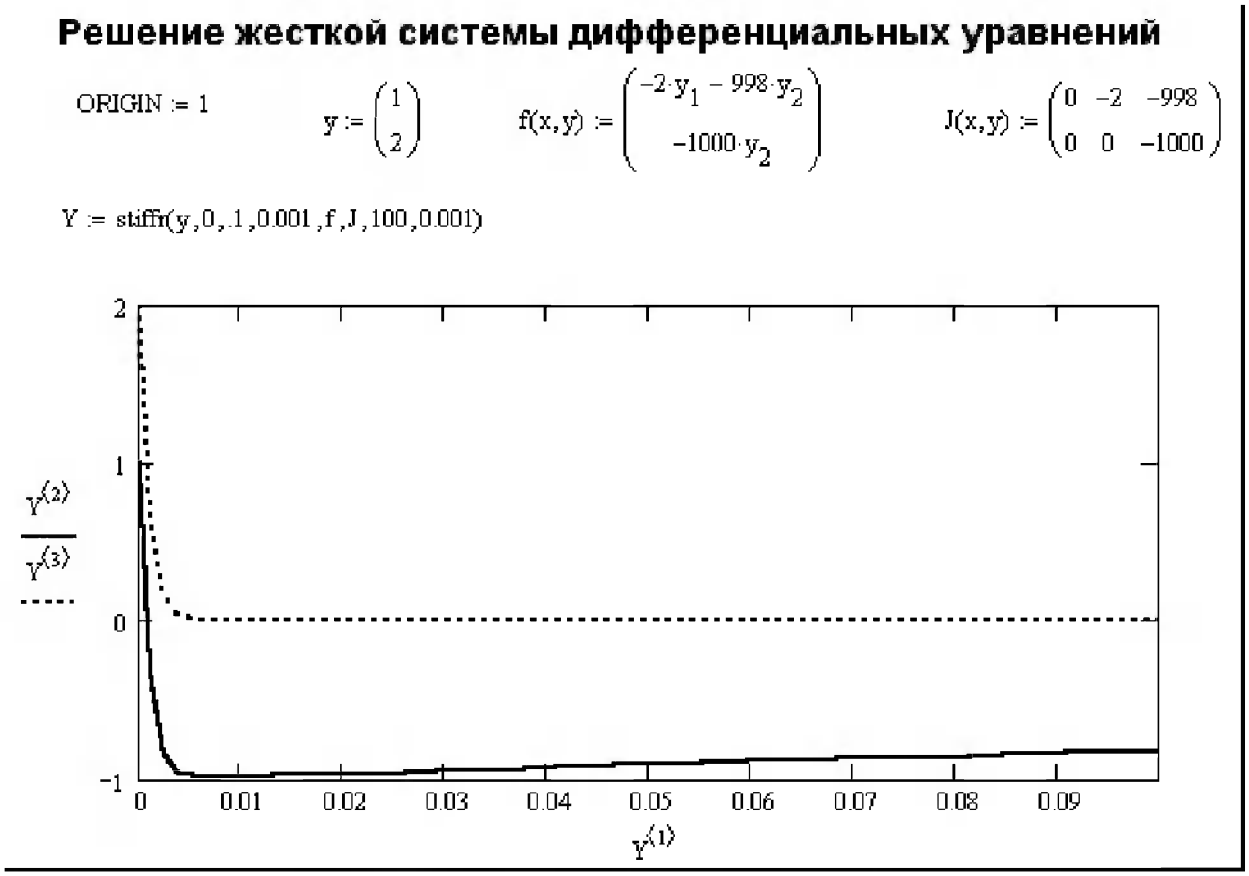


Рис. 9.37. Пример решения жесткой системы дифференциальных уравнений

В данном примере использована функция `stiffr`, которая реализует метод Розенброка. В справочной системе Mathcad можно найти и другие примеры решения жестких систем дифференциальных уравнений.

9.12.4. Пример решения жесткой системы ДУ химической кинетики

Теперь рассмотрим типичную задачу химической кинетики – изменение во времени концентрации трех веществ при их смешении. Опуская подробное описание этих превращений, хорошо известное из курса химии, зададим систему дифференциальных уравнений в виде функции F(t,y) и начальных условий y0 (это начальные концентрации фракций смеси). Далее составим матрицу Якоби и найдем решение с помощью функции **Stiffr**, реализующей метод Розенброка (вы можете убедиться в том, что другая функция **Stiffb**, реализующая метод Булирша-Штера, дает тот же результат). Подготовленный в соответствии с этим описанием документ представлен на рис. 9.38.

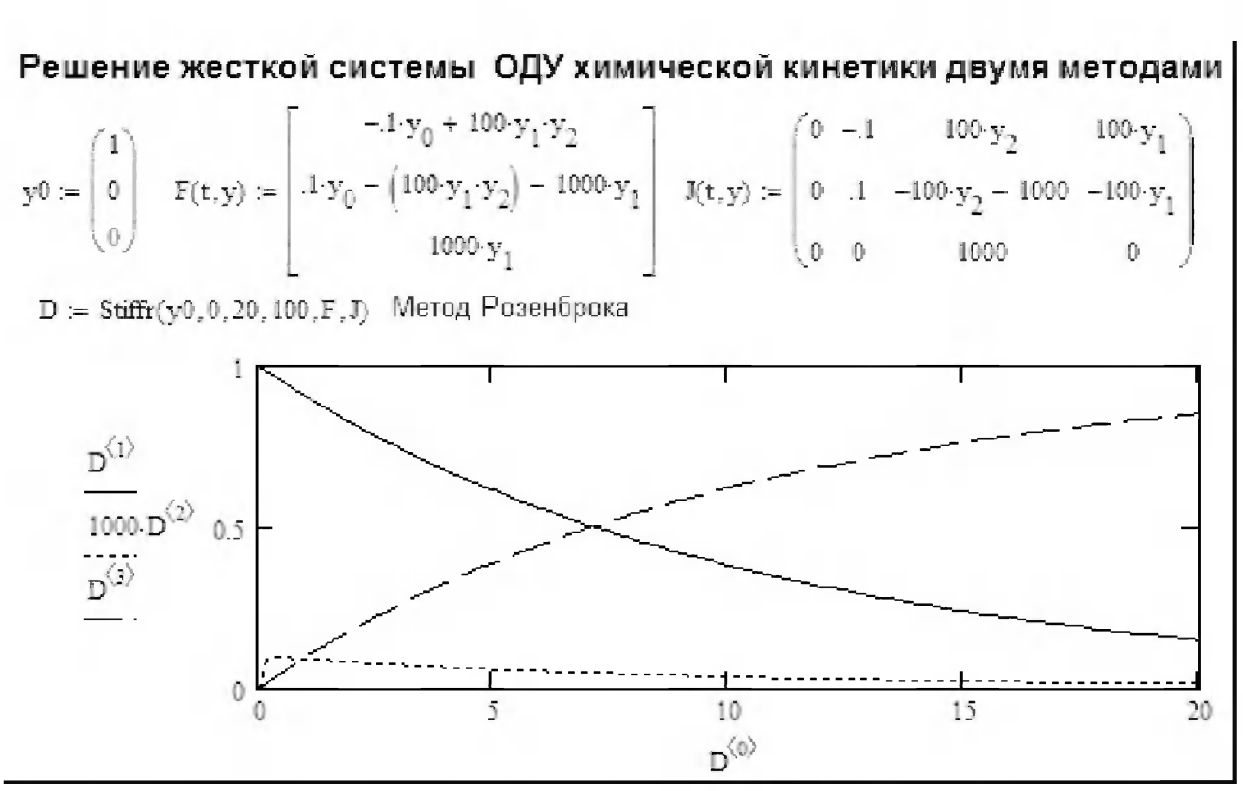


Рис. 9.38. Пример решения жесткой системы дифференциальных уравнений химической кинетики

При построении графика решения используется выделение столбцов из матрицы решения D. При этом столбец $D^{<0>}$ дает вектор значений времени *t*, а столбцы $D^{<1>}$, $D^{<2>}$ и $D^{<3>}$ – векторы значений переменных (концентрации компонентов смеси).

9.12.5. Функция Radau

Безусловно, необходимость вычисления матрицы Якоби (см. примеры на рис. 9.37 и 9.38) не всегда воспринимается с восторгом. В Mathcad 2001i/11 введена новая функция для решения жестких систем дифференциальных уравнений `Radau(y,x1,x2,n,F)`, которая реализует новый метод RADAU5, предложенный в 1996 г. и не требующий подготовки матрицы Якоби. Относящиеся к этой функции параметры уже обсуждались. Естественно, эта функция присутствует и в Mathcad 12.

Воспользовавшись примером на рис. 9.38, дополним его примером решения системы дифференциальных уравнений химической кинетики с помощью новой функции `Radau`. Пример этого представлен на рис. 9.39 (это конец документа, представленного на рис. 9.38).

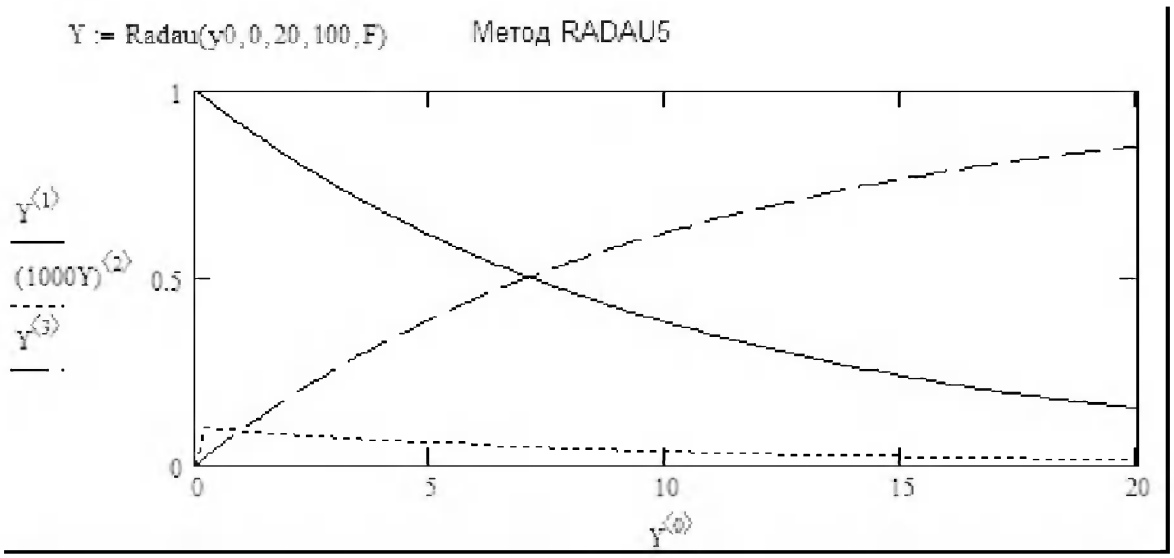


Рис. 9.39. Пример решения жесткой системы дифференциальных уравнений химической кинетики с помощью функции Radau

К сожалению, с применением функции `Radau` не все обстоит благополучно. При расширении диапазона времени шаг решения может превысить допустимую величину, и решение не будет получено. К примеру, заменив конечное время $t = 20$ на $t = 50$, можно наблюдать такую ситуацию. При этом выражение с функцией окрашивается в красный цвет и появляется всплывающее сообщение «Can't converge a solution. Encountered too many integrator steps.».

Еще один вариант этой функции `radau(y,x1,x2,acc,F,k,s)` служит для получения решения в конечной точке с заданной погрешностью `acc`. Параметры `k` и `s` задают максимальное число промежуточных точек, на которых ищется решение, и минимально допустимый интервал между ними.

9.12.6. Решение двухточечных краевых задач

Рассмотренные до сих пор методы численного решения дифференциальных уравнений исходили из задания начальных условий в одной точке – слева. Однако нередко надо найти такое решение дифференциального уравнения, при котором оно завершается также в заданной точке. Наглядным примером этого является решение задачи о попадании снаряда, выпущенного из пушки, точно в цель. При стрельбе полет снаряда, выпущенного из орудия с заданной скоростью, описывается системой дифференциальных уравнений. Неизвестным является угол, под которым надо выстрелить снаряд, чтобы он попал в цель. Задачи такого рода называются двухточечными краевыми задачами, а один из методов их решения именуется методом стрельбы, или пристрелки (shooting method).

Mathcad позволяет решать задачи данного типа, у которых часть начальных условий задана в начальной точке интервала решения, а остальная часть – в его конечной точке. Возможно также решение задач с граничными условиями в некоторой точке в середине интервала решения. Некоторые задачи имеют решения для определенных значений некоторого числового параметра – это задачи на собственные значения.

Для решения двухточечных краевых задач в Mathcad предназначены следующие функции:

- `bvalfit (v1,v2,x1,x2,xf,D,load1,load2,score)` – возвращает вектор недостающих начальных условий для краевой задачи, заданной в векторах `D`, `v1` и `v2` на интервале от `x1` до `x2`, где решение известно в некоторой промежуточной точке `xf`;
- `sbval(y,x1,x2,D,load,score)` – возвращает вектор недостающих `L` начальных условий на левой границе интервала решений для краевой задачи, определенной в символьном векторе `D`, вектор `y` определяет начальные условия на интервале `[x1,x2]`, прочие параметры определены ниже.

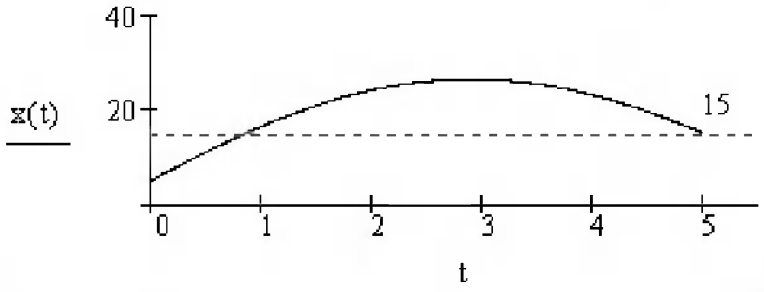
В этих функциях векторы `v`, `v1`, `v2` задают начальные условия, а `x`, `x1`, `x2` – граничные точки интервала решений. `D(x,y)` – функция, возвращающая N-компонентный вектор с первыми производными неизвестных функций. `load(x1,v)`, `load1(x1,v1)` и `load2(x2,v2)` – векторозначные функции, возвращающие значения начальных условий в точках `x1` (`x2`). `score(xf,y)` – векторозначная функция, возвращающая `n`-элементный вектор соответствия. Он указывает, насколько значения решений, начинающихся из точек `x1` и `x2`, должны соответствовать `xf`. Например, если нужно совпадение решений, то `score(xf,y) := y`.

Число элементов векторов `D` и `load` равно количеству уравнений `N`, а векторов `z` и результата выполнения функции `sbval` – количеству правых граничных условий `L`. Соответственно, число левых граничных условий должно быть (`N-L`).

Фрагмент документа Mathcad с двумя примерами решения краевой задачи показан на рис. 9.40. Оба примера достаточно очевидны и не нуждаются в особом

Решение ОДУ с двумя краевыми условиями:

Given $4\cdot\frac{d^2}{dt^2}x(t)+x(t)=t$ $\quad x(0)=5 \quad x(5)=15 \quad z:=\text{Odesolve}(t,5)$



Другой способ:

$g_0:=1 \quad t0:=0 \quad t1:=5 \quad D(t,X):=\begin{pmatrix} X_1 \\ \frac{t-X_0}{4} \end{pmatrix} \quad load(t,v):=\begin{pmatrix} 5 \\ v_0 \end{pmatrix}$

$score(t,w):=w_0-15 \quad IC:=sbval(g,t0,t1,D,load,score) \quad IC=(12.701)$

$ic:=load(0,IC) \quad S:=rkfixed(ic,t0,t1,1500,D) \quad T:=S^{\langle 0\rangle} \quad X:=S^{\langle 1\rangle}$

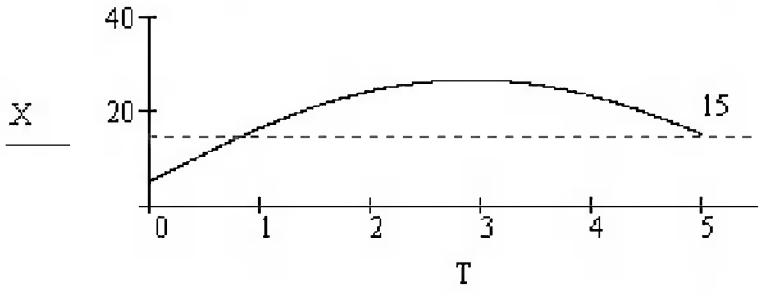


Рис. 9.40. Пример решения двухточечной краевой задачи

описании, тем более что представленные графические зависимости хорошо проясняют суть решаемых задач.

В приведенных примерах первые производные функций решения непрерывны. В этом случае можно применять функции `Odesolve` и `sbval`. Если это не так, то можно применять функцию `bvalfit`. Ограничимся приведенным на рис. 9.41 примером.

$$F(x,y):=\begin{bmatrix} y_1 \\ (x<0)\cdot y_0+(x\ge 0)\cdot -y_0 \end{bmatrix}$$

$v_0^1:=1 \quad v_0^2:=1 \quad xf:=0$

$$load1(x1,v1):=\begin{pmatrix} 1 \\ v_0^1 \end{pmatrix} \quad load2(x2,v2):=\begin{pmatrix} 2 \\ v_0^2 \end{pmatrix}$$

$score(xf,y):=y \quad S:=bvalfit(v1,v2,-1,1,0,F,load1,load2,score)$

$S=(0.092 \quad -0.678)$

$$F(x,y) := \begin{bmatrix} y_1 \\ (x < 0)\cdot y_0 + (x \geq 0)\cdot -y_0 \end{bmatrix}$$

$v1_0 := 1 \qquad v2_0 := 1 \qquad xf := 0$

$$load1(x1,v1) := \begin{pmatrix} 1 \\ v1_0 \end{pmatrix} \qquad load2(x2,v2) := \begin{pmatrix} 2 \\ v2_0 \end{pmatrix}$$

$score(xf,y) := y \qquad S := bvalfit(v1,v2,-1,1,0,F,load1,load2,score)$

$S = (0.092 \quad -0.678)$

Рис. 9.41. Пример применения функции bvalfit

Ряд примеров решения дифференциальных уравнений с описанными функциями имеется в самоучителе по разделу решения дифференциальных уравнений и в «быстрых шпаргалках» (QuickSheets) Центра ресурсов. Среди них есть интересные задачи на собственные значения. Для решения жестких систем описанные в этом разделе функции непригодны.

9.12.7. Решение дифференциальных уравнений Пуассона и Лапласа

Рассмотрим дифференциальные уравнения Пуассона (в частных производных второго порядка):

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \rho(x,y).$$

Для их решения, являющегося гомогенной формой уравнений Лапласа, в систему введены следующие функции:

- `multigrid(M,n)` – возвращает матрицу решения уравнения Пуассона, у которого решение равно нулю на границах;
- `relax(M1,M2,M3,M4,M5,A,U,r)` – возвращает квадратную матрицу решения уравнения Пуассона для спектрального радиуса `r`.

Фрагмент документа Mathcad с примерами задания и применения этих функций дан на рис. 9.42.

Эти функции предназначены для решения эллиптических уравнений.

9.12.8. Функции для решения ОДУ в частных производных

Иногда возникает необходимость в решении дифференциальных уравнений в частных производных. Классический пример решения такой задачи – вычисление ко-

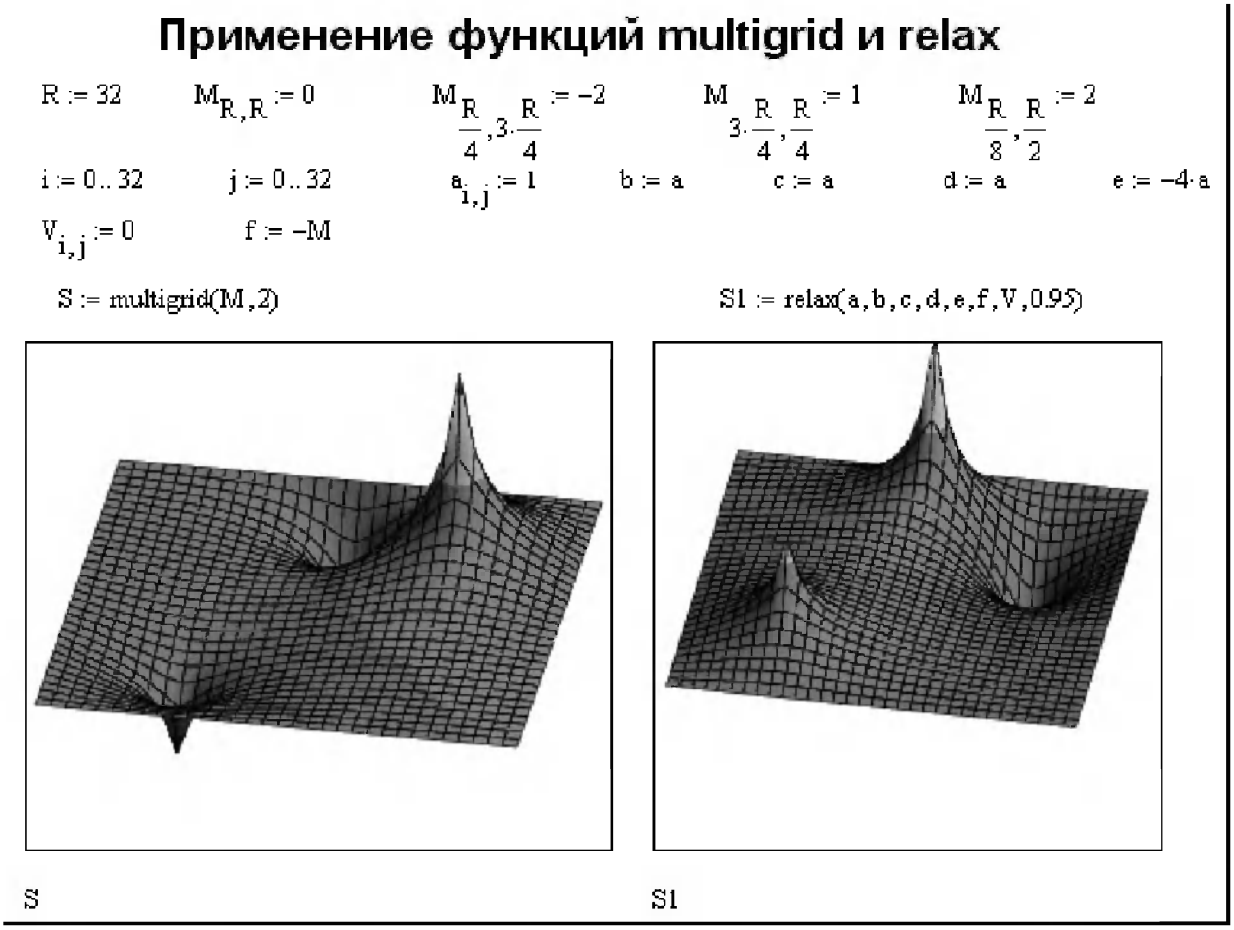


Рис. 9.42. Примеры использования функций multigrid и relax

лебаний струны, зажатой с двух сторон, или колебание мембраны, впрессованной в прямоугольное отверстие. Далеко не все системы компьютерной математики имеют средства для решения таких задач. Но в версии Mathcad 11 подобные средства впервые появились. Они реализованы двумя новыми функциями, предназначенными для решения гиперболических и параболических дифференциальных уравнений в частных производных.

Функция

`Pdesolve(u, x, xrange, t, trange [,xpts] [, tpts])`

возвращает функцию или вектор функций и `t` для решения систем дифференциальных уравнений как без ограничений, так и с ограничениями, заданными в виде алгебраических уравнений. В этой функции используются следующие параметры: `u` – вектор из имен заданных функций, `x` – пространственная переменная, `xrange` – двухэлементный вектор, задающий граничные значения переменной `x`, `t` – время, `trange` – двухэлементный вектор, задающий действительные граничные значения времени, `xpts` и `tpts` – целые числа, задающие дискретность изменения `x` и `t`.

Другая функция

`numol(x_endpts, xpts, t_endpts, tpts, num_pde, num_pae, pde_func, pinit, bc_func)`

возвращает матрицу решений размером `xpts` на `tpts` для одномерного дифференциального уравнения в частных производных. Здесь `x_endpts` и `t_endpts` –

двухэлементные векторы-столбцы, определяющие область интегрирования, `xpts` и `tpts` – целые числа, определяющие число точек в области интегрирования, `num_pde` и `num_pae` – целые числа, определяющие число дифференциальных уравнений и алгебраических уравнений, `pde_func` – вектор, задающий функцию от переменных `x`, `t`, `u`, `ux` и `uxx` с длиной (`num_pde` + `num_pae`), `pinit` – вектор начальных условий с длиной (`num_pde` + `num_pae`), а `bc_func` – матрица размера `num_pde`×3, строки которой задаются в форме:

- `(bc_left(t) bc_right(t) 'D')` – для граничных условий Дирихле;
- `(bc_left(t) bc_right(t) 'N')` – для граничных условий Неймана.

9.12.9. Анализ колебаний струны в одномерном случае

Рассмотрим реализацию классической задачи колебаний струны в плоскости (одномерный случай), если струна наглухо закреплена с обеих сторон. Волновое уравнение колебания струны и его решение с помощью функции `Pdesolve` представлены на рис. 9.43.

На этом рисунке показан также график решения данного уравнения при заданных параметрах.

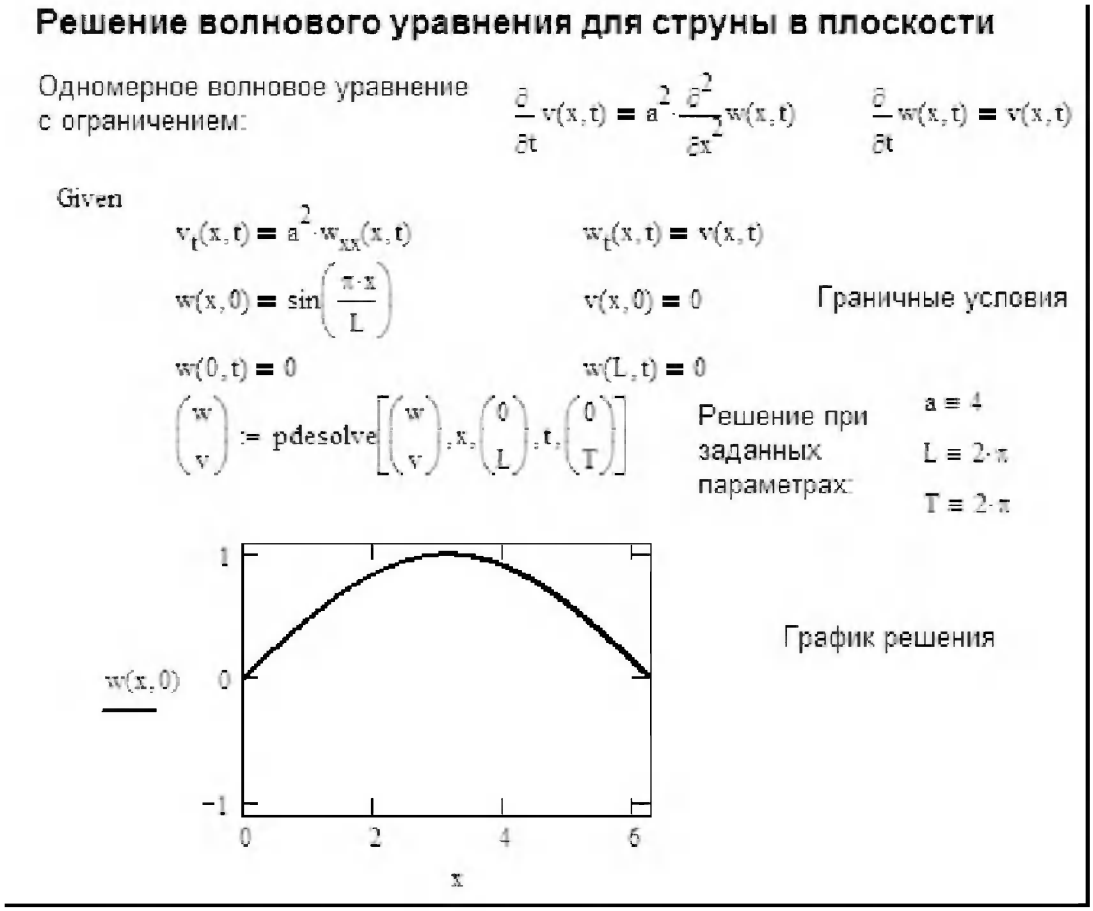


Рис. 9.43. Решение задачи на колебание струны

9.12.10. Анализ колебаний поверхности

Несколько более сложной является задача о колебаниях упругой поверхности (мембраны), размещенной в отверстии квадратной формы массивной плиты. Решение этой задачи с помощью функции командной строки numol представлено на рис. 9.44.

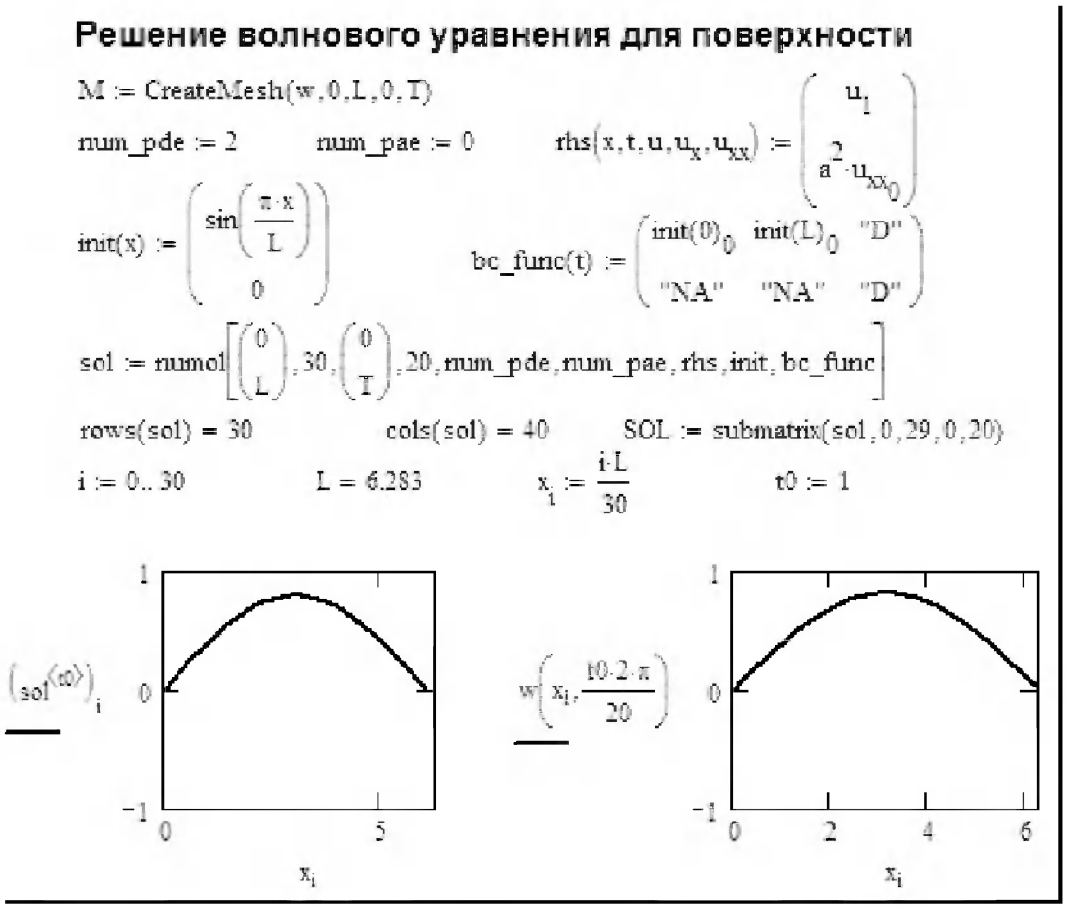


Рис. 9.44. Решение задачи на колебание поверхности

Представляет интерес форма разреза колеблющейся поверхности. Для некоторых заданных параметров она представлена рисунками. Разрезы выделяются формированием субматрицы из общей матрицы решения. А на рис. 9.45 показано сравнение поверхности, построенной с помощью функции CreateMesh (она дает идеализированное решение), и поверхности, полученной решением с помощью функции `numol`.

9.12.11. Анимация колебания поверхности

Наглядное представление о колебаниях поверхности дает анимация – представление об изменении во времени разреза поверхности. Для этого достаточно представить график разреза с выборкой его столбцов с помощью целочисленной переменной FRAME. На рис. 9.46 показана предварительная подготовка к созданию

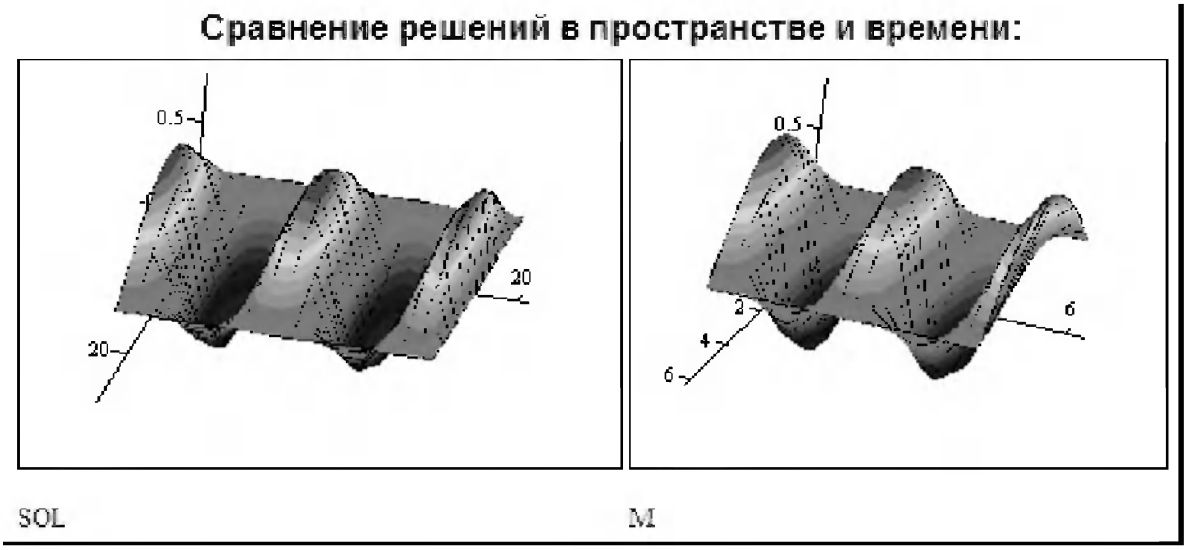


Рис. 9.45. Сравнение решений задачи на колебание поверхности

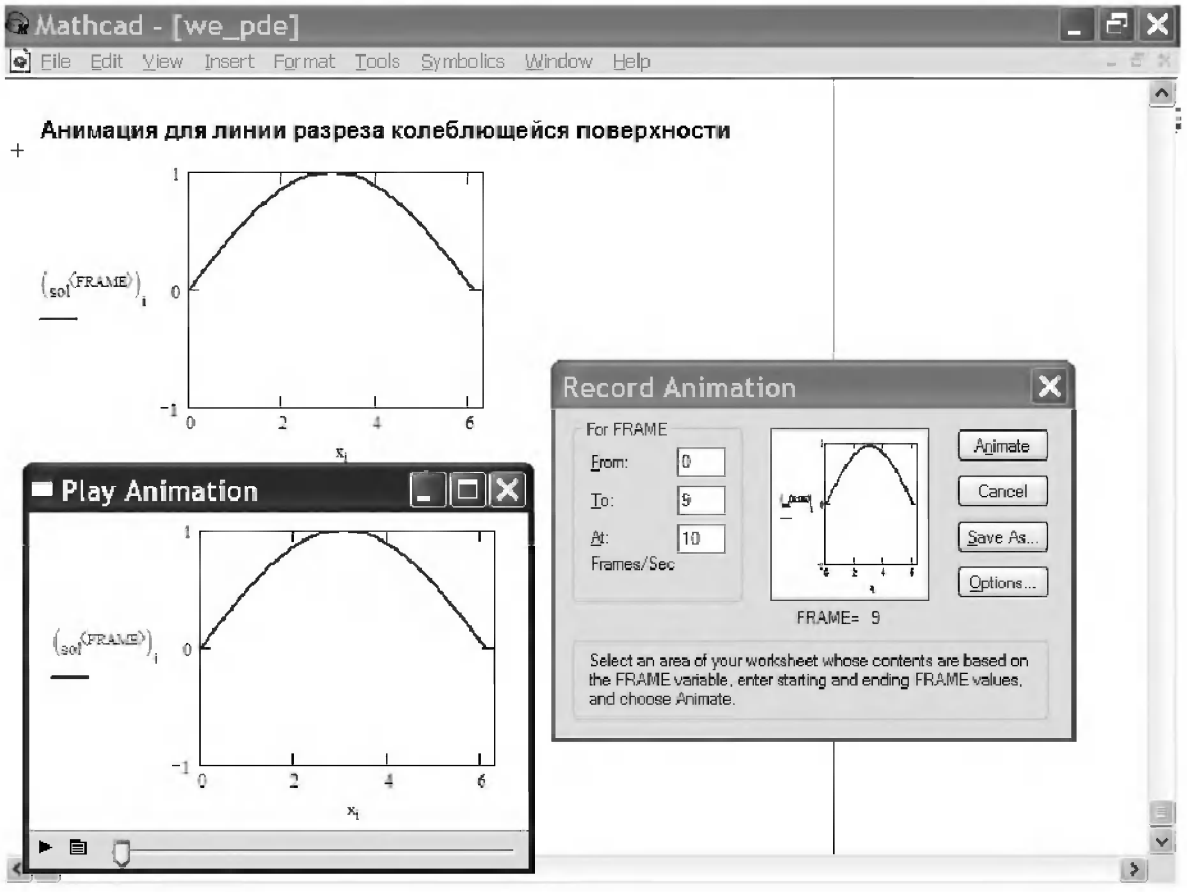


Рис. 9.46. Подготовка к анимации графика, описывающего колебание разреза поверхности

анимационного клипа – задан шаблон графика и выполнена команда **Record...** в подменю **Animation** позиции **Tools** меню Mathcad.

После выполнения указанных команд появляется окно **Record Animation**. В нем надо задать диапазон изменения переменной Frame и скорость ее изменения. Затем надо выделить график, анимацию которого мы решили наблюдать,

и нажать кнопку **Animation**. В окне будут последовательно видны кадры анимации, после чего появится проигрыватель анимационных клипов – он показан на рис. 9.46 под графиком. В окне проигрывателя виден первый кадр анимации. Остальные кадры можно последовательно просмотреть, нажав кнопку пуска (черный треугольник) проигрывателя.

На рис. 9.47 показан некоторый промежуточный кадр анимации выделенного графика – с малым прогибом поверхности по сравнению с показанным в окне проигрывателя на рис. 9.46.

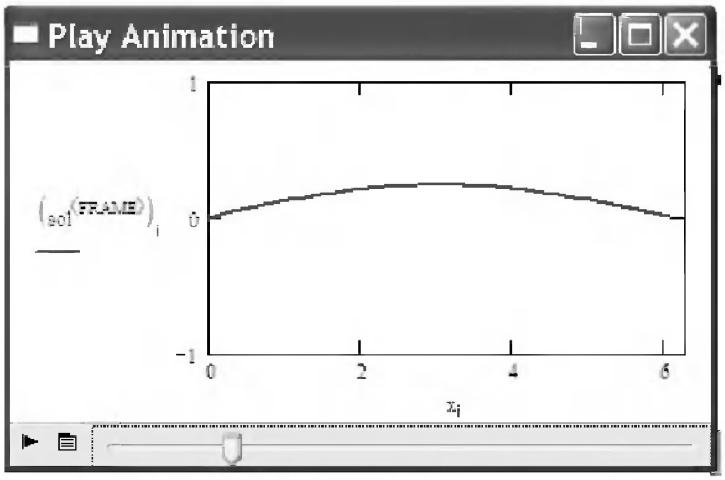


Рис. 9.47. Промежуточный кадр анимации с небольшим прогибом поверхности вверх

Наконец, на рис. 9.48 показан еще один кадр анимации с прогибом поверхности вниз. Заметим, что «вручную» наблюдать кадры анимации можно, просто перемещая ползунок проигрывателя слева направо.

Заинтересованный читатель может попытаться создать анимацию всей поверхности в целом. Это даст более красочную картину, но для детального анализа процесса колебаний наблюдать колебания точек поверхности ее разреза более удобно.

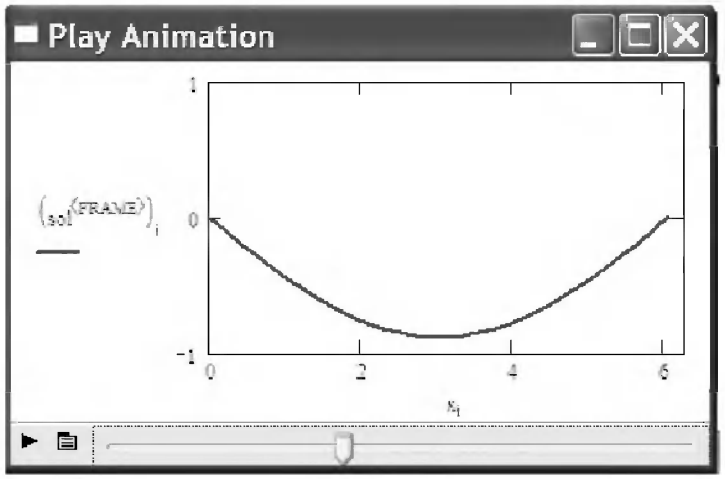


Рис. 9.48. Кадр анимации при прогибе поверхности вниз

9.12.12. Решение дифференциальных уравнений с комплексными параметрами

Изредка может понадобиться решение дифференциальных уравнений с комплексными параметрами, включая воздействие, записанное в комплексной форме. Фрагмент документа Mathcad с примером решения такой задачи представлен на рис. 9.49.

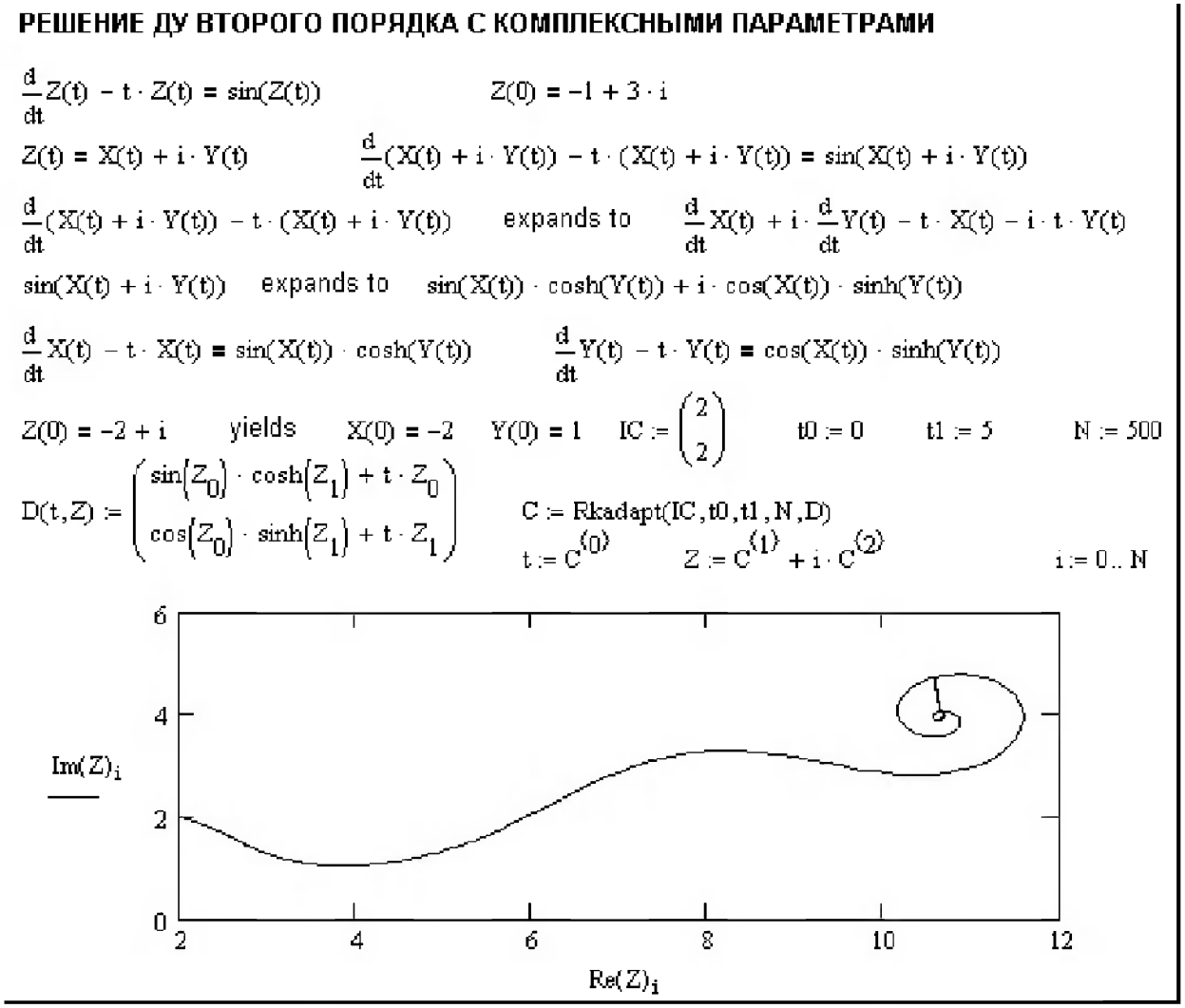


Рис. 9.49. Пример решения ДУ с комплексными параметрами

Обратите внимание на то, что в данном случае для решения использован адаптивный метод Рунге-Кутта, реализованный функцией `Rkadapt`.

9.13. Символьное решение ДУ в системе Mathcad

9.13.1. Применение преобразования Лапласа для решения ДУ

Для получения решения ДУ в аналитическом виде можно воспользоваться преобразованиями Лапласа, что и иллюстрирует фрагмент документа Mathcad, представленный на рис. 9.50. На этом рисунке подробно показан процесс получения результата. Приходится «вручную» инициировать прямое преобразование Лапласа, по его результатам составлять алгебраическое уравнение и после решения инициировать обратное преобразование Лапласа – оно дает решение в виде временной зависимости.

ПРИМЕР РЕШЕНИЯ В СИМВОЛЬНОМ ВИДЕ ДИФФЕРЕНЦИАЛЬНОГО УРАВНЕНИЯ ВТОРОГО ПОРЯДКА

Вводим левую часть уравнения $m \cdot \frac{d^2}{dt^2} y(t) - k \cdot \frac{d}{dt} y(t)$

Отметив переменную t, выполним прямое преобразование Лапласа:

$$m \cdot \left[s \cdot (s \cdot \text{laplace}(y(t), t, s) - y(0)) - \left| \begin{matrix} t \leftarrow 0 \\ \frac{d}{dt} y(t) \end{matrix} \right| \right] - k \cdot (s \cdot \text{laplace}(y(t), t, s) - y(0))$$

Введя обозначения L=laplace(y(t),t,s), C1=y(0) и C2=diff(y(0),0), имеем:

$$m \cdot L \cdot s^2 - m \cdot s \cdot C1 - m \cdot C2 - k \cdot L \cdot s + k \cdot C1 = 0$$

Решая это уравнение относительно переменной L, получим:

$$\frac{(k \cdot C1 - m \cdot s \cdot C1 - m \cdot C2)}{[s \cdot (k - m \cdot s)]}$$

Отметив переменную s и произведя обратное преобразование Лапласа, найдем временную зависимость - решение заданного уравнения:

$$\frac{-1}{k} \cdot m \cdot C2 + C1 + m \cdot \frac{C2}{k} \cdot \exp\left(k \cdot \frac{t}{m}\right)$$

Рис. 9.50. Пример решения дифференциального уравнения второго порядка с применением преобразований Лапласа

Данный пример наглядно показывает, что помещаемый в буфер обмена результат символьных операций может быть очень полезным и его порой невозможно получить прямым образом. Это значительно расширяет возможности применения системы Mathcad, особенно при решении аналитических задач. Наилучшие возможности в решении задач в символьном виде дает версия Mathcad 11. Напоминаем, что в новейшей Mathcad 14 применено ядро символьной математики **MuPAD**.

9.13.2. Решение задачи Коши для линейного неоднородного ДУ

Итак, мы рассмотрели характерный пример аналитического решения линейного ДУ второго порядка с применением преобразования Лапласа, осуществляемого командой из меню символьных операций. Однако область применения ДУ настолько широка и важна, что их решение заслуживает отдельного рассмотрения. Достаточно отметить, что решение ДУ и систем с ними лежит в основе моделирования различных динамических объектов, систем и физических явлений.

Для начала рассмотрим задачу на получение аналитического решения линейного ДУ первого порядка. Один из простейших способов заключается в применении известных общих формул для такого решения, получаемых интегрированием ДУ. Фрагмент документа Mathcad, представленный на рис. 9.51, дает пример подобного решения.

Обратите внимание на технику применения директивы `simplify` для проверки правильности решения путем его подстановки в исходное уравнение. Конт-

Решение задачи Коши для линейного неоднородного ДУ первого порядка

$y' + 2 \cdot x \cdot y = x \cdot \exp\left(-x^2\right) \cdot \sin(x) \qquad y_0 := 1$

Вычислим решение по готовой формуле для задачи Коши y' = a(x)y +b(x), y(xo)=yo:

$$y(x) := y_0 \cdot \exp\left(\int_0^x -2 \cdot t \, dt\right) + \int_0^x \exp\left(\int_z^x -2 \cdot t \, dt\right) \cdot z \cdot \exp\left(-z^2\right) \cdot \sin(z) \, dz$$

$$y(x) \rightarrow \exp\left(-x^2\right) + (\sin(x) - x \cdot \cos(x)) \cdot \exp\left(-x^2\right)$$

Проверим правильность результата.

$\frac{d}{dx} y(x) + 2 \cdot x \cdot y(x) - x \cdot \exp\left(-x^2\right) \cdot \sin(x)$ simplify $\rightarrow 0 \qquad y(0) \rightarrow 1$

После подстановки уравнение обратилось в тождество. Начальное условие выполняется. Решение задачи Коши вычислено верно.

Рис. 9.51. Пример решения задачи Коши для линейного неоднородного уравнения первого порядка

роль решения – важный аспект аналитических вычислений, поскольку целиком полагаться на них рискованно.

9.13.3. Общее решение неоднородного ДУ первого порядка

Другой пример, на этот раз на получение общего решения ДУ первого порядка, дает фрагмент документа Mathcad, представленный на рис. 9.52.

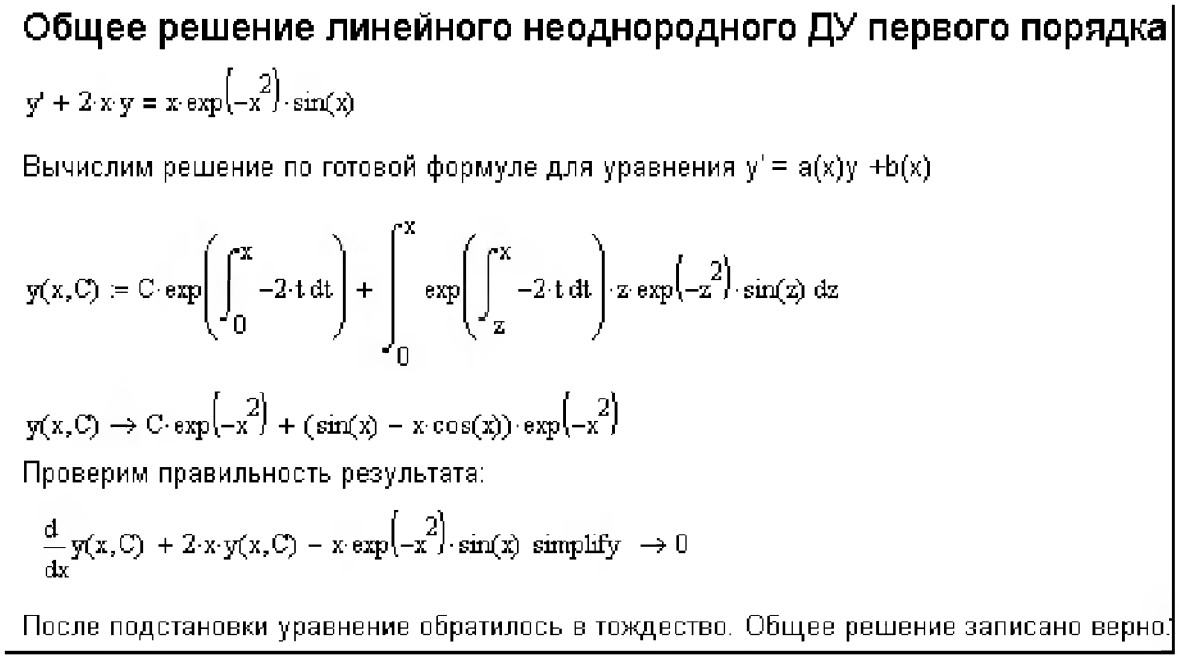


Рис. 9.52. Пример нахождения общего решения ДУ первого порядка

Обратите внимание: поскольку в данном примере начальные условия не заданы, в решении фигурирует произвольная постоянная `C`. Кроме того, как и в случае решения задачи Коши для линейного неоднородного ДУ, для проверки правильности решения используется директива `simplify` и выполняется подстановка полученного решения в исходное уравнение.

9.13.4. Нахождение всех решений ДУ первого порядка

Фрагмент документа Mathcad, представленный на рис. 9.53, дает пример нахождения всех решений линейного ДУ первого порядка с разделяющимися переменными. Решение опять-таки ищется с применением общей формулы решения, записанной в левой части функции пользователя `F(x, y)`.

Обратите внимание на проверку правильности решения (в нижней части рисунка).

Нахождение всех решений уравнения $(1+x^2)\cdot y' = 2\cdot x\cdot\sqrt{1-y^2}$

Запишем уравнение в нормальной форме: $y' = \frac{2\cdot x\cdot\sqrt{1-y^2}}{(1+x^2)}$

Получили уравнение с разделяющимися переменными.
Разделим переменные: $\frac{dy}{\sqrt{1-y^2}} = \frac{2\cdot x}{1+x^2}\cdot dx$

Запишем и вычислим выражение для общего интеграла этого уравнения, удовлетворяющего заданному начальному условию:

$$F(x,y) := \int \frac{1}{\sqrt{1-y^2}}\,dy - \int \frac{2\cdot x}{1+x^2}\,dx \qquad F(x,y) \to \operatorname{asin}(y) - \ln(1+x^2)$$

Общий интеграл уравнения записывается в виде F(x,y) = C: $\operatorname{asin}(y) - \ln(1+x^2) = C$

Однако общий интеграл описывает не все решения уравнения, а только те для которых y отличен от 1 и от -1.
Все решения уравнения описываются равенствами:

$\operatorname{asin}(y) - \ln(1+x^2) = C$ и $y = 1$ $y = -1$

Проверим правильность результата.Выражение F(x,y) = C задает решение уравнения y=y(x) как функцию переменной x в неявной форме. Для проверки решения вычислим производную y'(x) по формулам дифференцирования неявной функции и подставим ее в уравнение:

$$-\frac{\frac{d}{dx}F(x,y)}{\frac{d}{dy}F(x,y)}\cdot(1+x^2) - 2\cdot x\cdot\sqrt{1-y^2}\ \text{simplify} \to 0$$

После подстановки уравнение обратилось в тождество. Общий интеграл записан верно.

Рис. 9.53. Пример нахождения общего решения ДУ первого порядка

9.13.5. Решение задачи Коши для ДУ в полных дифференциалах

Теперь рассмотрим более сложный случай – решение задачи Коши для ДУ в полных дифференциалах. В этом случае в уравнении фигурируют производные по ряду переменных – в данном примере (рис. 9.54) это производные по переменным x и y.

Как и при решении других ДУ, после нахождения решения требуется проверить его правильность.

9.13.6. Нахождение частного решения ДУ третьего порядка

При моделировании сложных систем часто приходится сталкиваться с необходимостью решения ДУ высокого порядка. Рассмотрим пример, показанный на рис. 9.55. Решается ДУ третьего порядка.

Решение задачи Коши:

$$\left(1+\frac{1}{y}\cdot\exp\left(\frac{x}{y}\right)\right)\cdot dx+\left(1-\frac{x}{y^2}\cdot\exp\left(\frac{x}{y}\right)\right)\cdot dy=0 \qquad y(0)=1$$

Обозначим $P(x,y):=1+\frac{1}{y}\cdot\exp\left(\frac{x}{y}\right)$ $\qquad Q(x,y):=1-\frac{x}{y^2}\cdot\exp\left(\frac{x}{y}\right)$

Покажем, что уравнение является уравнением в полных дифференциалах:

$$\frac{d}{dx}(Q(x,y))-\frac{d}{dy}(P(x,y))\ \text{simplify}\ \rightarrow 0 \qquad +$$

Найдем частный интеграл уравнения

$$u(x,y):=\int_0^x P(t,y)\,dt+\int_1^y\left(Q(x,t)-\frac{d}{dt}\int_0^x P(z,t)\,dz\right)dt \qquad u(x,y)\rightarrow x+\exp\left(\frac{x}{y}\right)-2+y$$

Проверим правильность решения: покажем, что du(x,y)=P(x,y)dx+Q(x,y)dy:

$$du(x,y,dx,dy):=\left(\frac{d}{dx}u(x,y)\right)\cdot dx+\left(\frac{d}{dy}u(x,y)\right)\cdot dy$$

$$du(x,y,dx,dy)-[(P(x,y))\cdot dx+Q(x,y)\cdot dy]\ \text{simplify}\ \rightarrow 0$$

Проверим начальное условие $\qquad u(0,1)\rightarrow 0$

Частный интеграл уравнения имеет вид: $\qquad x+\exp\left(\frac{x}{y}\right)-2+y=0$

Рис. 9.54. Пример решения задачи Коши для ДУ в полных дифференциалах

В данном случае решение базируется на анализе характеристического уравнения для заданного ДУ. Решение достаточно подробно прокомментировано на рисунке и может рассматриваться как довольно характерное для этого метода решения.

9.13.7. Фундаментальная система уравнений и общее решение неоднородного ДУ четвертого порядка

Завершим рассмотрение аналитических методов решения линейных ДУ примером нахождения фундаментальной системы уравнений и общего решения неоднородного линейного ДУ четвертого порядка. Фрагмент документа Mathcad, иллюстрирующий решение этой задачи, представлен на рис. 9.56.

Приведенные примеры свидетельствуют о широких возможностях символьных вычислений, которые находят все возрастающее применение в математических и научно-технических расчетах. И о возможностях их реализации даже в СКМ Mathcad, изначально ориентированной на численные вычисления.

Частное решение неоднородного линейного дифференциального уравнения

$y''' + 3y'' - 4y = 1 - x^2$ Соответствующее однородное уравнение $y''' + 3y'' - 4y = 0$

Запишем характеристический многочлен уравнения и найдем его корни

$P(\lambda) := \lambda^3 + 3\cdot\lambda^2 - 4$ Given $P(\lambda) = 0$ $\text{Find}(\lambda) \rightarrow (1 \quad -2 \quad -2)$

Характеристическое уравнение имеет три корня λ1 = 1, λ2 = λ3 = - 2. Правая часть уравнения - многочлен второй степени. Среди корней характеристического уравнения нет нуля - резонанса нет. Будем искать частное решение уравнения в виде многочлена второй степени:

$yr(x,A,B,C) := A\cdot x^2 + B\cdot x + C$ Подставим частное решение в левую часть уравнения и упростим:

$$\frac{d^3}{dx^3}yr(x,A,B,C) + 3\cdot\frac{d^2}{dx^2}yr(x,A,B,C) - 4\cdot yr(x,A,B,C) \text{ simplify } \rightarrow 6\cdot A - 4\cdot A\cdot x^2 - 4\cdot B\cdot x - 4\cdot C$$

Вычислим неизвестные коэффициенты A, B и C:

Given $6\cdot A - 4\cdot C = 1$ $-4\cdot B = 0$ $-4\cdot A = -1$ $\text{Find}(A,B,C) \rightarrow \begin{pmatrix} \frac{1}{4} \\ 0 \\ \frac{1}{8} \end{pmatrix}$

Определим искомое частное решение y(x):

$y(x) := yr\left(x, \frac{1}{4}, 0, \frac{1}{8}\right)$ $y(x) \rightarrow \frac{1}{4}\cdot x^2 + \frac{1}{8}$

Покажем, что y(x) - решение уравнения. Подставим y(x) в уравнение и упростим:

$$\frac{d^3}{dx^3}y(x) + 3\cdot\frac{d^2}{dx^2}y(x) - 4\cdot y(x) \text{ simplify } \rightarrow 1 - x^2$$

Подстановка y(x) обращает уравнение в тождество.Решение найдено верно.

Рис. 9.55. Пример нахождения частного решения для ДУ третьего порядка

Фундаментальная система решений и общее решение однородного линейного ДУ
y''''-4y'''+14y''-20y' + 25y =0

Запишем характеристический многочлен уравнения и найдем его корни:

$P(\lambda) := \lambda^4 - 4\cdot\lambda^3 + 14\cdot\lambda^2 - 20\cdot\lambda + 25$ Given $P(\lambda) = 0$ $\text{Find}(\lambda) \rightarrow (1 + 2\cdot i \quad 1 - 2\cdot i \quad 1 + 2\cdot i \quad 1 - 2\cdot i)$

Характеристическое уравнение имеет две пары комплексных корней λ=1+2i и λ=1-2i. Запишем фундаментальную систему решений:

$y1(x) := \exp(\text{Re}(1 + 2\cdot i)\cdot x)\cdot\cos(\text{Im}(1 + 2\cdot i)\cdot x)$ $y2(x) := x\cdot y1(x)$ $y3(x) := \exp(\text{Re}(1 - 2\cdot i)\cdot x)\cdot\cos(\text{Im}(1 - 2\cdot i)\cdot x)$ $y4(x) := x\cdot y3(x)$

$y1(x) \rightarrow \exp(x)\cdot\cos(2\cdot x)$ $y2(x) \rightarrow x\cdot\exp(x)\cdot\cos(2\cdot x)$ $y3(x) \rightarrow \exp(x)\cdot\cos(2\cdot x)$ $y4(x) \rightarrow x\cdot\exp(x)\cdot\cos(2\cdot x)$

Запишем общее решение уравнения

$y(x,c1,c2,c3,c4) := c1\cdot y1(x) + c2\cdot y2(x) + c3\cdot y3(x) + c4\cdot y4(x)$

$y(x,c1,c2,c3,c4) \rightarrow c1\cdot\exp(x)\cdot\cos(2\cdot x) + c2\cdot x\cdot\exp(x)\cdot\cos(2\cdot x) + c3\cdot\exp(x)\cdot\cos(2\cdot x) + c4\cdot x\cdot\exp(x)\cdot\cos(2\cdot x)$

Покажем, что y(x,c1,c2,c3,c4) при любых значениях констант c1, c2, c3 и c4 является решением уравнения y''''-4y'''+14y''-20y' + 25y =0. Подставим y(x,c1,c2,c3,c4) в уравнение и упростим:

$$\left(\begin{array}{l} \frac{d^4}{dx^4}y(x,c1,c2,c3,c4) - 4\cdot\frac{d^3}{dx^3}y(x,c1,c2,c3,c4) \ldots \\ +14\cdot\frac{d^2}{dx^2}y(x,c1,c2,c3,c4) - 20\cdot\frac{d}{dx}y(x,c1,c2,c3,c4) + 25\cdot y(x,c1,c2,c3,c4) \end{array}\right) \text{ simplify } \rightarrow 0$$

Подстановка y(x,c1,c2,c3,c4) обращает уравнение в тождество.Общее решение найдено верно.

Рис. 9.56. Пример общего решения для ДУ четвертого порядка

9.14. О реализации в Mathcad вариационных методов

9.14.1. Особенности решения задач механики вариационными методами

Основные проблемы механики могут решаться наряду с решениями дифференциальных уравнений с помощью *вариационных методов*. Например, положение равновесия механической системы соответствует минимуму ее потенциальной энергии. Поэтому проблема решения краевой задачи для дифференциального уравнения, описывающего механическую систему, эквивалентна проблеме отыскания функции, для которой интеграл, выражающий потенциальную энергию системы, принимает наименьшее значение. Рассмотрим одну из типовых задач реализации вариационных методов (задача предложена доцентом Р. Е. Кристаллинским).

9.14.2. Решение задачи на прогиб струны

Пусть потенциальная энергия струны, закрепленной в точках $x = 0$ и $x = l$, под действием внешней нагрузки $f(x)$ определяется равенством

$$I = \frac{1}{2}\int_0^l \left(\frac{\mu}{2}(y')^2 + f \cdot y \right) dx, \qquad (9.5)$$

где $y(x)$ – уравнение струны, μ – некоторая постоянная, значение которой зависит от материала, из которого изготовлена струна. Таким образом, требуется найти функцию $y(x)$, удовлетворяющую краевым условиям $y(0) = y(l) = 0$, для которой приведенный интеграл принимает наименьшее значение.

К нахождению минимума некоторого интеграла сводится и решение краевых задач для дифференциальных уравнений в частных производных. Так, задача о нахождении решения уравнения Пуассона

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y)$$

в заданной области D, удовлетворяющего на контуре области Г краевому условию $u = \varphi(s)$, сводится к задаче о нахождении минимума интеграла

$$I = \iint_D \left[\left(\frac{\partial u}{\partial x} \right)^2 + \left(\frac{\partial u}{\partial y} \right)^2 + 2f(x,y)u \right] dxdy.$$

В 1908 году Вальтер Ритц предложил такой метод приближенного решения рассматриваемого класса задач. Сущность этого метода состоит в следующем. Искомая функция u представляется в виде

$$u = \phi_0 + c_1\phi_1 + \ldots + c_n\phi_n,$$

где функции ϕ_0, ϕ_1, ϕ_n линейно независимы и при всех значениях параметров c_1, c_2, ..., c_n выполняются заданные краевые условия. Эта функция подставляется в интеграл, минимум которого мы находим. После преобразований мы получаем функцию от переменных c_1, c_2, ..., c_n. Затем находится минимум этой функции.

При традиционном подходе задача нахождения минимума сводится к решению системы линейных уравнений. Однако если мы попытаемся увеличить точность решения за счет увеличения n, полученная система становится плохо обусловленной, надежность полученных результатов при этом существенно уменьшается.

9.14.3. Решение задачи на прогиб струны в среде Mathcad

Система Mathcad позволяет непосредственно найти минимум полученной функции, что способствует повышению точности решения рассматриваемой задачи. Интегралы, которые получаются в процессе решения, целесообразно находить сначала в символьном виде.

Рассмотрим в качестве примера решение задачи о нахождении минимума интеграла (9.5). Предположим, что искомая функция обращается в 0 на концах отрезка [0, 1], $\mu = 0.3$, $f(x) = x(1 - x)$. Решение будем искать в виде

$$y = \sum_{i=1}^{5} c_i \sin(i\pi x).$$

Тогда

$$\int_0^1 [y''(x)]^2 dx = \sum_{i=1}^{5}\sum_{j=1}^{5} c_i c_j i^2 j^2 \pi^4 \int_0^1 \sin(i\pi x)\sin(j\pi x)dx = \frac{1}{2}\sum_{i=1}^{5} c_i{}^2 i^4 \pi^4.$$

Положим

$$a_i = \frac{i^4\pi^4}{2}.$$

Далее положим

$$b_i = \int_0^1 x(1-x)\sin(i\pi x)dx = \frac{2}{i^3\pi^3}(1-\cos(\pi i)).$$

Функция, минимум которой нам надлежит определить, будет иметь вид:

$$I(c) = \frac{\mu}{2}\sum_{i=1}^{5} a_i c_i^2 + \sum_{i=1}^{5} b_i c_i.$$

Дальнейшие вычисления производим в системе Mathcad – рис. 9.57.

В результате решения получен профиль струны и найден его минимум.

Рис. 9.57. Решение вариационной задачи в среде системы Mathcad

9.15. Математическое моделирование в Mathcad колебательных систем

Удобное представление документов в системе Mathcad и прекрасный интерфейс пользователя делают Mathcad предпочтительной при решении относительно простых задач. К таковым относятся задачи этого раздела на моделирование колебательных систем, описываемых дифференциальными уравнениями.

9.15.1. Анализ линейной колебательной системы

Известно множество *линейных систем* и *устройств*, создающих почти синусоидальные колебания – самые простые из известных. Это струна музыкальных инструментов, маятник часов, *LCR*-колебательный контур, колеблющаяся молекула вещества и т. д. Все эти устройства и системы при малых амплитудах колебаний можно описать линейным дифференциальным уравнением второго порядка, вид

которого представлен в заголовке рис. 9.58. Там же даны типичные решения этого уравнения с помощью блока Given и функции Odesolve системы Mathcad.

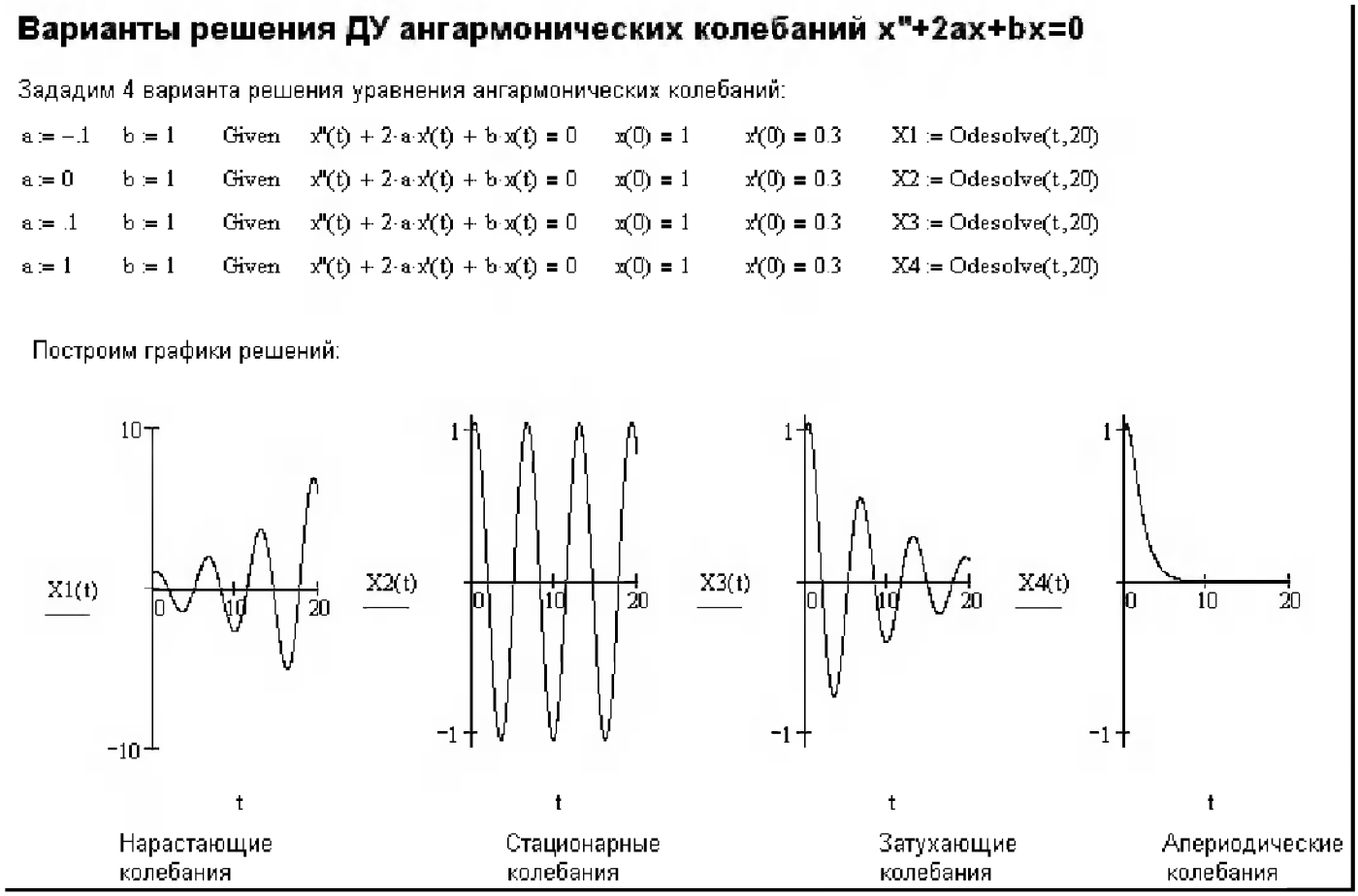


Рис. 9.58. Решения дифференциального уравнения второго порядка, описывающего поведение линейных колебательных систем

Поведение линейной системы сильно зависит от параметра a – затухания. При его отрицательных значениях амплитуда колебаний нарастает по экспоненциальному закону. При $a = 0$ создаются незатухающие синусоидальные колебания. Однако этот процесс нестабилен – малейшее изменение в ту или иную сторону приводит либо к нарастанию колебаний, либо к их затуханию. При больших положительных a (теоретически $a > 0.25$) переходный процесс в системе становится апериодическим. Все эти случаи можно анализировать аналитически, но численный метод решения с помощью функции Odesolve намного проще и нагляднее.

9.15.2. Анализ нелинейной колебательной системы Ван-дер-Поля

А теперь рассмотрим поведение *нелинейной колебательной системы второго порядка*, описываемой нелинейным дифференциальным уравнением второго порядка – *уравнением Ван-дер-Поля*. Рисунок 9.59 показывает документ системы Mathcad, в котором такое уравнение решается при параметре $\mu = 0{,}5$. Этот пара-

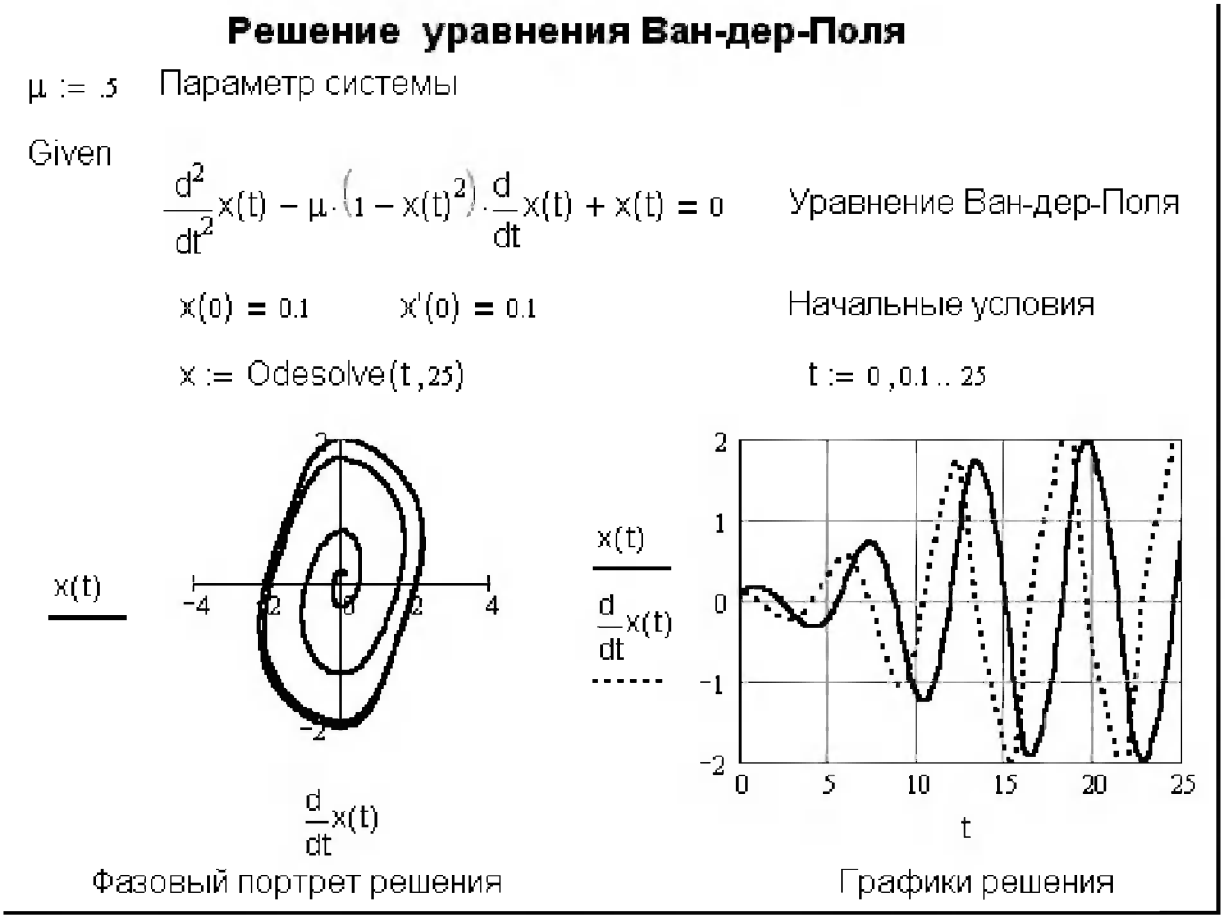


Рис. 9.59. Решение уравнения Ван-дер-Поля

метр задает характер решения, как и начальные условия для $x(t)$ и $\mathrm{d}x(t)/\mathrm{d}t$. При положительных значениях μ колебания в системе нарастают, но вследствие нелинейности системы их амплитуда ограничивается, а форма становится заметно отличной от синусоидальной. Решение выполняется с помощью функции Odesolve.

Системы, колебания в которых возникают без внешних воздействий, принято называть *автономными системами*. Помимо систем класса Ван-дер-Поля, к ним относятся генератор колебаний на туннельном диоде и большинство автогенераторов синусоидальных и релаксационных колебаний. В частности, выполненных на электронных лампах и полевых транзисторах.

9.15.3. Моделирование системы Дафинга с внешним воздействием

Поведение неавтономных нелинейных систем второго порядка, находящихся под внешним воздействием, может быть очень сложным. В этом можно убедиться на примере *системы Дафинга*, описывающей процессы в нелинейных резонаторах с внешним воздействием, – например, в лазерных резонаторах. Пример численного моделирования процессов в такой системе дан на рис. 9.60.

Дифференциальное уравнение Дафинга второго порядка имеет дополнительный кубический член в левой части, а правая часть представляет внешнее косину-

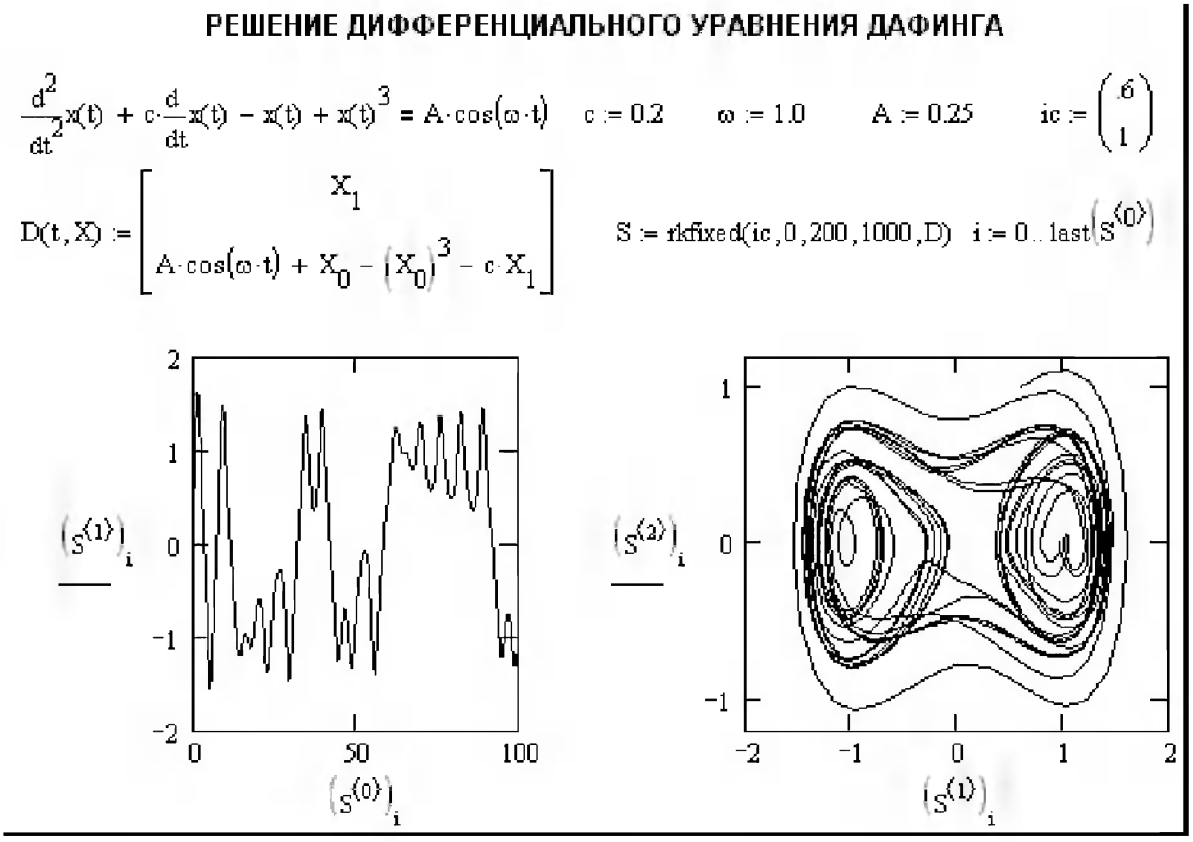


Рис. 9.60. Решение уравнения Дафинга

соидальное воздействие. Форма колебаний такой системы довольно сложна из-за наложения внутренних колебаний на внешние, причем частоты колебаний сильно различаются. В итоге время от времени может наступать автосинхронизация колебаний, но из-за нелинейности системы и изменения амплитуды собственных колебаний может наблюдаться срыв синхронизации, сопровождаемый скачкообразными и довольно хаотическими изменениями параметров системы. Тем не менее фазовый портрет системы имеет два фокуса, соответствующих более низкочастотной компоненте колебаний. Эти фокусы соответствуют статистической оценке наиболее вероятных видов (*мод*) колебаний.

9.15.4. Хаос и моделирование аттрактора Лоренца

Броуновское движение частиц, моделирование которого мы уже провели, и колебания в системе Дафинга являются проявлениями *хаоса* в природе. Наблюдая за изменениями курса акций, сходами ледников и снежных лавин или за колебаниями температуры, мы нередко убеждаемся в том, что наряду с вполне предсказуемыми изменениями того или иного параметра (например, повышения температуры летом и понижения зимой) нередко наблюдаются хаотические изменения, которые трудно или невозможно заранее предвидеть.

Чем сложнее система и чем большим количеством дифференциальных уравнений она описывается, тем больше вероятность возникновения в системе хаотиче-

ских режимов – даже если она автономна. Изучение этого вопроса показало, что уже в системах из трех дифференциальных уравнений возможно возникновение хаотических режимов даже в автономных системах. Наглядным примером этого является *аттрактор Лоренца*. Документ системы Mathcad, дающий пример моделирования аттрактора Лоренца, представлен на рис. 9.61.

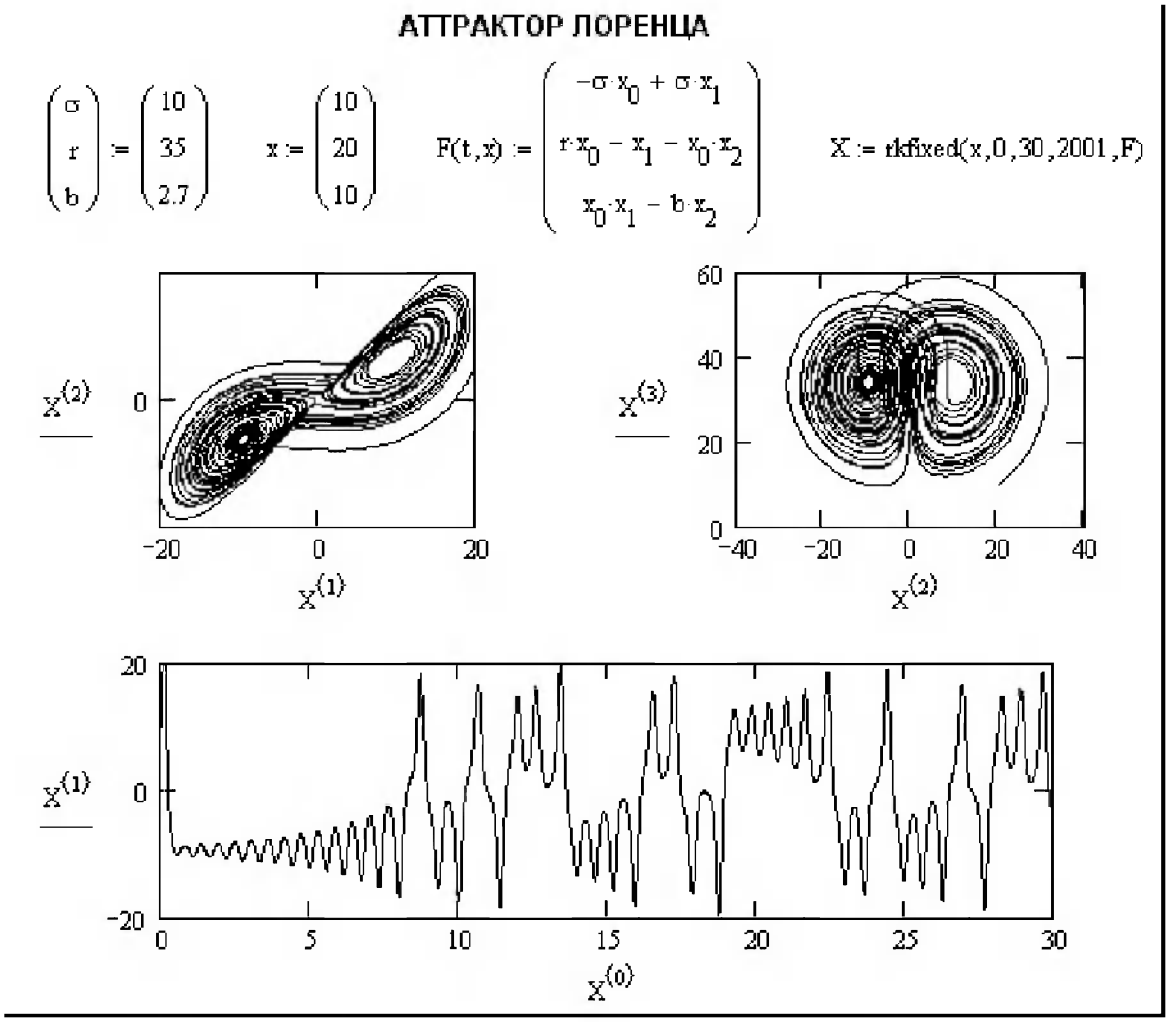


Рис. 9.61. Моделирование аттрактора Лоренца, заданного системой из трех дифференциальных уравнений

Тут аттрактор Лоренца описан уже как система из трех дифференциальных уравнений. При определенных значениях параметров r и b и начальных параметрах переменных поведение аттрактора (он в этом случае называется *странным аттрактором*) очень напоминает хаотические колебания в системе Дафинга.

Аттрактором в теории колебаний называется притягивающая область в фазовом пространстве. Причины неустойчивости аттракторов связаны с экспоненциальной неустойчивостью системы в малых областях фазового пространства. При этом наблюдаются хаотические переходы из одной области фазового пространства в другие, но колебания могут не выходить из некоторой более обширной области фазового пространства. «Обвал» системы означает переход в некоторое состояние, резко отличающееся от других состояний, то есть выход за пределы

ограниченного фазового состояния системы. Такое состояние может оказаться устойчивым и приводит к переходу системы в статическое состояние, при котором изменения ее параметров отсутствуют.

9.15.5. Моделирование математического маятника с анимацией

Но вернемся к системам, поведение которых все же предсказуемо и описывается решением систем дифференциальных уравнений, описывающих состояния системы, – они так и именуются: *уравнениями состояния*. И рассмотрим классическую физическую задачу на колебания математического маятника. Физически это камень или иной тяжелый предмет, подвешенный на веревке, закрепленной сверху. Маятник настольных часов – тоже хорошая иллюстрация на применение математического маятника в реальном устройстве.

Рисунок 9.62 содержит (в первой его половине) постановку *задачи на колебания маятника*. Они описываются нелинейным дифференциальным уравнением,

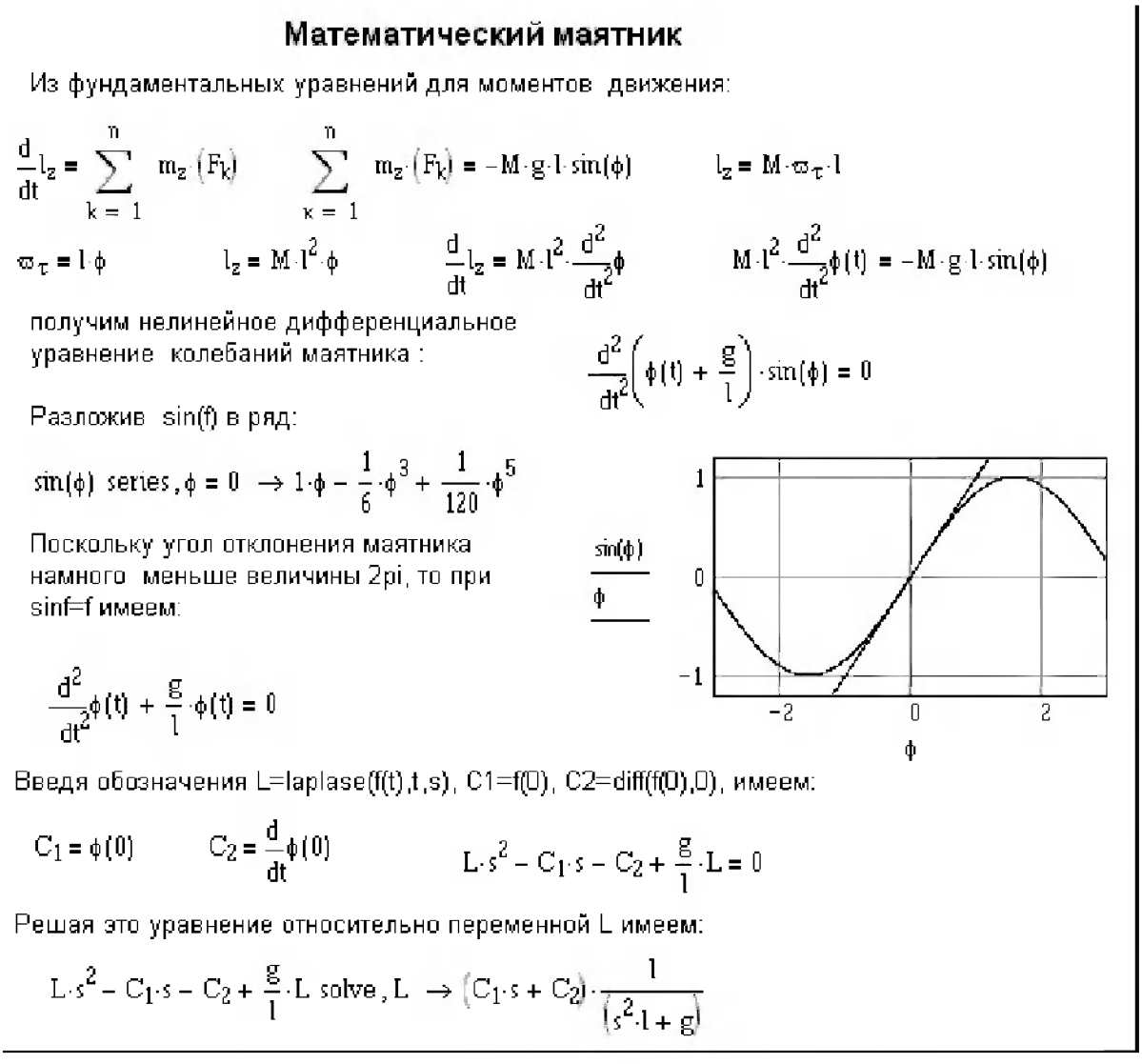


Рис. 9.62. Постановка задачи на моделирование математического маятника

что существенно осложняет анализ колебаний. Однако при малой амплитуде колебаний система линеаризируется, и мы сравнительно легко получаем уравнение колебаний в операторной форме.

Решение этого уравнения при воздействии на маятник толчка представлено на рис. 9.63. Оно иллюстрируется графиком колебаний – отклонением угла нити от вертикального положения. Нетрудно заметить, что форма колебаний синусоидальна.

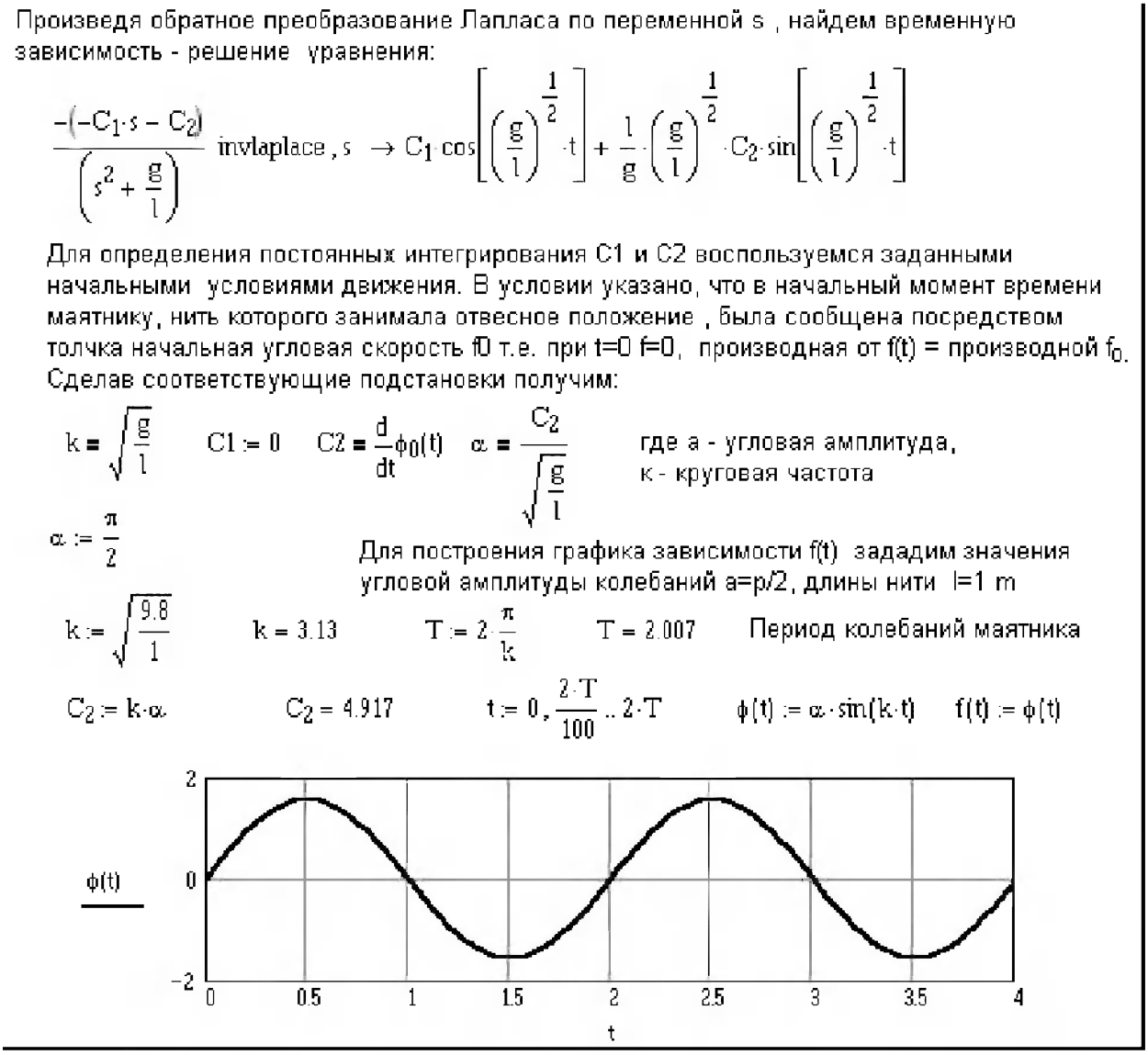


Рис. 9.63. Решение линеаризованного уравнения математического маятника и его графическое представление

Колебания маятника легко проиллюстрировать графически, используя технику анимации графиков. Такие иллюстрации производят большое впечатление на учащихся школ и даже вузов и университетов. Рисунок 9.64 поясняет подготовку к анимации. Здесь показаны уравнение движения груза (жирной точки) и окно подготовки фреймов (кадров) анимации. В этом окне, после выделения рисунка маятника и нажатия кнопки **Animation**, можно наблюдать подготовку фреймов (кадров) анимации.

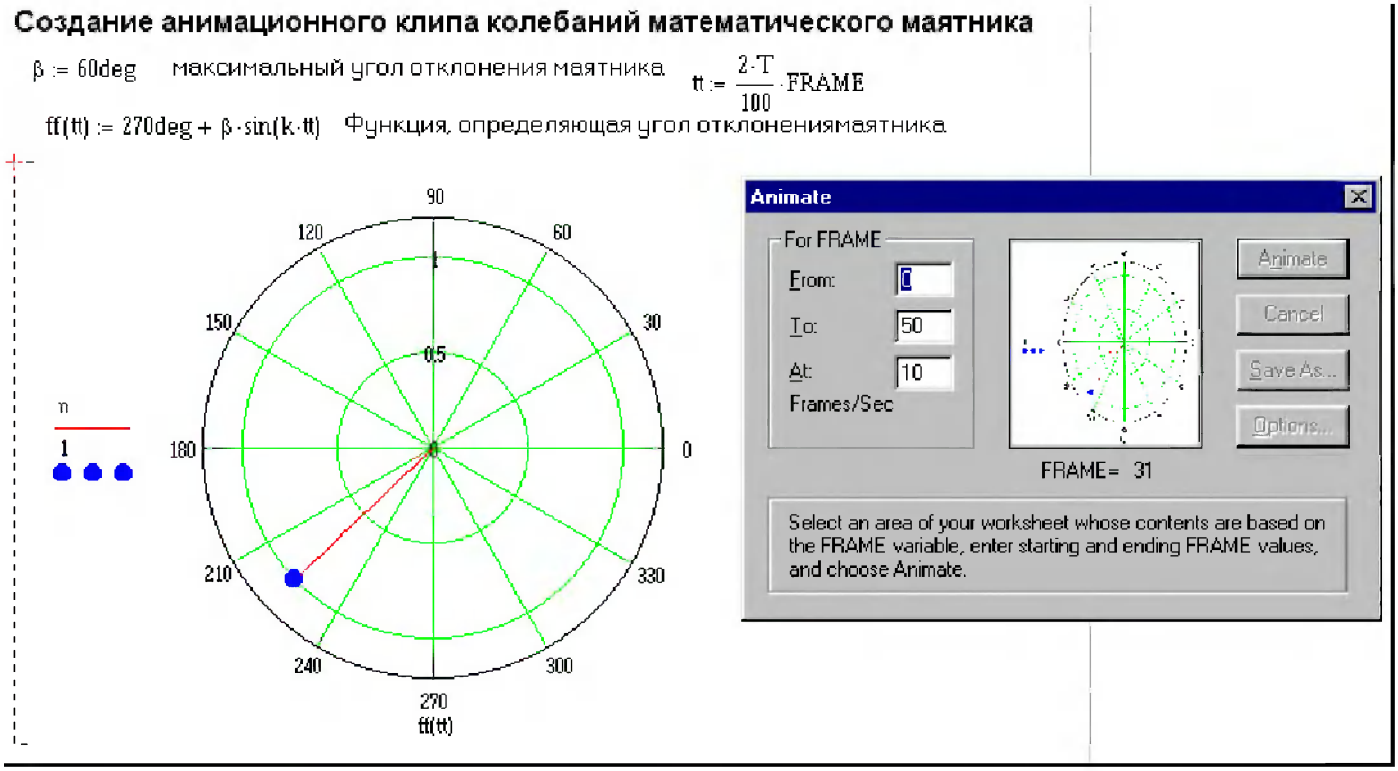


Рис. 9.64. Подготовка к анимации колебаний математического маятника

По окончании подготовки фреймов появляется окно анимационного проигрывателя, показанное на рис. 9.65. В нем и можно наблюдать колебания маятника в пределах одного периода. При этом маятник отклоняется вправо на угол до 60°, а затем влево на угол до −60° и возвращается в исходное состояние (угол 270°). Рисунок 9.65 иллюстрирует некоторый промежуточный кадр анимации (всего задано 50 фреймов на 1 цикл колебаний).

Но насколько линейная модель маятника отличается от более точной модели, основанной на решении нелинейных дифференциальных уравнений маятника? Ответ на этот вопрос дает рис. 9.66, на котором представлено решение этих уравнений численным методом.

Нетрудно заметить, что решение по нелинейной модели качественно почти не отличается от решения по линейной модели – форма колебаний очень близка к синусоидальной. Однако амплитуда колебаний и их период заметно отличаются от значений, характерных для линейной модели.

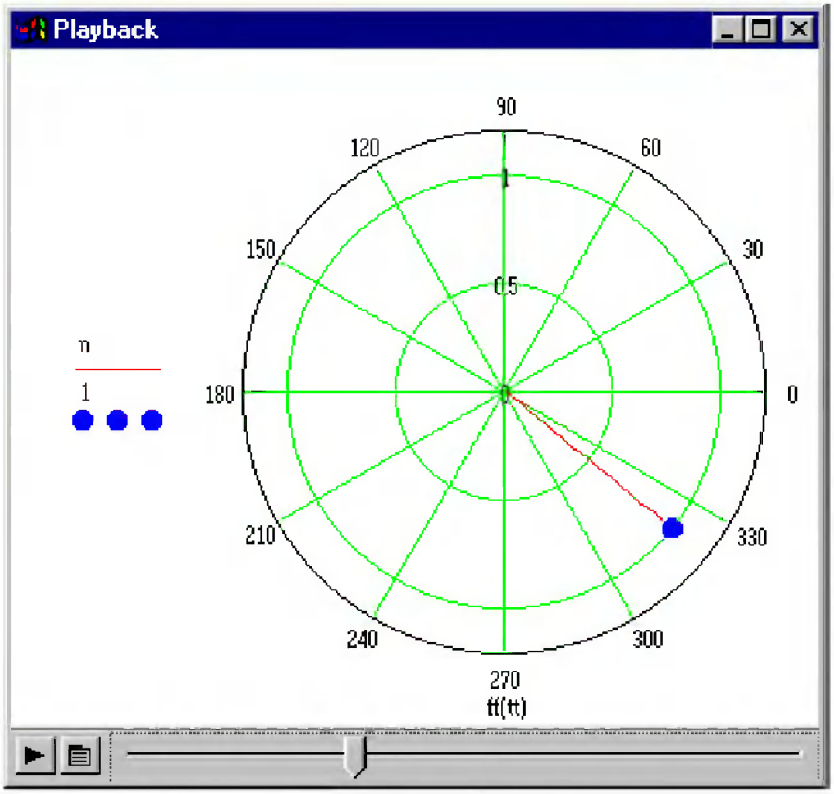


Рис. 9.65. Наблюдение колебаний маятника с помощью анимационного проигрывателя

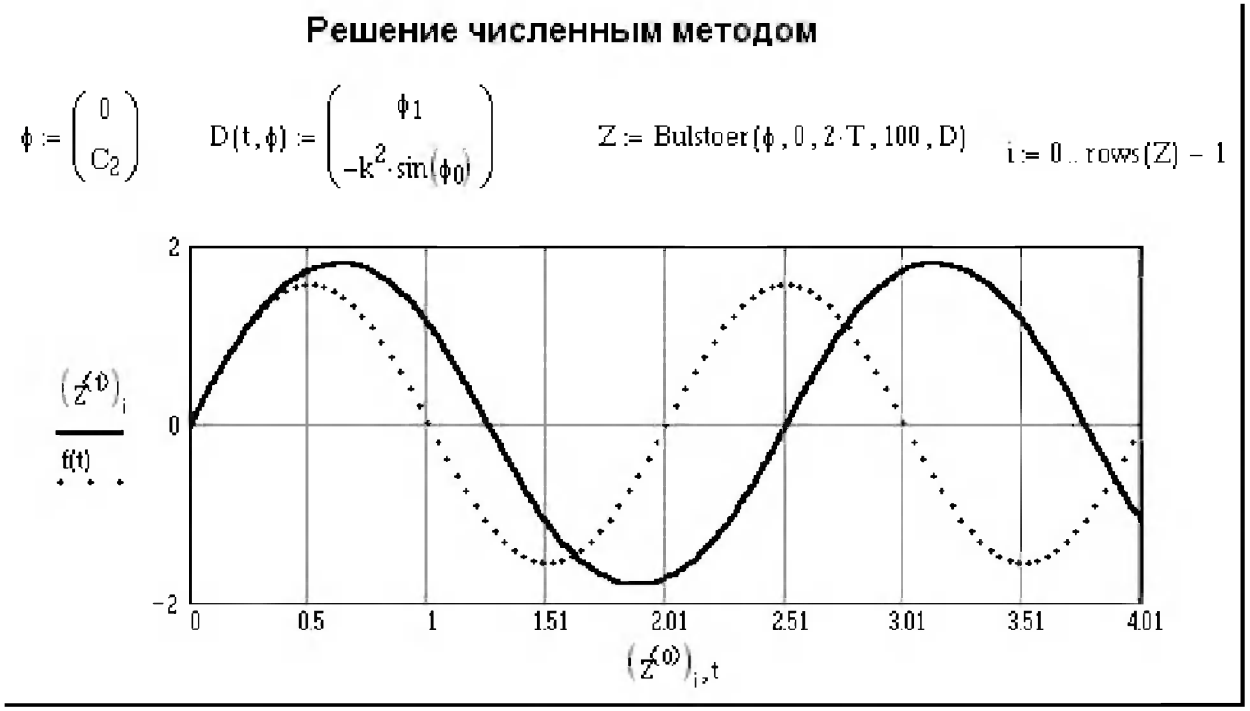


Рис. 9.66. Численное решение системы нелинейных дифференциальных уравнений математического маятника

9.16. Моделирование в Mathcad биологических и экономических систем

9.16.1. Модель системы «хищник–жертва» Лотки–Вольтера

Теперь рассмотрим типичную земную задачу о совместном проживании хищников и их жертв. Поскольку жертвы поедаются хищниками, число жертв начинает сокращаться, а число хищников – расти. Однако так не может продолжаться долго. Через некоторое время хищникам начинает не хватать пищи, и их популяция перестает расти и даже уменьшается. В итоге жертвы начинают размножаться более интенсивно, и их число растет. Далее эти процессы повторяются, и в них обнаруживается периодичность.

Одной из первых моделей (1925–1927 гг.) такой системы «хищник–жертва» стала *модель Лотки и Вольтера* (рис. 9.67). Пусть y_0 и y_1 – число жертв и хищников. Предположим, что относительный прирост жертв y_0'/y_0 равен $a - by_1$, где $a > 0$ –

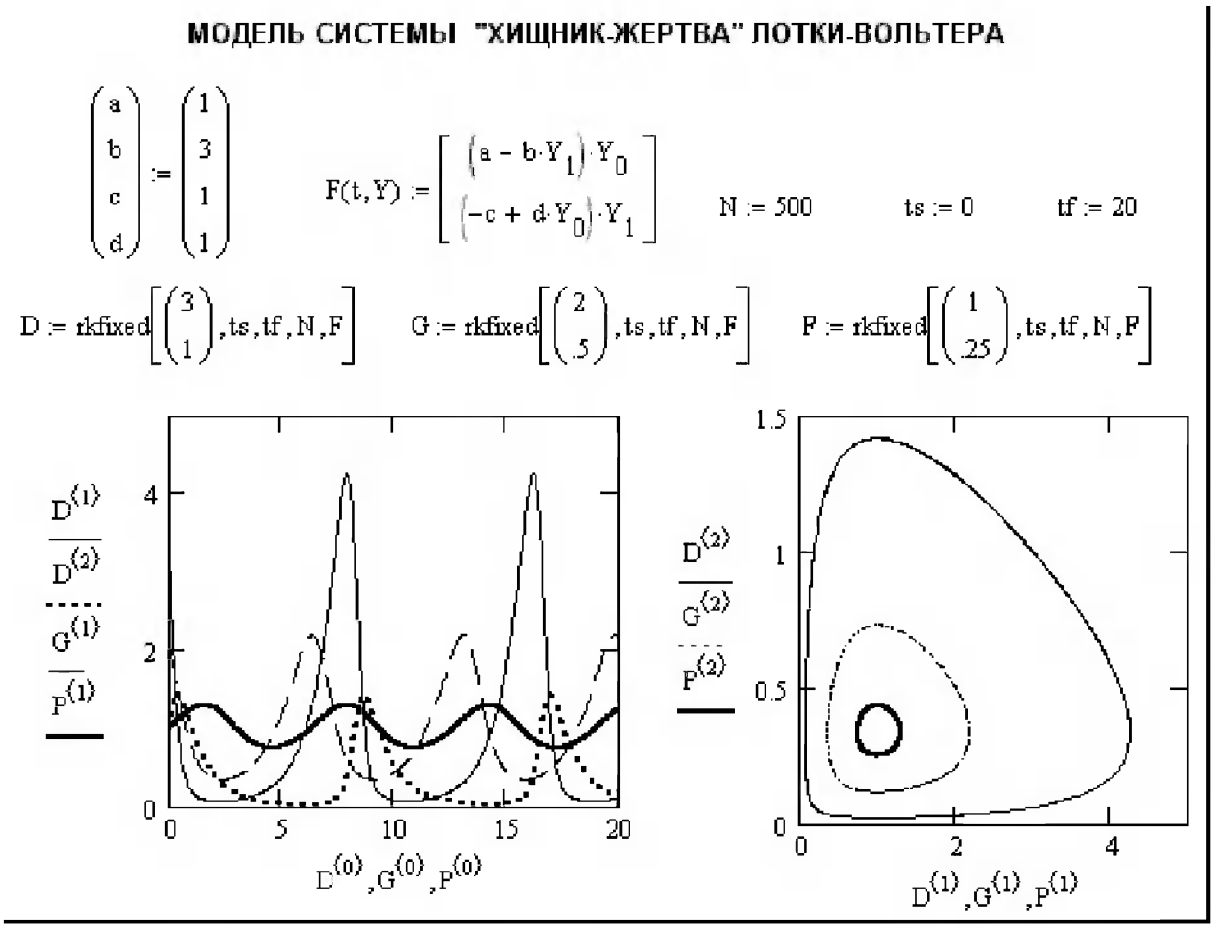


Рис. 9.67. Моделирование системы «хищник–жертва» по модели Лотки–Вольтера

скорость размножения жертв в отсутствие хищников, $-by_1$ ($b > 0$) – потери от хищников. Развитие популяции хищников зависит от количества пищи (жертв), при отсутствии пищи ($y_0 = 0$) относительная скорость изменения популяции хищников равна $y_1'/y_1 = c$, где $c > 0$. Наличие пищи компенсирует убывание хищников, и при $y_0 > 0$ имеем $y_1'/y_1 = (-c + dy_0)$, где $d > 0$.

Рассмотренная модель достаточно универсальна. Она может описывать не только изменение популяций хищников и жертв, но и поведение конкурирующих фирм, рост народонаселения, численность воюющих армий, изменение экологической обстановки, развитие науки и прочее. Рекомендуется поэкспериментировать с этой моделью и убедиться, что моделируемые процессы могут иметь не только колебательный, но и апериодический характер.

9.16.2. Модель системы «хищник–жертва» с логистической поправкой

Колебания популяций хищников и жертв на самом деле наблюдаются не всегда. Нередко мы наблюдаем стабильное количество тех и других, хотя процесс съедения жертв хищниками идет постоянно. Такой случай требует введения некоторой *логистической поправки*, которая учитывается в несколько иной модели системы «хищник–жертва», представленной на рис. 9.68.

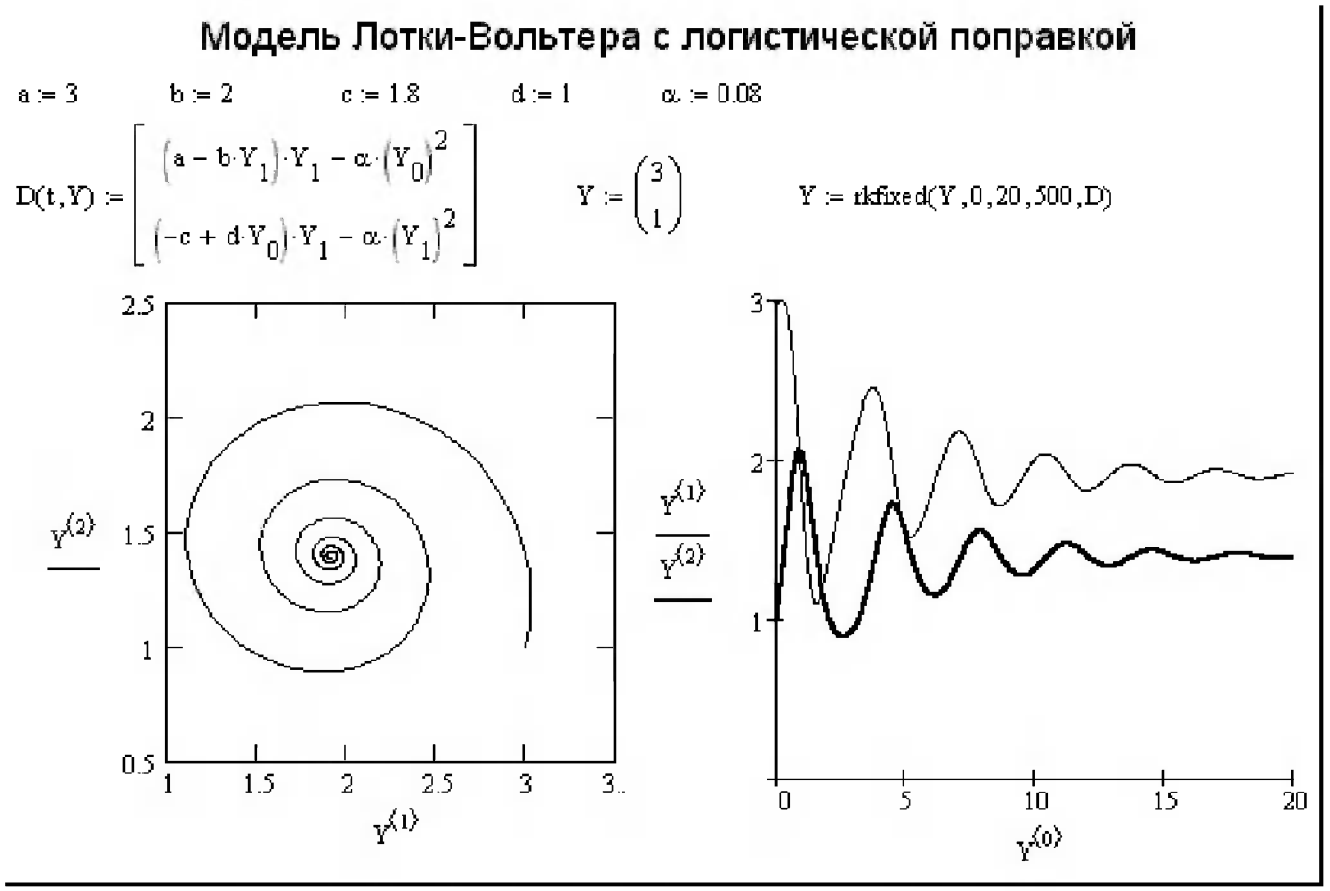


Рис. 9.68. Моделирование системы «хищник–жертва» по модели Лотки–Вольтера с логистической поправкой

Дополнительный параметр a в этой модели позволяет управлять затуханием осцилляций (колебаний) модели. Как нетрудно заметить, при указанных параметрах модели колебательный процесс в модели явно затухает, и устанавливается длительное равновесие между числом хищников и жертв. Фазовый портрет приобретает устойчивый *фокус*. Форма фазового портрета свидетельствует о довольно малой нелинейности этой системы. Поэтому колебания напоминают затухающую синусоиду. Однако при $a < 0$ образуется неустойчивый фокус, и колебания носят нарастающий характер.

9.16.3. Модель системы «хищник–жертва» Холлинга–Тэннера

Еще одна нелинейная модель системы «хищник–жертва» была предложена Холлингом и Тэннером – рис. 9.69. *Модель Холлинга и Тэннера* имеет две важные особенности. Ее нелинейность довольно сильна, что заметно из вида фазового портрета, витки которого достаточно отличны от эллипсов.

Главное свойство этой модели заключается в том, что в конечном счете колебания задаются предельным циклом фазового портрета, который может быть устой-

Рис. 9.69. Моделирование системы «хищник–жертва» по модели Холлинга–Тэннера

чивым. Он и определяет амплитуду колебаний, которые устанавливаются в стационарном режиме работы системы. При этом колебания могут как затухать во времени (пример чего и приведен), так и возрастать – приближаясь при этом к стационарным колебаниям.

9.16.4. Моделирование замкнутой экономической системы

В поведении биологических и экономических систем есть много общего. Поэтому при обсуждении моделей экономических систем ограничимся описанием *модели замкнутой экономической системы*, которая описывается дифференциальным уравнением второго порядка, – рис. 9.70.

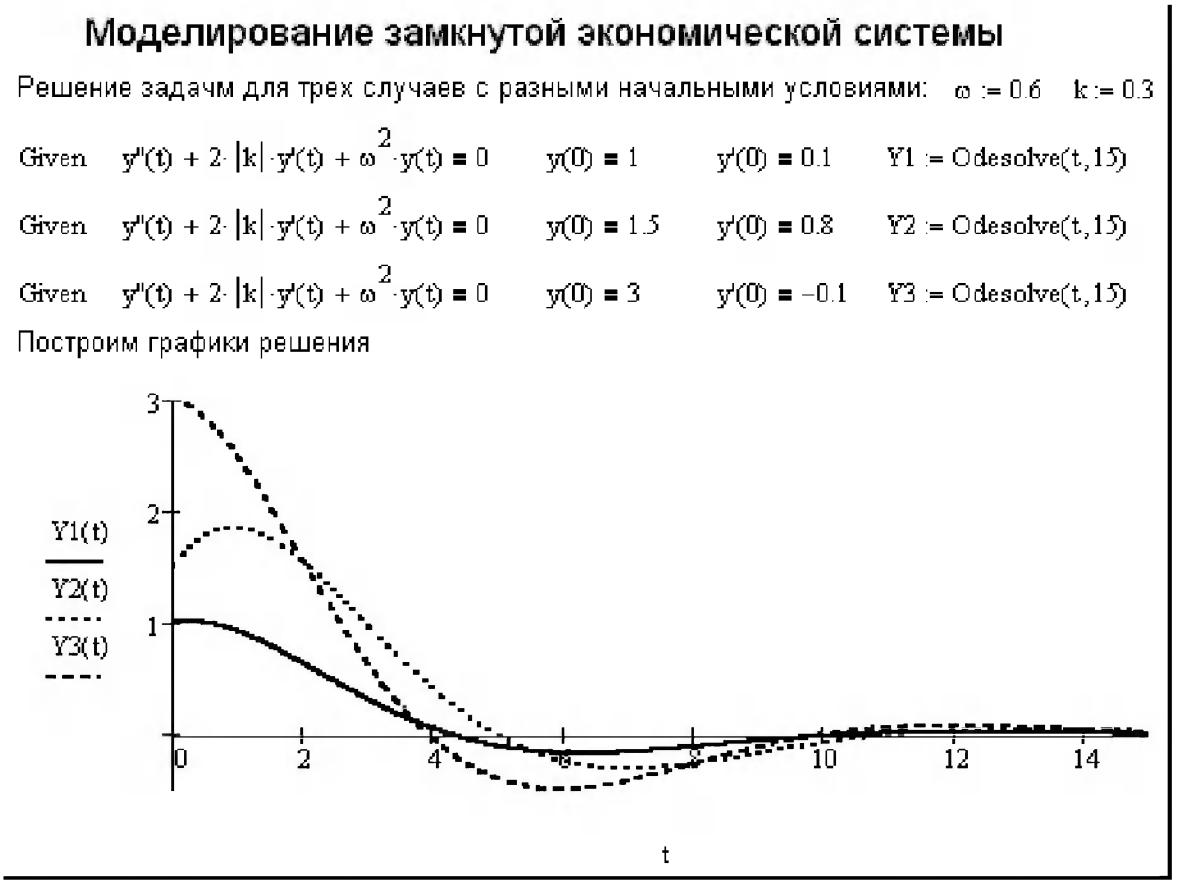


Рис. 9.70. Моделирование замкнутой экономической системы

Эта модель характеризуется двумя параметрами – круговой частотой осцилляций w и параметром затухания k. Поведение системы существенно зависит как от этих параметров, так и от начальных условий. Так, оно может иметь как апериодический, так и колебательный характер. В сущности, в этой модели для нас уже нет ничего нового – это еще одна модель системы второго порядка.

9.17. Новые возможности в решении дифференциальных уравнений в Mathematica

9.17.1. Пример решения линейного ДУ с нерациональными коэффициентами

В реализациях Mathematica 5.1/5.2 существенно улучшены алгоритмы решения дифференциальных уравнений в символьном виде. Ядро системы в этой части хорошо доработано, что позволяет уверенно решать дифференциальные уравнения самого различного класса. Разумеется, невозможно перечислить все новинки системы Mathenatica 5.1/5.2 в этой части, но несколько характерных примеров мы приведем.

На рис. 9.71 показано аналитическое решение линейного дифференциального уравнения с коэффициентами, которые выражены нерациональными числами, – как известно, в случае рациональных коэффициентов решение существенно упрощается. В нашем случае решение является далеко не простым, оно выражено через ряд специальных математических функций, но оно существует и с ним можно работать.

```
In[1]:= DSolve[x''[t] + e^-t x[t] == c, x[t], t]

                  {{x[t] → J0(2 Sqrt[e^-t]) c1 + 2 Y0(2 Sqrt[e^-t]) c2 -
Out[1]=
                    c π Y0(2 Sqrt[e^-t]) G[3,0,2,0](e^-t | 1, {0, 0, 0}) + c π J0(2 Sqrt[e^-t]) G[4,0,3,0](e^-t | -1/2, 1, {0, 0, 0, -1/2})}}
```

Рис. 9.71. Пример аналитического решения линейного дифференциального уравнения второго порядка с нерациональными коэффициентами

9.17.2. Решение дифференциального уравнения Абеля

В прежних версиях системы Mathematica определенные трудности вызывало решение даже некоторых дифференциальных уравнений классического типа, например *дифференциального уравнения Абеля* первого порядка. Как видно из рис. 9.72, в системе Mathematica 5.1 это уравнение уже решается.

```
In[2]:= DSolve[y'[x] == (x^7 - 4x/25 + y[x])/y[x], y[x], x]

Solve[c1 == (((2/(5x^3) - 2y(x)/x^4)^2 - 1)^(3/8)

Out[2]= {-5x^3/2 - (3 2F1(1/2, 5/8; 3/2; (2/(5x^3) - 2y(x)/x^4)^2)(2/(5x^3) - 2y(x)/x^4))/(4(1 - (2/(5x^3) - 2y(x)/x^4)^2)^(3/8)), y(x)}
```

Рис. 9.72. Пример решения дифференциального уравнения Абеля

9.17.3. Решение нелинейных дифференциальных уравнений

В общем случае *нелинейные дифференциальные уравнения* не имеют аналитического решения. Однако есть немало нелинейных дифференциальных уравнений частного вида, аналитические решения которых существуют и которые способна дать СКМ Mathematica 5.1/5.2. Один из примеров этого представлен на рис. 9.73.

```
In[3]:= DSolve[Max[t, t^2] y[t] + y'[t] == UnitStep[t] && y[0] == 1, y[t], t]

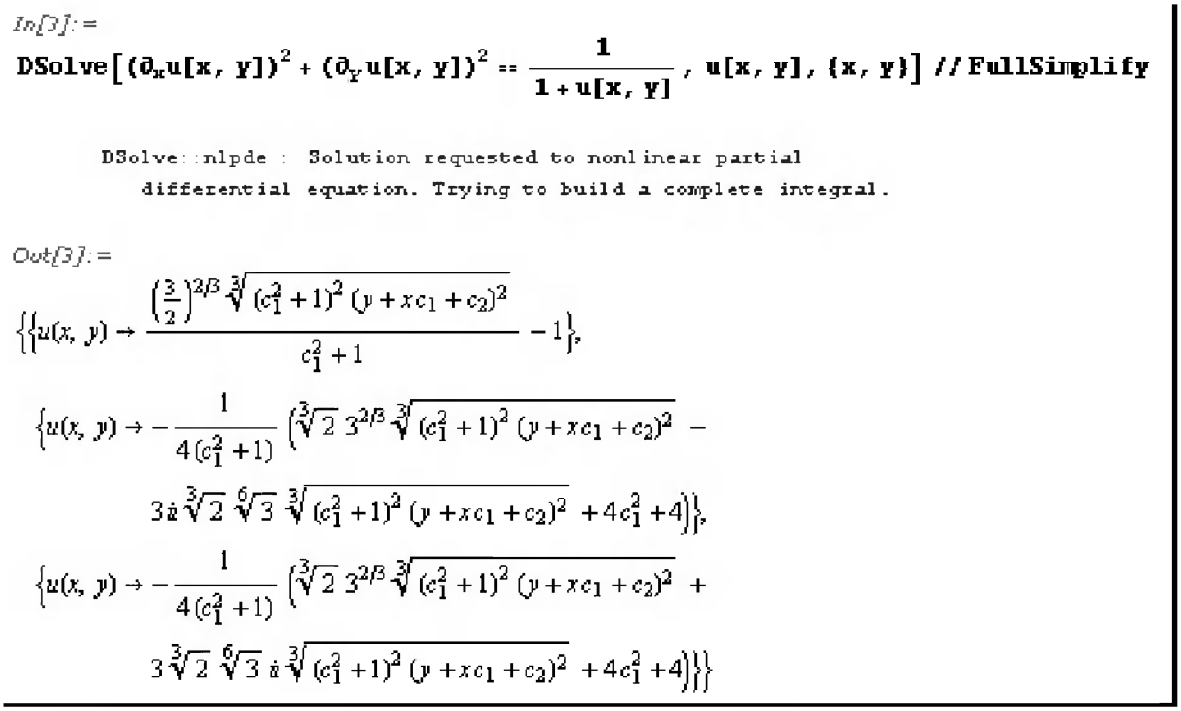
Out[3]= y(t) → e^(-t^3/3) + 1/(6t^2) (e^(-t^3/3 - 1/6) (2 3^(1/3) e^(1/6) Γ(1/3, -t^3/3) (-t^3)^(2/3)
        + 6t^2 + 3√(2π) t^2 erfi(1/√2) + 3^(1/3) e^(1/6) t^2 Γ(1/3, -1/3)
        - i 3^(5/6) e^(1/6) t^2 Γ(1/3, -1/3))) - e^(-t^3/3) θ(t)
      + θ(1 - t)(1/2 e^(-t^2/2) (√(2π) erfi(t/√2) + 2) θ(t)
        - 1/(6t^2) (e^(-t^3/3 - 1/6) (2 3^(1/3) e^(1/6) Γ(1/3, -t^3/3) (-t^3)^(2/3)
        + 6t^2 + 3√(2π) t^2 erfi(1/√2) + 3^(1/3) e^(1/6) t^2 Γ(1/3, -1/3)
        - i 3^(5/6) e^(1/6) t^2 Γ(1/3, -1/3))))
```

Рис. 9.73. Пример решения нелинейного дифференциального уравнения

9.17.4. Решение нелинейного ДУ в частных производных

Более сложный пример решения нелинейного дифференциального уравнения второго порядка в частных производных представлен на рис. 9.74.

Большое число примеров на решение дифференциальных уравнений с применением новых средств Mathematica 5.1/5.2/6 можно найти в справках по этим системам и на интернет-сайте ее разработчика.



```
In[3]:=
DSolve[(∂x u[x, y])^2 + (∂y u[x, y])^2 == 1/(1 + u[x, y]), u[x, y], {x, y}] // FullSimplify

DSolve::nlpde : Solution requested to nonlinear partial
   differential equation. Trying to build a complete integral.

Out[3]=
{{u[x, y] → (3/2)^(2/3) (c1^2 + 1)^(2/3) (y + x c1 + c2)^(2/3) / (c1^2 + 1) - 1},
 {u[x, y] → -1/(4 (c1^2 + 1)) ((2)^(1/3) 3^(2/3) (c1^2 + 1)^(2/3) (y + x c1 + c2)^(2/3) -
    3 i (2)^(1/3) (3)^(1/6) (c1^2 + 1)^(2/3) (y + x c1 + c2)^(2/3) + 4 c1^2 + 4)},
 {u[x, y] → -1/(4 (c1^2 + 1)) ((2)^(1/3) 3^(2/3) (c1^2 + 1)^(2/3) (y + x c1 + c2)^(2/3) +
    3 (2)^(1/3) (3)^(1/6) i (c1^2 + 1)^(2/3) (y + x c1 + c2)^(2/3) + 4 c1^2 + 4)}}
```

Рис. 9.74. Пример решения нелинейного дифференциального уравнения в частных производных

Глава 10

Визуализация вычислений в Maple

Эта глава книги посвящена уникальным возможностям системы Maple 9.5/10/11 в визуализации самых разнообразных вычислений. Вначале рассмотрены возможности и опции двумерной и трехмерной графики, имеющиеся во всех упомянутых версиях системы Maple. Особое внимание уделено визуализации математических и физических понятий и реализации различных возможностей машинной графики [155]. Затем описаны новые возможности графики последних версий – Maple 10/11.

10.1. Двумерная графика Maple

10.1.1. Введение в двумерную графику Maple

Средства для построения графиков в большинстве языков программирования принято считать графическими *процедурами*, или *операторами*. Однако в СКМ Maple 9.5/10 мы сохраним за ними наименование *функций* в силу двух принципиально важных свойств:

- графические средства Maple *возвращают* некоторые графические объекты, которые размещаются в окне документа – в строке вывода или в отдельном графическом объекте;
- эти объекты можно использовать в качестве значений переменных, то есть переменным можно присваивать значения графических объектов и выполнять над ними соответствующие операции (например, с помощью функции `show` выводить на экран несколько графиков).

В Maple (начиная с версии 5) введены новые функции *быстрого построения графиков*. Так, функция `smartplot(f)` предназначена для создания двумерных графиков. Параметр `f` может задаваться в виде одиночного выражения или набора выражений, разделяемых запятыми. Задание управляющих параметров в этих графических функциях не предусмотрено; таким образом, их можно считать первичными, или черновыми. Для функции построения двумерного графика по умолчанию задан диапазон изменения аргумента `-10..10`.

Графические функции заданы таким образом, что обеспечивают построение типовых графиков без какой-либо особой подготовки. Однако с помощью дополнительных необязательных параметров (опций) можно существенно изменить вид графиков, например настроить стиль и цвет линий, вывести титульную надпись, изменить вид координатных осей и т. д.

На некоторых графиках Maple по умолчанию вписывает легенду, то есть список линий с обозначениями. Иногда этот список оказывается просто некстати. Легенду можно убрать, расширив заодно место для графика, сняв флажок **Show Legend** в контекстном меню **Legend** правой клавиши мыши. Заодно запомните, что легенду можно редактировать, выполнив команду **Edit Legend**.

10.1.2. Функция plot для построения двумерных графиков и ее опции

Для построения *двумерных графиков* служит функция `plot`. Она задается в виде

```
plot(f, h, v)    plot(f, h, v, o)
```

где `f` – визуализируемая функция (или функции), `h` – переменная с указанием области ее изменения, `v` – необязательная переменная с указанием области изменения, `o` – параметр или набор параметров, задающих стиль построения графика (толщину и цвет кривых, тип кривых, метки на них и т. д.).

Выше уже приводилось множество примеров применения этой функции (см. рис. 1.4, 2.4, 3.1–3.3 и др.). Для функции `plot` возможны следующие дополнительные параметры:

- `adaptive` – включение адаптивного алгоритма построения графиков;
- `axes` – вывод различных типов координат (`axes=NORMAL` – обычные оси, выводятся по умолчанию, `axes=BOXES` – график заключается в рамку с осями-шкалами, `axes=FRAME` – оси в виде перекрещенных линий, `axes=NONE` – оси не выводятся);
- `axesfont` – задание шрифтов для подписи делений на координатных осях (см. также параметр `font`);
- `color` – задает цвет кривых;
- `coords` – задание типа координатной системы;
- `discont` – задает построение непрерывного графика (значения `true` или `false`), например для таких функций, как тангенс;
- `filled` – при `filled=true` задает окраску цветом, заданным параметром `color`, для области, ограниченной построенной линией и горизонтальной координатной осью *x*;
- `font` – задание шрифта в виде [семейство, стиль, размер];
- `labels` – задание надписей по координатным осям в виде [X,Y], где X и Y – надписи по осям x и y графика;
- `labeldirections` – задает направление надписей по осям [X,Y], где X и У могут иметь строковые значения HORISONTAL (горизонтально) и VERTICAL (вертикально);
- `labelfont` – задает тип шрифта метод (см. `font`);
- `legend` – задает вывод легенды (обозначения кривых);
- `linestyle` – задание стиля линий (`1` – сплошная, `2` – точками, `3` – пунктиром и `4` – штрихпунктиром);
- `numpoints` – задает минимальное количество точек на графике (по умолчанию `numpoints=49`);
- `resolutions` – задает горизонтальное разрешение устройства вывода (по умолчанию `resolutions=200`, параметр используется при отключенном адаптивном методе построения графиков);

- `sample` – задает список параметров для предварительного представления кривых;
- `scaling` – задает масштаб графика: `CONSTRAINED` (сжатый) или `UNCONSTRAINED` (несжатый – по умолчанию);
- `size` – задает размер шрифта в пунктах;
- `style` – задает стиль построения графика (`POINT` – точечный, `LINE` – линиями);
- `symbol` – задает вид символа для точек графика (возможны значения `BOX` – прямоугольник, `CROSS` – крест, `CIRCLE` – окружность, `POINT` – точка, `DIAMOND` – ромб);
- `symbolsize` – установка размеров символов для точек графика (в пунктах, по умолчанию 10);
- `title` – задает построение заголовка графика (`title="string"`, где `string` – строка);
- `titlefont` – определяет шрифт для заголовка (см. `font`);
- `thickness` – определяет толщину линий графиков (`0`, `1`, `2`, `3`, значение по умолчанию – `0`);
- `view=[A, B]` – определяет максимальные и минимальные координаты, в пределах которых график будет отображаться на экране, `A = [xmin..xmax]`, `B=[ymin..ymax]` (по умолчанию отображается вся кривая);
- `xtickmarks` – задает минимальное число отметок по оси x;
- `ytickmarks` – задает минимальное число отметок по оси y.

Параметр `adaptive` задает работу специального адаптивного алгоритма для построения графиков наилучшего вида. При этом Maple автоматически учитывает кривизну изменения графика и увеличивает число отрезков прямых в тех частях графиков, где их ход заметно отличается от интерполирующей прямой. При задании `adaptive=false` адаптивный алгоритм построения графиков отключается, а при `adaptive=true` включается (значение по умолчанию).

С помощью параметра `y=ymin..ymax` можно задать масштаб графика по вертикали.

Изредка встречаются графики функций $f(x)$, которые надо построить при изменении значения x от нуля до бесконечности или даже от минус бесконечности до плюс бесконечности. Бесконечность в таких случаях задается как особая константа `infinity`. В этом случае переменной x, устремляющейся в бесконечность, откладывается значение $\arctan(x)$.

В Maple 9.5 параметр `coords` задает 15 типов *координатных систем* для двумерных графиков. По умолчанию используется прямоугольная (декарто-

ва) система координат (`coords=cartesian`). При использовании других координатных систем координаты точек для них (u, v) преобразуются в координаты (x, y) как $(u, v) \rightarrow (x, y)$. Формулы преобразования координат можно найти в справке.

10.1.3. Управление стилем и цветом линий двумерных графиков

Maple 9.5 позволяет воспроизводить на одном графике множество кривых с разным *стилем*, который задается параметром `style`:

- `POINT` или `point` – график выводится по точкам;
- `LINE` или `line` – график выводится линией.

Если задано построение графика точками, то параметр `symbol` позволяет представить точки в виде различных символов, например прямоугольников, крестов, окружностей или ромбов. Другой параметр – `color` – позволяет использовать обширный набор цветов линий графиков. Различные цветовые оттенки получаются использованием RGB-комбинаций базовых цветов: `red` – красный, `gray` – зеленый, `blue` – синий. Приведем перевод ряда других составных цветов: `black` – черный, `white` – белый, `khaki` – цвет «хаки», `gold` – золотистый, `orange` – оранжевый, `violet` – фиолетовый, `yellow` – желтый и т. д. Естественно, что при черно-белой печати рисунков вместо цветов получаются градации серого цвета.

10.1.4. Графики нескольких функций на одном рисунке

Важное значение имеет возможность построения на одном рисунке графиков нескольких функций. В простейшем случае (рис. 2.4 и 3.3) для построения таких графиков достаточно перечислить нужные функции и установить для них общие интервалы изменения.

Обычно графики разных функций автоматически строятся разными цветами. Но это не всегда удовлетворяет пользователя – например, при распечатке графиков монохромным принтером некоторые кривые могут выглядеть слишком блеклыми или даже не пропечататься вообще. Используя списки параметров `color` (цвет линий) и `style` (стиль линий), можно добиться выразительного выделения кривых.

10.2. Специальные типы двумерных графиков

10.2.1. Графики функций, заданных своими именами и процедурами

К интересным возможностям графики Maple относится построение графиков функций, заданных только их функциональными именами – даже без указания параметров в круглых скобках. Например, график функций синуса и косинуса можно построить, используя функцию `plot` в виде: `plot({sin,cos},-10..10)`.

Некоторые виды функций, например кусочные, удобно задавать процедурами. Построение графиков функций, заданных процедурами, не вызывает никаких трудностей. Еще одна возможность функции `plot` – построение графиков функций, заданных *функциональными операторами*. Имена функций (без указания списка параметров в круглых скобках, например x->x^2) тоже по существу являются функциональными операторами. Так что они также могут использоваться при построении графиков упрощенными способами.

10.2.2. Графики функций, заданных параметрически

В ряде случаев для задания функциональных зависимостей используются заданные *параметрически* уравнения, например $x = f_1(t)$ и $y = f_2(t)$ при изменении переменной t в некоторых пределах. Точки (x, y) наносятся на график в декартовой системе координат и соединяются отрезками прямых. Для этого используется функция `plot` в следующей форме:

```
plot([f1(t),f2(t),t=tmin..tmax],h,v,p)
```

Если функции $f_1(t)$ и $f_2(t)$ содержат периодические функции (например, тригонометрические), то для получения замкнутых фигур диапазон изменения переменной t обычно задается равным `0..2*Pi` или `-Pi..Pi`. К примеру, если задать в качестве функций $f_1(t)$ и $f_2(t)$ функции $\sin(t)$ и $\cos(t)$, то будет получен график окружности. Задание диапазонов для изменений `h` и `v`, а также параметров `p` не обязательно. Но, как и ранее, они позволяют получить вид графика, удовлетворяющий всем требованиям пользователя.

10.2.3. Графики функций в полярной системе координат

Графики в *полярной системе координат* представляют собой линии, которые описывают конец радиус-вектора $\mathbf{r}(t)$ при изменении угла t в определенных преде-

лах – от t_{min} до t_{max}. Построение таких графиков также производится функцией `plot`, которая для этого записывается в следующем виде:

```
plot([r(t),theta(t),t=tmin..tmax],h,v,p,coords=polar)
```

Здесь существенным моментом является задание полярной системе координат параметра `coords=polar`. Рисунок 10.1 дает примеры построения графиков функций в полярной системе координат.

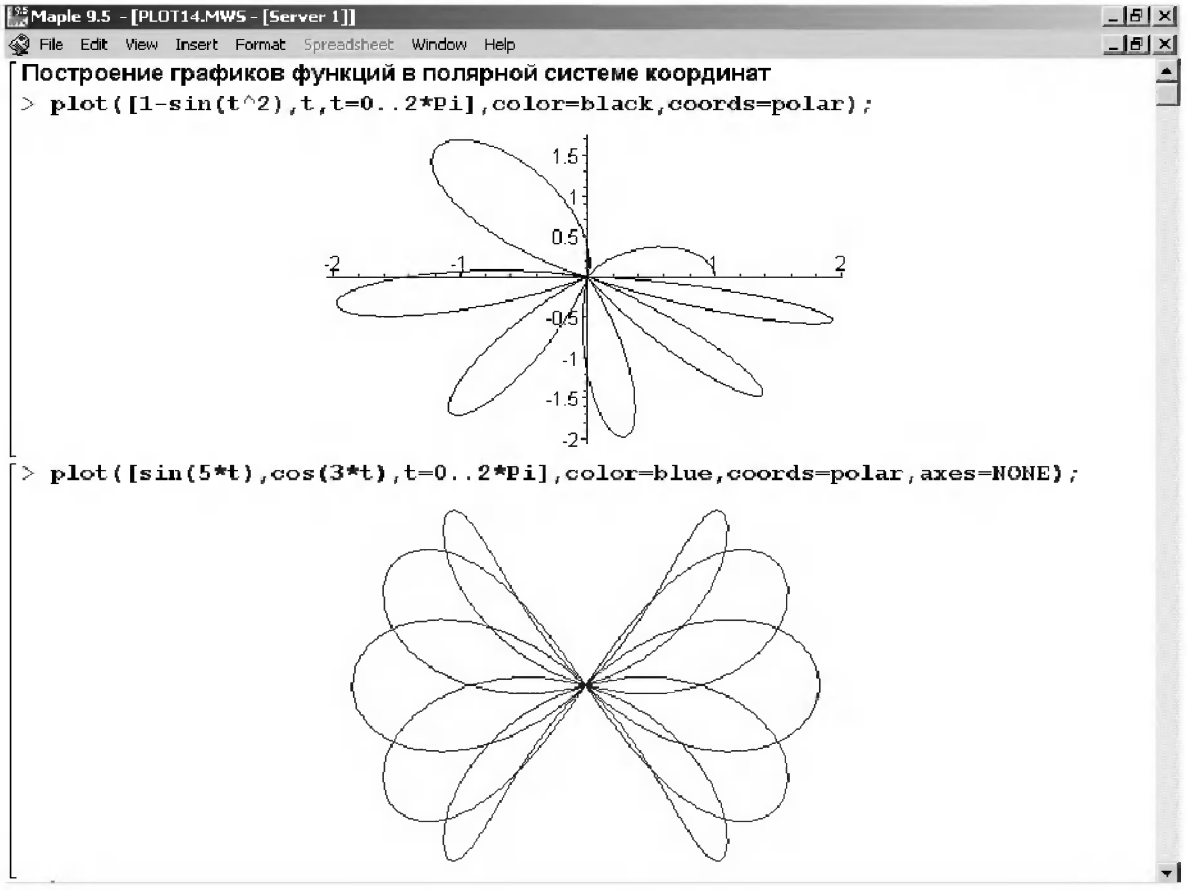


Рис. 10.1. Построение графиков функций в полярной системе координат

Графики параметрических функций и функций в полярной системе координат отличаются огромным разнообразием. Снежинки и узоры мороза на стеклах, некоторые виды кристаллов и многие иные физические объекты подчиняются математическим закономерностям, положенным в основу построения таких графиков.

10.3. Визуализация трехмерных объектов

10.3.1. Функция plot3d и ее параметры

Трехмерными графиками называют графики, отображающие функции двух переменных $z(x, y)$. Каждая точка z_i таких графиков является высотой (аппликатой) точки, лежащей в плоскости XY и представленной координатами (x_i,y_i). Для

построения графиков трехмерных поверхностей Maple имеет встроенную в ядро функцию plot3d. Она может использоваться в следующих форматах:

```
plot3d(expr1, x=a..b, y=c..d,p)
plot3d(f, a..b, c..d,p)
plot3d([exprf,exprg,exprh], s=a..b, t=c..d,p)
plot3d([f,g,h], a..b, c..d,p)
```

В двух первых формах plot3d применяется для построения обычного графика одной поверхности, в других формах – для построения графика с параметрической формой задания поверхности. В приведенных формах записи f, g и h – функции; expr1 – выражение, отражающее зависимость от x и y; exprf, exprg и exprh – выражения, задающие поверхность параметрически; s, t, a и b – числовые константы действительного типа; c и d – числовые константы или выражения действительного типа; x, y, s и t – имена независимых переменных; p – управляющие параметры. Пример применения функции plot3d дан на рис. 1.5 (Maple 11).

С помощью параметров p можно в широких пределах управлять видом трехмерных графиков, выводя или убирая линии каркасной сетки, вводя функциональную окраску поверхностей, меняя угол их обзора и параметры освещения, изменяя вид координатных осей и т. д. Следующие параметры функции plot3d задаются аналогично их заданию для функции plot:

```
axesfont   font     color  coords  font       labelfont  linestyle
numpoints  scaling  style  symbol  thickness  title      titlefont
```

Однако функция plot3d имеет ряд дополнительных специфических параметров:

- ambientlight=[r,g,b] – задает интенсивность красного (r), зеленого (g) и синего (b) цветов подстветки в относительных единицах (от 0 до 1);
- axes=f – задает вид координатных осей (BOXED, NORMAL, FRAME и NONE, по умолчанию NONE);
- grid=[m,n] – задает число линий каркаса поверхности;
- gridstyle=x – задает стиль линий каркаса x ('rectangular' или 'triangular');
- labels=[x,y,z] – задает надписи по осям (x, y и z – строки, по умолчанию пустые);
- light=[phi,theta,r,g,b] – задает углы, под которыми расположен источник освещения поверхности, и интенсивности составляющих цвета (r, g и b);
- lightmodel=x – задает схему освещения (соответственно 'none', 'light1','light2','light3' и 'light4');
- orientation=[theta,phi] – задает углы ориентации поверхности (по умолчанию 45°);
- projection=r – задает перспективу при обзоре поверхности (r может быть числом 0 или 1, задающим включение или выключение перспективы, а также одной из строк 'FISHEYE', 'NORMAL' или 'ORTHOGONAL' (это соответствует численным значениям r, равным 0, 0,5, или 1, причем по умолчанию задано projection=ORTHOGONAL);

- `shading=s` – задает направления, по которым меняется цвет функциональной окраски (значения `s` могут быть `XYZ`, `XY`, `Z`, `ZGREYSCALE`, `ZHUE`, `NONE`);
- `tickmarks=[l,n,m]` – задает характер маркировки по осям x, y и z (числа `l`, `n` и `m` имеют значения не менее 1);
- `view=zmin..zmax` или `view=[xmin..xmax, ymin..ymax, zmin..zmax]` – задает минимальные и максимальные координаты поверхности для ее видимых участков.

По умолчанию в Maple строится поверхность с функциональной окраской и стилем `style=patch`. Функциональная окраска делает рисунки более информативными, но, увы, на рисунках в книге она превращается в окраску оттенками серого цвета. Параметр `style=hidden` строит каркасную поверхность с функциональной окраской тонких линий каркаса и удалением невидимых линий.

Помимо значения `patch`, для построения трехмерных поверхностей можно задавать ряд других стилей: `point` – точками, `contour` – контурными линиями, `line` – линиями, `hidden` – линиями каркаса с удалением невидимых линий, `wireframe` – линиями каркаса со всеми видимыми линиями, `patchnogrid` – с раскраской, но без линий каркаса, `patchcontour` – раскраска с линиями равного уровня.

Цвет трехмерного графика может задаваться (как и для двумерного) параметром `color=c`, где `c` – цвет (оттенки цвета перечислялись ранее). Возможны еще два алгоритма задания цвета:

- `HUE` – алгоритм с заданием цвета в виде `color=f(x,y)`;
- `RGB` – алгоритм с заданием цвета в виде `color=[exprr,exprg,exprb]`, где выражения `exprr`, `exprg` и `exprb` задают относительную значимость (от 0 до 1) основных цветов (красного – `exprr`, зеленого – `exprg` и синего – `exprb`).

Удачный выбор углов обзора фигуры и применение функциональной окраски позволяют придать построениям трехмерных фигур весьма эффектный и реалистический вид.

10.3.2. Построение 3D-фигур в различных системах координат

Для трехмерных графиков возможно задание множества типов координатных систем с помощью параметра `coords=Тип_координатной_системы`. Поскольку на экране монитора поверхность отображается только в прямоугольной системе координат и характеризуется координатами x, y и z, то для представления поверхности, заданной в иной системе координат с координатами u, v и w, используются известные формулы для преобразования (u, v, w) –> (x, y, z). Их можно найти в справке. Вид графиков трехмерных поверхностей очень сильно различается в разных координатных системах. По умолчанию трехмерные графики строятся в прямоугольной системе координат – `rectangular`.

Как отмечалось, вид графика трехмерной поверхности (или 3D-фигуры) существенно зависит от выбора координатной системы. Рисунок 10.2 показывает пример построения шаровой поверхности с вырезом в сферической системе координат. Здесь функция задана элементарно просто – в виде числа 1. Но поскольку выбрана сферическая система координат, в результате строится поверхность шара единичного радиуса.

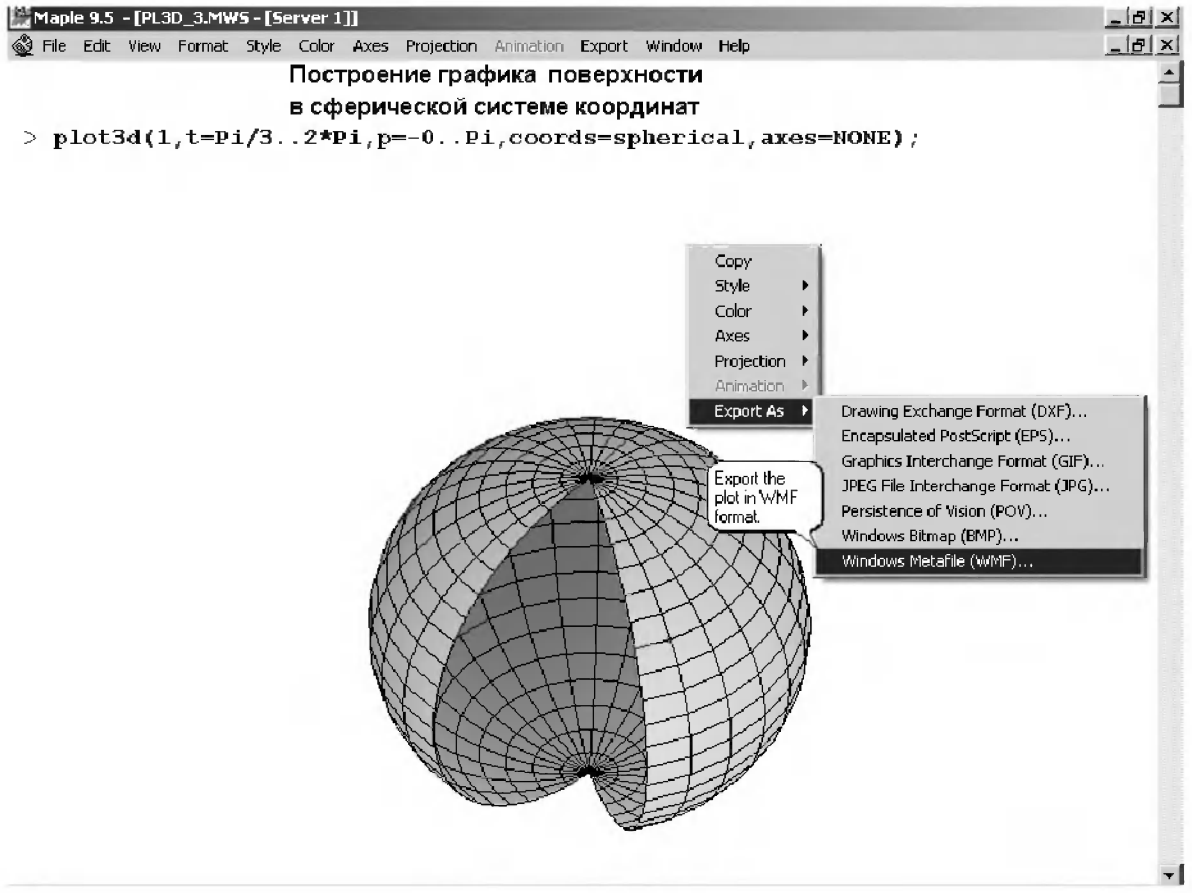


Рис. 10.2. Построение шарообразной поверхности с вырезом в сферической системе координат

Полезно просмотреть построение графиков разных функций в различных системах координат. Можно при этом можно получить самые необычные фигуры.

10.3.3. Графики параметрически заданных поверхностей

Большие возможности открывает параметрическое задание поверхности, при котором все три координаты описываются функциональными зависимостями. В Maple есть способ явно задать углы обзора с помощью параметра

orientation=[theta, phi],

где `theta` и `phi` – углы, через которые задаются параметрические уравнения трехмерной фигуры или поверхности. Параметрическое задание уравнений по-

верхности открывает почти неисчерпаемые возможности построения занимательных и сложных фигур самого различного вида. Приведем пример построений такого рода – рис. 10.3.

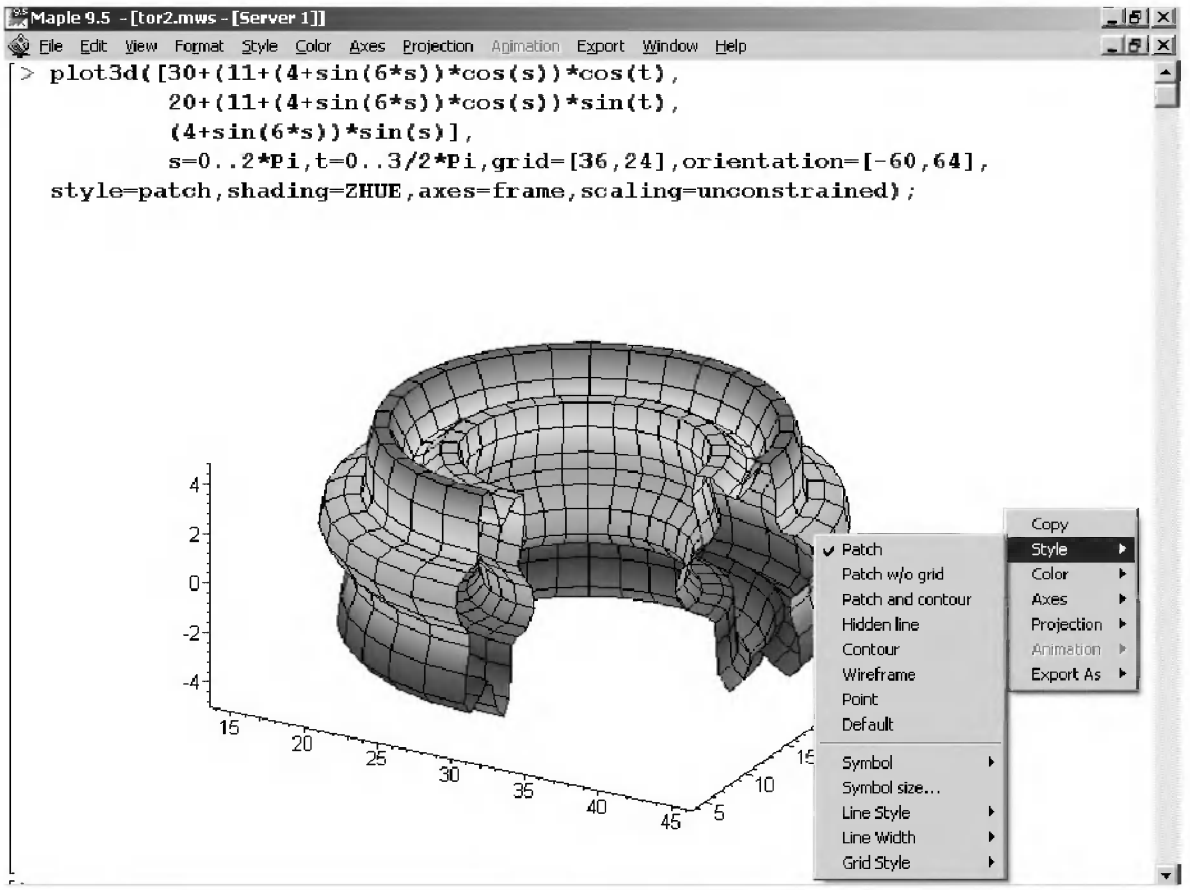


Рис. 10.3. Пример сложной 3D-фигуры с параметрическим ее заданием

Здесь строится тор, сечение которого имеет вид сплюснутой шестиконечной звезды. Вырез в фигуре дает прекрасный обзор ее внутренней поверхности, а цветная функциональная окраска и линии сетки, построенные с применением алгоритма удаления невидимых линий, дают весьма реалистичный вид фигуры. Замените параметр `scaling=unconstrained` на `scaling=constrained`, и вы получите тор с неискаженным сечением. Обратите внимание на возможности контекстного меню правой клавиши мыши, которое видно на рис. 10.3.

Быстрое (не в смысле ускорения самого построения, а лишь в смысле более быстрого задания построения графиков) построение трехмерных графиков обеспечивает функция `smartplot3d`. Для этой функции задан диапазон изменения обоих аргументов `-5..5`.

10.3.4. Построение ряда трехмерных фигур на одном графике

Функция `plot3d` позволяет строить одновременно несколько фигур, пересекающихся в пространстве. Для этого достаточно вместо описания одной поверхности за-

дать список описаний ряда поверхностей. Это создает изображения, выглядящие вполне естественно. Пример такого построения для двух функций показан на рис. 10.4.

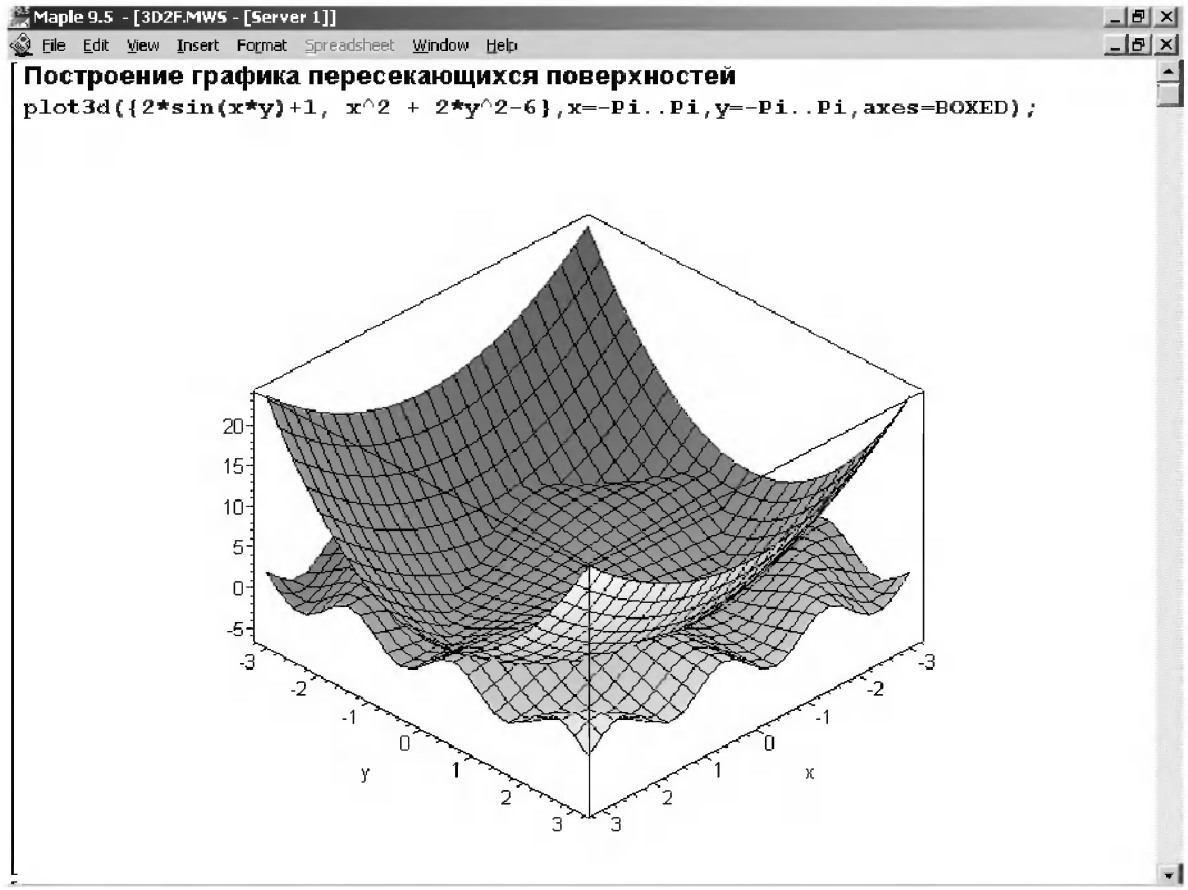


Рис. 10.4. Пример построения двух трехмерных фигур, пересекающихся в пространстве

Из рис. 10.4 видно, что функция `plot3d` обладает возможностью автоматически вычислять точки пересечения фигур и показывает только видимые части поверхностей.

10.4. Работа с графическими структурами

10.4.1. Работа с графическими структурами двумерной графики

Функции `PLOT` и `PLOT3D` (с именами, набранными большими буквами) позволяют создавать *графические структуры*, содержащие ряд графических объектов `s1`, `s2`, `s3` и т. д. Каждый объект может представлять собой точку или фигуру, полигон, надпись и т. д., позиционированную с высокой точностью в заданной системе координат. Координатные оси также относятся к графическим объектам. Важно

отметить, что функции PLOT и PLOT3D одновременно являются данными, описывающими графики. Их можно записывать в виде файлов и (после открытия файлов) представлять в виде графиков. Особые свойства этих функций подчеркиваются их записью прописными буквами.

Графическая структура двумерной графики задается в виде

```
PLOT(s1, s2, s3,…,o);
```

где s1, s2, s3 ... – графические объекты (или элементарные структуры – примитивы), o – общие для структуры параметры).

Основными объектами являются:

- POINTS([x1,y1],[x2,y2],…[xn,yn]) – построение точек, заданных их координатами;
- CURVES([[x11,y11],…[x1n,y1n]], [[x21,y21],…[x2n,y2n]],… [[xm1,ym1],…[xmn,ymn]]) – построение кривых по точкам;
- POLYGONS([[x11,y11],…[x1n,y1n]], [[x21,y21],…[x2n,y2n]], … [[xm1,ym1],…[xmn,ymn]]) – построение замкнутой области-полигона (многоугольника, так как последняя точка должна совпадать с первой);
- TEXT([x,y],'string',horizontal,vertical) – вывод текстовой надписи 'string', позиционированной в точке с координатами [x,y], с горизонтальной или вертикальной ориентацией. Параметр horizontal может иметь значения ALIGNLEFT или ALIGNRIGHT, указывающие, в какую сторону (влево или вправо) идет надпись. Аналогично параметр vertical может иметь значения ALIGNABOVE или ALIGNBELOW, указывающие, в каком направлении (вверх или вниз) идет надпись.

При задании графических объектов (структур) s1, s2, s3 и т. д. можно использовать описанные выше параметры графических объектов. Однако следует отметить, что параметры в графических структурах задаются несколько иначе – с помощью круглых скобок. Например, для задания шрифта TIMES ROMAN с размером символов 16 пунктов надо записать FONT(TIMES,ROMAN,16), для задания стиля координатных осей в виде прямоугольника – AXESSTYLE(BOX) и т. д.

На рис. 10.5 показан пример графических построений при использовании основных структур двумерной графики. Как видно из этого примера, графическая двумерная структура позволяет задавать практически любые двумерные графики и текстовые надписи в пределах одного рисунка.

10.4.2. Работа с графическими структурами трехмерной графики

Графические структуры трехмерной графики строятся функцией PLOT3D:

```
PLOT3D(s1,s2,s3,…,o)
```

В качестве элементарных графических структур можно использовать уже описанные выше объекты POINTS, CURVES, POLYGONS и TEXT – разумеется, с добавлением в списки параметров третьей координаты. Пример такого построения дан на рис. 10.6.

Maple 9.5 - [STR2D.MWS - [Server 1]]

File Edit View Insert Format Spreadsheet Window Help

```
> PLOT(POINTS([0,0.5],SYMBOL(CIRCLE)),
    TEXT([-1,0.4],``Структура 2D``,ALIGNBELOW,ALIGNRIGHT,
    FONT(HELVETICA,OBLIQUE,10)),
    CURVES([[-2,0.5],[0,1],[3,0.25]],THICKNESS(3)),
    TEXT([3.1415,0.6],'p',FONT(SYMBOL,12)),
    TEXT([-3.2,0.6],'P',FONT(SYMBOL,12)),
    POLYGONS([[-2,0],[-2,-.5],[2,-.5],[2,0]],COLOR(HUE,3)),
    TEXT([0,-0.25],``Красный``,COLOR(RGB,1,1,1)),
    AXESSTYLE(FRAME),VIEW(-4..4,-1..1));
```

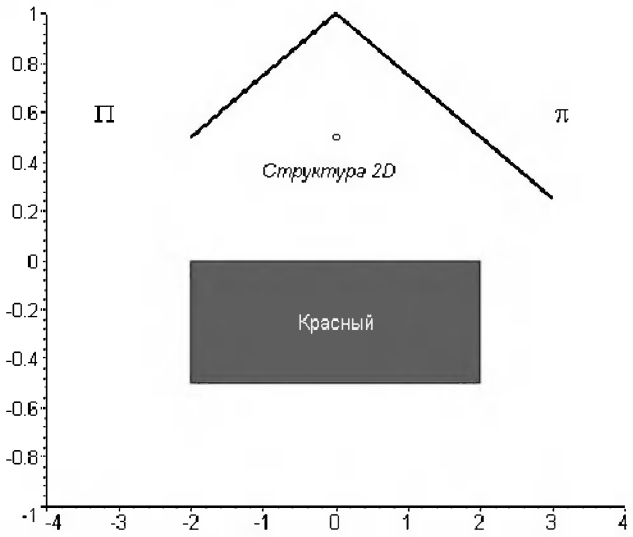


Рис. 10.5. Пример использования двумерных структур

Maple 9.5 - [CUBE.MWS - [Server 1]]

File Edit View Insert Format Spreadsheet Window Help

```
> PLOT3D(POLYGONS([[0,0,0],[1,0,0],[1,1,0],[0,1,0]],
                  [[0,0,0],[0,1,0],[0,1,1],[0,0,1]],
                  [[1,0,0],[1,1,0],[1,1,1],[1,0,1]],
                  [[0,0,0],[1,0,0],[1,0,1],[0,0,1]],
                  [[0,1,0],[1,1,0],[1,1,1],[0,1,1]],
                  [[0,0,1],[1,0,1],[1,1,1],[0,1,1]]),
  LIGHT(0,0,0.0,0.7,0.0), LIGHT(100,45,0.7,0.0,0.0),
  LIGHT(100,-45,0.0,0.0,0.7), AMBIENTLIGHT(0.4,0.4,0.4),
  TITLE(CUBE),STYLE(PATCH),COLOUR(ZHUE));
```

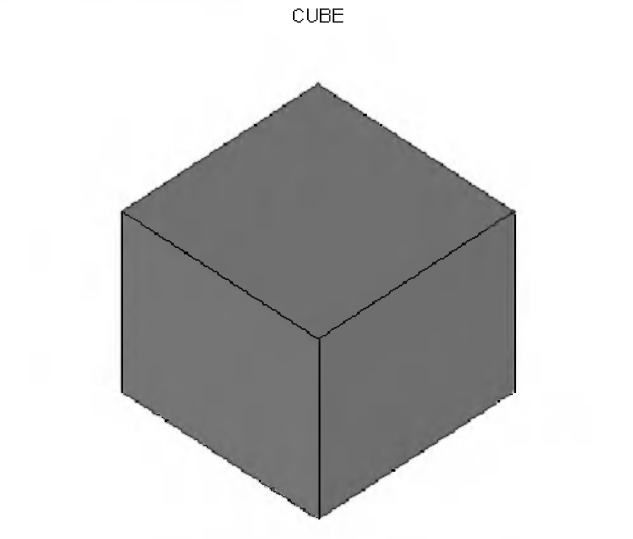


Рис. 10.6. Пример создания структуры трехмерной графики

Кроме того, могут использоваться некоторые специальные трехмерные структуры. Одна из них – структура GRID:

- GRID(a..b,c..d,listlist) – задание поверхности над участком координатной плоскости, ограниченной отрезками $[a, b]$ и $[c, d]$, по данным, заданным переменной-списком listlist:=[[z11,…z1n], [z21,…z2n], …[zm1…zmn]] с размерностью $n \times m$. Заметим, что эта переменная задает координату z для равноотстоящих точек поверхности.

Еще один тип трехмерной графической структуры – это MESH:

- MESH(listlist) – задание трехмерной поверхности по данным списочной переменной listlist, содержащей полные координаты всех точек поверхности (возможно задание последней при неравномерной сетке).

Обычная форма задания этой структуры следующая:

```
MESH([[x11,y11,z11],…[x1n,y1n,z1n]],
[[x21,y21,z21],…[x2n,y2n,z2n]], … [[xm1,ym1,zm1]…[xmn,ymn,zmn]])
```

Описанные структуры могут использоваться и в программных модулях. Много таких примеров описано в книгах, поставляемых с системой Maple.

10.5. Применение графики пакета plots

10.5.1. Общая характеристика пакета plots

Пакет plots содержит почти полсотни графических функций, существенно расширяющих возможности построения двумерных и трехмерных графиков в Maple. Для вызова полного списка функций этого пакета надо использовать команду:

```
> with(plots)
```

Она выводит список большого числа графических функций пакета:

- animate – создает анимацию двумерных графиков функций;
- animate3d – создает анимацию трехмерных графиков функций;
- animatecurve – создает анимацию кривых и т. д.

Среди этих функций надо отметить прежде всего средства построения графиков ряда новых типов (например, в виде линий равного уровня, векторных полей и т. д.), а также средства объединения различных графиков в один. Особый интерес представляют две первые функции, обеспечивающие анимацию как двумерных (animate), так и трехмерных графиков (animate3d).

Учитывая ограниченный объем и направленность данной книги, мы рассмотрим лишь несколько характерных примеров его применения. Заметим, что для использования приведенных функций нужен вызов пакета, например командой with(plots). К примеру, в пакете plots есть функция для построения двумерных (2D) графиков в полярной системе координат. Она имеет вид polarplot(L,o), где L – объекты для задания функции, график которой строится, и o – необязательные параметры.

10.5.2. Построение импликативных графиков

В математике часто встречается особый тип задания геометрических фигур, при котором переменные x и y связаны неявной зависимостью. Например, окружность задается выражением $x^2 + y^2 = R^2$, где R – радиус окружности. Для задания двумерного графика такого вида служит функция импликативной графики:

```
implicitplot(eqn,x=a..b,y=c..d,options)
```

Трехмерные поверхности также могут задаваться уравнениями *неявного вида*. В этом случае для построения их графиков используется функция `implicitplot3d`:

```
implicitplot3d(expr1,x=a..b,y=c..d,z=p..q,<options>)
implicitplot3d(f,a..b,c..d,p..q,<options>)
```

На рис. 10.7 показаны два примера (взяты из справки) построения любопытных объемных фигур с помощью функции `implicitplot3d`. С ее помощью можно строить весьма своеобразные фигуры, что, впрочем, видно и из приведенных примеров.

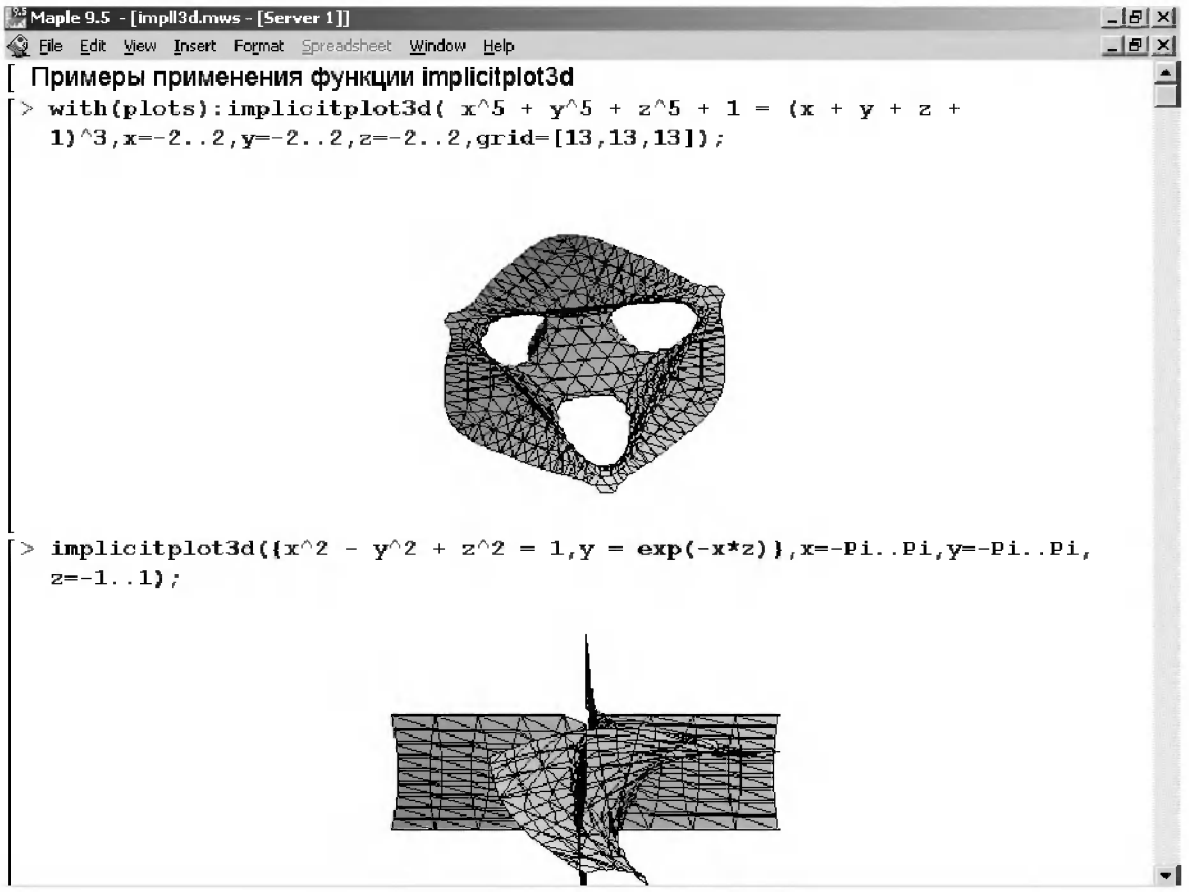


Рис. 10.7. Примеры применения функции implicitplot3d

10.5.3. Построение графиков линиями равного уровня

Графики, построенные с помощью линий равного уровня (их также называют контурными графиками), часто используются в картографии. Для построения таких графиков используется функция `contourplot`, которая может применяться в нескольких форматах:

```
contourplot(expr1,x=a..b,y=c..d)
contourplot(f,a..b,c..d)
contourplot([exprf,exprg,exprh ],s=a..b,t=c..d)
contourplot([f,g,h ],a..b,c..d)
contourplot3d(expr1,x=a..b,y=c..d)
contourplot3d(f,a..b,c..d)
contourplot3d([exprf,exprg,exprh],s=a..b,t=c..d)
contourplot3d([f,g,h ],a..b,c..d)
```

Здесь `f`, `g` и `h` – функции; `expr1` – выражение, описывающее зависимость высоты поверхности от координат x и y; `exprf`, `exprg` и `exprh` – выражения, зависящие от s и t, описывающие поверхность в параметрической форме; `a` и `b` – константы вещественного типа; `c` и `d` – константы или выражения вещественного типа; `x`, `y`, `s` и `t` – имена независимых переменных.

На рис. 10.8 показано построение графика линиями равного уровня для одной функции. Параметр `filled=true` обеспечивает автоматическую функциональ-

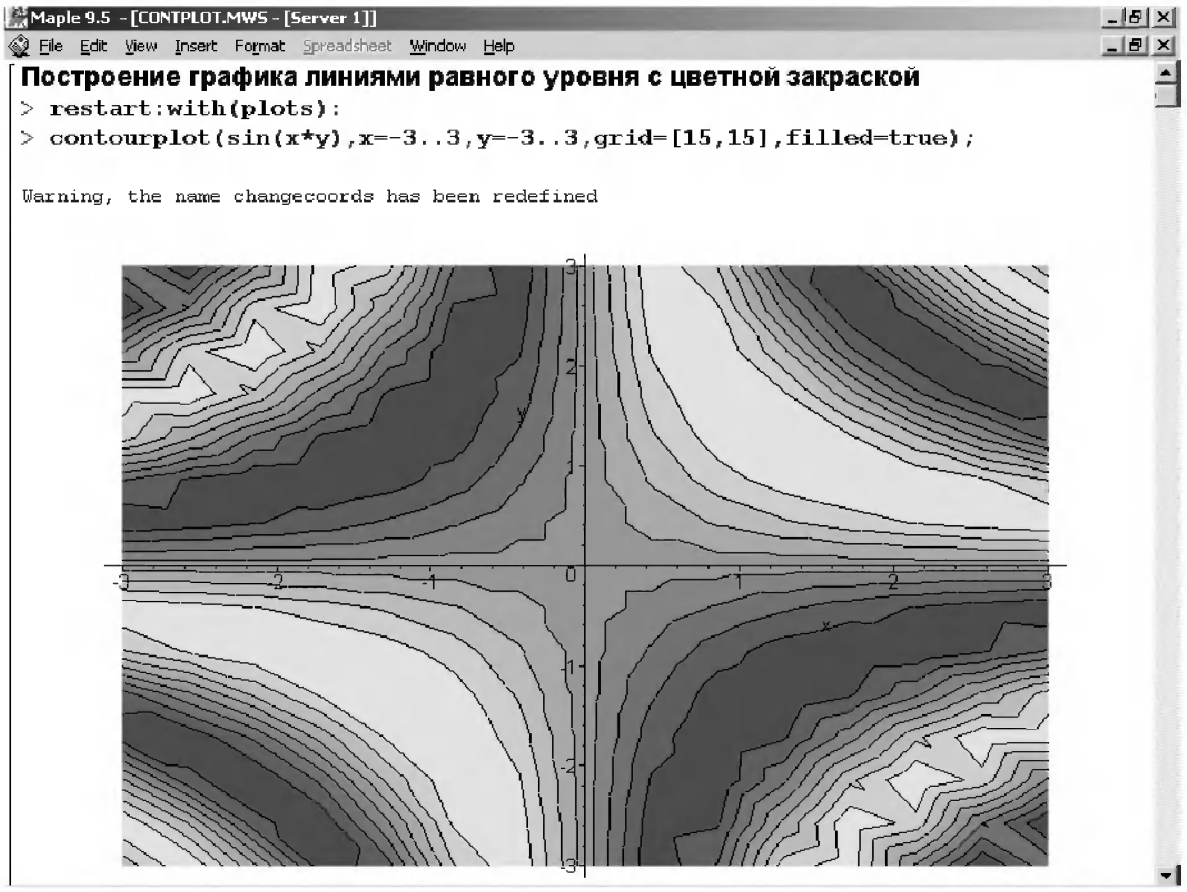


Рис. 10.8. Пример построения графика функции линиями равного уровня

ную окраску замкнутых фигур, образованных линиями равного уровня. Порой это придает графику большую выразительность, чем при построении только линий равного уровня.

К сожалению, в данном варианте окраски сами контурные линии получаются черными и их невозможно отличить. Однако если убрать параметр `filled=true`, то контурные линии (и линии легенды) будут иметь разный цвет и легко различаться.

Следует отметить, что хотя графики в виде линий равного уровня выглядят не так эстетично и естественно, как обычные графики трехмерных поверхностей (ибо требуют осмысления результатов), у них есть один существенный плюс – экстремумы функций на таких графиках выявляются порой более четко, чем на обычных графиках. Например, небольшая возвышенность или впадина за большой «горой» на обычном графике может оказаться невидимой, поскольку заслоняется «горой». На графике линий равного уровня этого эффекта нет. Однако выразительность таких графиков сильно зависит от числа контурных линий.

10.5.4. График плотности и векторного поля

Иногда поверхности отображаются на плоскости как *графики плотности* – чем выше высота поверхности, тем плотнее (темнее) окраска. Такой вид графиков создается функцией `densityplot`. Она может записываться в двух форматах:

`densityplot(expr1,x=a..b,y=c..d)` **`densityplot(f,a..b,c..d)`**,

где назначение параметров соответствует указанному выше для функции `contourplot`.

Еще один распространенный способ представления трехмерных поверхностей – *графики полей векторов*. Они часто применяются для отображения полей, например электрических зарядов. Особенность таких графиков в том, что для их построения используют стрелки, направление которых соответствует направлению изменения градиента поля, а длина – значению градиента. Так что термин «поле векторов» надо понимать в смысле, что поле графика заполнено векторами.

Для построения таких графиков в двумерной системе координат используется функция `fieldplot`:

`fieldplot(f, r1, r2)` **`fieldplot(f, r1, r2, …)`**,

где `f` – вектор или множество векторов, задающих построение; `r1` и `r2` – пределы.

Наглядность ряда графиков можно существенно увеличить, строя их в трехмерном представлении. Например, для такого построения графиков *полей из векторов* можно использовать графическую функцию `fieldplot3d`. В отличие от функции `fieldplot` она строит стрелки как бы в трехмерном пространстве (рис. 10.9). Возможности смены осей и оформления «ящика» графика иллюстрирует контекстное меню правой клавиши мыши, показанное на рис. 10.9.

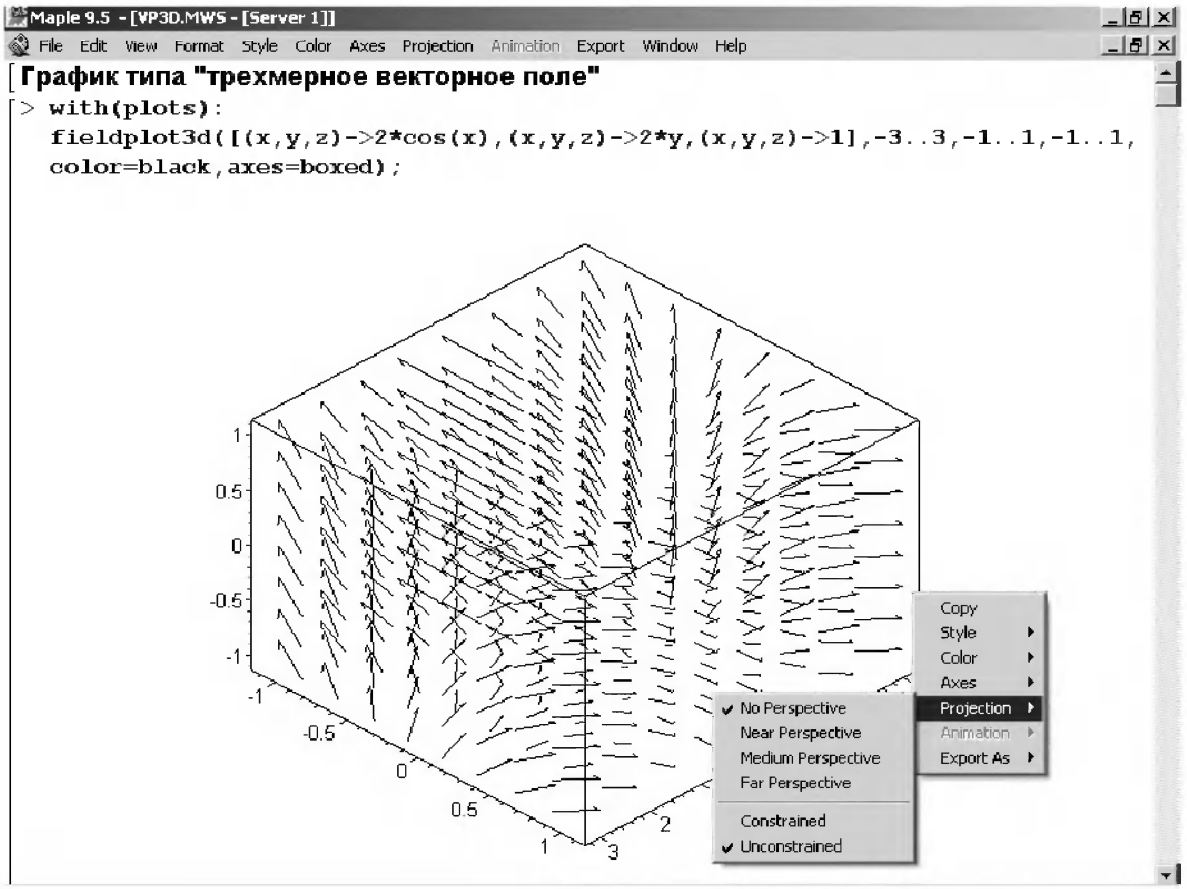


Рис. 10.9. Построение поля в трехмерном пространстве с помощью векторов

10.5.5. Контурные трехмерные графики

В отличие от векторных графиков, *контурные графики* поверхностей, наложенные на сами эти поверхности, нередко повышают восприимчивость таких поверхностей – подобно изображению линий каркаса. Для одновременного построения поверхности и контурных линий на них служит функция `contourplot3d`. Пример ее применения показан на рис. 10.10.

Для повышения наглядности этот график доработан с помощью контекстной панели инструментов графиков. В частности, включена функциональная окраска и подобраны углы обзора фигуры, при которых отчетливо видны ее впадина и пик. О возможностях переформатирования графика свидетельствует контекстное меню, показанное на рис. 10.10.

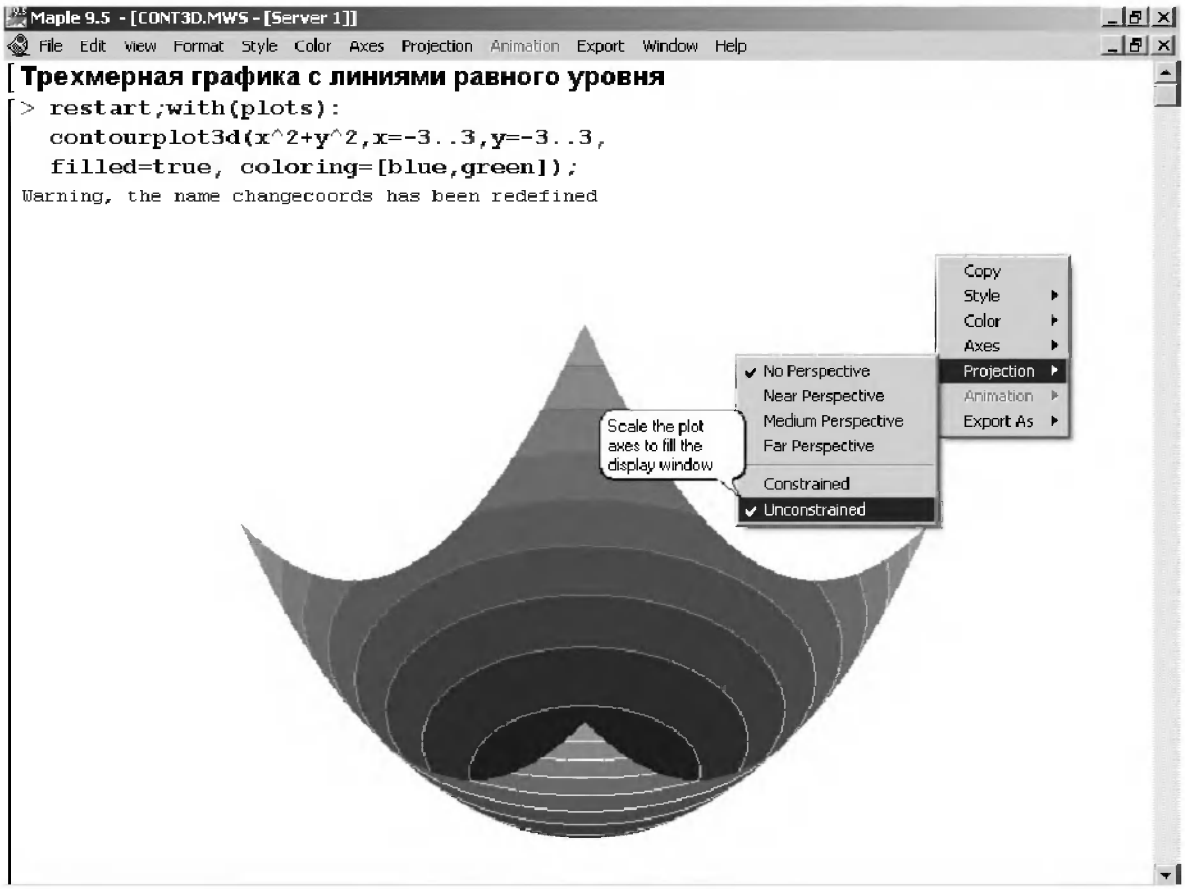


Рис. 10.10. График поверхности с контурными линиями

10.5.6. Визуализация сложных пространственных фигур

Рассмотрим еще пару примеров визуализации трехмерных фигур. Многие видели катушки индуктивности, у которых провод того или иного диаметра намотан на тороидальный магнитный сердечник. Некую математическую абстракцию такой катушки иллюстрирует рис. 10.11.

Наглядность таких графиков, как графики плотности и векторных полей, может быть улучшена их совместным применением. Такой пример показан на рис. 10.12.

Этот пример иллюстрирует использование «жирных» стрелок для обозначения векторного поля. Наглядность графика повышается благодаря наложению стрелок на график плотности, который лучше, чем собственно стрелки, дает представление о плавности изменения высоты поверхности, заданной функцией *f*.

10.5.7. Новая функция сравнения двух зависимостей от комплексного аргумента

В пакет Plots СКМ Maple 9.5 введена новая функция для сравнения двух зависимостей f(z) и g(z) комплексного аргумента z. Функция может использоваться в нескольких формах:

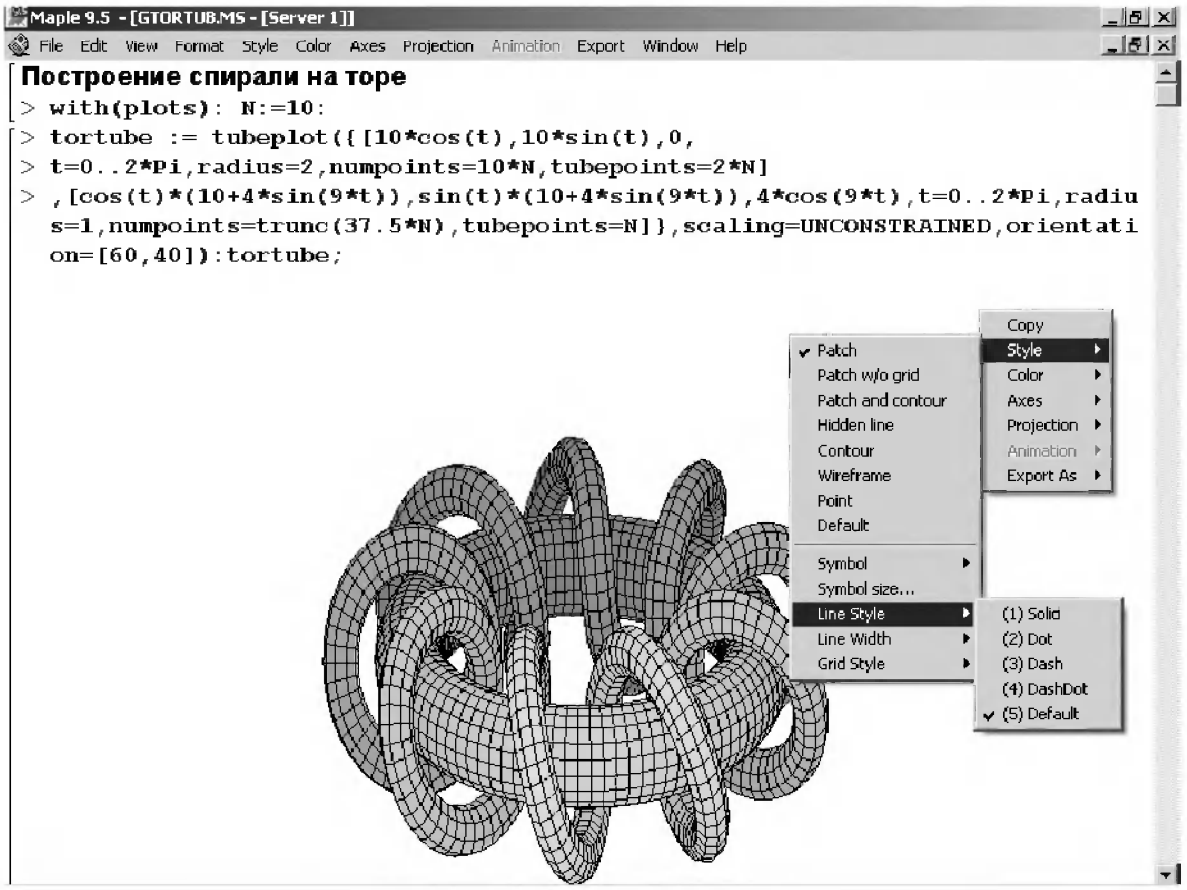


Рис. 10.11. Тор с обмоткой – толстой спиралью

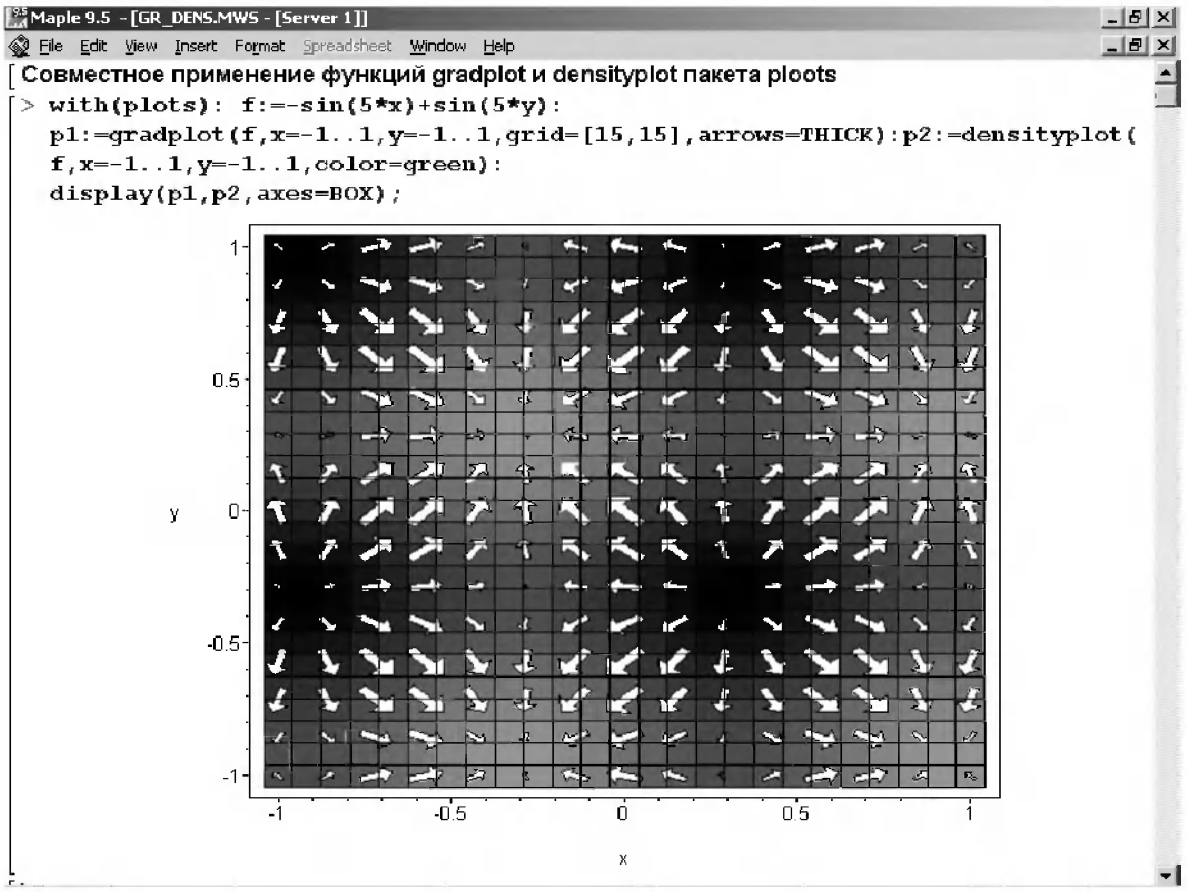


Рис. 10.12. Пример совместного применения графиков плотности и векторного поля

```
plotcompare(f(z), g(z), z = a+c*I..b+d*I, options);
plotcompare(f(z) = g(z), ...);
plotcompare(f, g, a+c*I..b+d*I, options);
plotcompare(f = g, ...);
```

Здесь a, b, c, d – константы реального типа. Функция на одном рисунке строит графики действительной и мнимой частей зависимостей f(z) и g(z). С помощью опций можно менять цветовую гамму рисунков, их ориентацию в пространстве и другие характеристики графиков. В справке по данной функции дается множество примеров ее применения, так что ограничимся одним, показанным на рис. 10.13.

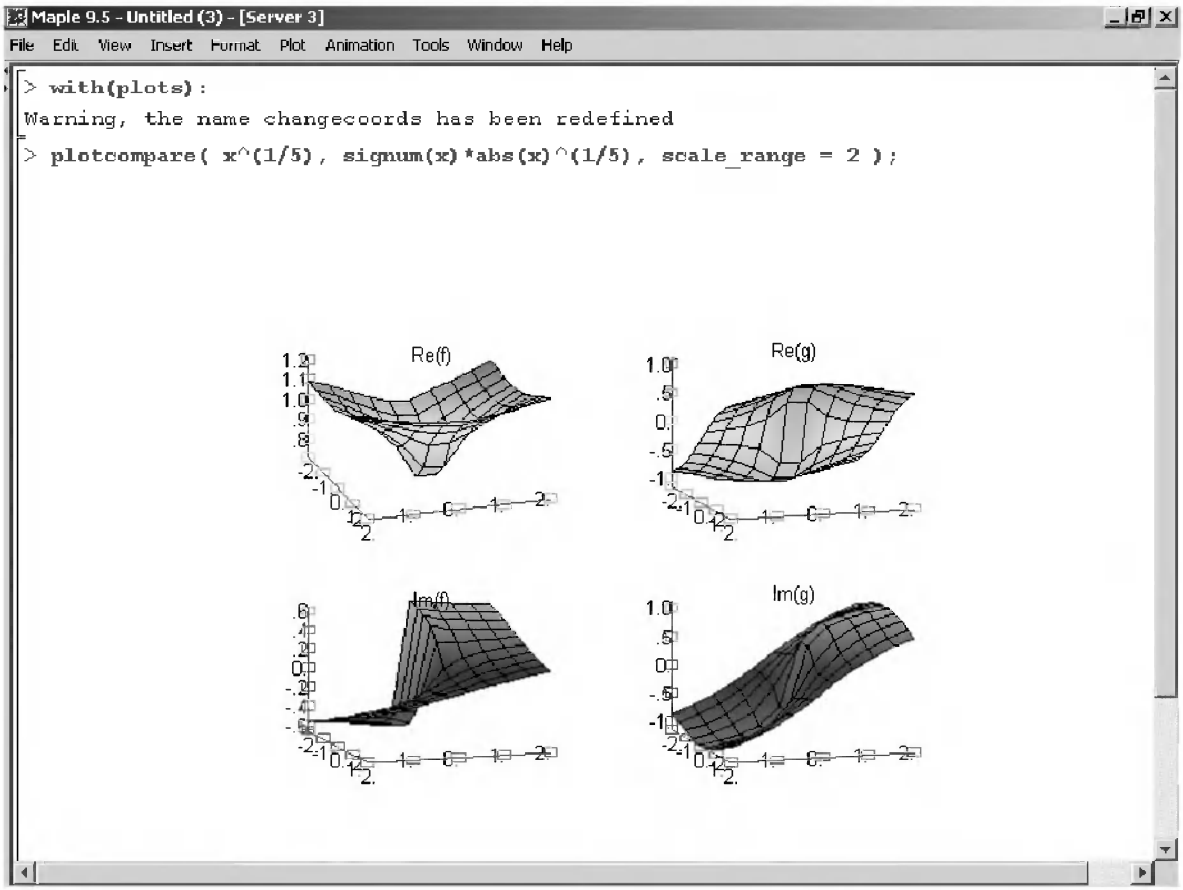


Рис. 10.13. Пример графического сопоставления двух зависимостей от комплексного аргумента

Сравнение графиков двух зависимостей, представленных на рис. 10.13, наглядно выявляет существенные отличия этих зависимостей. Достаточно отметить, что на графиках действительных частей зависимостей в одном случае видна выпуклая, а в другом случае – вогнутая поверхности. Еще сильнее отличия в графиках мнимых частей сопоставляемых зависимостей.

10.6. Динамическая графика

10.6.1. Анимация двумерных графиков

Визуализация графических построений и результатов моделирования различных объектов и явлений существенно повышается при использовании средств «оживления» (анимации) изображений. Пакет plots имеет две простые функции для создания динамических (анимированных) графиков.

Первая из этих функций служит для создания анимации графиков, представляющих функцию одной переменной $F(x)$:

```
animatecurve(F, r, …)
```

Эта функция просто позволяет наблюдать медленное построение графика. Формат ее применения подобен используемому в функции `plot`.

При вызове данной функции вначале строится пустой шаблон графика. Если активизировать шаблон мышью, то в строке главного меню появляется меню **Animation**. Меню **Animation** содержит команды управления анимацией. Такое же подменю появляется и в контекстном меню – см. рис. 10.14.

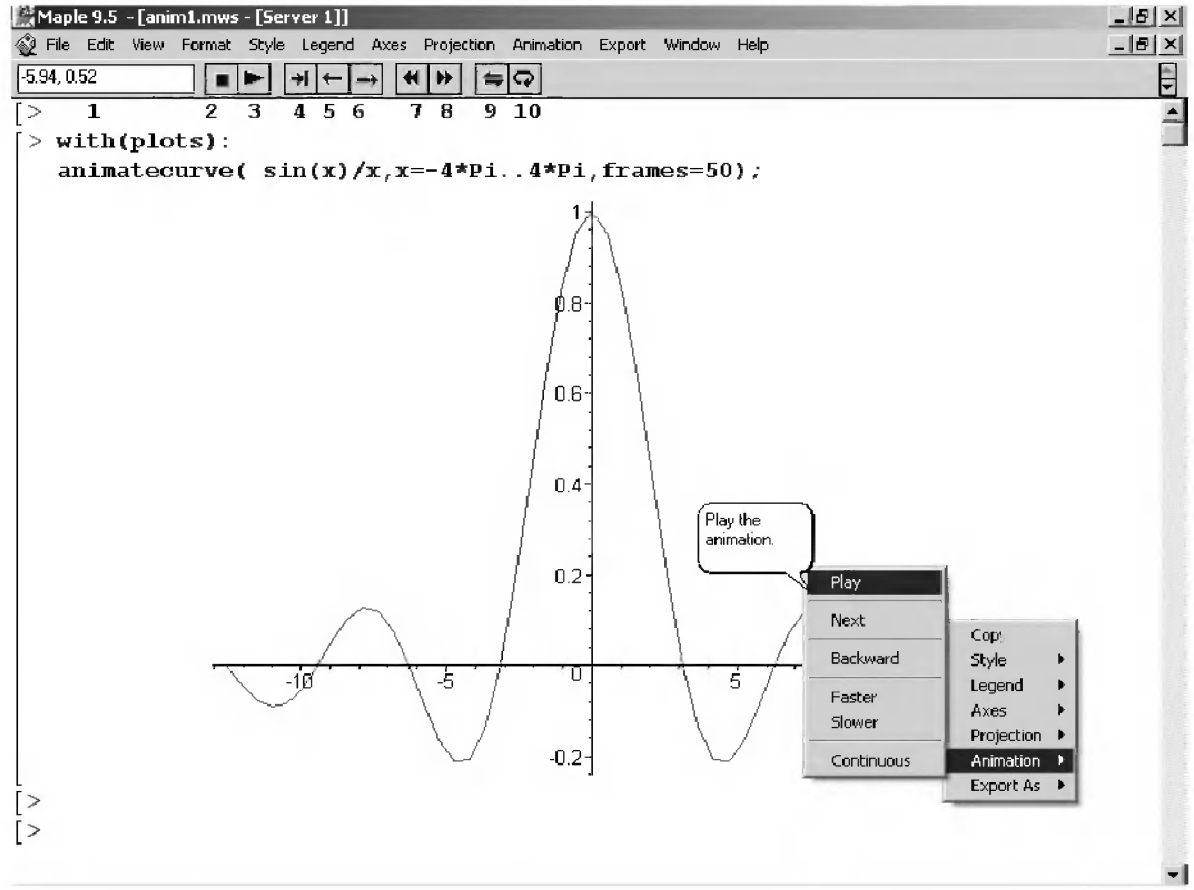


Рис. 10.14. Пример анимационного построения графика функцией animatecurve

Указанное подменю содержит следующие команды анимации:

- **Play** – запуск построения графика;
- **Next** – выполнение следующего шага анимации;
- **Backward/Forward** – переключение направления анимации (назад/вперед);
- **Faster** – ускорение анимации;
- **Slower** – замедление анимации;
- **Continiuus/Single cycle** – цикличность анимации.

При исполнении команды **Play** происходит построение кривой (или нескольких кривых) с помощью *проигрывателя анимационной графики*. В зависимости от выбора команд **Faster** или **Slower** построение идет быстро или медленно. Команда **Next** выполняет один шаг анимации – построение очередного фрагмента кривой. Переключатель **Backward/Forward** позволяет задать направление построения кривой – от начала к концу или от конца к началу. Построение может быть непрерывным или циклическим в зависимости от состояния позиции **Continiuus/Single cycle** в подменю управления анимацией. При циклической анимации число циклов задается параметром `frames=n`.

Более обширные возможности анимации двумерных графиков обеспечивает функция `animate`:

```
animate(F, x, t)        animate(F, x, t, o)
```

В ней параметр `x` задает пределы изменения переменной x, а параметр `t` – пределы изменения дополнительной переменной t. Суть анимации при использовании данной функции заключается в построении серии кадров (как в мультфильме), причем каждый кадр связан со значением изменяемой во времени переменной `t`. Если надо явно задать число кадров анимации `N`, то в качестве `o` следует использовать `frame=N`.

10.6.2. Анимация трехмерных графиков

Аналогичным образом может осуществляться и анимирование трехмерных фигур. Для этого используется функция `animate3d`:

```
animate3d(F,x, y,t,o)
```

Здесь `F` – описание функции (или функций); `x`, `y` и `t` – диапазоны изменения переменных x, y и t. Для задания числа кадров N надо использовать необязательный параметр `o` в виде `frame=N`. Примеры применения этой функции мы рассмотрим позже.

На рис. 10.15 показано построение графика с анимацией. После задания функции, график которой строится, необходимо выделить график и запустить проигрыватель, как это описывалось для анимации двумерных графиков.

На рис. 10.15 показано также контекстное меню поля выделенного графика. Нетрудно заметить, что с помощью этого меню (и содержащихся в нем подменю) можно получить доступ к параметрам трехмерной графики и выполнить необходимые операции форматирования, такие как включение цветовой окраски, выбор ориентации фигуры и т. д.

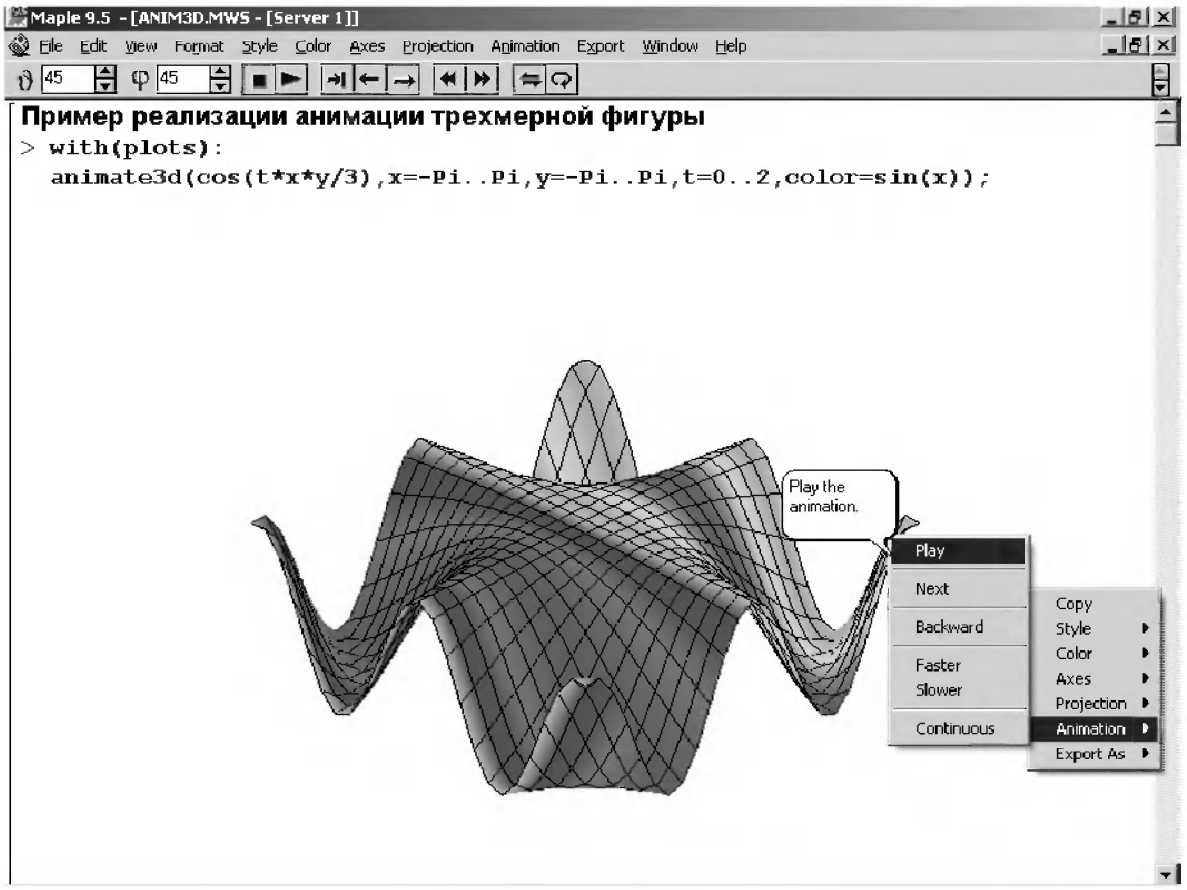


Рис. 10.15. Подготовка трехмерного анимационного графика

Назначение параметров, как и средств управления проигрывателем анимационных клипов, было описано выше.

Еще один путь получения анимационных рисунков – создание ряда графических объектов `p1`, `p2`, `p3` и т. д. и их последовательный вывод с помощью функций `display` или `display3d`:

```
display(p1,p2,p3,…,insequence=true)
display3d(p1,p2,p3…,insequence=true)
```

Здесь основным моментом является применение параметра `insequence=true`. Именно он обеспечивает вывод одного за другим серии графических объектов `p1`, `p2`, `p3` и т. д. При этом объекты появляются по одному, и каждый предшествующий объект стирается перед появлением нового объекта. Этот метод анимации мы рассмотрим чуть позже.

10.7. Графика пакета plottools

10.7.1. Примитивы пакета plottools

Инструментальный пакет графики plottools служит для создания графических примитивов, строящих элементарные геометрические объекты на плоскости и в пространстве: отрезки прямых и дуг, окружности, конусы, кубики и т. д. Его

применение позволяет разнообразить графические построения и строить множество графиков специального назначения. В пакет входят следующие графические примитивы:

```
arc          arrow        circle       cone         cuboid
curve        cutin        cutout       cylinder     disk
dodecahedron ellipse      ellipticArc  hemisphere   hexahedron
hyperbola    icosahedron  line         octahedron   pieslice
point        polygon      rectangle    semitorus    sphere
tetrahedron  torus
```

Их назначение ясно из названий. Вызов перечисленных примитивов осуществляется после загрузки пакета в память компьютера командой `with(plottools)`. Обычно примитивы используются для задания графических объектов, которые затем выводятся функцией `display`.

Большинство примитивов пакета plottools имеют довольно очевидный синтаксис. Например, для задания дуги используется примитив `arc(c, r, a..b,…)`, где `c` – список с координатами центра окружности, к которой принадлежит дуга, `r` – радиус этой окружности, `a..b` – диапазон углов. На месте многоточия могут стоять обычные параметры, задающие цвет дуги, толщину ее линии и т. д. Конус строится примитивом `cone(c,r,h…)`, где `c` – список с координатами центра, `r` – радиус основания, `h` – высота и т. д. Все формы записи графических примитивов и их синтаксис можно найти в справке системы Maple.

10.7.2. Примеры применения примитивов пакета plottools

Применение примитивов двумерной графики вполне очевидно, и читатель может придумать примеры этого сам. Во избежание искажений пропорций фигур надо согласовывать диапазон изменения переменной x. Обычно параметр `scalling=constrained` выравнивает масштабы и диапазоны по осям координат, что гарантирует отсутствие искажений у окружностей и других геометрических фигур. Однако при этом размеры графика нередко оказываются малыми. Напоминаем, что этот параметр можно задать и с помощью подменю **Projection**.

Аналогичным образом используются примитивы построения трехмерных фигур. Это открывает возможность создания разнообразных иллюстрационных рисунков и графиков, часто применяемых при изучении курса стереометрии. Могут строиться самые различные объемные фигуры и поверхности – конусы, цилиндры, кубы, полиэдры и т. д. Использование средств функциональной окраски делает изображения очень реалистичными.

На рис. 10.16 показано совместное построение двух пересекающихся кубов и сферы в пространстве. Нетрудно заметить, что графика пакета приблизительно (с точностью до сегмента) вычисляет области пересечения фигур. С помощью контекстно-зависимого меню правой кнопки мыши (оно показано на рис. 10.16) можно устанавливать условия обзора фигур, учитывать перспективу при построении и т. д. В частности, фигуры на рис. 10.23 показаны в перспективе.

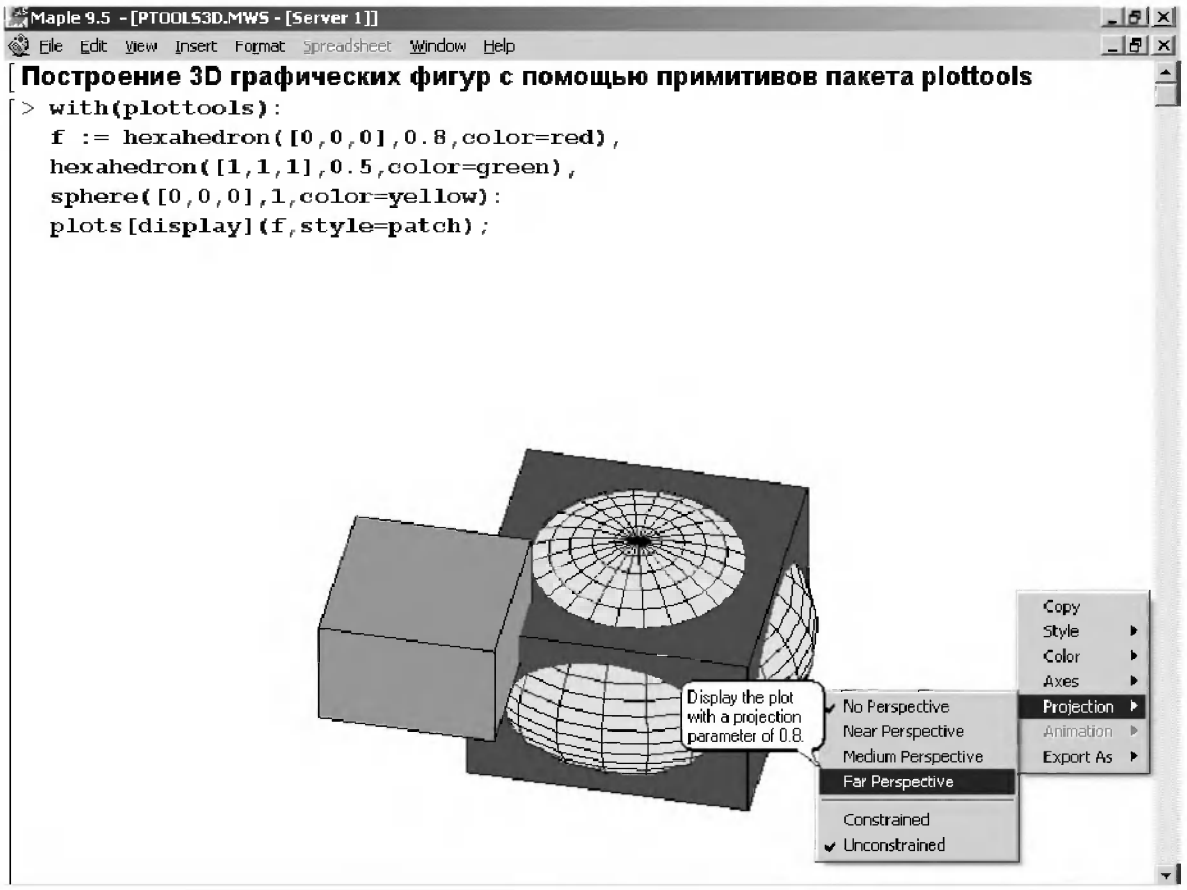


Рис. 10.16. Примеры применения примитивов трехмерной графики пакета plottools

С другими возможностями этого пакета читатель теперь справится самостоятельно или с помощью данных справочной системы.

10.7.3. Анимация в пакете plottols

Пакет plottools открывает возможности реализации анимационной графики. Мы ограничимся одним примером анимации двумерных графиков – рис. 10.17. В этом примере показана анимационная иллюстрация решения дифференциального уравнения, описывающего незатухающий колебательный процесс. Строится качающийся объект – стрелка с острием вправо, решение дифференциального уравнения в виде синусоиды и большая стрелка с острием влево, которая соединяет текущую точку графика синусоиды с острием стрелки колеблющегося объекта. Этот пример наглядно показывает возможности применения анимации для визуализации достаточно сложных физических и математических закономерностей.

Хорошим примером 3D-анимации является документ, показанный на рис. 10.18. Представленная на нем процедура `springPlot` имитирует поведение упругой системы, первоначально сжатой, а затем выстреливающей шар, установленный на ее верхней пластине. Упругая система состоит из неподвижного основания, на котором расположена упругая масса (например, из пористой резины), и верхней пластины.

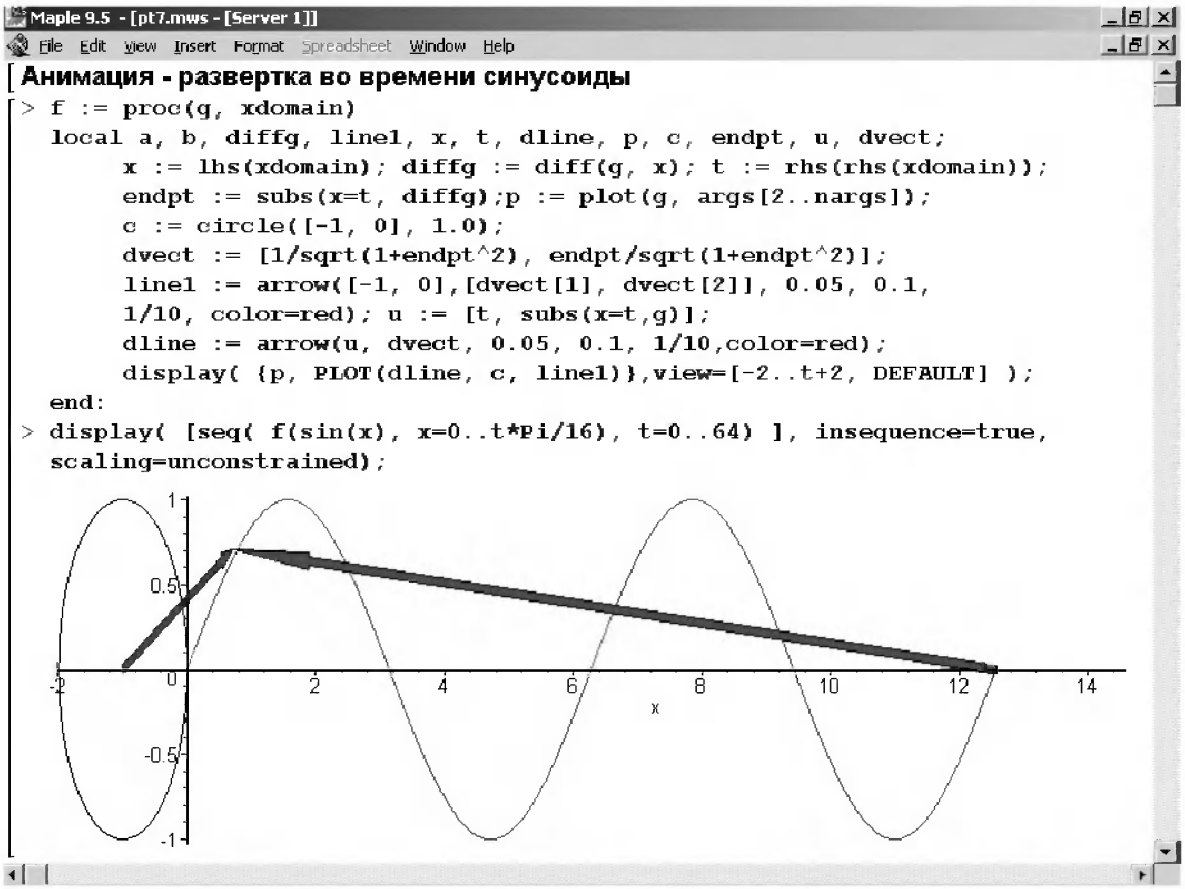


Рис. 10.17. Пример анимации двумерной графики

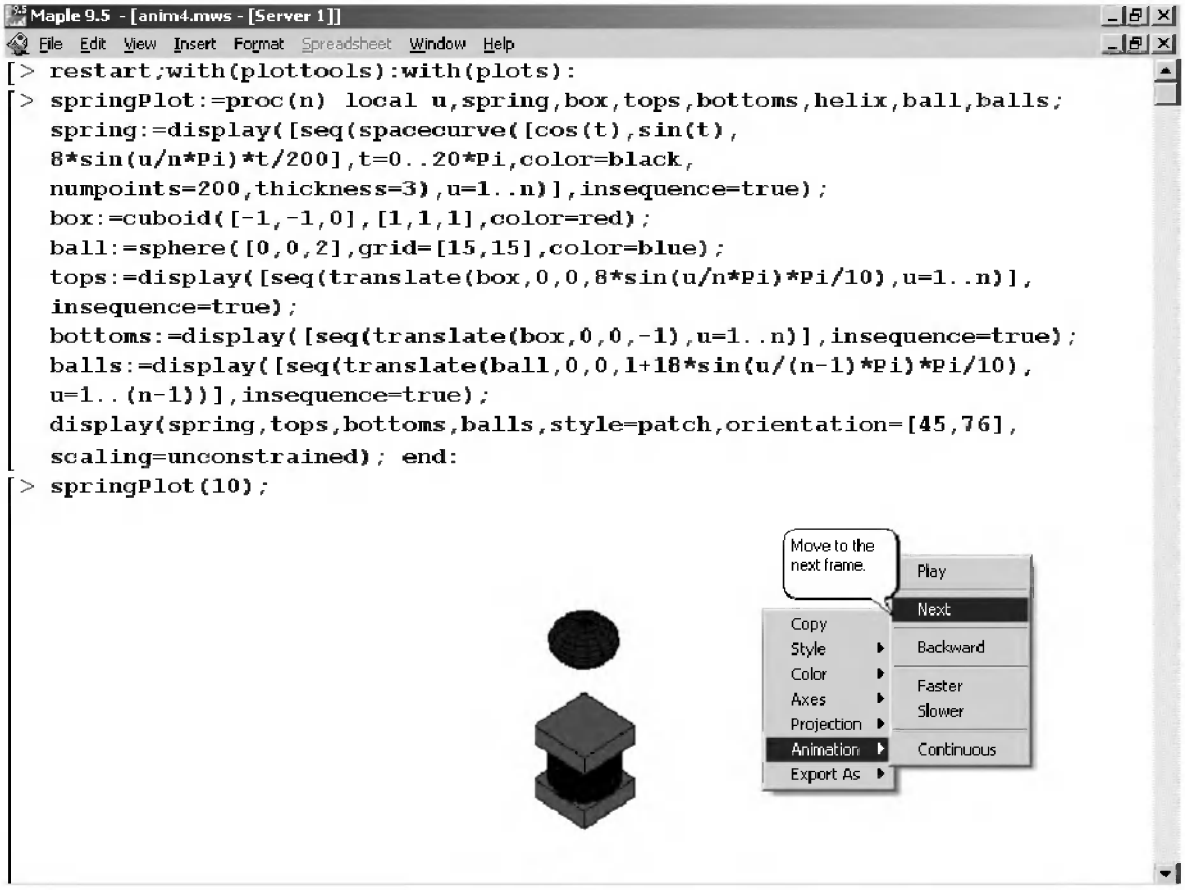


Рис. 10.18. Имитация отстрела шара сжатой упругой системой

Управление анимацией, реализованной средствами пакета `plottools`, подобно уже описанному ранее. Последний пример также прекрасно иллюстрирует возможности применения Maple при математическом моделировании различных явлений, устройств и систем.

10.8. Расширенные средства графической визуализации

10.8.1. Построение ряда графиков, расположенных по горизонтали

Обычно если в строке ввода задается построение нескольких графиков, то в строке вывода все они располагаются по вертикали. Это не всегда удобно. Применяя функции `plots` и `display`, можно разместить ряд двумерных графиков в строке вывода по горизонтали – см. рис. 10.19. Пример достаточно прост и нагляден.

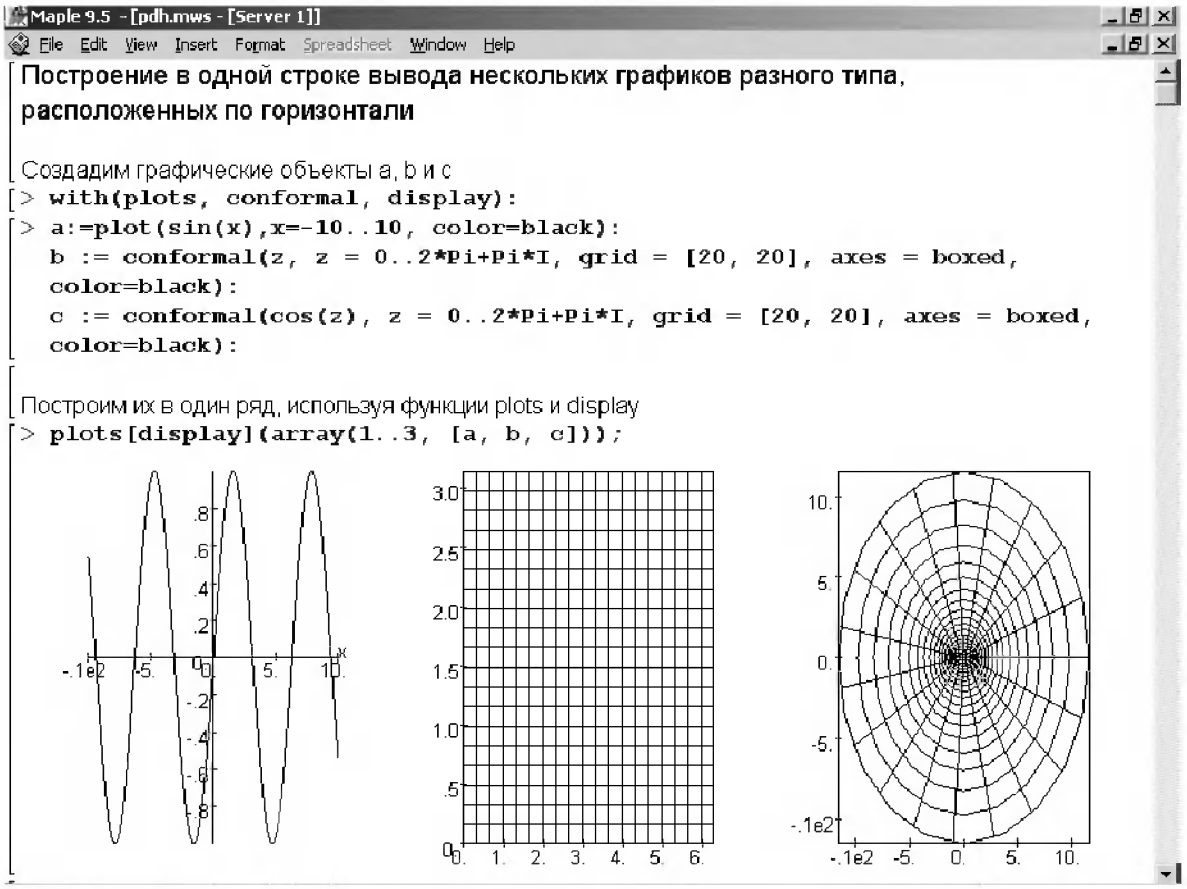


Рис. 10.19. Пример расположения трех графиков в строке вывода по горизонтали

10.8.2. Конформные отображения на комплексной плоскости

В пакете `plots` имеется функция для *конформных отображений*:

`conformal(F,r1,r2,o)`,

где `F` – комплексная процедура или выражение; `r1`, `r2` – области, задаваемые в виде `a..b` или `name=a..b`; `o` – управляющие параметры. Таким образом, для построения нужного графика достаточно задать нужное выражение и области изменения `r1` и `r2`. Пример построения конформных изображений для трех выражений дан на рис. 10.20.

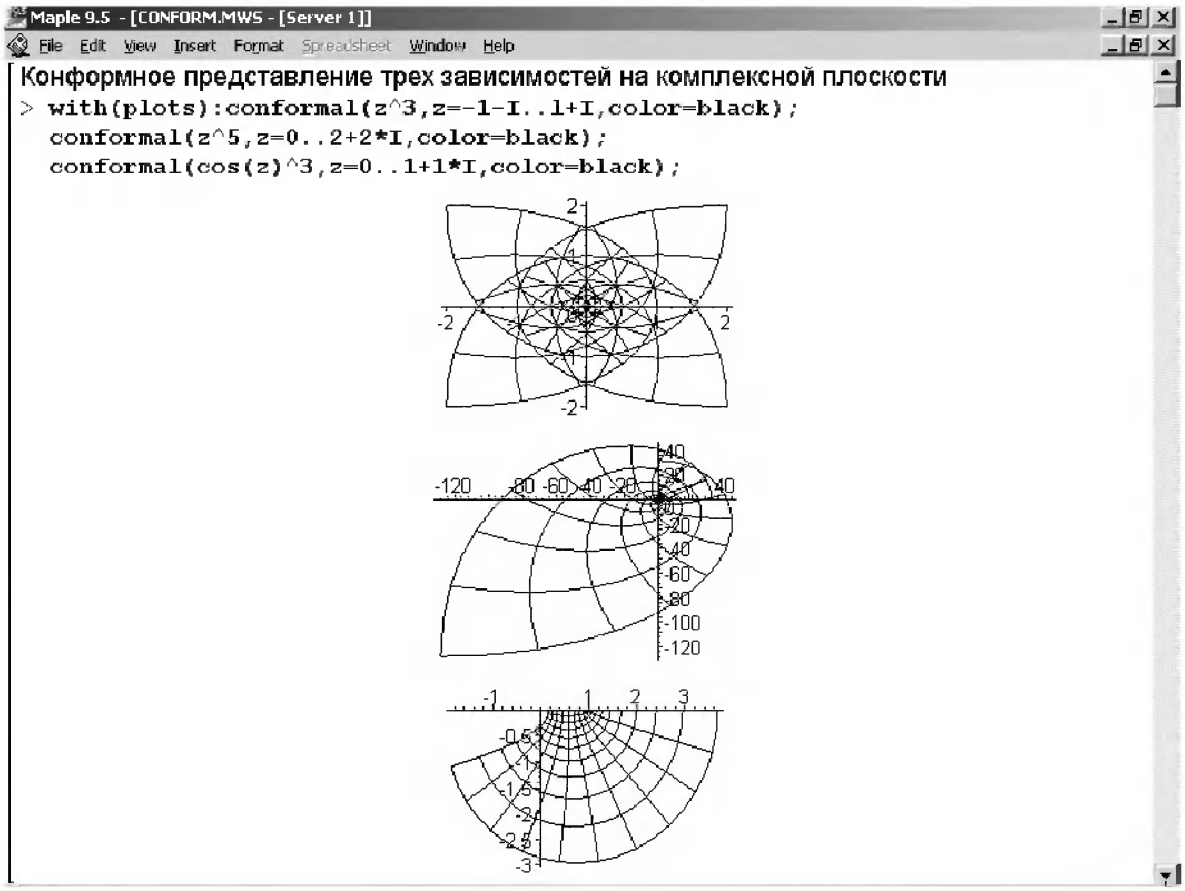


Рис. 10.20. Конформное отображение на комплексной плоскости графиков трех зависимостей

10.8.3. Визуализация поверхностей со многими экстремумами

Maple дает прекрасные возможности для визуализации поверхностей, имеющих множество пиков и впадин, другими словами, экстремумов. Рисунок 10.21 показывает задание «вулканической» поверхности с глубокой впадиной, окруженной

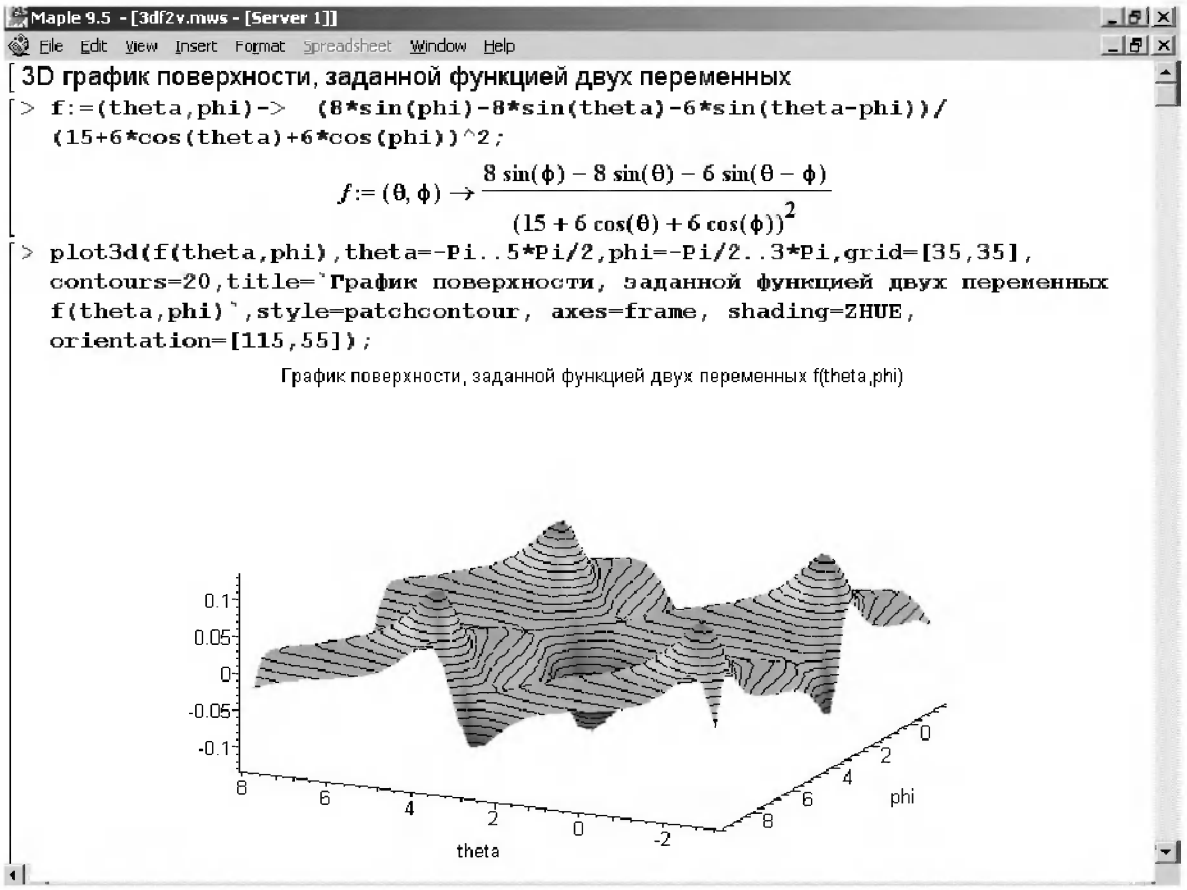


Рис. 10.21. Построение графика поверхности с множеством экстремумов

пятью пиками. Вид графика задается с помощью функции двух угловых переменных $f(\theta, \varphi)$.

При построении этого рисунка также используются функциональная окраска и построение контурных линий.

10.8.4. Визуализация решения систем линейных уравнений

Замечательной возможностью функции `solve` является возможность решения относительно ограниченного числа переменных. Например, систему линейных уравнений с переменными x, y, z, t и v можно решить относительно только первых трех переменных x, y и z. При этом решения будут функциями относительно переменных t и v и можно будет построить наглядный график решения (рис. 10.22).

На рис. 10.22 система задана пятью равенствами: `e1`, `e2`, `e3`, `e4` и `e5`. Затем функцией `solve` получено вначале решение для всех переменных (для иллюстрации), а потом для трех переменных x, y и z. Для получения решения в виде списка, а не множества, как в первом случае для всех переменных, использована функция подстановки `subs`. После этого функция `plot3d` строит плоскость решения в пространстве.

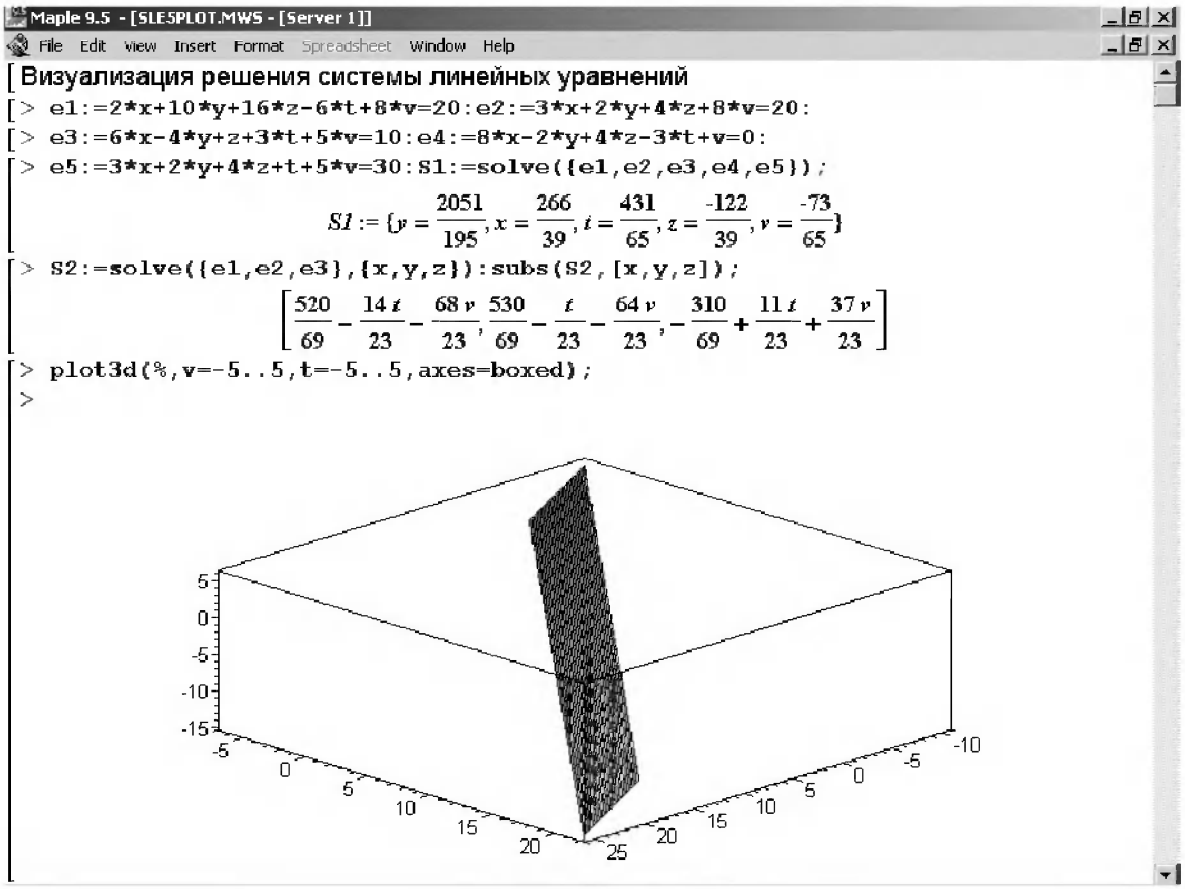


Рис. 10.22. График, представляющий решения системы линейных уравнений

10.8.5. Визуализация решения систем неравенств

Пожалуй, еще более полезным и наглядным средством является визуализация решения системы уравнений в виде неравенств. В пакете plots имеется специальная графическая функция `inequal`, которая строит все граничные линии неравенств и позволяет раскрасить разделенные ими области различными цветами:

```
inequal(ineqs, xspec, yspec, options)
```

Параметры этой функции следующие: `ineqs` – одно или несколько неравенств или равенств или список неравенств или равенств; `xspec` – `xvar=min_x..max_x`; `yspec` – `yvar=min_y..max_y`; `o` – необязательные параметры, например, указывающие цвета линий, представляющих неравенства или равенства, и областей, образованных этими линиями и границами графика. Пример применения этой функции представлен на рис. 10.23.

Обратите внимание на задание цветов: `optionsfeasible` задает цвет внутренней области, для которой удовлетворяются все неравенства (равенства), `optionsopen` и `optionsclosed` задают цвета открытых и закрытых границ областей графика, `optionsexcluded` используется для цвета внешних областей. График дает весьма наглядную интерпретацию действия ряда неравенств (или равенств).

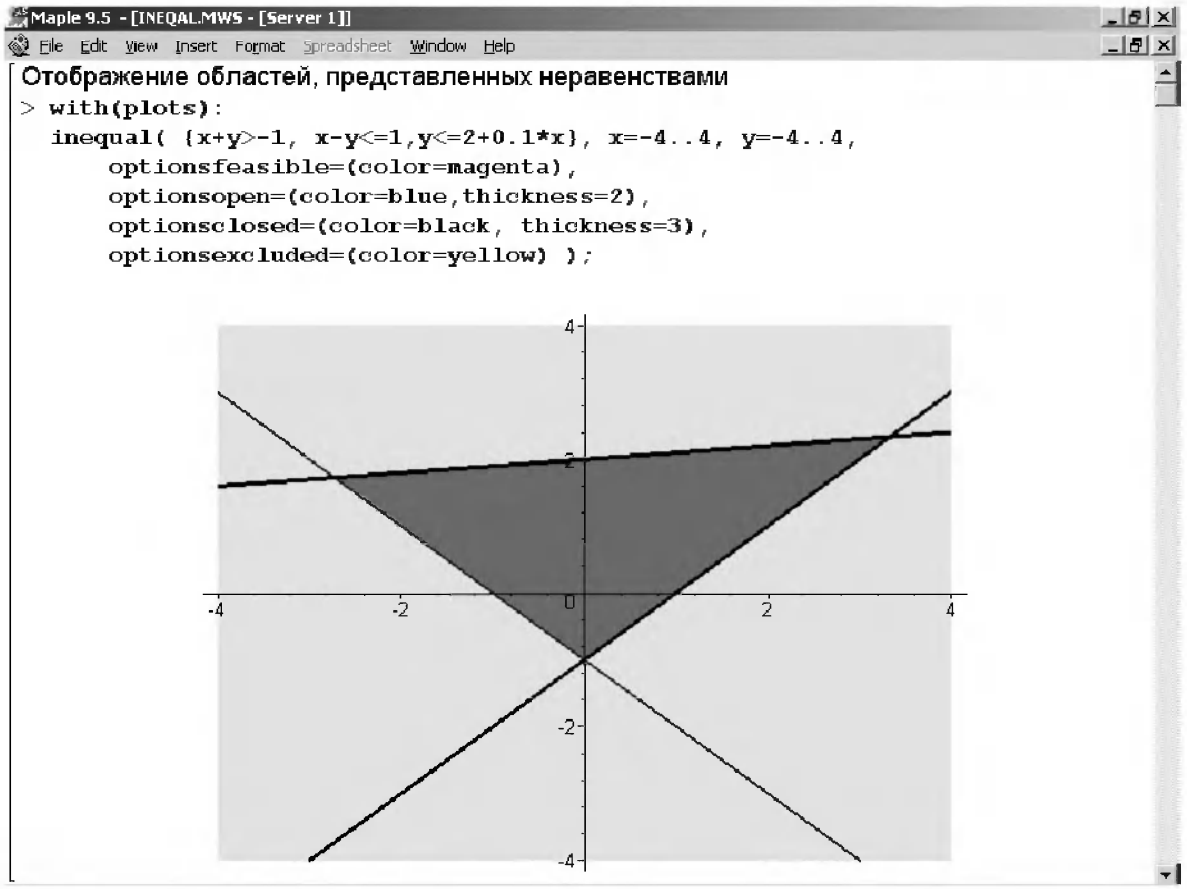


Рис. 10.23. Пример графической интерпретации решения системы неравенств

10.8.6. Иллюстрация итерационного решения уравнения f(x) = x

Классическим методом решения нелинейных уравнений является сведение их к виду $x = f(x)$ и применение метода простых итераций $x_k = s(x_{k-1})$ при заданном значении x_0. Приведем пример такого решения:

```
> f := x ->3*ln(x+1);
```

$$f := x \to 3\ln(x+1)$$

```
> x||0 := 0.5;
```

$$x0 := .5$$

```
> x0 := .5;
```

$$x0 := .5$$

```
> for k from 1 to 16 do x||k := evalf( f(x||(k-1) )); od;
```

$$x1 := 1.216395324$$
$$x2 := 2.387646445$$
$$x3 := 3.660406248$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$x15 := 5.711257755$$
$$x16 := 5.711359134$$

Нетрудно заметить, что значения x_k в ходе итераций явно сходятся к некоторому значению. Проведем проверку решения, используя встроенную функцию `solve`:

```
> f(x) = x; solve(%, x);
```

$$3\ln(x+1) = x$$

$$0, -3\,\mathrm{LambertW}\left(-1, -\frac{1}{3}\mathrm{e}^{(-1/3)}\right) - 1$$

Результат выглядит необычно – помимо довольно очевидного корня $x = 0$, значение другого корня получено в виде специальной функции Ламберта. Впрочем, нетрудно найти и его численное значение:

```
> evalf(%);
```

$$0., 5.711441084$$

К нему и стремятся промежуточные результаты решения. Однако как сделать процесс решения достаточно наглядным? Обычно для этого строят графики двух зависимостей – прямой x и кривой $f(x)$ – и наносят на них ступенчатую линию перемещения точки x_k. Специальной функции для графиков подобного рода Maple не имеет. Но можно составить специальную процедуру для их построения. Ее листинг, взятый из примера, описанного в пакете обучения системе Maple – `PowerTools`, —представлен на рис. 10.24.

На рис. 10.24 представлено задание процедуры **`rec_plot( f1, a, b, x0)`**.

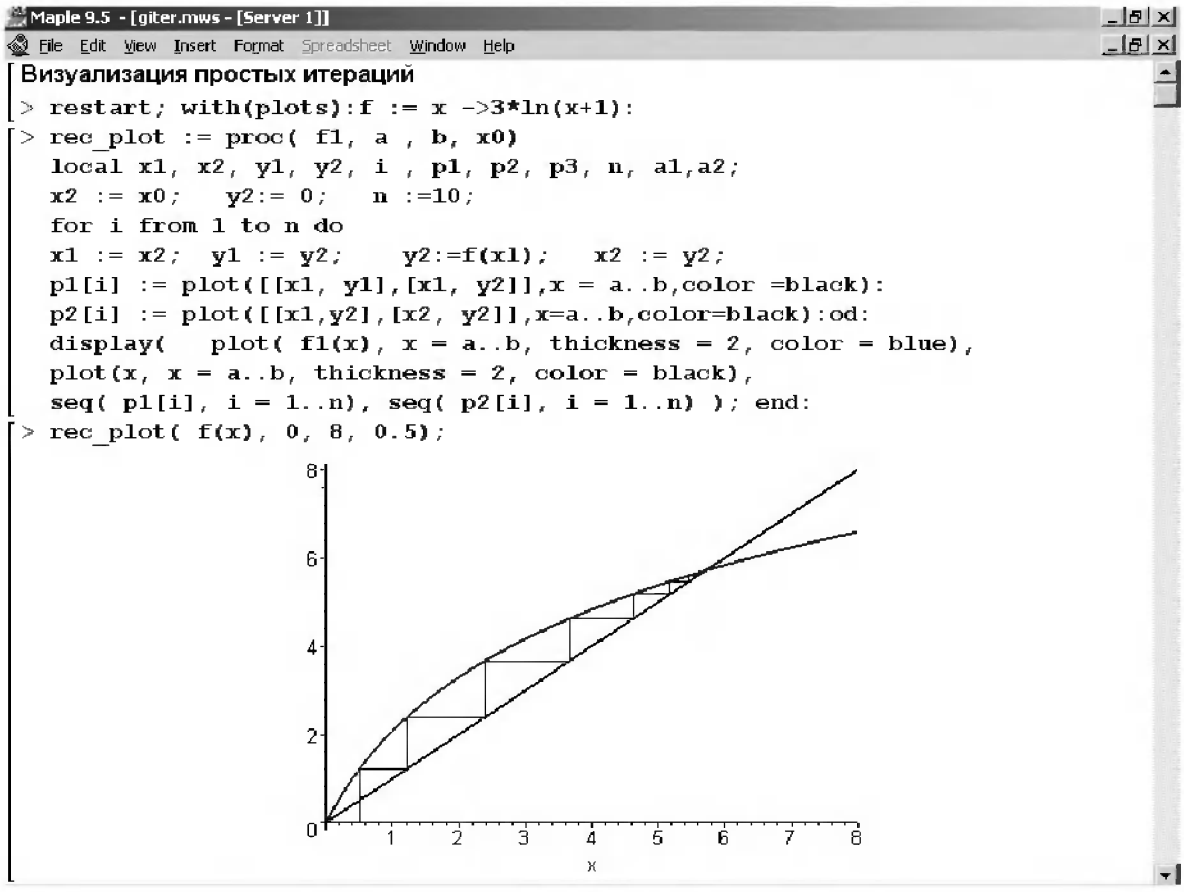


Рис. 10.24. Иллюстрация процесса итераций

Параметрами этой процедуры являются: `f1` – функция $f(x)$; `a` и `b` – пределы изменения x при построении графика; `x0` – значение x, с которого начинаются итерации. Используя эту процедуру, можно наблюдать график, иллюстрирующий итерационный процесс. Он представлен на рис. 10.33 снизу.

Нетрудно заметить, что для данной функции процесс итераций хотя и не очень быстро, но уверенно сходится к точке пересечения прямой $y = x$ и кривой $y = f(x)$. Вы можете, меняя зависимость $f(x)$, провести исследования сходимости уравнений $x = f(x)$.

10.8.7. Визуализация ньютоновских итераций в комплексной области

Теперь займемся довольно рискованным экспериментом – наблюдением *ньютоновских итераций* с их представлением на комплексной плоскости. На рис. 10.25 задана функция $f(z)$ комплексного аргумента. Проследить за поведением этой функции на комплексной плоскости в ходе ньютоновских итераций в соответствии с выражением z=f(z) позволяет графическая функция `complexplot3d` из пакета `plots`.

Наблюдаемая картина весьма необычна и свидетельствует о далеко не простом ходе итерационного процесса. А рискованной эта задача названа потому, что в старых версиях Maple она нередко вела к «зависанию» компьютера.

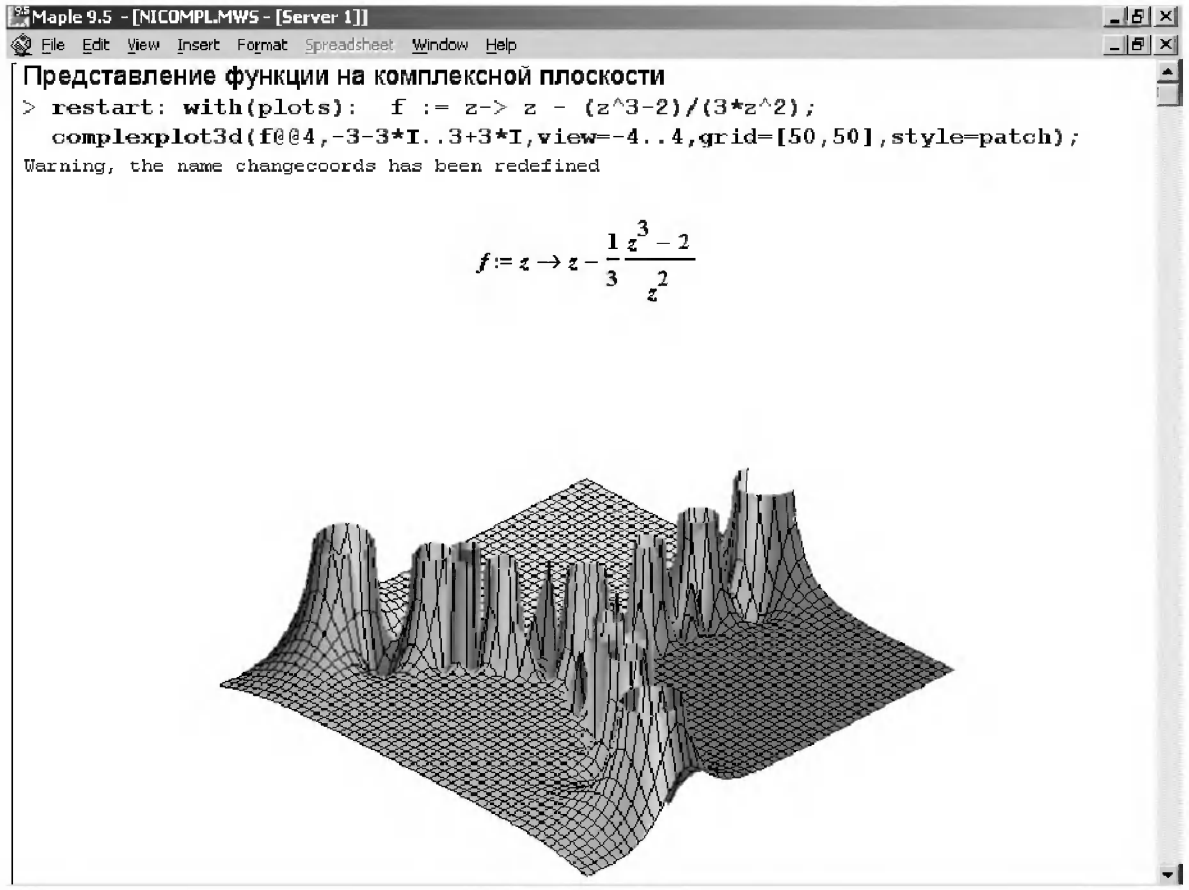


Рис. 10.25. Наблюдение за процессом ньютоновских итераций в трехмерном пространстве

10.8.8. Визуализация геометрических понятий

Средства Maple 9.5/10 весьма удобны для *визуализации геометрических построений*.

Примером наглядного геометрического представления математических понятий является визуализация известной *теоремы Пифагора*, которую моэно найти в справке по системе Maple.

В ряде геометрических построений нужно строить касательную и перпендикуляр к кривой, отображающей произвольную функцию $f(x)$ в заданной точке $x = a$. Рисунок 10.26 поясняет, как это можно сделать. Линии касательной $T(x)$ и перпендикуляра $N(x)$ определены аналитически через производную в заданной точке.

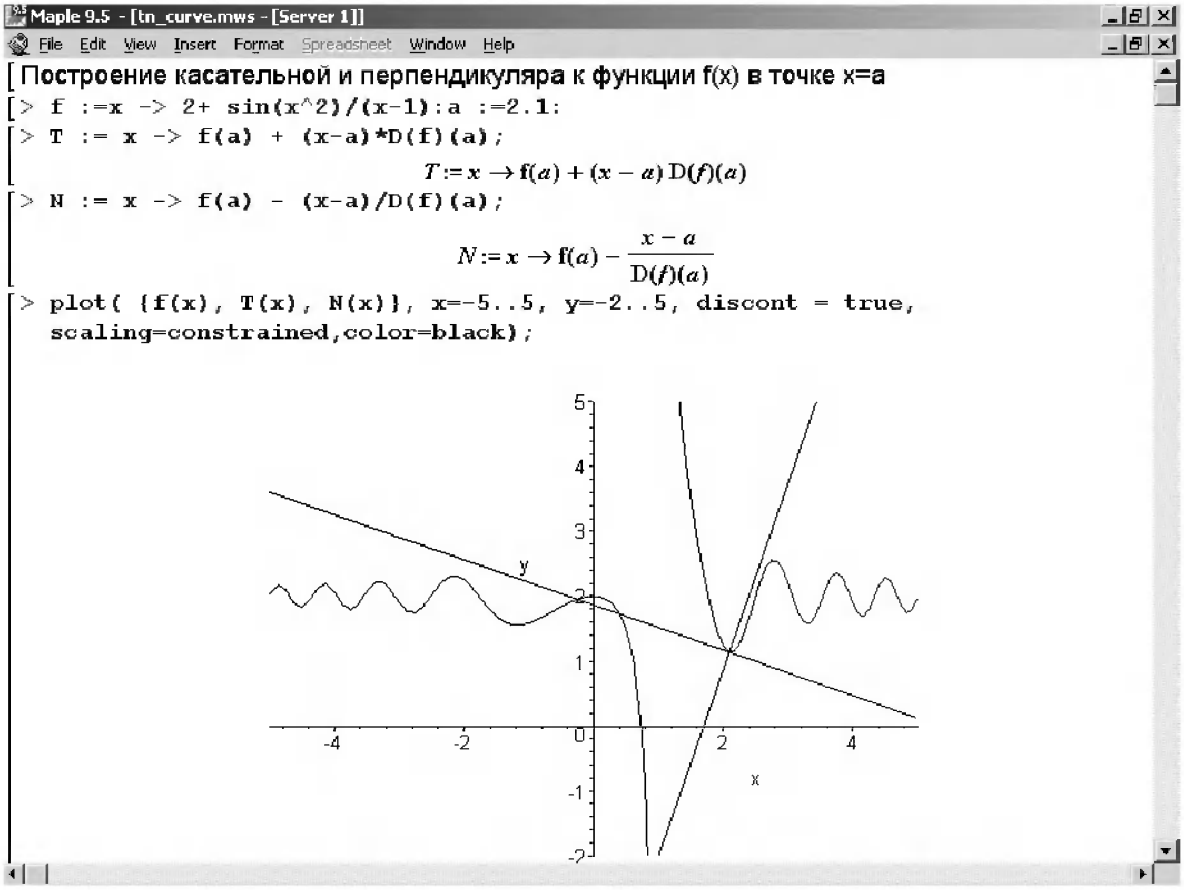


Рис. 10.26. Построение касательной и перпендикуляра к заданной точке графика функции f(x)

Во избежание геометрических искажений положения касательной и перпендикуляра при построении графика функцией `plot` надо использовать параметр `scaling=constrained`.

10.8.9. Визуализация вычисления определенных интегралов

Часто возникает необходимость в геометрическом представлении определенных интегралов в виде алгебраической суммы площадей, ограниченных кривой подынтегральной функции $f(x)$, осью абсцисс x и вертикалями $x = a$ и $x = b$ (пределами интегрирования). При этом желательно обеспечение закраски верхней и нижней (отрицательной и положительной) площадей разными цветами, например зеленым для верхней площади и красным для нижней. Как известно, численное значение определенного интеграла есть разность этих площадей.

На рис. 10.27 представлена процедура `a_plot`, решающая эту задачу. Параметрами процедуры являются интегрируемая функция $f(x)$ (заданная как функция пользователя), пределы интегрирования `a` и `b` и пределы слева `am` и справа `bm`, задающие область построения графика $f(x)$.

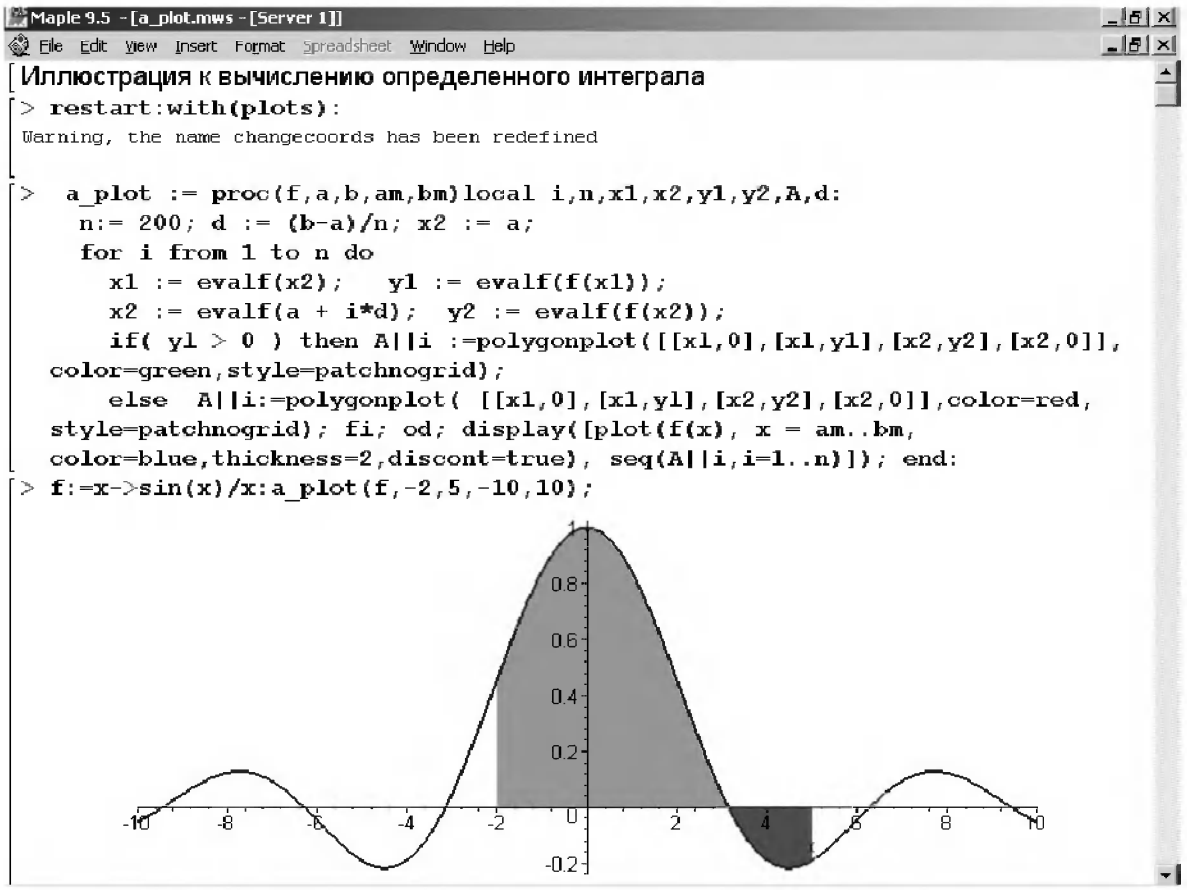


Рис. 10.27. Графическое представление определенного интеграла

Рисунок 10.27 дает прекрасное представление о сущности интегрирования для определенного интеграла. Приведенную на этом рисунке процедуру можно использовать для подготовки эффектных уроков по интегрированию разных функций.

10.8.10. Построение стрелок в пространстве

В пакет plots была введена новая функция построения стрелок в пространстве `arrow`. Она задается в виде:

`arrow(u,[v,]opts)` или **`arrow(U,opts)`**

Построение стрелок задается одномерными массивами координат начала стрелок и их направлениями `u` и `v` или двумерным массивом `U`, которые могут быть представлены векторами, списками или множествами. Вид стрелок задается параметром `opts`, который может иметь значения `shape`, `length`, `width`, `head_width`, `head_length` или `plane` и задает вид стрелок (форму, длину, ширину и т. д.). Детали задания параметров можно найти в справке по данной функции. Рисунок 10.28 дает наглядное представление о ее возможностях.

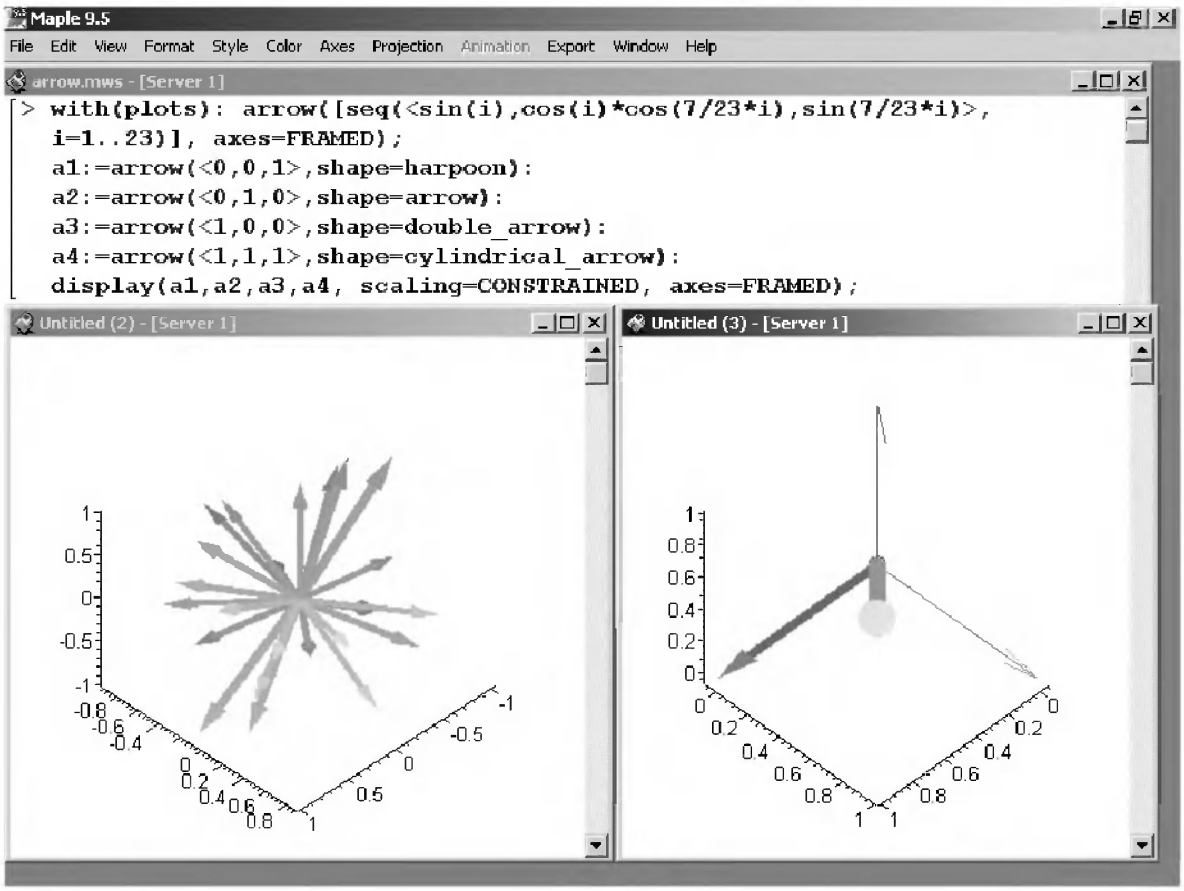


Рис. 10.28. Построение стрелок с помощью функции arrow

10.8.11. Построение сложных комбинированных графиков

Maple 9.5 позволяет строить достаточно сложные *комбинированные графики*, содержащие различные графические и текстовые объекты. Пример построения такого графика представлен на рис. 10.29.

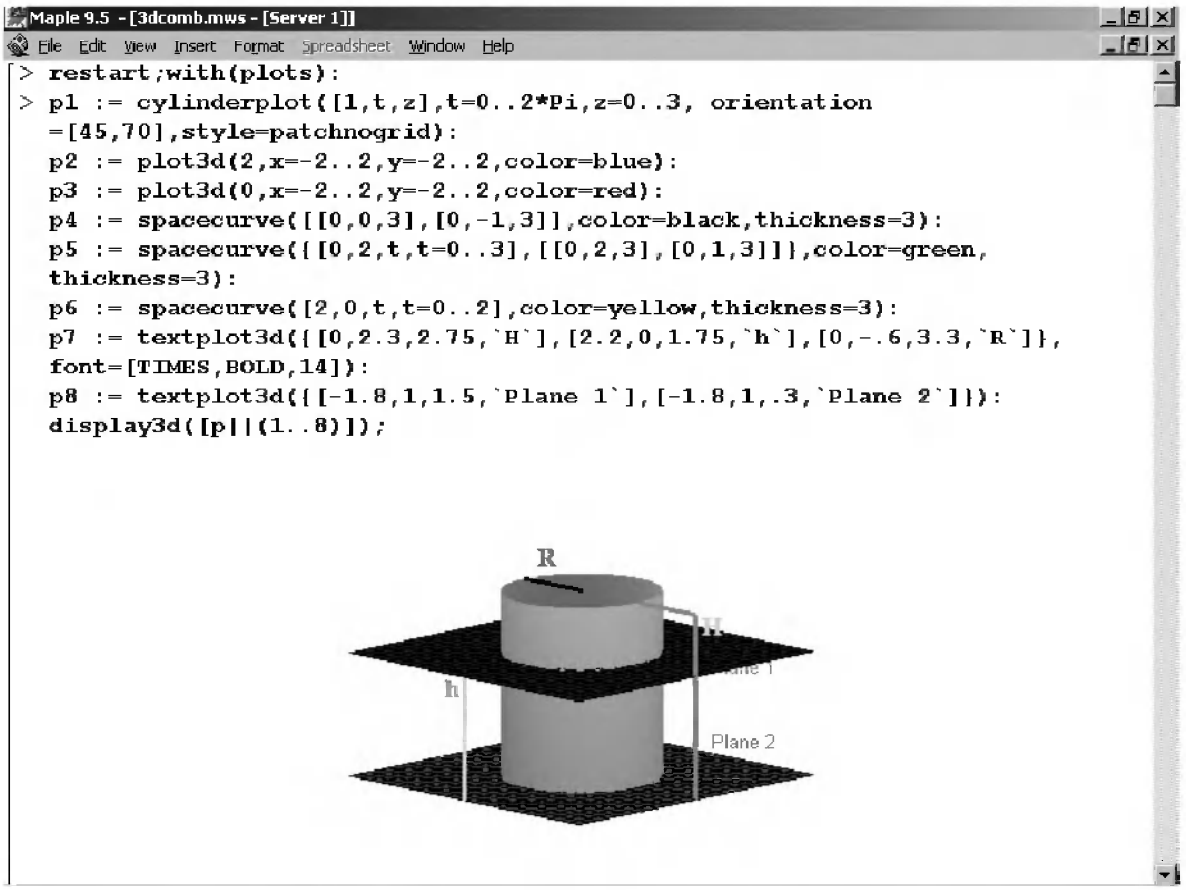


Рис. 10.29. Пример построения сложного объекта, состоящего из 8 графических и текстовых объектов

Представленный на рис. 10.29 объект задает построение восьми графических объектов от `p1` до `p10`. Среди них цилиндр, две пересекающие его плоскости и иные (в том числе текстовые) объекты. Обратите внимание на способ вывода этих объектов функцией display3d. Этот пример показывает, что с помощью графических программных средств Maple 9 можно строить достаточно замысловатые графики, которые могут использоваться для визуализации тех или иных геометрических и иных объектов.

10.8.12. Визуализация дифференциальных параметров кривых

Дифференциальные параметры функции $f(x)$, описывающей некоторую кривую, имеют большое значение для анализа ее особых точек и областей существования. Так, точки с нулевой первой производной задают области, где кривая нарастает (первая производная положительна) или убывает (первая производная отрицательна) с ростом аргумента x. Нули второй производной задают точки перегиба кривой.

Для такого анализа особенно удобен новый пакет Calculus 1, включенный в пакет расширения Student. На рис. 10.30 показано применение функции `FunctionChart` для визуализации дифференциальных параметров кривой, которая представляет

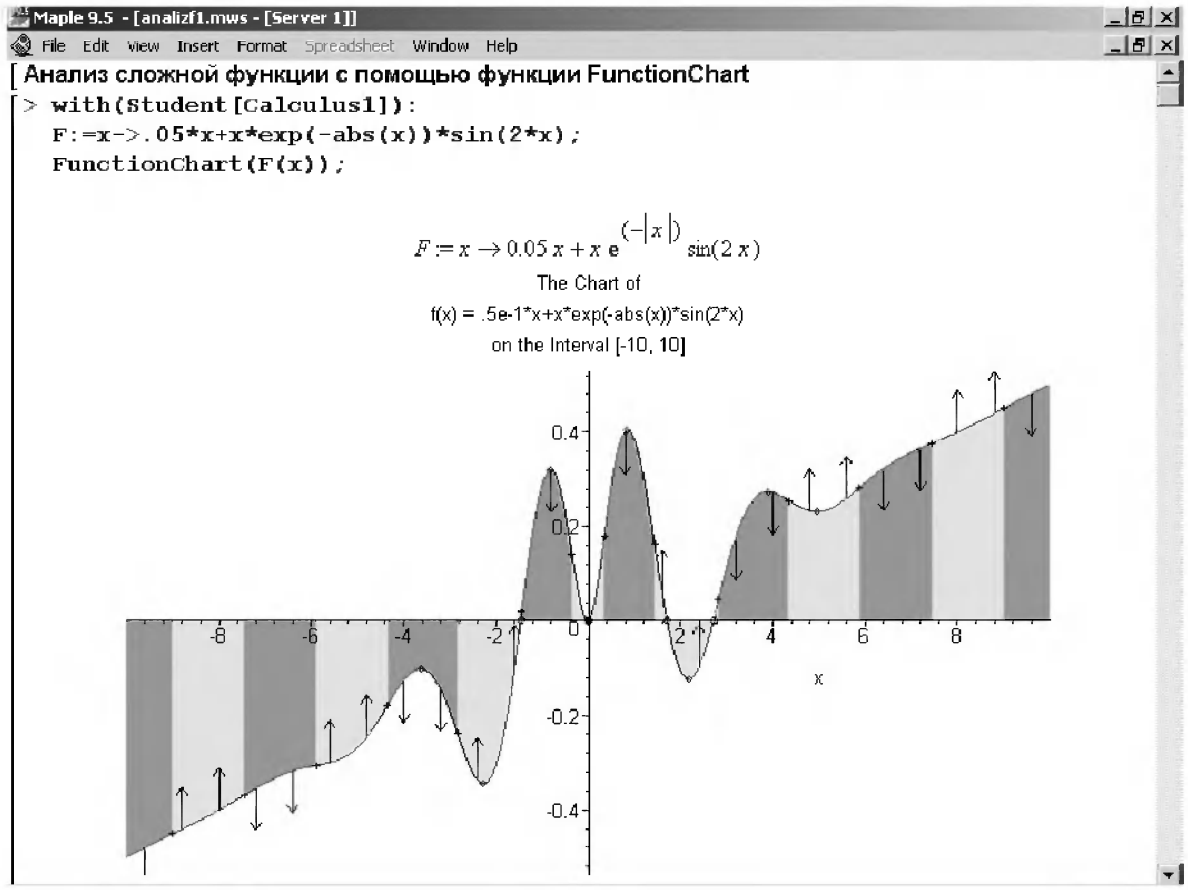


Рис. 10.30. Анализ и визуализация сложной функции, заданной функцией пользователя

собой сложную функцию. По умолчанию анализ ведется в интервале изменения x от –10 до +10. Экстремальные точки помечаются ромбиком, точки перегиба – крестиком, нули – кружочками, а области кривых – заливкой цветом.

Функция FunctionChart может использоваться с многочисленными опциями, существенно влияющими на вид рисунка. Визуализация функций с ее помощью весьма полезна в учебных целях при детальном изучении свойств той или иной функции.

10.8.13. Визуализация колебаний мембраны

В ряде случаев обычная техника анимации оказывается не очень подходящей из-за ограничений на выбор опций. Такова, например, ситуация, когда желательно обеспечить анимацию с большим числом кадров сложной поверхности, освещаемой от некоторого источника света. Пример такого рода представлен на рис. 10.31.

На рис. 10.32 задана упругая поверхность мембраны, закрепленной по периметру. Мембрана имеет ряд выпуклостей и впадин, переходящих друг в друга. Начальный вид мембраны представлен на рис. 10.32. Там же показано контекстное меню мыши, с помощью которой можно запустить анимацию, в том числе по кадрам. При этом можно наблюдать «оживление» колебаний мембраны.

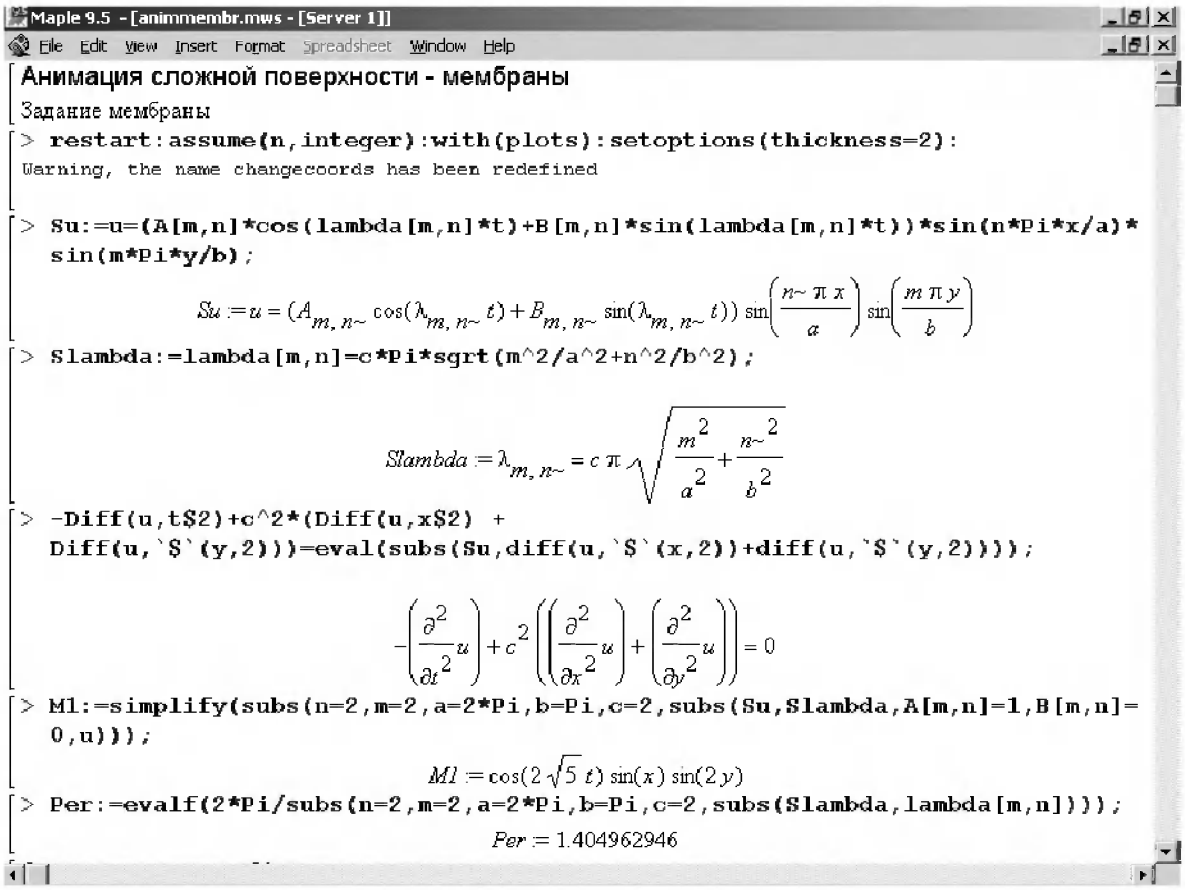


Рис. 10.31. Задание поверхности – мембраны

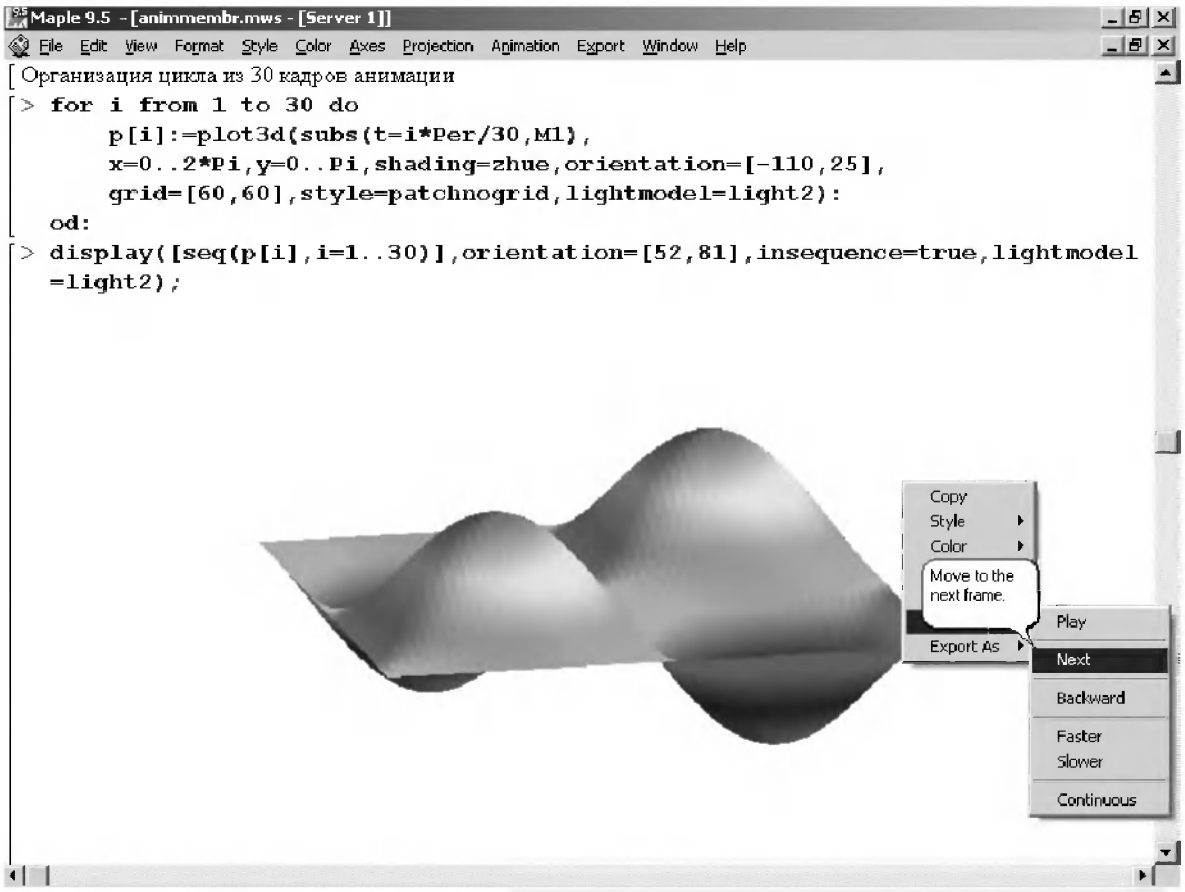


Рис. 10.32. Организация анимации мембраны и ее начальное положение

10.8.14. Визуализация экстремумов поверхности

Различные поля (электрические, гидравлические, гравитационные и иные) относятся к достаточно сложным понятиям, визуализация которых представляет значительные трудности в связи с большим объемом вычислений параметров поля, выполняемых во многих точках пространства с разными системами координат и отсутствием у людей органов для наблюдения полей.

В пакете VectorCalculus можно найти функции, которые совместно с графикой ряда других пакетов обеспечивают высокую степень визуализации полей с помощью стрелок, направления и размеры которых указывают на количественные оценки полей и изменения их градиентов в различных точках полей. Рисунок 10.33 иллюстрирует поиск трех локальных экстремумом поверхности, представленной функцией двух переменных. Экстремумы-минимумы ищутся с помощью функции fsolve по нулям частных производных по поверхности, вычисляемым функцией diff пакета VectorCalculus. Это расширяет методы поиска экстремумов функций двух переменных.

В найденных точках минимумов размещены черные сферы, что позволяет наглядно представить положение точек минимума поверхности. Правда, сферы выглядят как эллипсоиды, поскольку при выводе графиков выравнивание масштабов изображения по всем трем осям не предусматривалось.

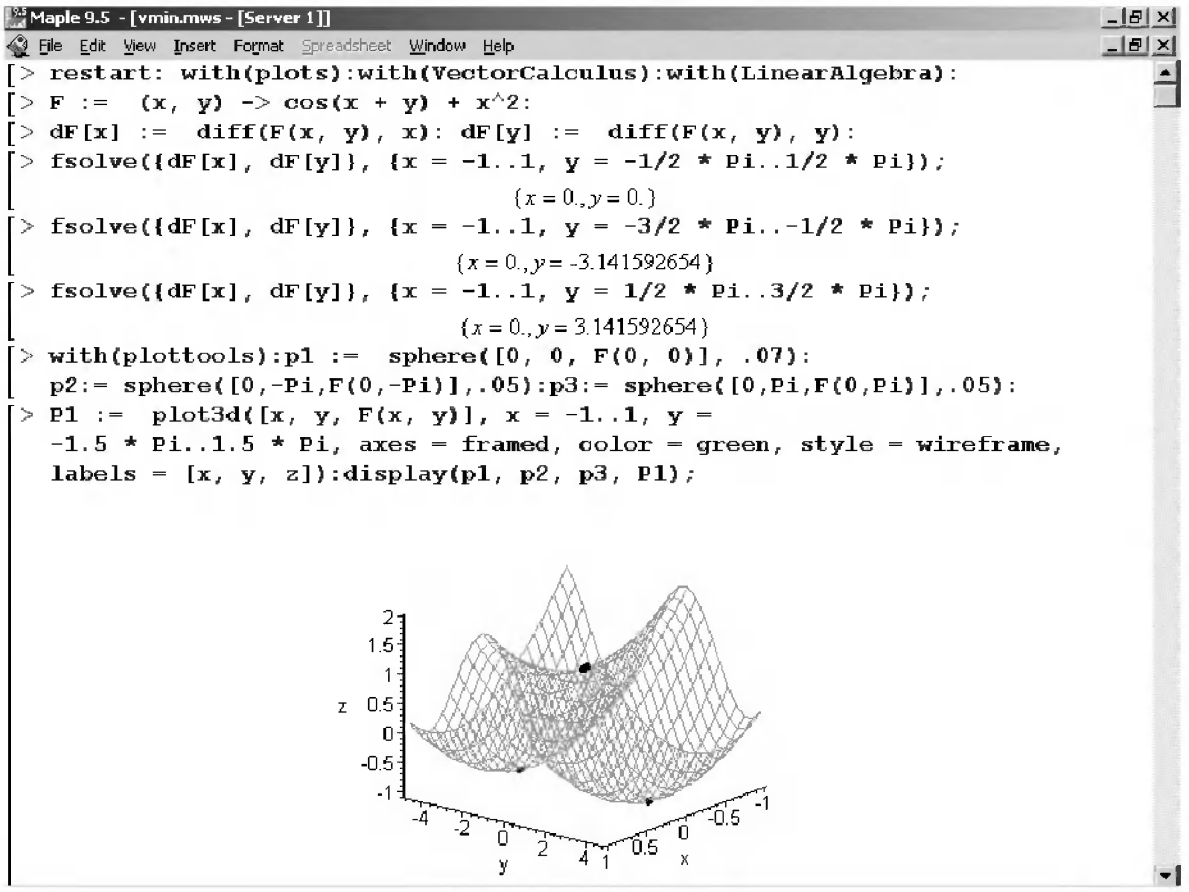


Рис. 10.33. Визуализация поиска локальных минимумов поверхности

10.8.15. Визуализация теоремы Стокса

С интегральными представлениями в теории поля связан ряд фундаментальных теорем Грина, Стокса, Остроградского и др. Средства пакета VectorCalculus открывают обширные возможности по наглядному представлению этих и иных положений теории поля. В качестве примера на рис. 10.34 представлена визуализация положений теоремы Стокса.

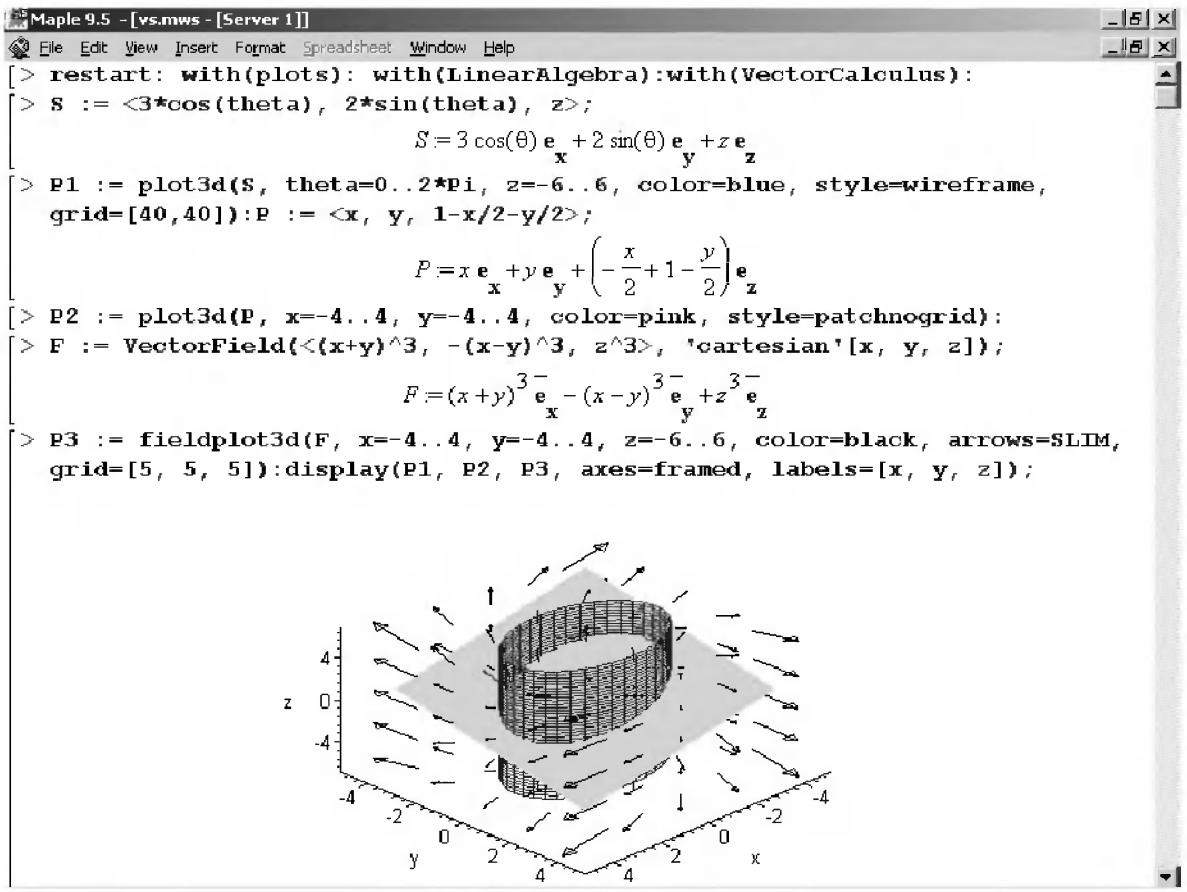


Рис. 10.34. Иллюстрация теоремы Стокса

10.8.16. Визуализация поля электрических зарядов

Рассмотрим поле двух точечных единичных зарядов, расположенных на расстоянии 0,25 м друг от друга. Нас интересует визуализация картины электрического поля между ними для случая, когда заряды равны по величине и противоположны по знаку. Воспользуемся упрощенной формулой для вычисления потенциала любой точки (x,y), из которой удален множитель $1/4\pi\varepsilon$. Рисунок 10.35 показывает применение этой формулы для построения трехмерного графика распределения потенциала для двух зарядов (положительного и отрицательного), расположенных на расстоянии 0,25 (единицы условные).

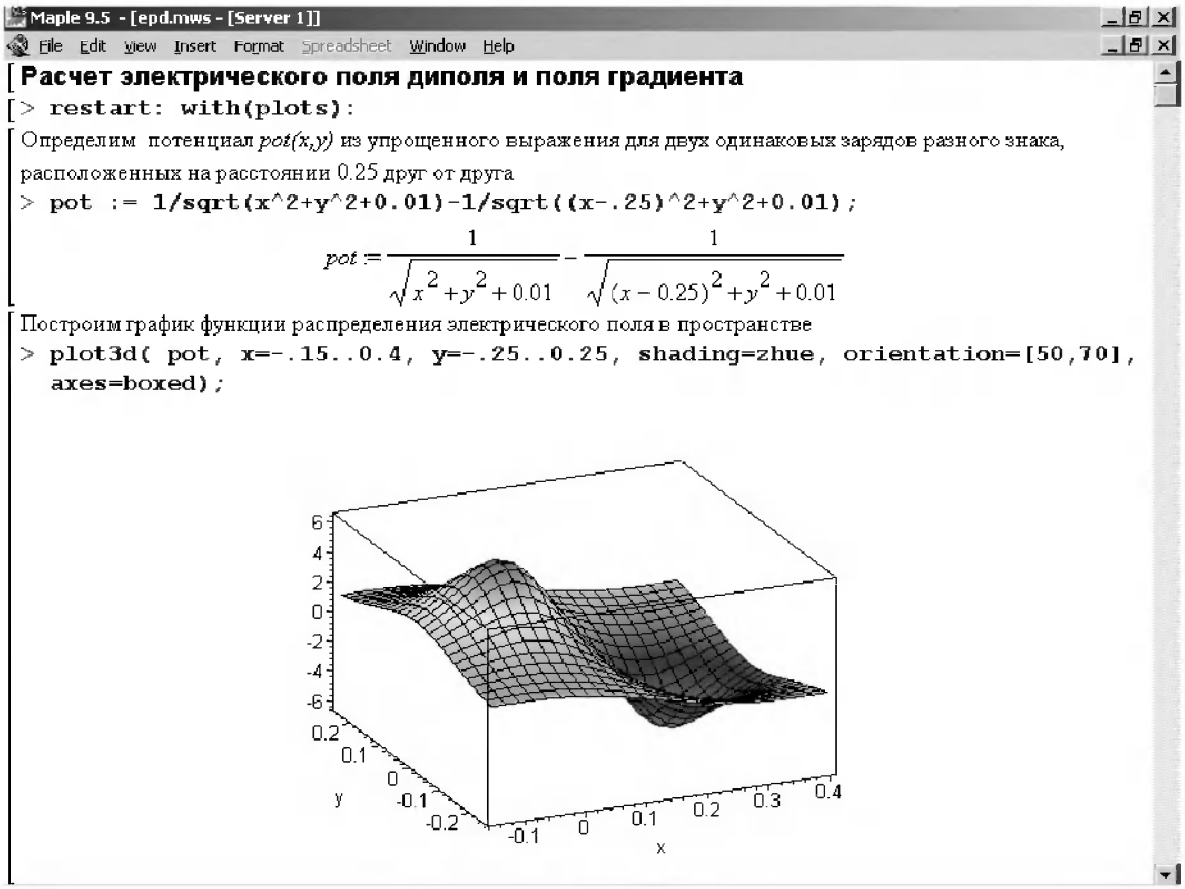


Рис. 10.35. Расчет и построение графика нормированного распределения поля двух зарядов противоположных знаков

Потенциал зарядов pot(x,y) представляет поверхность с пиком для положительного заряда в точке (0,0) и впадиной в точке (0.25,0). Ввиду упрощения выражения для заряда точных данных эта зависимость не несет и используется для качественных представлений.

Так, в электротехнике, однако, чаще используют графики поля в виде контурных эквипотенциальных линий. Построение такого графика для поля и его градиента (графика напряженности поля) показано на рис. 10.36. К сожалению, функциональная окраска графика поля отчетливо видна лишь на экране цветного дисплея.

Иногда полезно отслеживать изменение графика электрического поля при изменении одного из зарядов. Рисунок 10.37 показывает пример построения такого анимационного графика при изменении величины правого заряда с отрицательного на положительный. Представлен один из кадров анимации.

Большое число других примеров на вычисление параметров полей и визуализацию их можно найти в размещенных в Интернете (на сайте корпорации Waterloo Maple) пакетах Calculus IV и V.

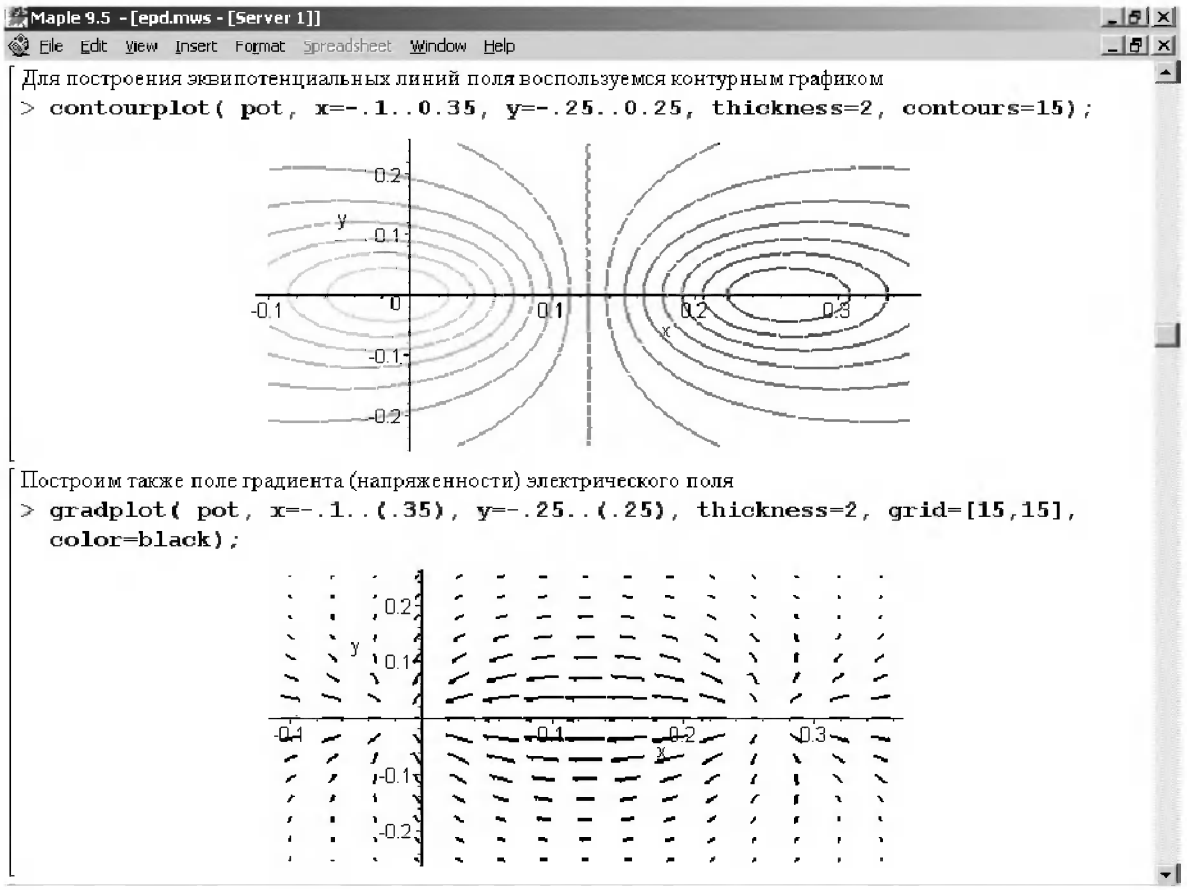


Рис. 10.36. Контурные графики электрического поля и его градиента

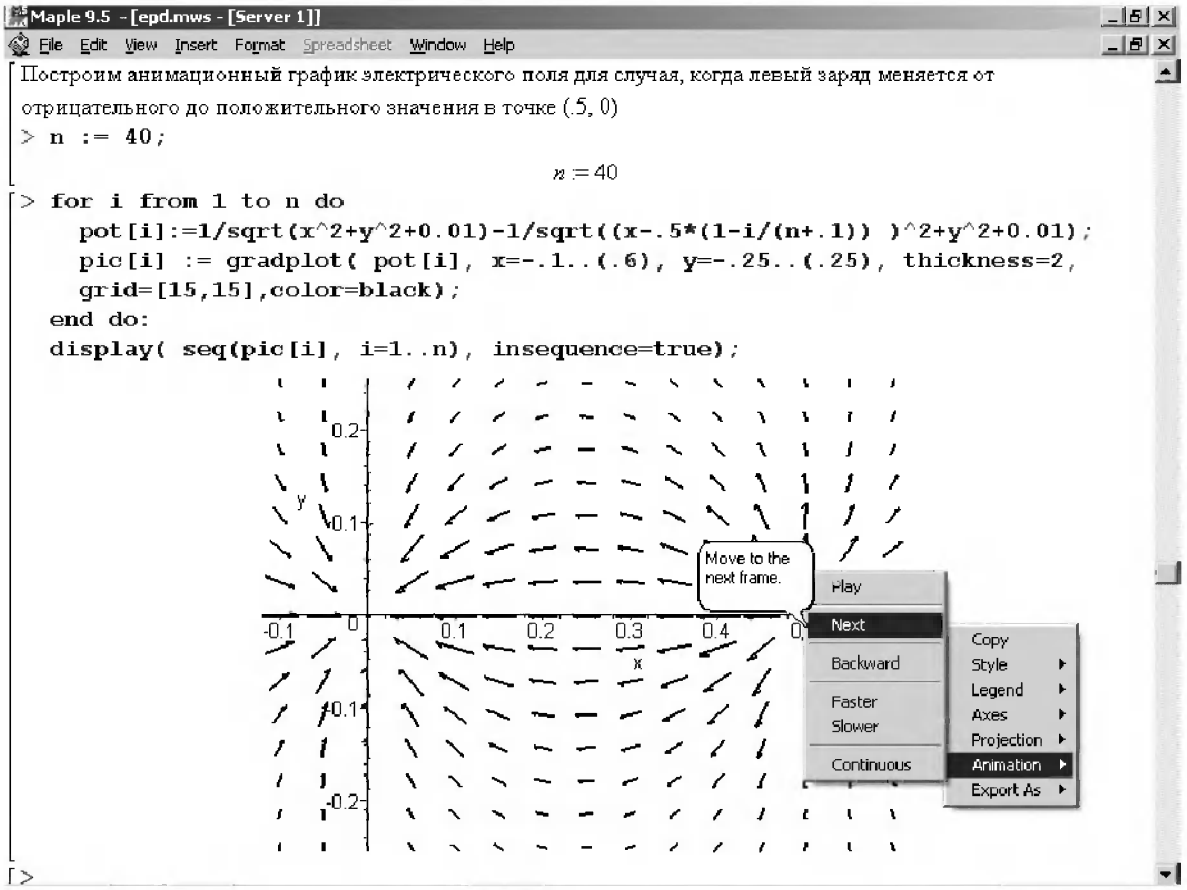


Рис. 10.37. Пример построения анимационного графика поля двух зарядов при изменении правого заряда

10.9. Работа с геометрическими пакетами

10.9.1. Набор функций пакета geometry

Пакет геометрических расчетов загружается командой

```
> with(geometry)
```

Она возвращает весьма внушительный список из более чем 100 функций. Ввиду его громоздкости список не приводится. Функции пакета имеют типовые для объектов двумерной графики имена и рассчитаны на выборочное использование (это, кстати, характерно для средств большинства других пакетов расширения системы Maple).

Данный пакет содержит средства расчета основных параметров ряда геометрических объектов. Для каждого объекта возможно задание различных исходных величин, так что пакет охватывает практически все виды классических геометрических расчетов на плоскости. Несомненно, этот пакет заинтересует всех, кто работает в области геометрии и смежных областях.

10.9.2. Пример применения расчетных функций пакета geometry

Учитывая идентичность идеологии при работе с функциями этого пакета, большинство из которых имеет вполне прозрачные имена (правда, англоязычные), работу с пакетом поясним на примере одной из функций – `circle`. Она позволяет математически задать окружность и определить все ее геометрические параметры. Функция может иметь несколько форм записи. Например, в форме

```
circle(c, [A, B, C], n, 'centername'=m)
```

она определяет построение окружности, проходящей через три точки `A`, `B` и `C`. Необязательный параметр `n` – список с именами координатных осей. Параметр `'centername'=m` задает имя центра.

В форме

```
circle(c, [A, B], n, 'centername'=m)
```

задается окружность, проходящая через две точки `A` и `B`, а в форме

```
circle(c, [A, rad], n, 'centername'=m)
```

задается окружность, проходящая через одну точку `A` с заданным (и произвольным) радиусом `rad` и центром `c`. Наконец, функция `circle` в форме

```
circle(c, eqn, n, 'centername'=m )
```

позволяет задать окружность по заданным уравнению `eqn` и центру `c`.

Проиллюстрируем применение функции `circle` на следующих примерах. Зададим характеристические переменные:

```
> _EnvHorizontalName := m: _EnvVerticalName := n:
```

Определим окружность c1, проходящую через три заданные точки A, B и C с указанными после их имен координатами и найдем координаты центра этой окружности:

```
> circle(c1,[point(A,0,0), point(B,2,0),point(C,1,2)],
'centername'=O1):
> center(c1), coordinates(center(c1));
```

$$O1, \left[1, \frac{3}{4}\right]$$

Далее найдем радиус окружности

```
> radius(c1);
```

$$\frac{\sqrt{25}\sqrt{16}}{16}$$

и уравнение окружности, заданное в аналитическом виде:

```
> Equation(c1);
```

$$m^2 + n^2 - 2m - \frac{3}{2}n = 0$$

Наконец, с помощью функции detail получим детальное описание окружности:

```
> detail(c1);
```

name of the oblect: c1

form of the object: circle2d
name of the center: O1
coordinates of the center: [1, 3/4]
*radius of the circle: 1/16*25^(1/2)*16^(1/2)*
*equation of the circle: m^2+n^2-2*m-3/2*n = 0*

10.9.3. Визуализация геометрических построений

Одно из важных достоинств пакета geometry – возможность наглядной визуализации различных геометрических понятий, например графической иллюстрации доказательства теорем или геометрических преобразований на плоскости. Проиллюстрируем это на примере графической иллюстрации к одной из теорем Фейербаха – рис. 10.38. Здесь эффектно используются средства выделения геометрических фигур цветом, что, увы, нельзя оценить по книжной черно-белой иллюстрации.

Рисунок 10.39 показывает гомологические преобразования квадрата. Заинтересовавшийся читатель может легко разобраться с деталями простого алгоритма этой программы.

Обратите особое внимание на последний параметр в функции draw. Он задает построение титульной надписи с заданными шрифтом и размером символов.

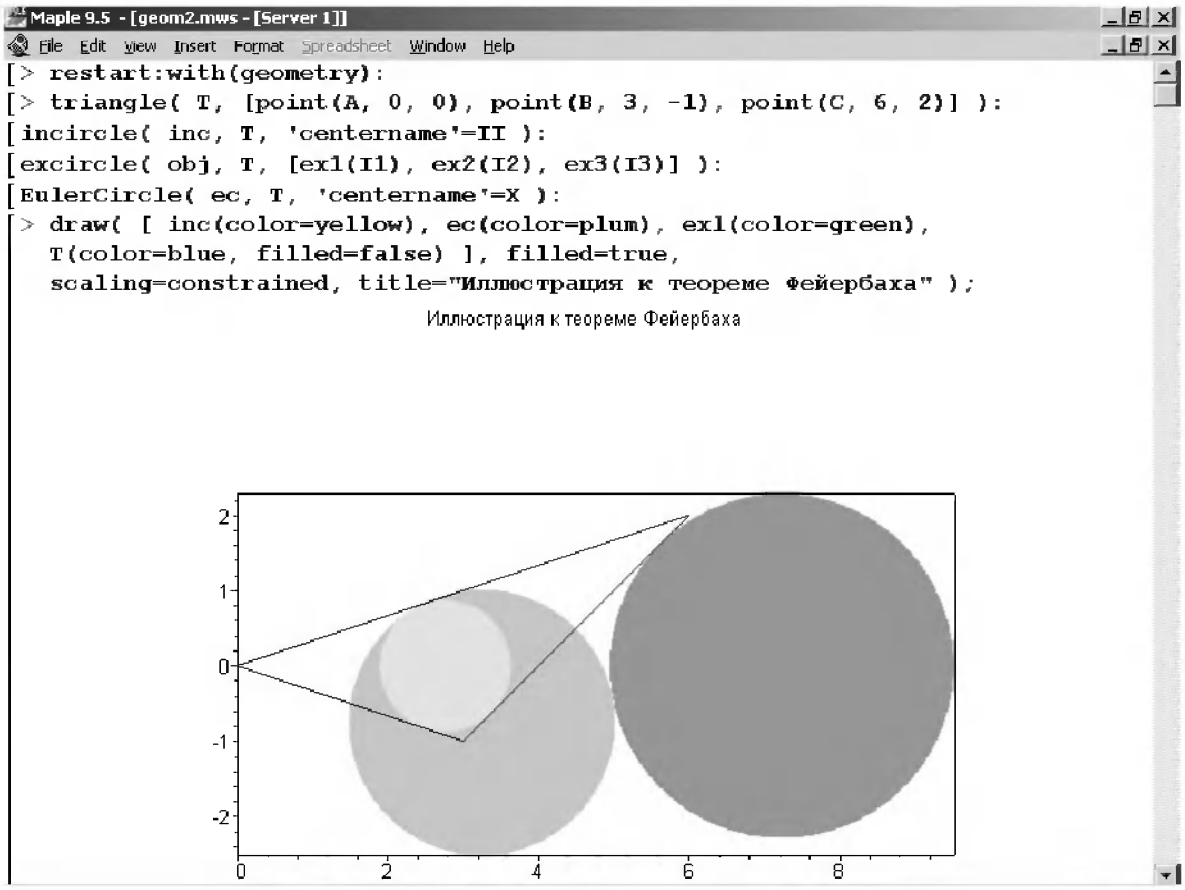


Рис. 10.38. Графическая иллюстрация к теореме Фейербаха

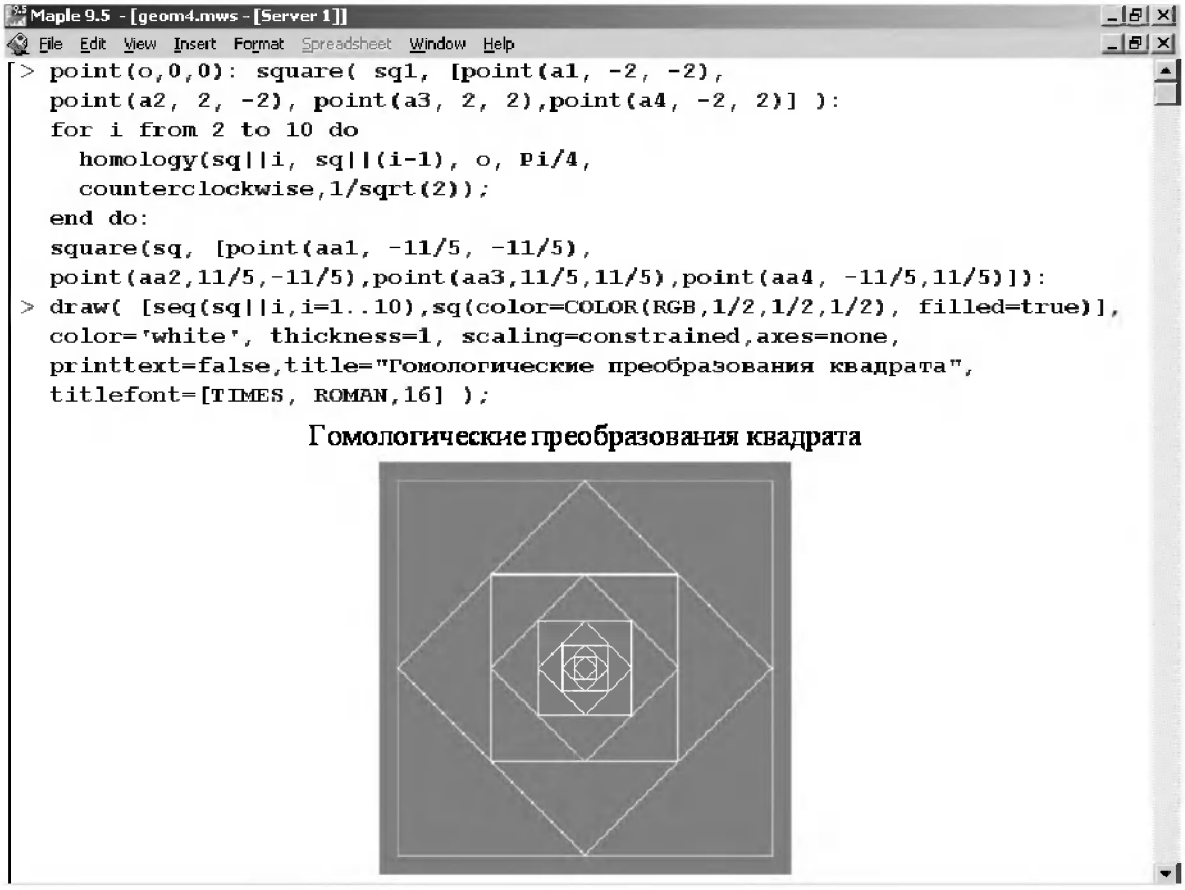


Рис. 10.39. Гомологические преобразования квадрата

Приятно, что нет преград для использования символов кириллицы и создания надписей на русском языке.

Множество других примеров применения всех функций пакета `geometry` дано в одноименном с ним файле примеров.

10.9.4. Набор функций пакета geom3d

Помимо существенного расширения пакета `geometry`, в систему Maple 7/8 был введен новый геометрический пакет `geom3d`. Он предназначен для решения задач в области стереометрии (трехмерной геометрии). Пакет есть и в новых реализациях Maple.

При загрузке пакета командой

```
with(geom3d)
```

появляется доступ к весьма большому (свыше 140) числу новых функций. Ввиду громоздкости списка он также не приводится, но читатель может просмотреть его самостоятельно.

Функции этого пакета обеспечивают задание и определение характеристик и параметров многих геометрических объектов: точек в пространстве, сегментов, отрезков линий и дуг, линий, плоскостей, треугольников, сфер, регулярных и квазирегулярных полиэдров, полиэдров общего типа и др. Назначение многих функций этого пакета ясно из их названия, а характер применения тот же, что для функций описанного выше пакета `geometry`.

10.9.5. Пример применения пакета geom3d

Учитывая сказанное, ограничимся примером, представленным на рис. 10.40.

Здесь даны две объемные фигуры, одна из которых расположена внутри в другой.

Напоминаем, что цель пакета – не в построении рисунков геометрических фигур, а в аналитическом представлении объектов в пространстве. Поэтому в обширной базе данных справочной системы по этому пакету вы встретите очень мало рисунков.

10.9.6. Набор функций пакета networks

Графы широко используются при решении многих прикладных и фундаментальных задач. Пользователей, занятых решением таких задач, наверняка порадует пакет `networks`, содержащий весьма представительный набор функций. Его можно увидеть, исполнив команду:

```
> with(networks);
```

Рассмотрим некоторые избранные функции этого пакета, которые наиболее часто используются при работе с графами.

Функции создания графов:

- `new` – создает пустой граф (без ребер и узлов);
- `void` – создает пустой граф (без ребер);

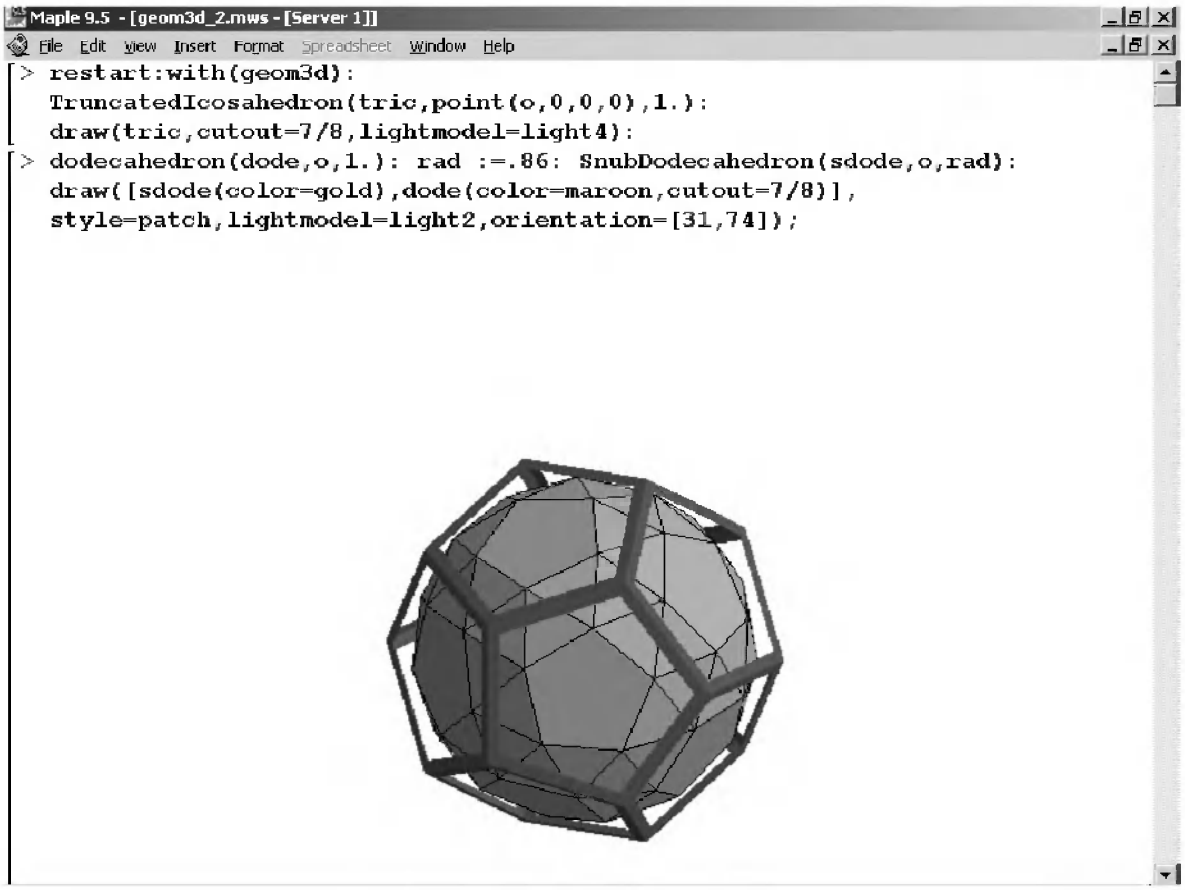


Рис. 10.40. Еще один пример применения пакета geom3d

- `duplicate` – создает копию графа;
- `complete` – создает полный граф;
- `random` – возвращает случайный граф;
- `petersen` – создает граф Петерсена.

Функции модификации графов:

- `addedges` – добавляет в граф ребро;
- `addvertex` – добавляет в граф вершины;
- `connect` – соединяет одни заданные вершины с другими;
- `delete` – удаляет из графа ребро или вершину.

Функции контроля структуры графов:

- `draw` – рисует граф;
- `edges` – возвращает список ребер графа;
- `vertices` – возвращает список узлов графа;
- `show` – возвращает таблицу с полной информацией о графе;
- `ends` – возвращает имена вершин графа;
- `head` – возвращает имя вершины, которая является головой ребер;
- `tail` – возвращает имя вершины, которая является хвостом ребер;
- `incidence` – возвращает матрицу инцидентности;

- `adjacency` – возвращает матрицу смежности;
- `eweight` – возвращает веса ребер;
- `vweight` – возвращает веса вершин;
- `isplanar` – упрощает граф, удаляя циклы и повторяющиеся ребра, и проверяет его на планарность (возвращает `true`, если граф оказался планарным, и `false` в противном случае).

Функции с типовыми возможностями графов:

- `flow` – находит максимальный поток в сети от одной заданной вершины к другой;
- `shortpathtree` – находит кратчайший путь в графе с помощью алгоритма Дейкстры.

Каждая из этих команд имеет одну или несколько синтаксических форм записи. Их можно уточнить с помощью справочной системы. С ее помощью можно ознакомиться и с назначением других функций этого обширного пакета. Объем, да и направленность данной энциклопедии делает нецелесообразным подробное рассмотрение данного пакета расширения. Тем более что его возможности детально описаны в книге [197].

10.10. Новые средства графики Maple 10/11

10.10.1. Новые средства двумерной графики в Maple 10/11

Возможности двумерной графики в системах Maple 9.5 и 10 практически равноценны. В Maple 10 расширены возможности форматирования двумерных графиков с помощью средств позиции Plot меню и контекстного меню правой клавиши мыши. Введена новая опция `gridlines` для функции `plot`, позволяющая выводить (в том числе раздельно по осям) линии масштабной сетки.

Новые возможности двумерной графики появились в Maple 11. Прежде всего это опция вывода наименования графика caption, используемая совместно с typeset – рис. 10.41. На рисунке представлено раскрытое контекстное меню правой клавиши мыши с активной позицией Title (титул графика) и выбором команды редактирования наименования графика.

Функция textplot служит для нанесения надписи в заданное место графика. Ее применение вполне очевидно из рис. 10.42, на котором построена синусоида и выведены надписи, поясняющие наличие локальных минимума и максимума. Обратите внимание на корректный вывод таких надписей на русском языке, сделанных символами кириллицы.

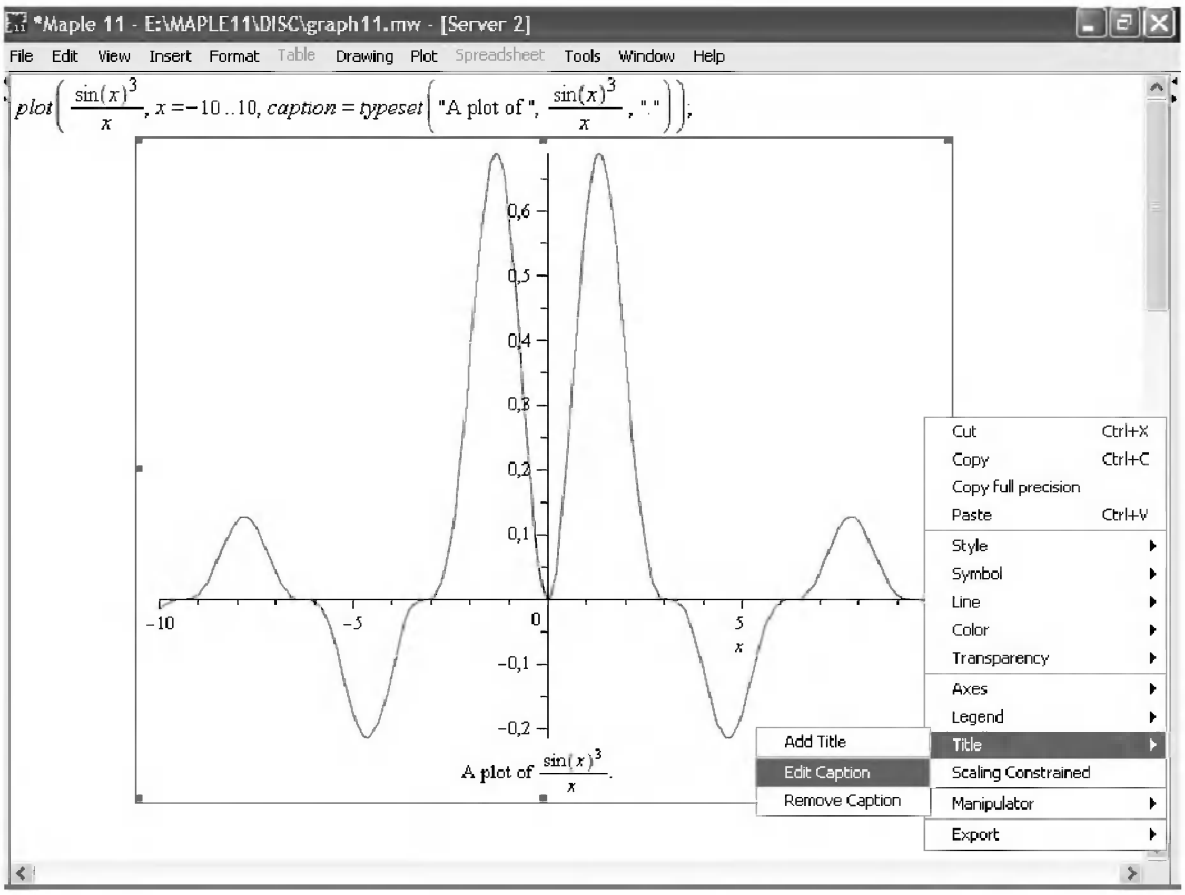


Рис. 10.41. Двумерный график с титульной (подрисуночной) надписью

```
with(plots) :
p := plot(sin(x), x=-Pi..Pi) :
t1 := textplot([Pi/2, 1, typeset("Локальный максимум (", Pi/2, ", ", 1, ")")], align=above) :
t2 := textplot([-Pi/2, -1, typeset("Локальный минимум (", -Pi/2, ", ", -1, ")")], align=below) :
display({p, t1, t2});
```

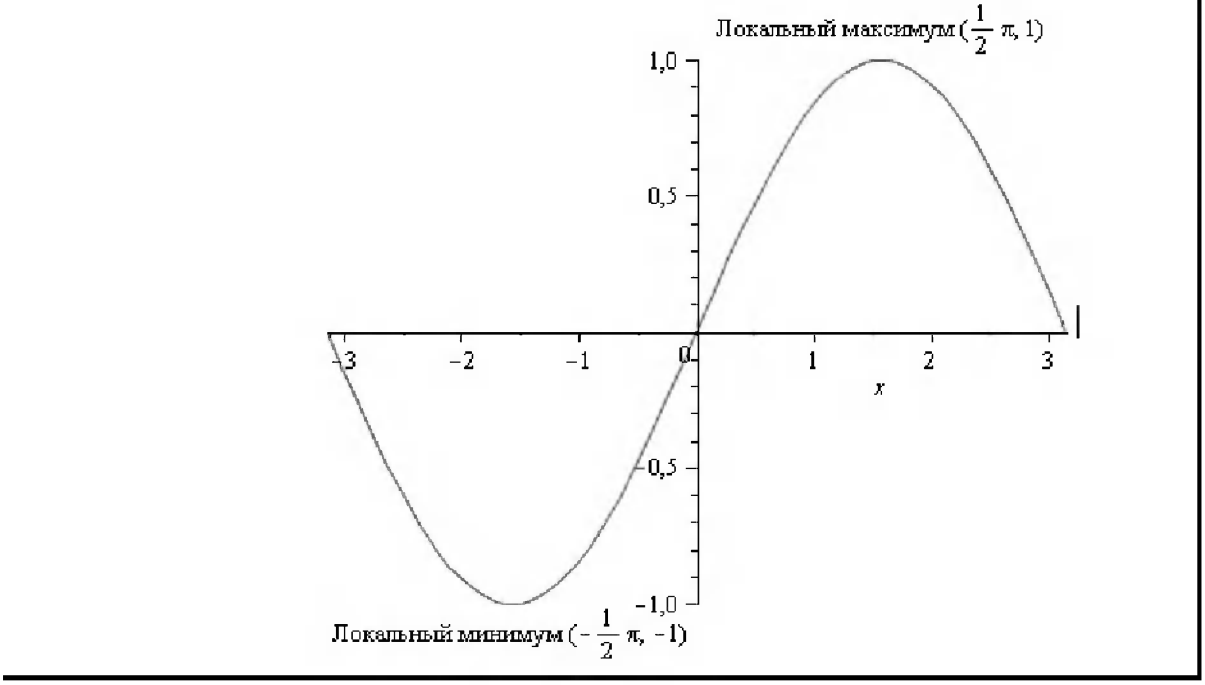


Рис. 10.42. Построение графика с поясняющими надписями в его поле

10.10.2. Новые средства трехмерной графики в Maple 10/11

Незначительному улучшению подверглись средства трехмерной графики. Введена новая опция `glossiness` (лоск, глянец), улучающая вид графика при окраске поверхности. Ее влияние иллюстрирует рис. 10.43 (сверху).

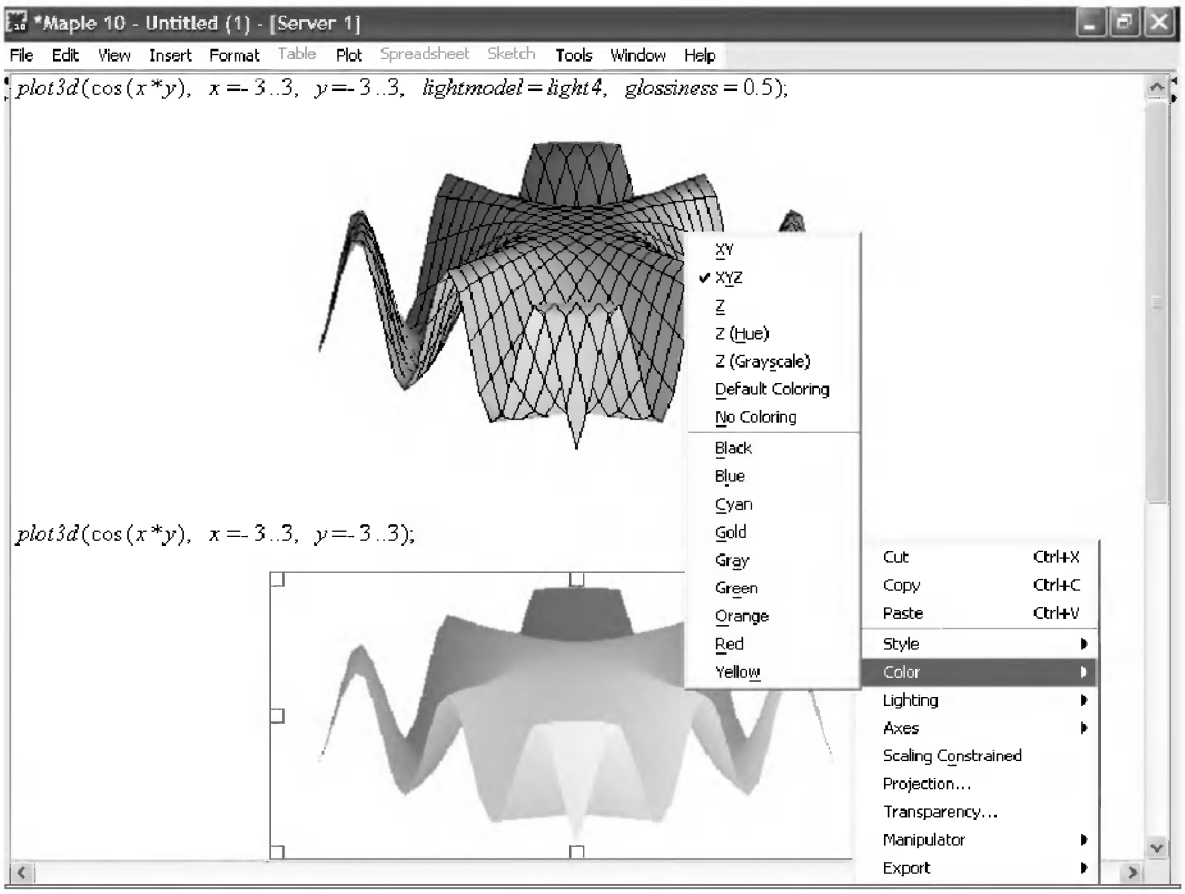


Рис. 10.43. Трехмерная графика в Maple 10/11

Здесь также особенно важно отметить возможность форматирования уже построенных графиков с помощью средств позиции `Plot` меню и контексного меню правой клавиши мыши.

10.10.3. Массивы разнотипных графиков

Весьма эффектным и наглядным средством представления на одном (не обязательно) рисунке нескольких графиков различного типа являются *массивы графиков*. Их задание и применение хорошо иллюстрирует рис. 10.44.

В каждом графике могут задаваться свои опции для задания типа координатных осей, вывода масштабной сетки и т. д. Могут одновременно строиться двумерные и трехмерные графики.

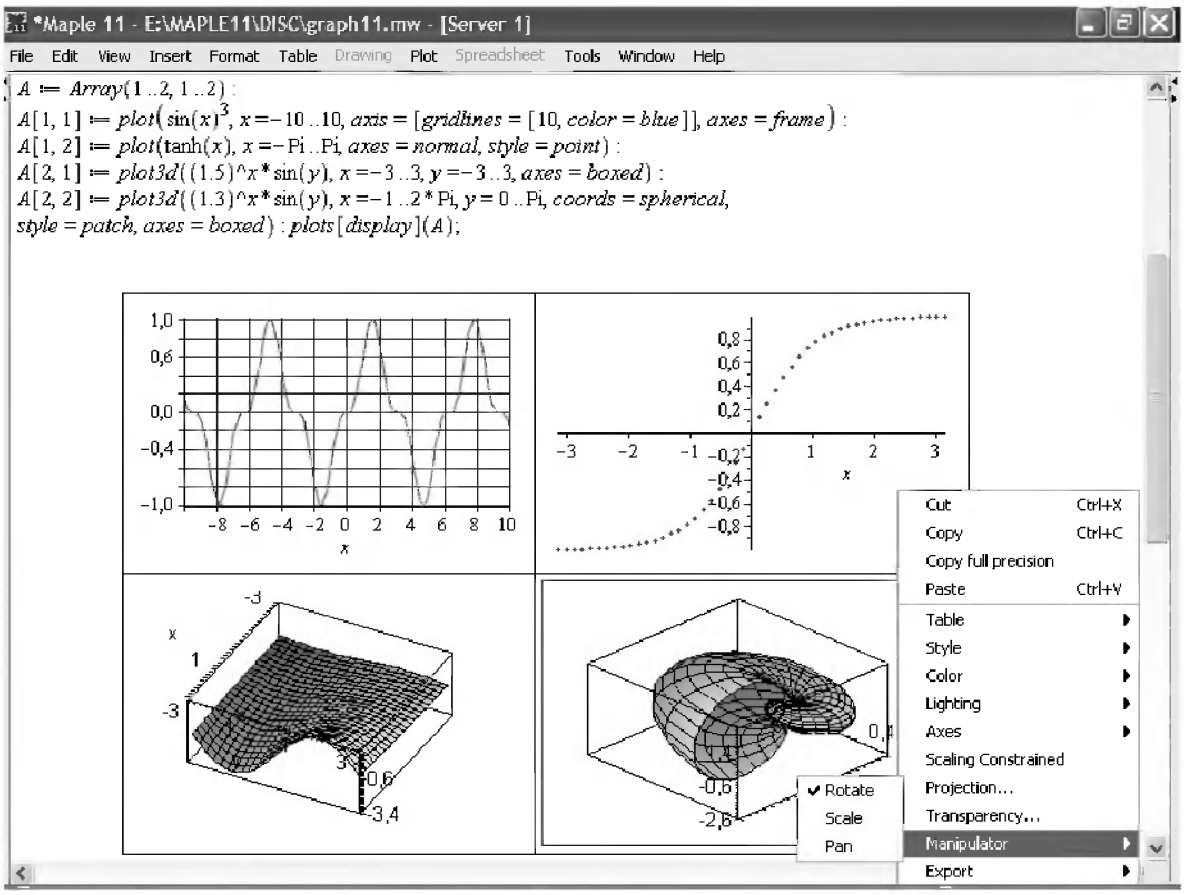


Рис. 10.44. Задание и применение массивов разнотипных графиков

10.10.4. Графические наброски

В Maple 10/11 введены новые графические объекты – наброски (от *sketch* – набросок). Они напоминают бумагу, разлинованную в клетку, на которой можно рисовать электронным карандашом **(Pencill)** или распылителем **(Highlighter)**. Есть также стиралка **Eraser**. Вставка наброска выполняется командой `sketch` в позиции меню `Insert`. Средства работы с набросками сосредоточены в специальной позиции меню **Sketch**. Те же самые средства имеются в контекстном меню правой клавиши мыши. Для редактирования набросков служит команда **Canvas Style...**, которая выводит окно редактирования. В окне можно задавать вывод вертикальных и горизонтальных линий сетки, устанавливать расстояние между ними (шаг) и изменять цвета объектов наброска – линий разметки и фона.

В Maple 11 те же возможности реализованы немного иначе. Явно вспомогательные средства (позиция **Sketch**) из меню были удалены. Зато в позиции **Insert** меню появилась команда **Canvas** (полотно, или холст), которая выводит заготовку наброска. В нее можно вводить графические объекты и надписи – рис. 10.45 и редактировать их. В частности, возможны перемещение объектов, их растяжение в том или ином направлении и т. д.

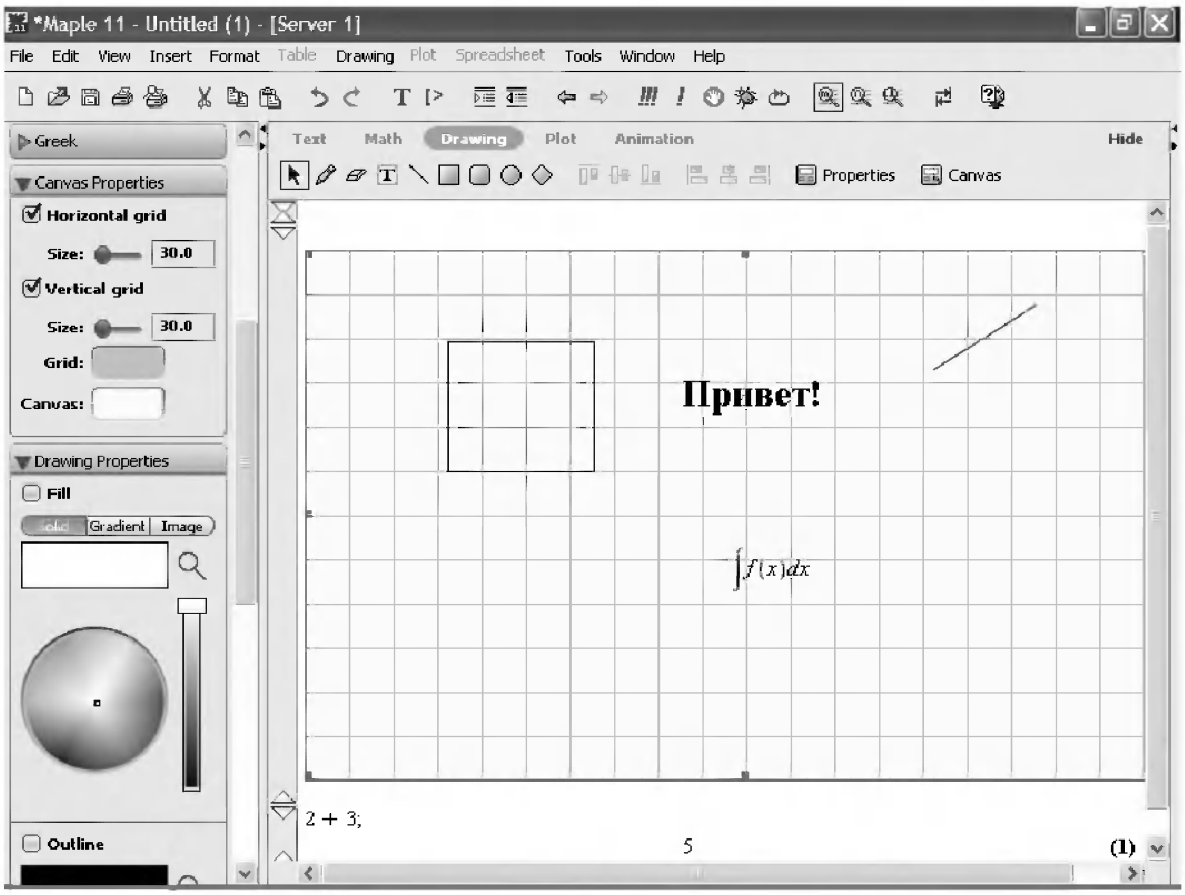


Рис. 10.45. Создание наброска в Maple 11

В Maple 11 в наборе панелей есть панели свойств набросков **Canvas Properties** и свойств рисунков **Drawing Properties**. Они показаны на рис. 10.45 слева. Установка свойств, в частности цветов, стала более удобной и современной, а удобная и наглядная панель инструментов облегчает создание набросков.

Глава 11

Визуализация в Mathematica 4/5/6

11.1. Средства двумерной графики системы Mathematica

11.1.1. Графическая функция Plot и ее опции

Концептуально графики в системе Mathematica являются *графическими объектами*, которые создаются (возвращаются) соответствующими *графическими функциями*. Их немного – всего порядка десятка, и они охватывают построение практически всех типов математических графиков. Это достигается это за счет применения опций и директив графики.

Поскольку графики являются объектами, то они могут быть значениями переменных. Mathematica допускает такие конструкции, как:

- `Plot[Sin[x],{x,0,20}]` – построение графика синусоиды;
- `g:=Plot[Sin[x],{x,0,20}]` – задание объекта – графика синусоиды с отложенным выводом;
- `g=Plot[Sin[x],{x,0,20}]` – задание объекта – графика синусоиды с немедленным выводом.

Для построения двумерных графиков функций вида $f(x)$ используется встроенная в ядро функция **Plot**:

- **Plot[f, {x, xmin, xmax}]** – возвращает объект – график функции f аргумента x в интервале от xmin до xmax.
- **Plot[{f1, f2, ...}, {x, xmin, xmax}]** – возвращает объект в виде графиков ряда функций fi.

Функция **Plot** используется для построения одной или ряда линий, дающих графическое представление для указанных функций f, f1, f2 и т. д.

По мере усложнения задач, решаемых пользователем, его рано или поздно не устроят графики, полученные при автоматическом выборе их стиля и иных параметров. Для точной настройки графиков Mathematica использует специальные **опции** графических функций. Так, с функцией Plot используются следующие опции:

```
Options[Plot]
```

$$\left\{\text{AspectRatio} \to \frac{1}{\text{GoldenRatio}}, \text{Axes} \to \text{Automatic}, \text{AxesLabel} \to \text{None},\right.$$

AxesOrigin → Automatic, AxesStyle → Automatic, Background → Automatic,
ColorOutput → Automatic, Compiled → True, DefaultColor → Automatic,
DefaultFont :→ $DefaultFont, DisplayFunction :→ $DisplayFunction,
Epilog → {}, FormatType :→ $FormatType, Frame → False, FrameLabel → None,
FrameStyle → Automatic, FrameTicks → Automatic, GridLines → None,
ImageSize → Automatic, MaxBend → 10., PlotDivision → 30.,
PlotLabel → None, PlotPoints → 25, PlotRange → Automatic,
PlotRegion → Automatic, PlotStyle → Automatic, Prolog → {},
RotateLabel → True, TextStyle :→ $TextStyle, Ticks → Automatic$\left.\right\}$

В приведенном обширном списке опций, помимо их названий, даны значения опций по умолчанию. Рассмотрим более подробно наиболее важные из опций (знаком * отмечены опции, применяемые как для 2D- так и для 3D-графики):

- ***AspectRatio** – задает пропорцию графика – отношение высоты к ширине (1/GoldenRatio задает отношение по правилу золотого сечения – около 1.6111...).
- ***Axes** – задает прорисовку координатных осей (False – осей нет, True – строятся обе оси и {Boolean, Boolean} – задает построение осей раздельно).
- ***AxesLabel** – задает вывод меток для осей в виде {"stringX", "stringY"}.
- **AxesOrigin** – задает начало отсчета для осей (указывает точку пересечения осей).
- ***AxesStyle** – задает стиль вывода осей с помощью ряда директив.
- ***BackGround** – задает цвет фона в одной из трех цветовых систем.
- ***ColorOutput** – задает цвет построений в одной из трех цветовых систем.
- ***DefaultFont** – задает шрифт для текста в графиках.
- **Frame** – задает прорисовку рамки вокруг графика при True и ее отсутствие при False.
- **FrameLabel** – задает надписи на гранях рамки (FrameLabel -> {"Text1", "Text2", "Text3", "Text4"}, причем построение идет по часовой стрелке, начиная с нижней надписи).
- **FrameStyle** – задает стиль граней рамки с помощью ряда директив.
- **FrameTicks** – задает прорисовку штриховых меток для граней рамки.
- **GridLines** – задает прорисовку линий сетки.
- ***PlotLabel** – задает вывод титульной надписи (PlotLabel->"Text").
- ***PlotRange** – задает масштаб построения в относительных единицах.
- ***PlotRegion** – задает область построения в относительных единицах.
- **RotateLabel** – задает разворот символьных меток на вертикальных осях фрейма, с тем чтобы они стали вертикальными.
- ***Ticks** – устанавливает штриховые метки для осей.

Кроме того, имеется ряд характерных для этой функции дополнительных функций:

- **Compiled** – задает компиляцию функции перед выводом.
- **MaxBend** – задает максимальную кривизну между сегментами кривой.
- **PlotDivision** – задает количество делений при построении гладкой кривой.
- **PlotPoints** – задает число точек выборки, участвующих в построении.
- **PlotStyle** – задает стиль линий или точек графика.

Опции внутри графических функций задаются своим именем name и значением value в виде:

```
name -> value
```

Наиболее распространенные символьные значения опций:

- **Automatic** – используется автоматический выбор.
- **None** – опция не используется.
- **All** – используется в любом случае.

- **True** – используется.
- **False** – не используется.

Многие опции могут иметь числовые значения. В сомнительных случаях рекомендуется уточнять форму записи опций и их значений по оперативной справочной системе.

По умолчанию система строит графики, не указывая надписей ни по осям координат (кроме букв x и y), ни в верхней части графика. Такая надпись на графике по центру сверху называется титульной. Рисунок 11.1 показывает построение графика с надписями у координатных осей. Для создания подобных надписей используется опция **AxesLabel**. После нее указывается список, содержащий две надписи: одну – для оси x и другую – для оси y. Надписи указываются в кавычках.

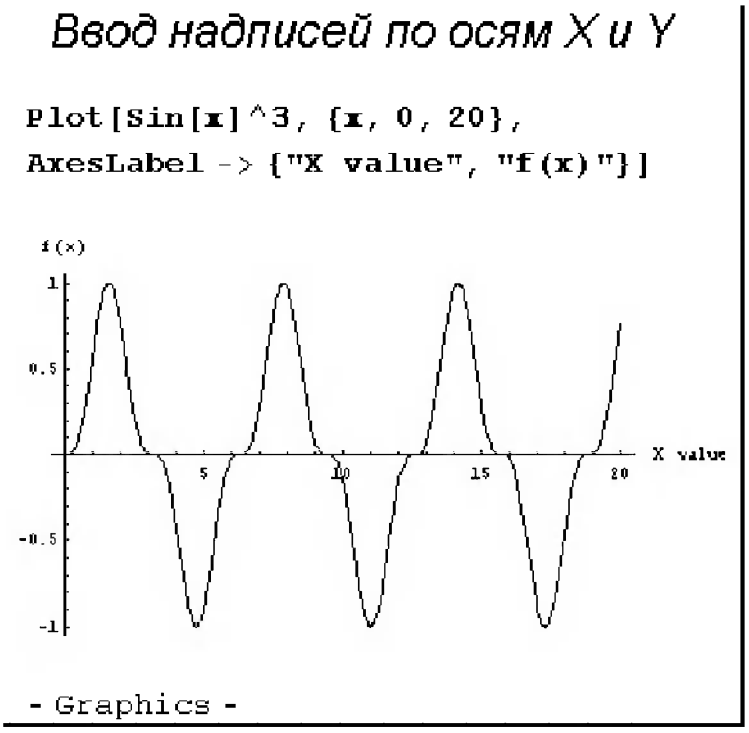


Рис. 11.1. Двумерный график с надписями по координатным осям

С помощью опции **Axes** с параметром **None** можно убрать с графика отображение осей. Опция **PlotLabel** обеспечивает вывод указанной в качестве ее параметра титульной надписи.

Часто возникает необходимость построения на одном рисунке нескольких графиков одной и той же функции, но при разных значениях какого-либо параметра – например, порядка специальных математических функций. В этом случае они могут быть заданы в табличной форме. Рисунок 11.2 дает пример построения пяти графиков функций Бесселя.

Рисунок 11.2 иллюстрирует недостаток простого представления на одном рисунке нескольких графиков – все они построены одинаковыми линиями, и сразу неясно, какой график к какой функции относится. С помощью опции **PlotStyle** можно, однако, менять стиль линий.

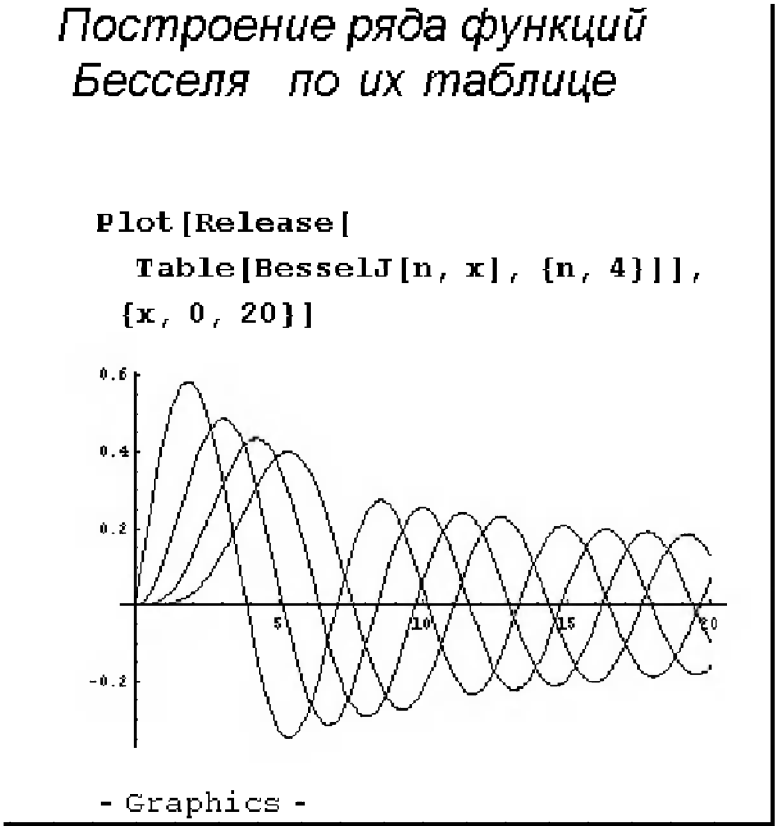


Рис. 11.2. Семейство функций Бесселя на одном графике

Применение других опций позволяет задавать массу других свойств графиков, например цвет линий и фона, вывод различных надписей и т. д. Помимо приведенных примеров, полезно просмотреть и множество примеров на построение двумерных графиков, приведенных в справочной системе Mathematica.

11.1.2. Директивы двумерной графики и их применение

Еще одним важным средством настройки графиков являются *графические директивы*. Синтаксис их подобен синтаксису функций. Однако опции не возвращают объектов, а лишь влияют на их характеристики. Используются также следующие директивы двумерной графики:

- **AbsoluteDashing[{d1, d2, ...}]** – задает построение последующих линий пунктиром со смежными (последовательными) сегментами, имеющими абсолютные длины d1, d2, ... (повторяемые циклически). Значения длины di задаются в пикселях.
- **AbsolutePointSize[d]** – задает построение последующих точек графика в виде кружков с диаметром d (в пикселях).
- **AbsoluteThickness[d]** – задает абсолютное значение толщины для последующих рисуемых линий (в пикселях).
- **Dashing[{r1, r2,...}]** – задает построение последующих линий пунктиром с последовательными сегментами длиной r1, r2, ..., повторяемыми циклически, причем ri задается как дробная часть от полной ширины графика.

- **PointSize[d]** – задает вывод последующих точек графика в виде кружков с относительным диаметром d, заданным как дробная часть от общей ширины графика.
- **Thickness[r]** – устанавливает толщину r для всех последующих линий, заданную как дробная часть от полной ширины графика.

Применение графических директив совместно с опциями позволяет создавать графики самого различного вида, вполне удовлетворяющие как строгим требованиям, так и различным «извращениям» в их оформлении. Попробуйте применить примитивы для построения графиков различного вида.

11.1.3. Построение графика по точкам – функция ListPlot

Часто возникает необходимость построения графика по точкам. Это обеспечивает встроенная в ядро графическая функция **ListPlot**:

- **ListPlot[{y1, y2, ...}]** – выводит график списка величин. Координаты x для каждой точки принимают значения 1, 2, ...
- **ListPlot[{{x1, y1}, {x2, y2}, ...}]** – выводит график списка величин с указанными x- и y-координатами.

В простейшем случае (рис. 11.3) она задает сама значения координаты x=0,1,2,3,... и строит точки на графике с координатами *(x,y)*, выбирая y последовательно из списка координат.

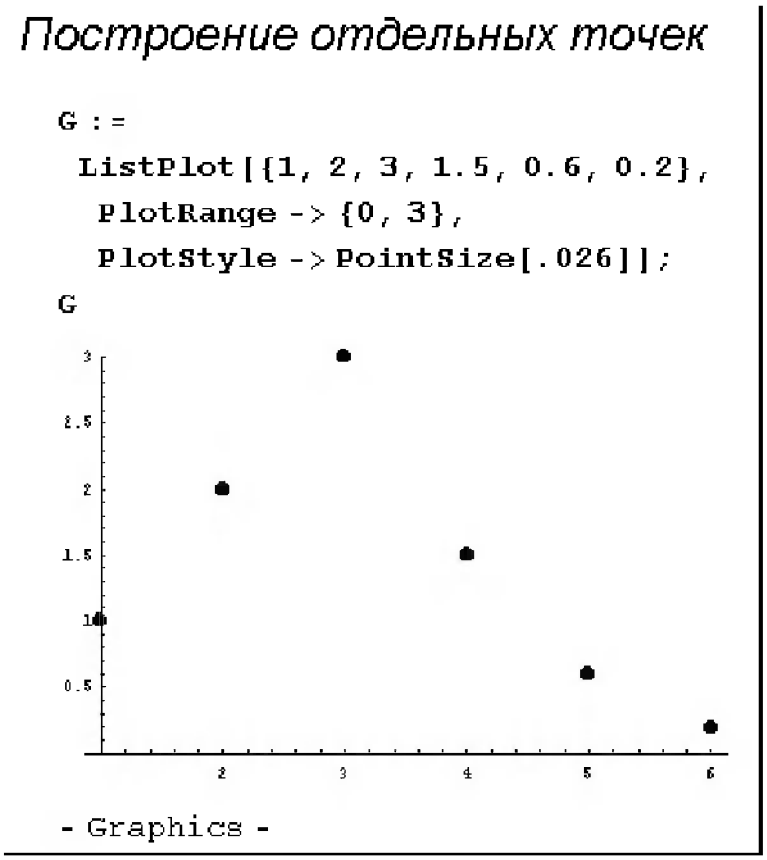


Рис. 11.3. Построение ряда точек графика

Можно подметить характерный недостаток построений – точки (особенно при небольшом размере) имеют вид, заметно отличающийся от закрашенной идеальной окружности – круга. Эта функция, особенно в ее второй форме (с заданными координатами x и y), удобна для вывода на график экспериментальных точек.

11.1.4. Получение информации о графических объектах

Порой некоторые детали построения графиков оказываются для пользователя неожиданными и не вполне понятными. Причина этого кроется во множестве опций, которые могут использоваться в графиках, причем в самых различных сочетаниях. Поэтому полезно знать, как можно получить информацию о свойствах графических объектов. Порой небольшая модификация опций (например, замена цвета линий или фона) делает график полностью удовлетворяющим требованиям пользователя.

Следующие функции дают информацию об опциях графического объекта g:

- **FullAxes[g]** – возвращает список опций координатных осей;
- **Options[g]** – возвращает упрощенный список опций;
- **FullOptions[g]** – возвращает полный список опций;
- **InputForm[g]** – возвращает информацию о графике (включая таблицу точек).

В списке функции **FullOptions** имеются численные данные обо всех параметрах графика. Аналогично можно получить и иные данные – они не приводятся ввиду громоздкости выводимой информации. Анализ графиков с применением этих функций может оказаться весьма полезным при задании построения сложных графиков и их редактировании.

Рекомендуется просмотреть с помощью этих функций опции графического объекта, например

```
g:=Plot[Sin[x],{x,-10,10}];
```

Ввиду громоздкости списков опций они не приводятся. Функции **OptionsFull** и **Options** можно также использовать в виде:

- **Options[g, option]** – возвращает значение указанной опции;
- **FullOptions[g, option]** – возвращает значение указанной опции.

В этом случае можно получить информацию по отдельной опции.

11.1.5. Директива Show

При построении графиков требуется изменение их вида и тех или иных параметров и опций. Этого можно достичь повторением вычислений, но при этом скорость работы с системой заметно снижается. Для ее повышения удобно использовать специальные функции перестройки и вывода графиков, учитывающие, что их узловые точки уже рассчитаны и большая часть опций уже задана.

В этом случае удобно использовать следующую функцию-директиву:

- **Show[plot]** – построение графика.
- **Show[plot, option -> value]** – построение графика с заданной опцией.
- **Show[plot1, plot2,...]** – построение нескольких графиков с наложением их друг на друга.

Директива **Show** полезна также и в том случае, когда желательно, не трогая исходных графиков, просмотреть их при иных параметрах. Соответствующие опции, меняющие параметры графиков, можно включить в состав директивы **Show**. Другое полезное применение директивы – объединение на одном графике нескольких графиков различных функций или экспериментальных точек и графика теоретической зависимости.

Директива **Show** часто применяется, когда надо построить на одном графике кривую некоторой функции и представляющие ее узловые точки (например, при построении кривых регрессии в облаке точек исходных данных). Примеры такого построения даны на рис. 5.12 и 5.13.

11.1.6. Функция Graphics и ее примитивы

Примитивами двумерной графики называют дополнительные указания, вводимые в функцию **Graphics** для построения некоторых заданных геометрических фигур. Функция **Graphics** задается в виде:

- **Graphics[primitives, options]** – представляет двумерное графическое изображение.

Применение примитивов в составе функции **Graphics** избавляет пользователя от задания математических выражений, описывающих эти фигуры. Примитивы могут выполнять и иные действия. Они заметно увеличивают число типов графиков, которые способна строить система Mathematica.

Используются следующие примитивы двумерной графики:

- **Circle[{x, y}, r]** – строит окружность с радиусом r и центром в точке {x, y}.
- **Circle[{x, y}, {rx, ry}]** – строит эллипс с центром {x,y}и полуосями rx и ry.
- **Circle[{x, y}, r, {theta1, theta2}]** – представляет дугу окружности радиуса r с центром {x,y} и углами концевых точек theta1 и theta2.
- **Disk[{x, y}, r]** – является примитивом двумерной графики, представляющим закрашенный круг радиусом r с центром в точке {x, y}.
- **Disk[{x, y}, {rx, ry}]** – строит закрашенный овал с полуосями rx и ry и центром {x,y}.
- **Disk[{x, y}, r, {theta1, theta2}]** – строит сегмент круга радиуса r с центром {x,y} и углами концевых точек theta1 и theta2.
- **Line[{pt1, pt2, ...}]** – строит линию, соединяющую последовательность точек.
- **Point[{x,y}]** – строит точку с координатами x и y.
- **Polygon[{x1,y1},{x2,y2},...]** – построение полигона с закраской.
- **PostScript["string"]** – построение объекта, заданного на языке PostScript.
- **Rectangle[{xmin, ymin}, {xmax, ymax}]** – строит закрашенный прямоугольник, ориентированный параллельно осям, намеченный координатами точек противолежащих углов.

- **Rectangle[{xmin, ymin}, {xmax, ymax}, graphics]** – строит закрашенный прямоугольник, заполненный в соответствием с указаниями в функции graphics и заданный координатами противолежащих углов.
- **Raster[{{a11, a12, ...}, ...}]** – строит прямоугольный массив ячеек яркости.
- **RasterArray[{{g11, g12, ...}, ...}]** – строит прямоугольный массив ячеек, окрашенных в соответствии с графическими директивами gij.
- **Text[expr, coords]** – выводит текст, соответствующий печатной форме выражения expr, центрированный в точке с указанными координатами coords.

Рисунок 11.4 показывает применение функции **Graphics** для построения одновременно четырех графических объектов: отрезка прямой, заданного координатами его концевых точек, окружности с центром (0,0) и радиусом 0.8, текстовой надписи «Hello!» и жирной точки. Каждый объект задан своим примитивом.

Применение примитивов 2D-графики

```
g1 := Graphics[Line[{{-1, -1}, {1, 1}}]];
  g2 := Graphics[Circle[{0, 0}, 0.8]];
  g3 := Graphics[Text["Привет!", {0.25, 0.5}]];
  Show[g1, g2, g3, Graphics[PointSize[0.032],
  Point[{-.5, .2}]], Axes -> True]
```

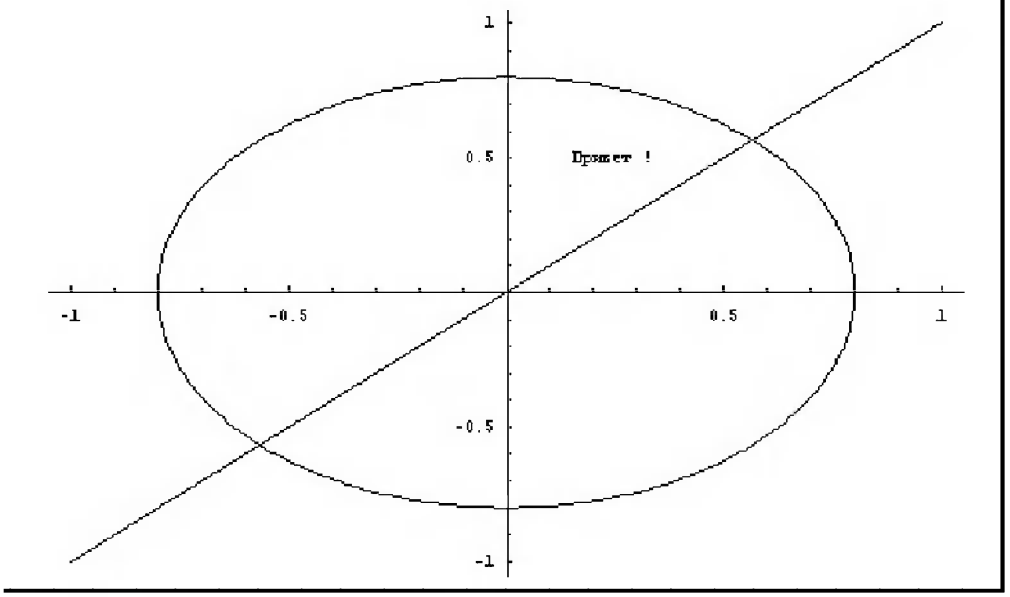


Рис. 11.4. Построение четырех графических объектов с помощью примитивов двумерной графики

Приведенный пример поясняет технику применения графических примитивов. Но они, разумеется, не исчерпывают всех возможностей этого метода построения геометрических фигур и объектов. Все указанные примитивы используются при построении как двумерных, так и трехмерных графиков.

11.1.7. Функции для построения параметрически заданных графиков

Для построения параметрически заданных функций используются следующие графические средства:

- **ParametricPlot[{fx, fy}, {t, tmin, tmax}]** – строит параметрический график с координатами fx и fy (соответствующими x и y), получаемыми как функции от t.
- **ParametricPlot[{{fx, fy}, {gx, gy}, ...}, {t, tmin, tmax}]** – строит графики нескольких параметрических кривых.

Функции fx, fy и т. д. могут быть как непосредственно вписаны в список параметров, так и определены как функции пользователя.

На одном графике можно строить две и более фигуры с заданными параметрически уравнениями. На рис. 11.5 показан пример такого построения – строятся две фигуры Лиссажу, причем одна из них – окружность. Больше двух фигур строить нерационально, так как на черно-белом графике их трудно различить.

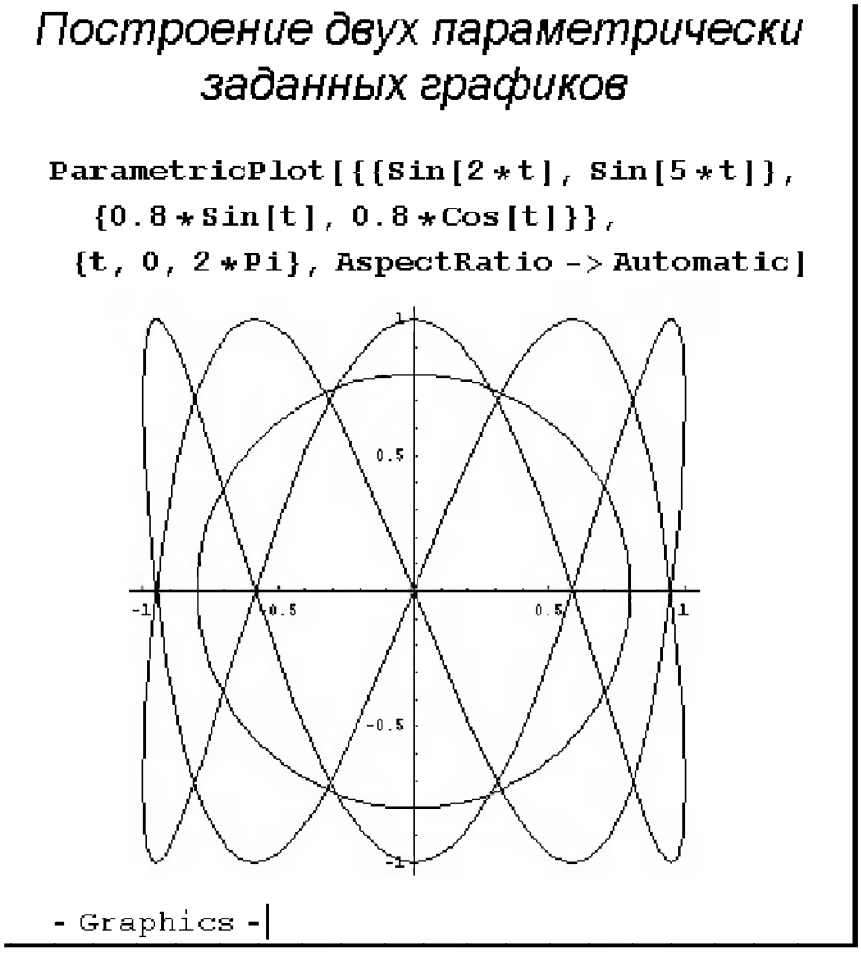


Рис. 11.5. Построение на одном графике двух фигур Лиссажу

11.1.8. Построения графиков в полярной системе координат

Теперь рассмотрим другой способ построения графиков в полярной системе координат – рис. 11.6. Здесь каждая точка является концом радиус-вектора R(t), причем угол t меняется от 0 до 2π. На рис. 11.6 функция R(t) задана как функция пользователя R[t_] с использованием образца для задания локальной переменной t в теле функции.

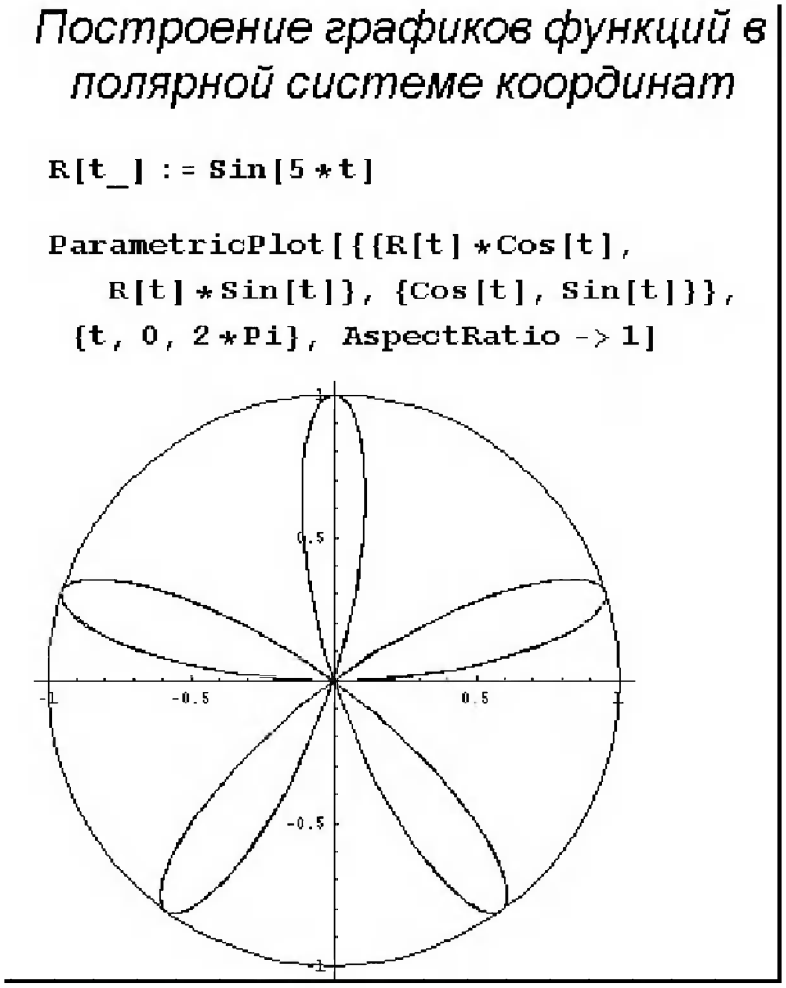


Рис. 11.6. Построение графика функции в полярной системе координат

Изменение параметра R позволяет заметно увеличить число отображаемых функций – фактически их бесконечно много. Помимо описанной фигуры, на рис. 11.6 дополнительно построена линия окружности единичного радиуса. Чтобы она имела вид окружности, задана опция **AspectRatio->1**.

11.2. Контурные графики и графики плотности

11.2.1. Функции для построения контурных графиков

Основными функциями и директивами для построения контурных графиков в системе Mathematica являются следующие функции:

- **ContourPlot[f,{x, xmin, xmax}, {y, ymin, ymax}]** – порождает контурный график f как функции от x и y.
- **ContourGraphics[array]** – представляет контурный график массива array.
- **ListContourPlot[array]** – формирует контурный график из массива величин высот.

Этих функций достаточно для построения практически любых монохромных графиков такого типа.

11.2.2. Опции для функций контурной графики

Для управления возможностями графической функции **ContourPlot** используются опции:

```
Options[ContourGraphics]
{AspectRatio→1,Axes→False,AxesLabel→None,AxesOrigin→Automatic,
AxesStyle→Automatic,Background→Automatic,ColorFunction→Automatic,
ColorFunctionScaling→True,ColorOutput→Automatic,ContourLines→True,
Contours→10,ContourShading→True,ContourSmoothing→True,
ContourStyle→Automatic,DefaultColor→Automatic,DefaultFont¦$DefaultFont,
DisplayFunction¦$DisplayFunction,Epilog→{},FormatType¦$FormatType,
Frame→True,FrameLabel→None,FrameStyle→Automatic,FrameTicks→Automatic,
ImageSize→Automatic,MeshRange→Automatic,PlotLabel→None,
PlotRange→Automatic,PlotRegion→Automatic,Prolog→{},RotateLabel→True,
TextStyle¦$TextStyle,Ticks→Automatic}
```

Аналогично можно получить данные об опциях других функций этого раздела. Помимо уже рассмотренных ранее опций, используются следующие:

- **ColorFunction** – задает окраску областей между линиями.
- **Contours** – задает число контурных линий.
- **ContourLines** – задает прорисовку явных (explicit) контурных линий.
- **ContourShading** – задает затенение областей между контурными линиями.
- **ContourSmoothing** – задает сглаживание контурных линий.

- **ContourStyle** – задает стиль рисуемых линий для контурных графиков.
- **MeshRange** – задает области изменения X- и Y-координат.

Как видно из приведенного выше перечня опций, помимо указанных, возможно применение и множества других опций.

11.2.3. Примеры построения контурных графиков

Рисунок 11.7 показывает построение контурного графика с окраской промежуточных областей между линиями. Окраска обеспечивается опцией **ColorFunction ->Hue**. Опция **ContourSmoothing->True** задает сглаживание контурных линий.

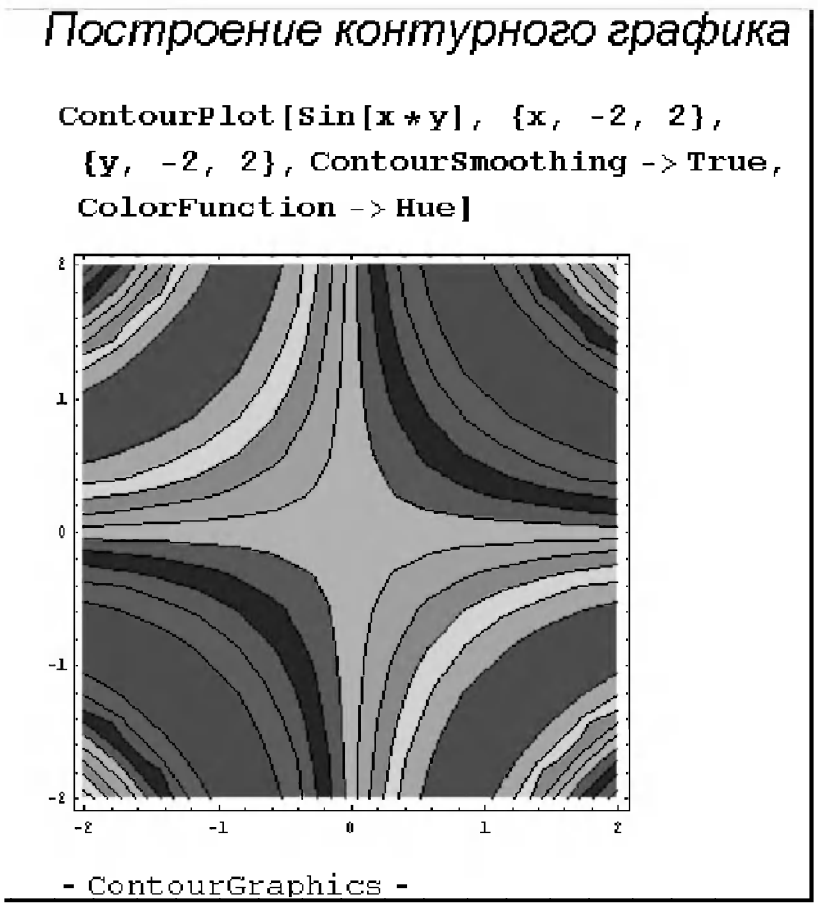


*Рис. 11.7. Контурный график поверхности sin(x*y) с закраской областей между линиями равного уровня оттенками серого цвета*

Можно проверить эффективность применения опции ContourShading. Если задать ее значение False, то заполнение пространства между линиями будет отсутствовать. Таким образом, в данном случае строятся только линии равного уровня. Иногда график оказывается более наглядным, если убрать построение контурных линий, но оставить закраску областей между линиями. Такой вариант графика более предпочтителен, если нужно наблюдать качественную картину.

11.2.4. Функции графиков плотности

Функцией двух переменных $f(x,y)$ может описываться плотность некоторой среды. Для построения графиков плотности используются следующие графические функции:

- **DensityGraphics[array]** – является представлением графика плотности.
- **DensityPlot[f, {x, xmin, xmax}, {y, ymin, ymax}]** – строит график плотности f как функции от x и y.
- **ListDensityPlot[array]** – формирует график плотности из массива величин высот.

С этими функциями используется множество в основном уже рассмотренных опций. Их перечень можно получить с помощью функции **Options**. Внешне график плотности похож на контурный график. Однако для него характерно выделение элементарных участков (с равной плотностью) в форме квадратиков – рис. 11.8.

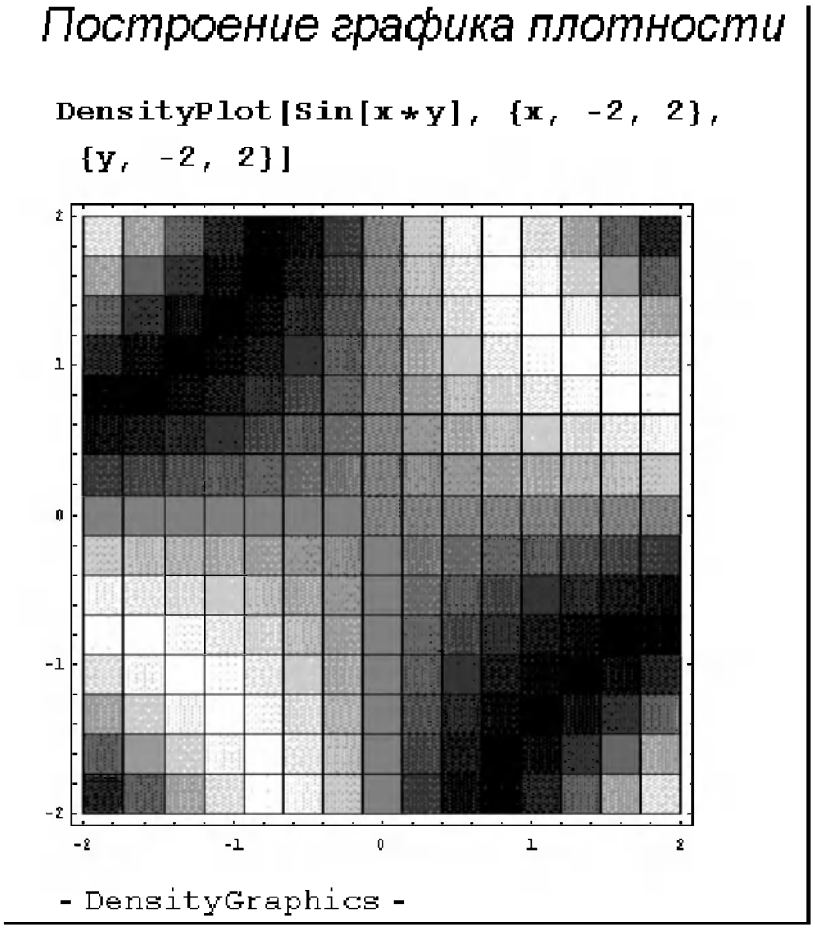


Рис. 11.8. График плотности

11.3. Построение графиков поверхностей

11.3.1. Основные функции для построения 3D-графиков

Для построения 3D-графиков поверхностей используется основная графическая функция **Plot3D**:

- **Plot3D[f, {x, xmin, xmax}, {y, ymin, ymax}]** – строит трехмерный график функции f переменных x и y.
- **Plot3D[{f, s}, {x, xmin, xmax}, {y, ymin, ymax}]** – строит трехмерный график, в котором высоту поверхности определяет параметр f, а затенение – s.

Для модификации трехмерных графиков могут использоваться следующие опции:

- **AmbientLight** – задает функциональную засветку от постоянного источника света с заданными координатами.
- **AxesEdge** – определяет, на каких гранях ограничительного параллелепипеда (ящика) должны выводиться оси.
- **Boxed** – указывает, надо ли рисовать ли контуры (ребра, грани) ограничительного параллелепипеда в трехмерном изображении.
- **BoxRatios** – задает значение отношений длин сторон для ограничительной рамки трехмерного изображения.
- **BoxStyle** – задает прорисовку ограничительной рамки.
- **Background** – задает цвет фона.
- **ClipFill** – определяет, как отсекаемые части поверхности должны выводиться.
- **ColorFunction** – определяет функцию, используемую для функциональной окраски.
- **ColorOutput** – задает тип цветового выхода для вывода.
- **DefaultFont** – возвращает шрифт по умолчанию для текста в графике.
- **DefaultColor** – задает цвет по умолчанию для линий, точек и т. д.
- **$DisplayFunction** – задает установочное значение по умолчанию для опции DisplayFunction в графических функциях.
- **DisplayFunction** – определяет функцию, которая применяется к графическим и звуковым примитивам для их отображения.
- **Epilog** – опция для графических функций, дающая список графических примитивов, которые должны воспроизводиться после воспроизведения главной части графики.
- **FaceGrids** – опция для функций трехмерной графики; устанавливает вывод линий сетки на гранях (лицевых сторонах) ограничительной рамки.

- **HiddenSurface** – определяет, нужно или нет удалять невидимые линии каркаса.
- **Lighting** – указывает, следует ли использовать моделируемую освещенность (simulated illumination) в трехмерных изображениях.
- **LightSources** – опция к Graphics3D и родственным функциям, которая устанавливает возможности (свойства) точечных источников света для моделируемого освещения.
- **Mesh** – указывает, следует ли вырисовывать явно заданную x-y сетку.
- **MeshRange** – устанавливает диапазон (область изменения) x- и y-координат, которые соответствуют массиву заданных величин z.
- **MeshStyle** – задает стиль вывода линий сетки.
- **SphericalRegion** – указывает, следует ли конечный образ масштабировать так, чтобы сфера, рисуемая вокруг трехмерной отображения, охватывала его целиком.
- **Plot3Matrix** – опция к Graphics3D и родственным функциям, которая может использоваться для определения матрицы преобразования явной гомогенной (однородной) перспективы.
- **PolygonIntersections** – опция для Graphics3D, которая определяет, следует ли пересекающиеся многоугольники оставлять без изменения.
- **Prolog** – опция для графических функций, дающая список графических примитивов, которые предоставляются, до главной части графики.
- **RenderAll** – опция к Graphics3D, которая указывает, должен или нет генерироваться PostScript для всех многоугольников.
- **Shading** – опция для SurfaceGraphics, указывающая, следует ли выполнять затенение поверхностей.
- **ToColor[color, form]** – превращает color в form; если form представляет собой функцию GrayLevel, RGBColor или CMYKColor, то color превращается в нее. В противном случае вычисляется **form[color]** и как результат ожидается правильная цветовая директива.
- **ViewCenter** – задает масштабные координаты точки, оказывающейся в центре области отображения в окончательном (последнем, целевом) графике.
- **ViewPoint** – меняет точку пространства, из которой рассматривается объект.
- **ViewVertical** – устанавливает, какое направление в относительных координатах должно быть вертикальным в окончательном образе.

11.3.2. Директивы трехмерной графики

Помимо опций, для трехмерной графики используется ряд графических директив и функций:

- **CMYKColor[cyan, magenta, yellow, black]** – устанавливает составляющие цвета.
- **EdgeForm[g]** – указывает, что грани многоугольников должны быть нарисованы с применением графической директивы или списка директив.

- **FaceForm[gf, gb]** – указывает, что передние грани (лицевые поверхности) многоугольников должны выводиться с применением графического примитива gf, а задние грани (невидимые поверхности) – посредством gb.
- **FullAxes[graphics]** – возвращает опции осей графического объекта.
- **FullGraphics[g]** – берет графический объект и производит новый, в котором объекты, определяемые графическими опциями, даются как явные (точные) списки графических примитивов.
- **FullOptions[expr]** – возвращает полные установки опций, которые явно определены в выражении типа графического объекта.
- **FullOptions[expr, name]** – возвращает полное установочное значение для объекта с именем name.
- **Hue[h]** – указывает, что графические объекты, которые последуют, должны будут отображаться по возможности в цвете h.
- **Hue[h, s, b]** – определяет цвета в значениях оттенка h, насыщенности s и яркости b.
- **LineForm[g]** – устанавливает, что вывод линий следует выполнять с применением графической директивы g или списка графических директив g.
- **PointForm[g]** – указывает форму точек объекта g.
- **PointSize[r]** – указывает, что точки при последующем выводе должны изображаться по возможности в виде кругов с радиусом r (дробь от общей ширины графика).
- **RGBColor[red, green, blue]** – указывает, что последующие графические объекты должны отображаться заданной совокупностью цветов. Значения red (красный), green (зеленый) и blue (синий) указываются в относительных единицах – от 0 до 1.
- **SurfaceColor[dcol]** – устанавливает, что последующие многоугольники должны действовать как рассеивающие (диффузные) отражатели света с заданным цветом dcol.
- **SurfaceColor[dcol, scol]** – указывает, что должен содержаться компонент зеркального отражения с цветом, заданным scol.
- **SurfaceColor[dcol, scol, n]** – указывает, что отражение должно происходить с показателем зеркального отражения n.

Применение указанных функций и опций позволяет строить большое число графиков различных типов даже при задании одной и той же поверхности. В качестве примера рассмотрим отдельные кадры документа, демонстрирующего влияние опций на вид 3D-математической поверхности.

11.3.3. Примеры модификации 3D-графиков с помощью опций

На рис. 11.9 показана поверхность, построенная с применением опции PlotPoint->50. Это означает, что поверхность по каждой оси делится на 50 частей (в исходном графике по умолчанию используется деление в 10 раз). Масштаб по вертикали задается автоматически, с тем чтобы все высоты поверхности не ограничивались.

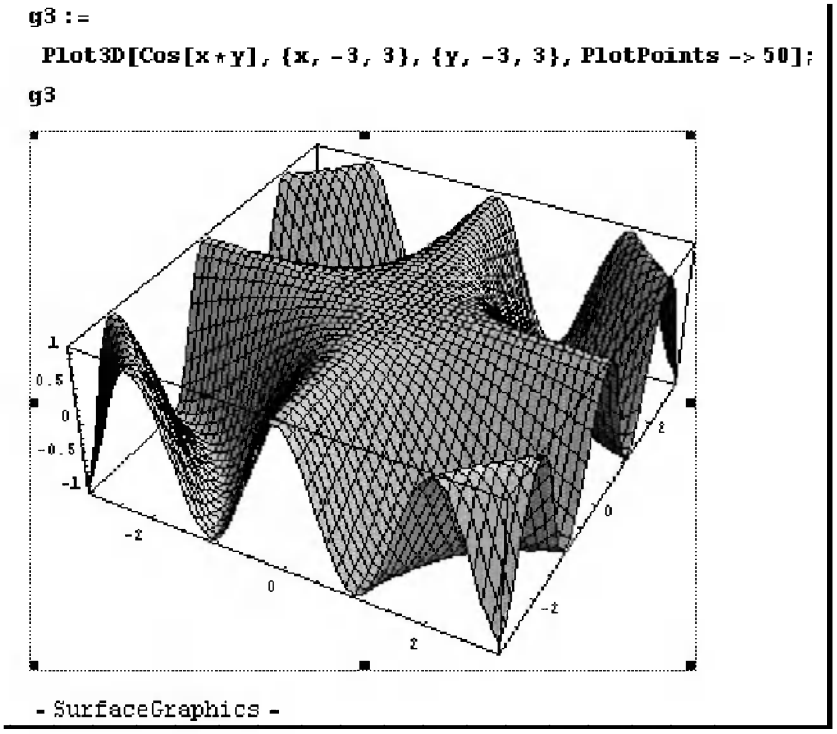


Рис. 11.9. Исходная математическая поверхность

На рис. 11.10 та же поверхность показана с применением при построении опции **PlotRange**, срезающей верхнюю часть поверхности. График поверхности при этом существенно меняется (сравните с рис. 11.9).

Опция **Boxed->False** удаляет ограничивающие рамки, образующие ящик, в который вписывается строящаяся трехмерная поверхность.

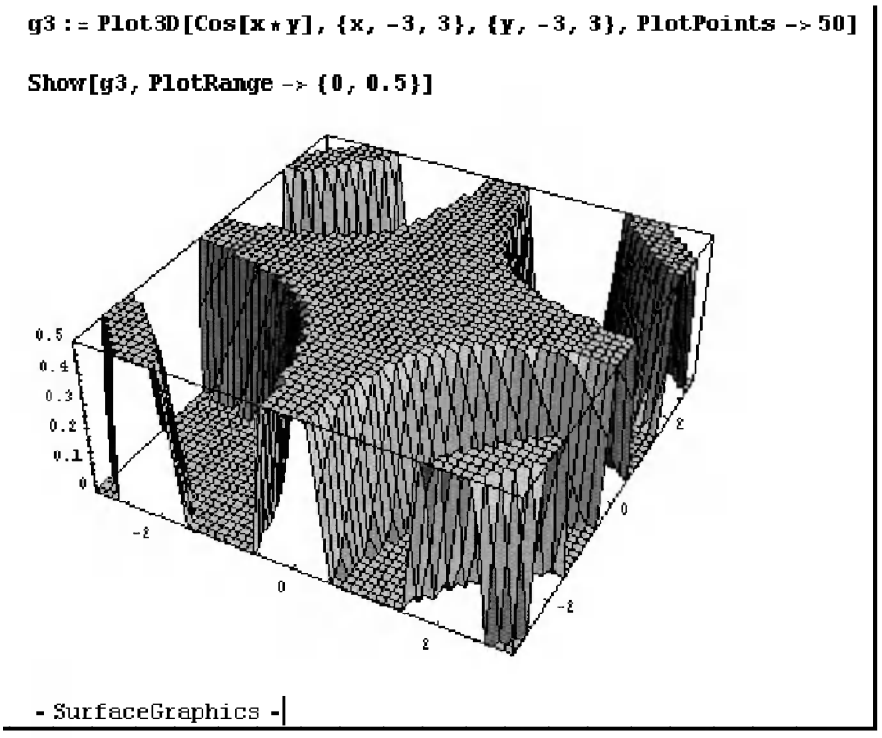


Рис. 11.10. Математическая поверхность с отсеченной верхней частью

Опция **ViewPoint** позволяет включить при построении отображение перспективы и изменять углы, под которыми рассматривается фигура. Рисунок 11.11 иллюстрирует применение этой опции.

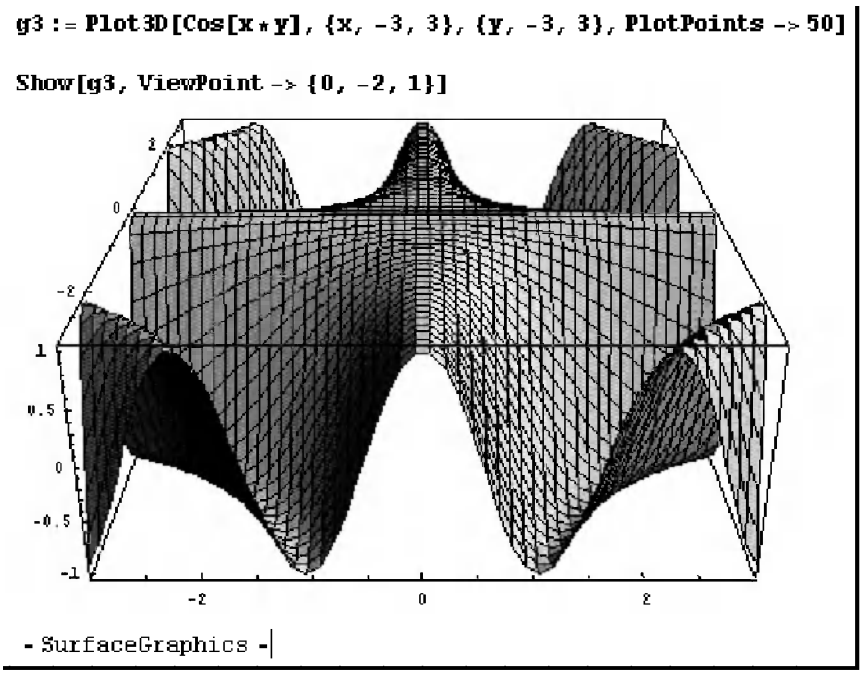


Рис. 11.11. Математическая поверхность, построенная в перспективе

Опция **Mesh->False** позволяет удалить линии каркаса фигуры. Нередко это придает фигуре более естественный вид – обычно мы наблюдаем такие фигуры без линий каркаса.

В ряде случаев, напротив, именно линии каркаса несут важную информацию. Система строит каркас 3D-поверхности для двух типов построений – с использованием алгоритма удаления невидимых линий и без использования этого алгоритма. Рисунок 11.12 показывает построение каркаса без удаления невидимых линий. Такой вид математическая поверхность имеет, если представить ее построенной из тонких проволочек, висящих в пространстве. Это дает дополнительную информацию о пространственной фигуре, но эстетически она выглядит хуже, чем фигура, построенная с применением алгоритма удаления невидимых линий каркаса.

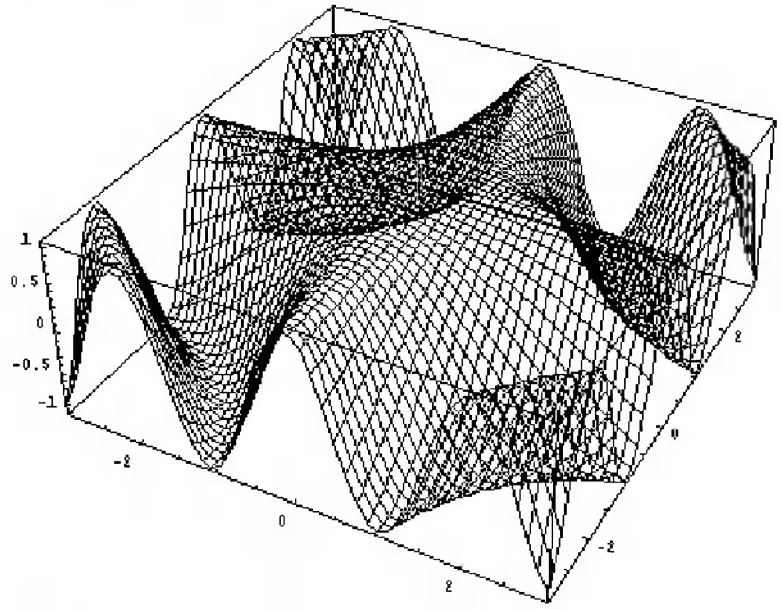
11.3.4. Графическая функция ListPlot3D

Часто 3D-поверхность задается массивом своих высот (аппликат). Для построения графика в этом случае используется графическая функция **ListPlot3D**:

- **ListPlot3D[array]** – строит трехмерный график поверхности, представленной массивом значений высот (рис. 11.13).
- **ListPlot3D[array, shades]** – строит график так, что каждый элемент поверхности штрихуется (затеняется) согласно спецификации в shades.
- **PlotJoined** – дополнительная опция для ListPlot, указывающая, следует ли точки, нанесенные на график, соединять линией.

```
g3 :=
 Plot3D[Cos[x*y], {x, -3, 3}, {y, -3, 3}, PlotPoints -> 50]

Show[g3, HiddenSurface -> False]
```

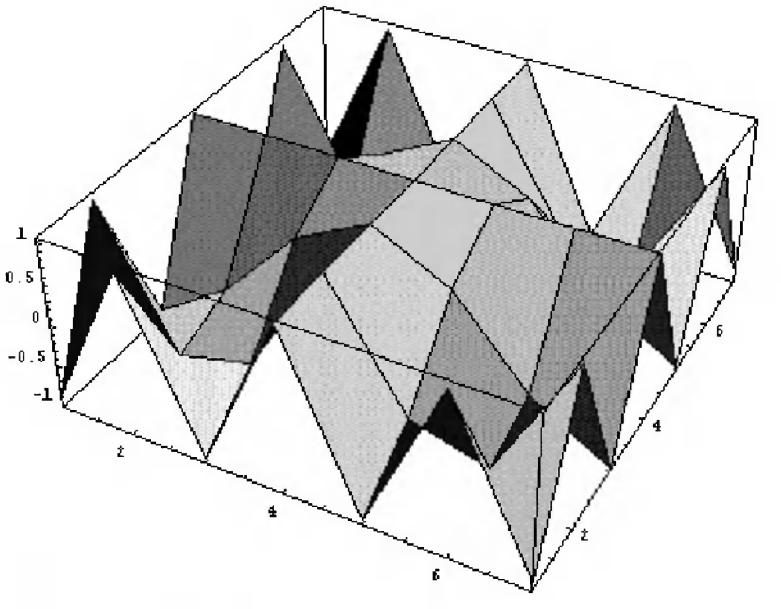



```
- SurfaceGraphics -
```

Рис. 11.12. Построение каркаса математической поверхности без использования алгоритма удаления невидимых линий

```
t3 = Table[Cos[x*y], {y,-3, 3}, {x,-3, 3}] ;

ListPlot3D[t3]
```




```
- SurfaceGraphics -
```

Рис. 11.13. Пример применения функции ListPlot3D

Командой **Options[ListPlot3D]** можно вывести полный список опций данной функции и использовать их для модификации графиков, которые строит эта функция.

11.3.5. Параметрическая 3D-графика

Особый шик построениям 3D-фигур и поверхностей придает функция **ParametricPlot3D**, в которой предусмотрено параметрическое задание всех трех функций, описывающих координаты каждой точки. Данная функция используется в следующих видах:

- **ParametricPlot3D[{fx, fy, fz}, {t, tmin, tmax}, {u, umin, umax}]** – строит трехмерную поверхность, параметризованную по t и u.
- **ParametricPlot3D[{fx, fy, fz}, {t, tmin, tmax}]** – выполняет трехмерную пространственную кривую, параметризованную переменной t, которая изменяется от tmin до tmax.
- **ParametricPlot3D[{fx, fy, fz, s},...]** – выполняет затенение графика в соответствии с цветовой спецификацией s.
- **ParametricPlot3D[{{fx, fy, fz}, {gx, gy, gz}, ...}, ...]** – строит несколько объектов вместе.

Эта функция имеет множество опций, список которых выводит команда

```
Options[ParametricPlot3D]
```

Большая часть из них уже рассматривалась ранее. При этом даже при использовании только опций, заданных по умолчанию, можно получить любопытные построения. На рис. 11.14 показан простой пример применения функции **ParametricPlot3D** для построения замкнутой линии, расположенной в пространстве. Это так сказать объемный вариант фигур Лиссажу, построение которых было описано выше.

Параметрическое задание функций позволяет легко строить сложные пространственные фигуры, визуально весьма напоминающие реальные объекты.

11.3.6. Построение фигур, пересекающихся в пространстве

Пожалуй, наиболее впечатляющими являются построения 3D-фигур, пересекающихся в пространстве. Для этого достаточно каждую фигуру представить в виде графического объекта, а затем с помощью директивы **Show** вывести их на одном графике. При этом Mathematica автоматически рассчитывает линии пересечения фигур и строит график так, чтобы заслоненные ячейки фигур не были видны.

Проиллюстрируем это с помощью рис. 11.15. На нем показаны задание и построение одного графического объекта g1 – объемной спирали, полученной сворачиванием ленты.

Второй объект, построение которого представлено на рис. 11.16, – это объемное кольцо. Его построение было описано выше. В конце части документа, показанного на рис. 11.16, задана функция **Show** вывода объектов на одном графике.

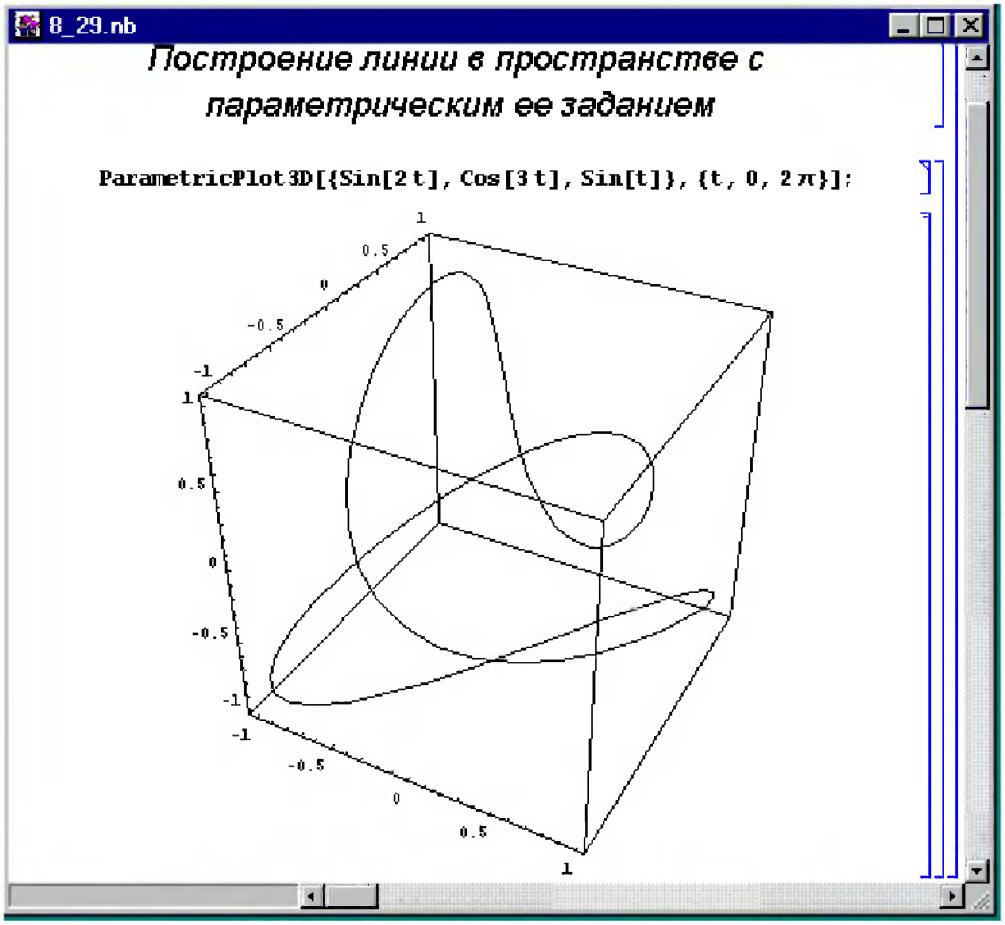


Рис. 11.14. Построение кривой в пространстве, заданной в параметрической форме

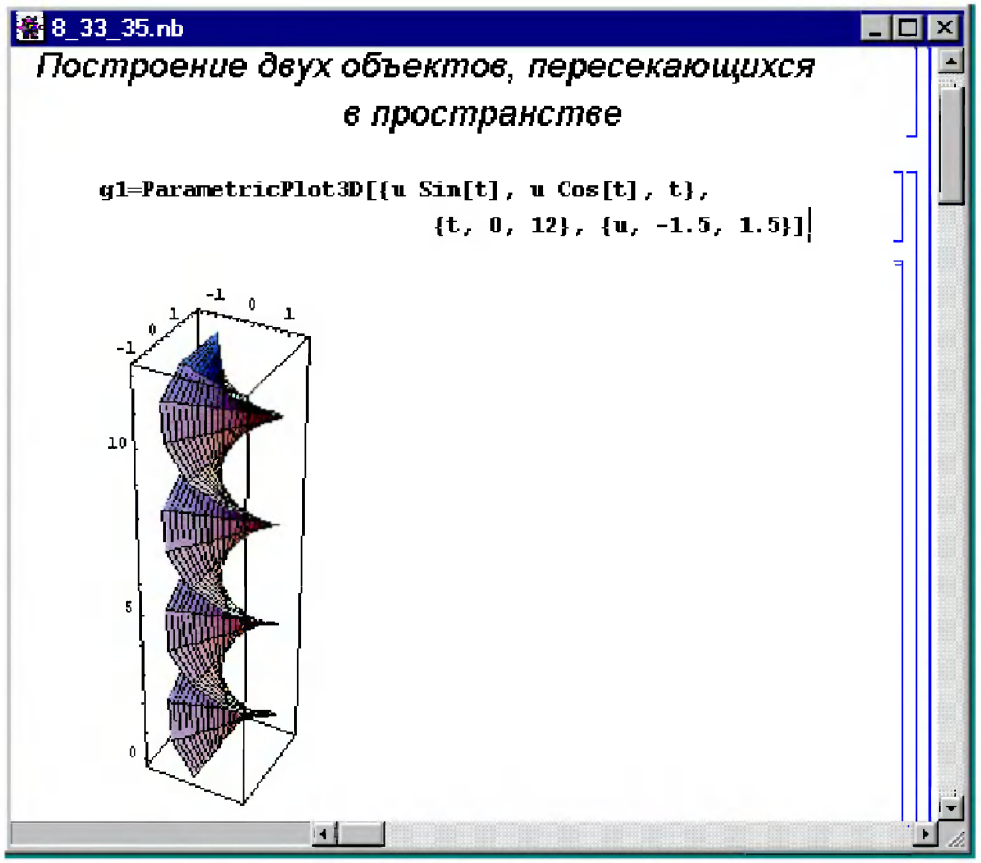


Рис. 11.15. Построение объекта g1 – объемной спирали

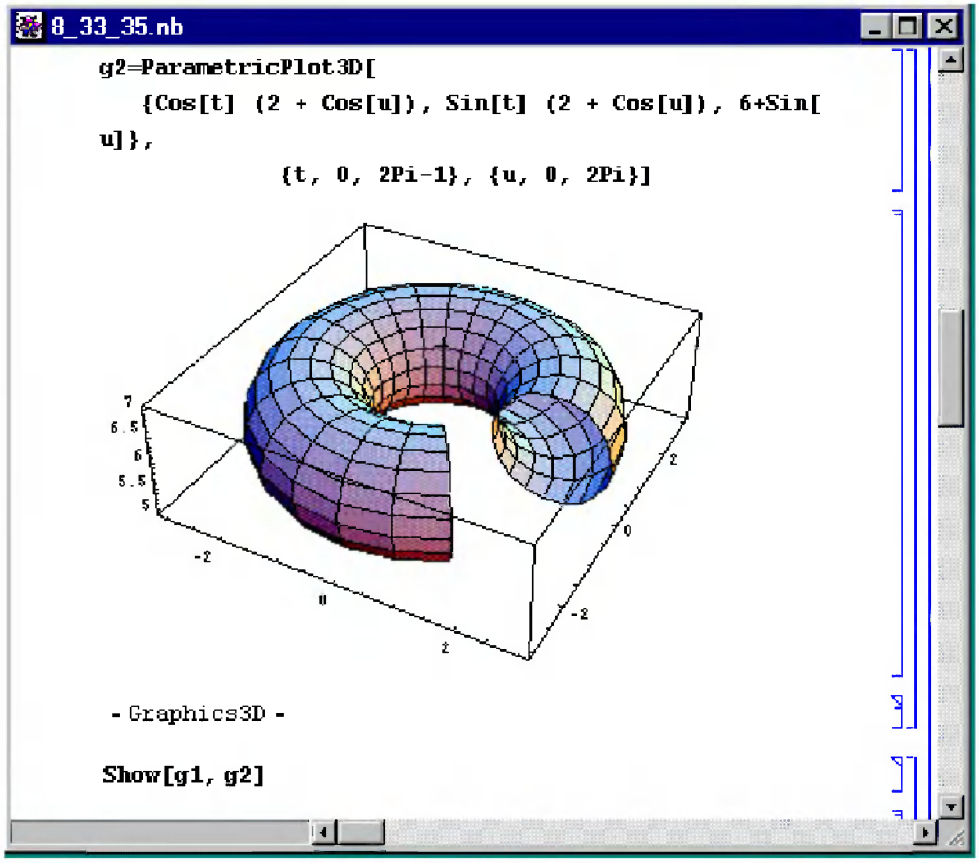


Рис. 11.16. Построение объекта g2 – объемного кольца с удаленным сегментом

Рисунок 11.17 демонстрирует комбинированный график, построенный функцией Show. Он показывает кольцо, через отверстие которого проходит объемная спираль. Вырез в кольце показывает, как проходит спираль внутри кольца.

Графики такого типа дают высокую степень визуализации трехмерных поверхностей и фигур.

11.3.7. Функция Graphics3D и ее опции и примитивы

Наряду с построением графиков поверхностей, заданных аналитическими выражениями, имеется возможность создания графиков из различных элементарных геометрических объектов, называемых примитивами. Они включаются в список параметров функции **Graphics3D**:

- **Graphics3D[primitives, options]** – представляет трехмерное графическое изображение.

С ней, помимо примитивов 2D-графики, могут использоваться следующие графические примитивы:

- **Cuboid[{xmin, ymin, zmin}]** – представляет единичный куб, ориентированный параллельно осям.
- **CellArray[{{a11, a12, ...}, ...}]** – представляет прямоугольный массив элементов яркости.

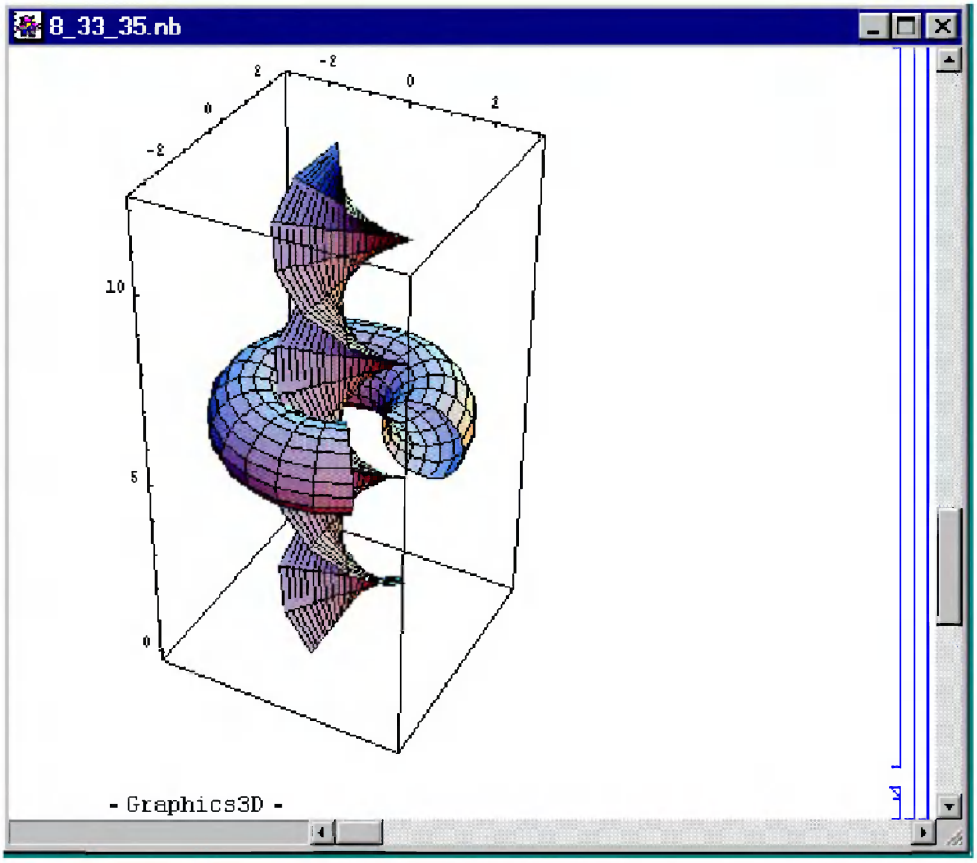


Рис. 11.17. Построение комбинированного объекта – спираль внутри кольца

- **Cuboid[{xmin, ymin, zmin}, {xmax, ymax, zmax}]** – представляет прямоугольный параллелепипед, заданный координатами противоположных вершин.
- **PostScript[\"string1\", \"string2\", ...]** – графический примитив, задающий построение графика по кодам языка PostScript.
- **SurfaceGraphics[array, shades]** – представляет поверхность, части которой затеняются согласно массиву shades.
- **SurfaceGraphics[array]** – представляет трехмерный график поверхности, для которого значения высоты каждой точки на сетке заданы элементами массива.
- **SurfaceGraphics[array, shades]** – представляет поверхность, части которой затеняются согласно массиву shades.
- **SurfaceGraphics[array]** – представляет трехмерный график поверхности, для которого значения высоты каждой точки на сетке заданы элементами массива.

Функция **Graphics3D** со своими примитивами может использоваться для построения в пространстве различных объектов, например точек, кубиков или многоугольников. На рис. 11.18 показано построение в пространстве ряда небольших

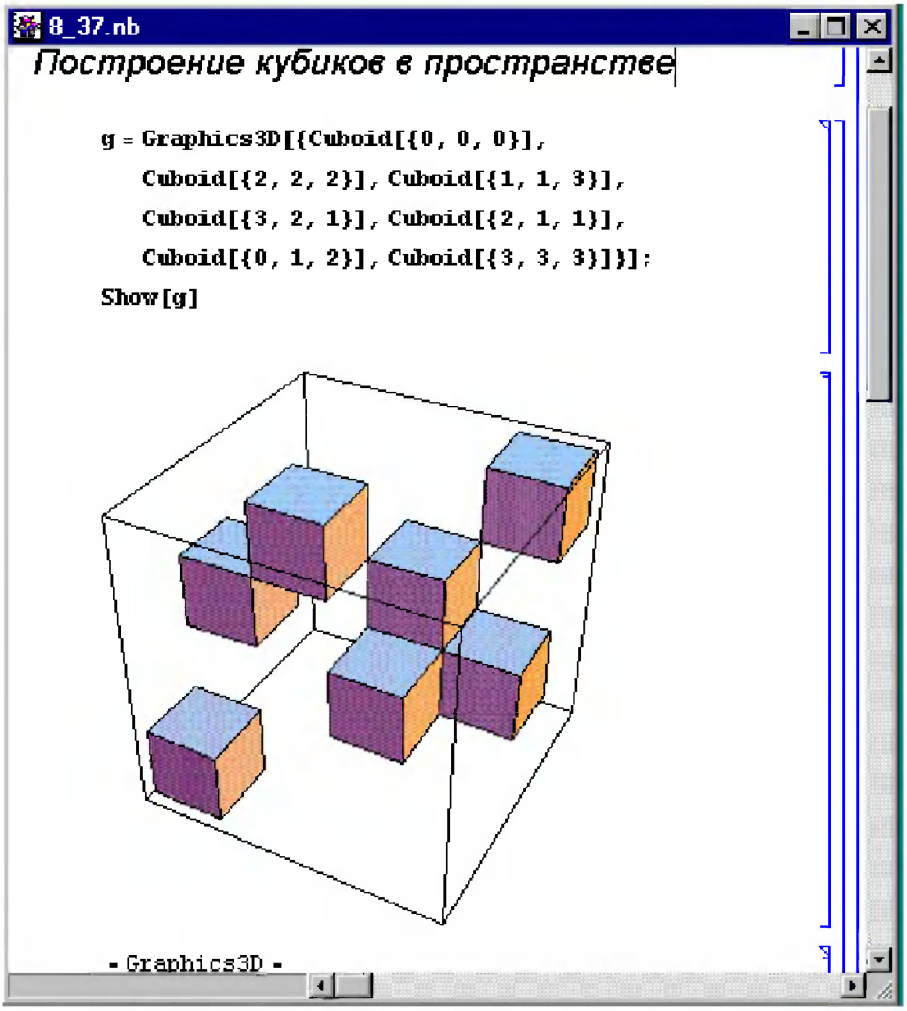


Рис. 11.18. Построение в пространстве ряда кубиков

кубиков. Для этого используется примитив **Cuboid**, повторенный 7 раз. Для воспроизведения набора кубиков, перечисленных в функции **Graphics3D**, применяется функция-директива **Show**.

Нетрудно заметить, что и здесь неплохо работают встроенные алгоритмы удаления невидимых частей объектов. Это дает довольно реалистическое изображение их в пространстве.

Еще более наглядное представление об этом алгоритме дает рис. 11.19. На нем показано построение в пространстве ряда плоских многоугольников, частично проникающих друг в друга. Нетрудно заметить, что и здесь алгоритм удаления невидимых поверхностей работает превосходно.

Здесь каждый из многоугольников формируется с помощью функции пользователя randpoly[n_], в теле которой используется примитив **Polygon**. Эта функция формирует случайные многоугольники, выводимые затем функцией-директивой **Show**.

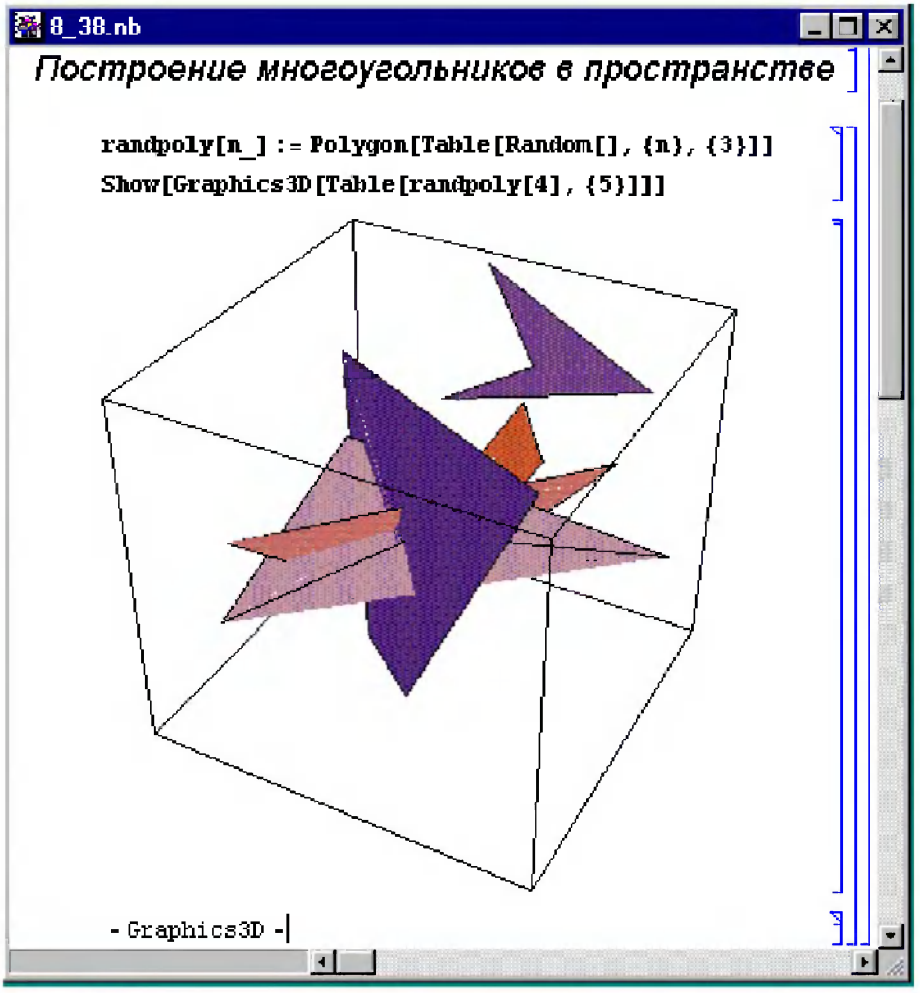


Рис. 11.19. Построение в пространстве взаимно пересекающихся плоских многоугольников

11.4. Графика пакета дискретной геометрии DiscreteMath

Пакет DiscreteMath содержит два класса функций дискретной математики. Первый – функции комбинаторики. Они уже были описаны. Второй – графические функции теории графов. Они описаны ниже.

11.4.1. Графы и их функции

Mathematica имеет самые обширные возможности в решении задач, связанных с графами. Задание графов и манипуляции с ними также включены в пакет комбинаторики. Они представлены четырьмя группами функций:

- представление графов;
- создание графов;
- свойства графов;
- алгоритмическая теория графов.

Ввиду большого числа этих функций рассмотрим применение лишь наиболее характерных из функций.

Одной из самых важных функций группы представления графов является функция **ShowGraph** (Показать граф). Она обеспечивает визуальное представление графа, заданного аргументом функции. Покажем работу избранных функций этой группы на нескольких примерах.

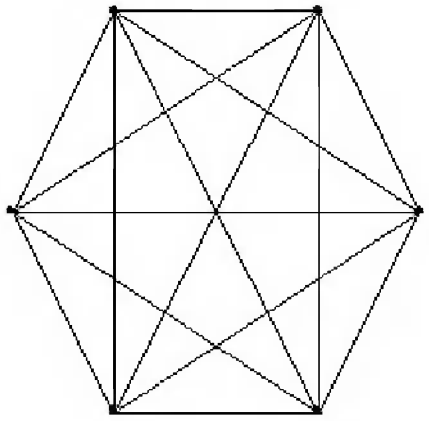
На рис. 11.20 показано построение полного графа и его таблицы. Параметром графа является число 6, характеризующее число узловых точек графа, соединенных друг с другом.

Изменяя значение параметра графа, можно получить множество других графов.

На рис. 11.21 показан вид двух разных графов. Верхний граф – многолучевая звезда, построенная отрезками прямых с помощью функции **AddEdge**. Первый аргумент задает число лучей графа, а второй – соединяемые отрезком прямой точки. Нижний рисунок иллюстрирует построение субграфа.

```
<<DiscreteMath`Combinatorica`

ShowGraph[ CompleteGraph[6] ];
```

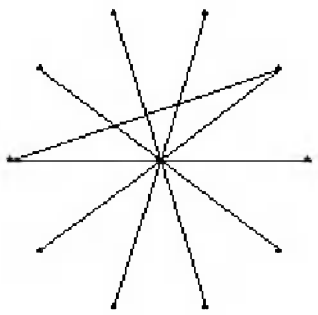


```
TableForm[ Edges[CompleteGraph[6]] ]
```

0	1	1	1	1	1
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	0	1	1
1	1	1	1	0	1
1	1	1	1	1	0

Рис. 11.20. Пример построения полного графа и его таблицы

```
ShowGraph[ AddEdge[Star[11], {1,5}]];
```



```
ShowGraph[ InduceSubgraph[CompleteGraph[
18], RandomSubset[Range[18]]]];
```

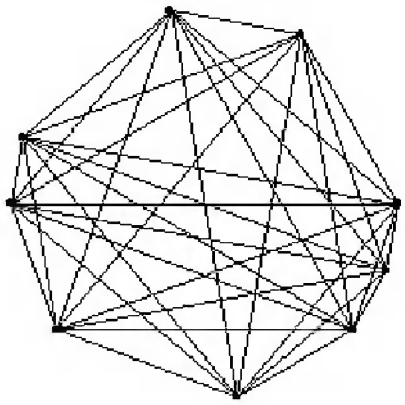


Рис. 11.21. Построение графа звезды и субграфа

Еще пара графов представлена на рис. 11.22. Этот рисунок иллюстрирует применение функций **Contract** и **GridGraph**. Последняя из них строит сеточный граф.

Набор функций данного пакета позволяет строить практически любые виды графов и обеспечивает высокую степень их визуализации. Из функций создания

графов рассмотрим две функции, представленные на рис. 11.23 (**GraphUnion** – верхний график и **GraphProduct** – нижний график).

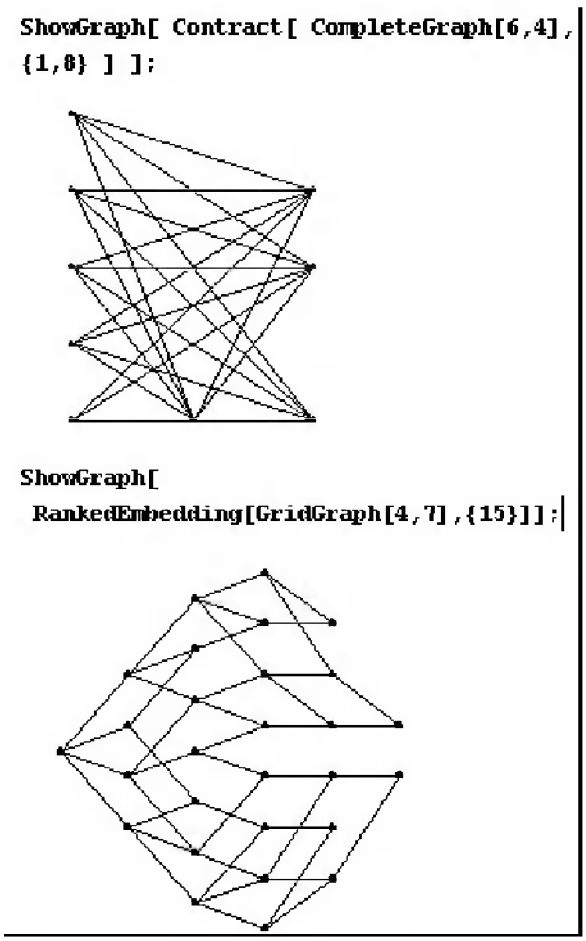


Рис. 11.22. Примеры построения графов с помощью функций Contract и GridGraph

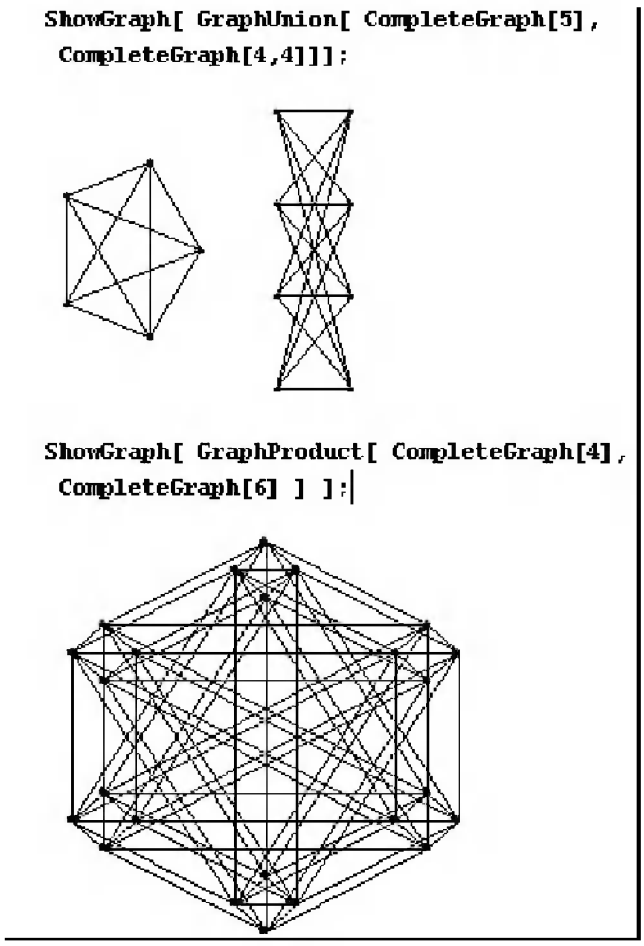


Рис. 11.23. Создание графов с помощью функций GraphUnion и GraphProduct

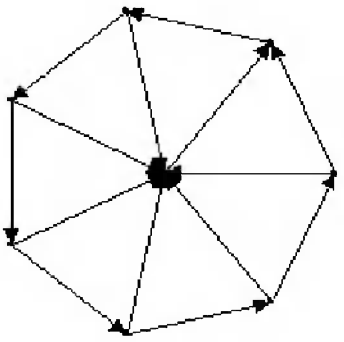
С действием других функций этой группы нетрудно ознакомиться самостоятельно. Рисунок 11.24 (сверху) показывает применение функции **OrientGraph** для построения ориентированного графа, который представляется стрелками. Там же показано применение функции **ShowLabelGraph** (снизу) для построения графа с маркированными числами вершинами. Напомним, что функция **ShowGraph** позволяет наблюдать графы без маркировки вершин.

Построение широко используемой в теории графов диаграммы Хассе (Hasse) иллюстрирует рис. 11.25.

Рисунок 11.26 показывает действие функции **MinimumSpanningTree** с выводом графа с метками узловых точек. Это лишь одна из многих функций алгебраической теории графов.

В целом следует отметить, что набор функций в области создания, визуализации и теории графов весьма представителен, и специалисты в области графов могут найти в этом наборе как типовые, так и уникальные средства.

```
ShowGraph[ OrientGraph[Wheel[8]],
Directed];
```



```
ShowLabeledGraph[g = MakeGraph[Range[9],
(Mod[#1,#2]==0)&] ];
```

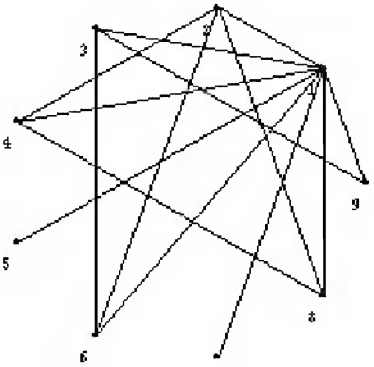


Рис. 11.24. Построение графов – ориентированного (сверху) и с маркированными вершинами (снизу)

```
ShowLabeledGraph[
  HasseDiagram[MakeGraph[Subsets[4],
   ((Intersection[#2,#1]==#1)&&(#1 !=
#2))&]], Subsets[4]];
```

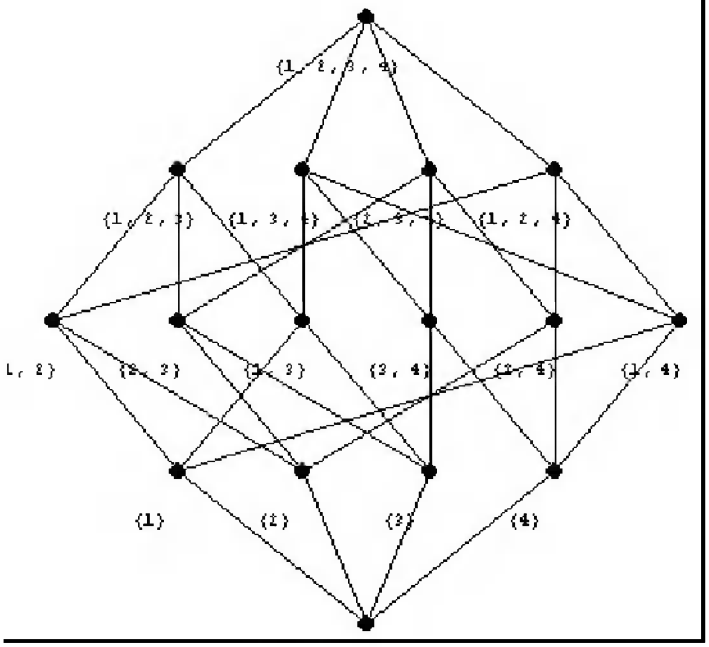


Рис. 11.25. Построение диаграммы Хассе

```
ShowLabeledGraph[ MinimumSpanningTree[
CompleteGraph[5,5,6,7]]];
```

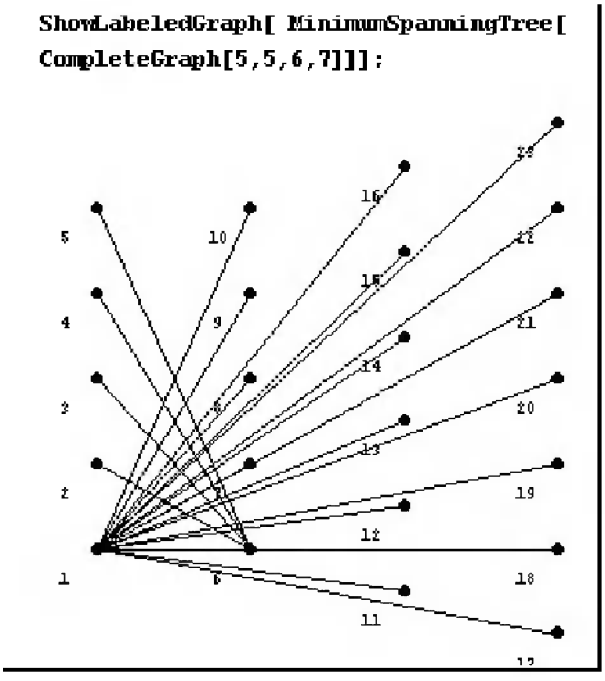


Рис. 11.26. Пример применения функции MinimumSpanningTree

11.4.2. Функции вычислительной геометрии

В подпакете ComputationalGeometry заданы следующие функции, относящиеся к **геометрическим поверхностям**:

- **ConvexHull[{{x1,y1...},{x2,y2,...},...]** – вычисляет выпуклость оболочки в точках плоскости.
- **DelaunayTriangulation[{{x1,y1...},{x2,y2,...},...]** – вычисляет триангуляцию Делоне (разбиение на выпуклые треугольники) в точках плоскости.
- **DelaunayTriangulationQ[{{x1,y1...},{x2,y2,...},...},trival]** – тестирует триангуляцию Делоне в точках плоскости.
- **DiagramPlot[{{x1,y1...},{x2,y2,...},...]** – построение диаграммы по заданным точкам (возможны спецификации после листа параметров в виде списков diagvert, diagval).
- **PlanarGraphPlot[{{x1,y1...},{x2,y2,...},...]** – построение планарного графа по заданным точкам (возможна спецификация после списка параметров в виде списка indexlist или vals).
- **TriangularSurfacePlot[{{x1,y1,z1},{x2,y2,z2},...]** – строит поверхность из треугольников по заданным точкам.
- **VoronoiDiagramm[{{x1,y1...},{x2,y2,...},...]** – вычисляет данные для построения диаграммы Вороного.

Примеры на применение этих функций даны ниже:

```
<<DiscreteMath`ComputationalGeometry`
 ConvexHull[{{0,2},{1,1},{0,0},{2,0},{1,2}}]
 {4  ,5  , ,3  }
  delval = (DelaunayTriangulation[{{1,2},{0,3},{1,1}}])
  // Short[#,2]&
 (  (#1&))[{{1,{2,3}},{2,{3,1}},{3,{1,2}}}]
  VoronoiDiagram[{{1,2},{0,3},{1,1}}]
```

$$\left\{\left\{-\frac{1}{2}, \frac{3}{2}\right\}, \text{Ray}\left[\left\{-\frac{1}{2}, \frac{3}{2}\right\}, \left\{\frac{3}{2}, \frac{7}{2}\right\}\right],\right.$$
$$\text{Ray}\left[\left\{-\frac{1}{2}, \frac{3}{2}\right\}, \left\{2, \frac{3}{2}\right\}\right], \text{Ray}\left[\left\{-\frac{1}{2}, \frac{3}{2}\right\}, \left\{-\frac{5}{2}, \frac{1}{2}\right\}\right]\},$$
$$\left.\{\{, \{, 3, 2\}\}, \{2, \{, 2, 4\}\}, \{3, \{, 4, 3\}\}\}\right\}$$

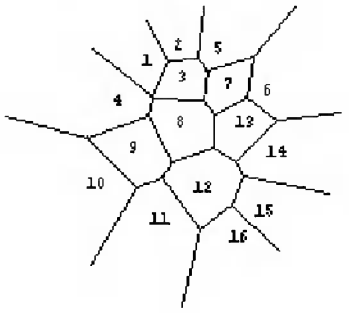
На рис. 11.27 показаны результаты действия еще двух функций – построение диаграммы и триангуляции в пространстве.

11.4.3. Дискретные функции единичного скачка

В подпакете KroneckerDelta системы Mathematica 4/5 заданы дискретные функции *единичного скачка*:

- **DiscreteStep[n]** – возвращает единичный скачок при целом n=0.
- **DiscreteStep[n1,n2,...]** – функция многоразмерного единичного скачка.

```
DiagramPlot[data2D]
```




```
data3D = Map[Append[#, Sqrt[64-(#[[1]]-8)^2-(#[[2]]-8)^
2]]&, data2D];TriangularSurfacePlot[data3D]
```

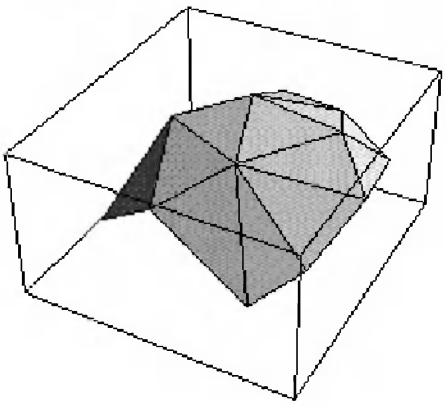


Рис. 11.27. Построение диаграммы (сверху) и триангуляция в пространстве (снизу)

Примеры применения одномерной функции представлены ниже:

```
<<DiscreteMath`DiscreteStep`
Table[DiscreteStep[{n}], {n, -3, 3}]
{{0}, {0}, {0}, { }, { }, { }, { }}
Table[DiscreteStep[n], {n, -3, 3, 1/2}]
{0, 0, 0, 0, 0, 0,  , 0,  , 0,  , 0,  }
DiscreteStep[{23, 0, -1, 0.25, -0.5}]
{1, 1, 0, 0, 0}
DiscreteStep[{2+3, 3-2-1, -1, 0.25, -0.5}]
{1, 1, 0, 0, 0}
```

В системе Mathematica 3 функция в данный пакет входила **KroneckerDelta**, но в Mathematica 4/5 она стала встроенной функцией.

В данный подпакет входит еще одна функция:

- **SimplifyDiscreteStep[expr]** – упрощение выражения expr с функциями дискретного скачка.

Действие этой функции демонстрирует следующий пример:

```
<<DiscreteMath`DiscreteStep`
 DiscreteStep[n — 1] — DiscreteStep[n- 2] //
  SimplifyDiscreteStep
 DiscreteDelta[-1+n]
```

11.4.4. Построение деревьев

Подпакет Tree задает функции задания и применения древовидных структур, именуемых **деревьями**. Вот эти функции:

- **MakeTree[list]** – создает дерево по информации, представленной в списке list.
- **TreeFind[tree,x]** – возвращает позицию наименьшего элемента, превосходящего x в списке list, представляющем дерево.

Действие этих функций поясняют следующие примеры:

```
<<DiscreteMath`Tree`
  MakeTree[{e1, e2, e3, e4}]
{{e2,2},{{e1,1},{},{}},{{e3,3},{},{{e4,4},{},{}}}}
 tree = MakeTree[{6.05, 4.48, 6.40, 2.46,
             3.43, 5.4, 2.0}]
{{4.48,4},{{2.46,2},{{2.,1},{},{}},{{3.43,3},{},{}}},{{6.05,6},{{5.4,5},
{},{}},{{6.4,7},{},{}}}}
  TreeFind[tree, 5.1]
4
 TreeFind[tree,1]
0
```

Для визуализации деревьев служат функции:

- **TreePlot[tree]** – строит график дерева tree.
- **ExprPlot[expr]** – строит график, представляющий expr в виде дерева.

Примеры построения графиков деревьев представлены на рис. 11.28. Верхний график построен по данным дерева tree, определенного в приведенных выше примерах, а нижний – по данным случайного дерева.

Построение графиков деревьев по выражению expr с помощью функции **ExprPlot** демонстрирует рис. 11.29.

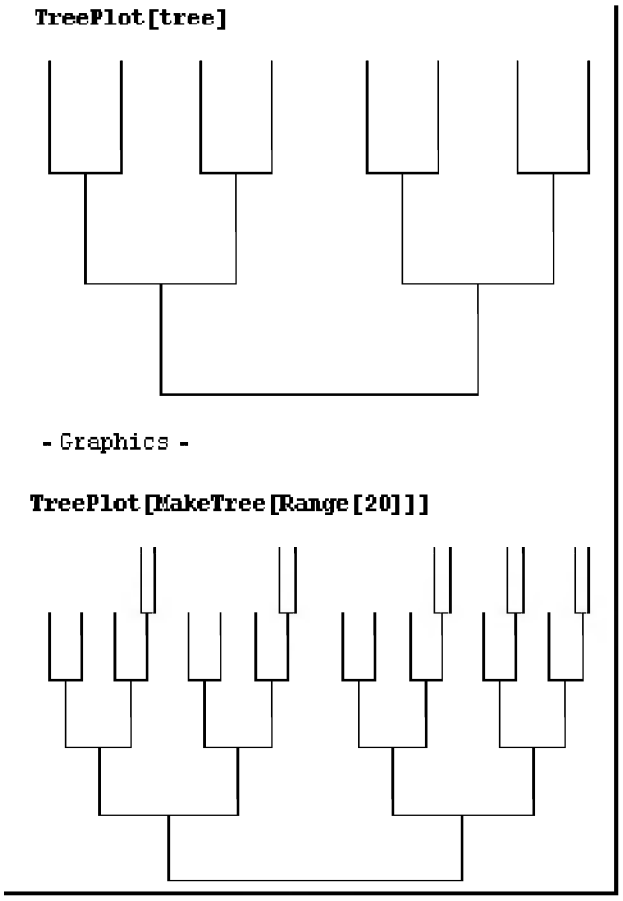


Рис. 11.28. Примеры визуализации деревьев

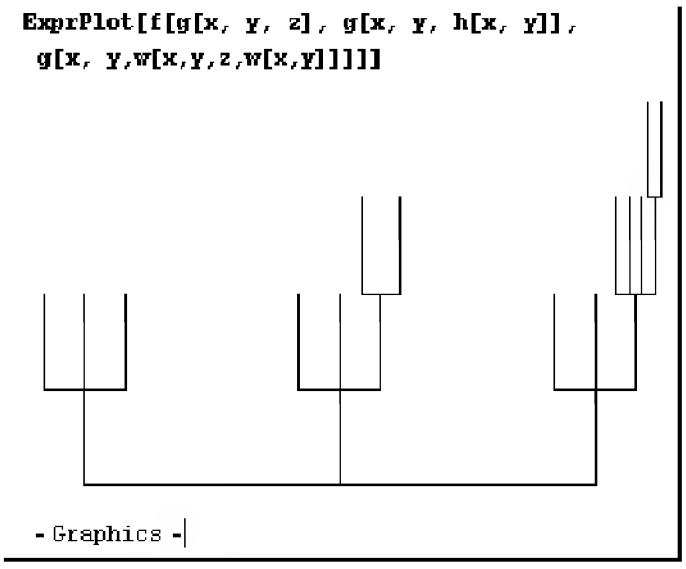


Рис. 11.29. Построение графиков деревьев с помощью функции ExprPlot

11.5. Пакет геометрических расчетов Geometry

Для проведения геометрических расчетов и их визуализации в системах Mathematica 4/5 служит пакет Geometry. Ниже описаны его основные возможности.

11.5.1. Характеристики регулярных полигонов и полиэдров – Polytopes

Подпакет Polytopes содержит ряд функций для *регулярных полигонов* (многоугольников). Эти функции позволяют вычислить следующие параметры фигур:

- **NumberOfVertices[p]** – число вершин углов у полигона.
- **NumberOfEdges[p]** – число сторон у полигона.
- **NumberOfFaces[p]** – число граней у полигона.
- **Vertices[p]** – список координат вершин углов у полигона.
- **Area[p]** – площадь полигона при длине каждой стороны, равной 1.
- **InscribedRadius[p]** – радиус вписанной в полигон окружности.
- **CircumscribedRadius[p]** – радиус описывающей полигон окружности.

В функциях наименование полигона p может быть следующим (в скобках дано число сторон):

Digon (2) **Triangle (3)** **Square (4)** **Pentagon (5)** **Heagon (6)** **Heptagon (7)**
Octagon (8) **Nonagon(9)** **Decagon (10)** **Undecagon (11)** **Dodecagon (12)**

На рис. 11.30 показаны примеры применения некоторых из этих функций и построение крупными точками вершин полигона – пентагона (пятиугольника).

Для объемных фигур – полиэдров имеются следующие функции:

- **NumberOfVertices[p]** – число вершин углов у полиэдра.
- **NumberOfEdges[p]** – число сторон у полиэдра.
- **NumberOfFaces[p]** – число граней у полиэдра.
- **Vertices[p]** – список координат вершин углов у полиэдра.
- **Area[p]** – площадь полиэдра при длине каждой стороны, равной 1.
- **InscribedRadius[p]** – радиус вписанной в полиэдр окружности
- **CircumscribedRadius[p]** – радиус окружности, описывающей полиэдр.
- **Volume[p]** – объем полиэдра.
- **Dual[p]** – дуальный полиэдр.
- **Schlafli[p]** – символ полиэдра.

Здесь наименование полиэдра может быть следующим:

Tetrahedron (4) **Cube (6)** **Octahedron (8)**
Didecahedron (12) **Icosahedron (20)**

Примеры на применение функций полиэдров представлены ниже:

```
<< Geometry`Polytopes`
  Volume[Octahedron]
```

$$\frac{\sqrt{2}}{3}$$

```
<< Geometry`Polytopes`

NumberOfEdges[Pentagon]

5

Area[Pentagon]

5 (1 + Sqrt[5])
---------------
4 Sqrt[2 (5 - Sqrt[5])]

Vertices[Pentagon]

{{0.309017, 0.951057}, {-0.809017, 0.587785},
 {-0.809017, -0.587785}, {0.309017, -0.951057}, {1., 0}}

Show[Graphics[{PointSize[.05], Point /@ %},
        AspectRatio -> 1]]
```

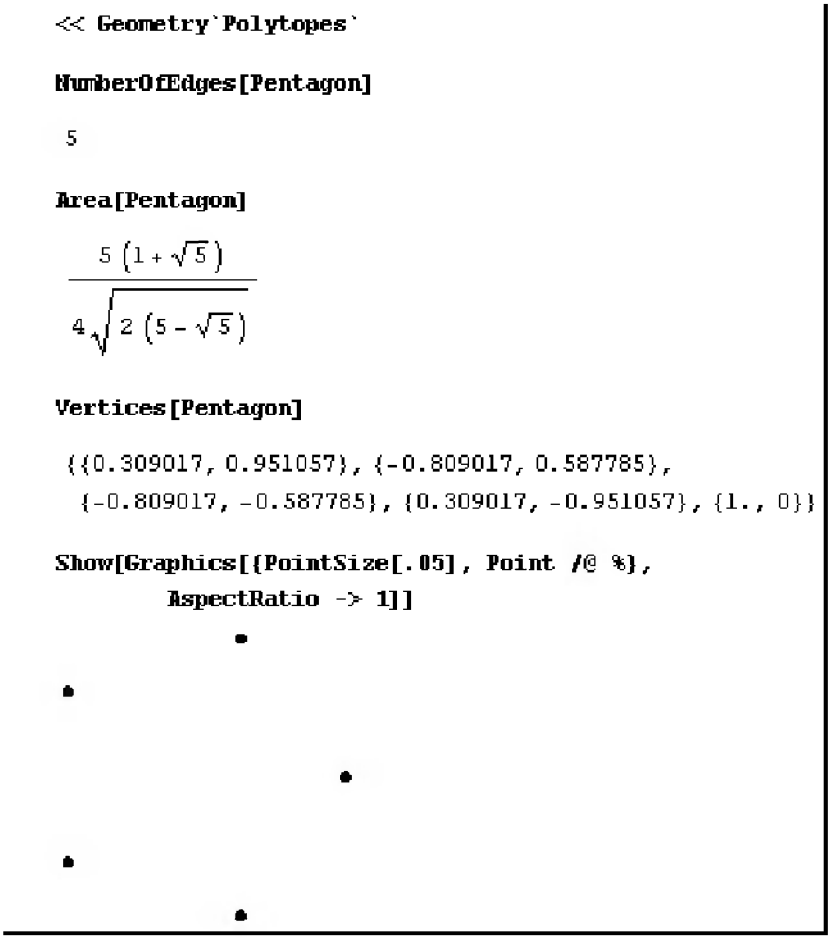


Рис. 11.30. Примеры работы с функциями полигонов

```
Vertices[Octahedron]
{{0, 0, Sqrt[2]}, {Sqrt[2], 0, 0}, {0, Sqrt[2], 0},
 {0, 0, -Sqrt[2]}, {-Sqrt[2], 0, 0}, {0, -Sqrt[2], 0}}
Dual[Octahedron]
Cube
InscribedRadius[Octahedron]
1
---
Sqrt[6]
CircumscribedRadius[Cube]
Sqrt[3]
---
2
```

11.5.2. Вращение фигур на плоскости и в пространстве – Rotations

Для задания *поворота* плоских фигур на заданный угол в подпакете Rotations заданы функции:

- **RotationMatrix2D[theta]** – дает данные поворота матрицы на угол theta.
- **Rotate2D[vec,theta]** – дает данные поворота вектора на угол theta.
- **Rotate2D[vec,theta,{x,y}]** – дает данные поворота вектора на угол theta относительно точки с координатами {x,y}.

Аналогичные функции заданы для получения данных поворота у трехмерных фигур:

- **RotationMatrix3D[psi,theta,phi]** – дает данные поворота матрицы 3D на заданные углы.
- **Rotate3D[vec,psi,theta,phi]** – дает данные поворота вектора в 3D-пространстве на заданные углы.
- **Rotate3D[vec,psi,theta,phi,{x,y,z}]** – дает данные поворота вектора в 3D-пространстве на заданные углы theta относительно точки с координатами {x,y,z}.

Приведем пример на вращение матрицы:

```
<<Geometry`Rotations`
  RotationMatrix3D[Pi, Pi/2, Pi/6]
```

$$\left\{\left\{-\frac{\sqrt{3}}{2}, 0, \frac{1}{2}\right\}, \left\{\frac{1}{2}, 0, \frac{\sqrt{3}}{2}\right\}, \{0, 1, 0\}\right\}$$

```
  MatrixForm[%]
```

$$\begin{pmatrix} -\frac{\sqrt{3}}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{\sqrt{3}}{2} \\ 0 & 1 & 0 \end{pmatrix}$$

11.6. Новые возможности визуализации в Mathematica 5.1

11.6.1. Новые средства визуализации графов

В системе Mathematica 5.1 в пакете дискретной математики DiscreteMath возможности построения графов расширены. В частности, появилась возможность построения объемных графов, представленная на рис. 11.31. Для построения используется список подстановок.

Для построения графов могут использоваться также новые графические функции GraphPlot и GraphPlot3D, которые строят графики, подобные приведенным на рис. 11.60. Возможно как прямое задание данных для построения графов, так и импорт данных из файла.

11.6.2. Средства GUI Mathematica 5.1

В Mathematica 5.1 введены также новые средства визуализации на основе графического интерфейса пользователя GUI. Из примерно десятка таких средств, представляющих инструментарий GUI Kit, мы рассмотрим только одно – Equation Trekker. На рис. 11.32 представлено окно этого инструмента, представляющее фа-

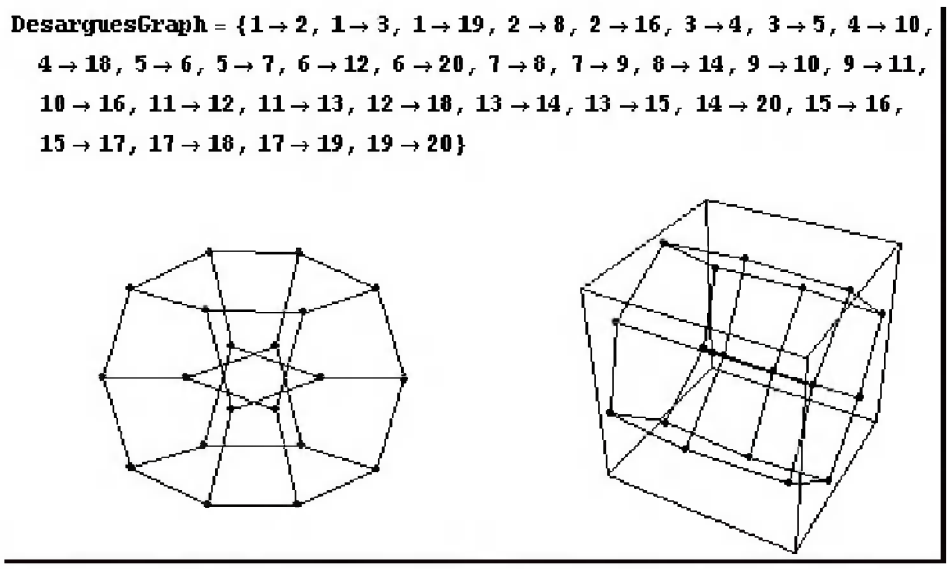


Рис. 11.31. Пример применения функции DesarguesGraph

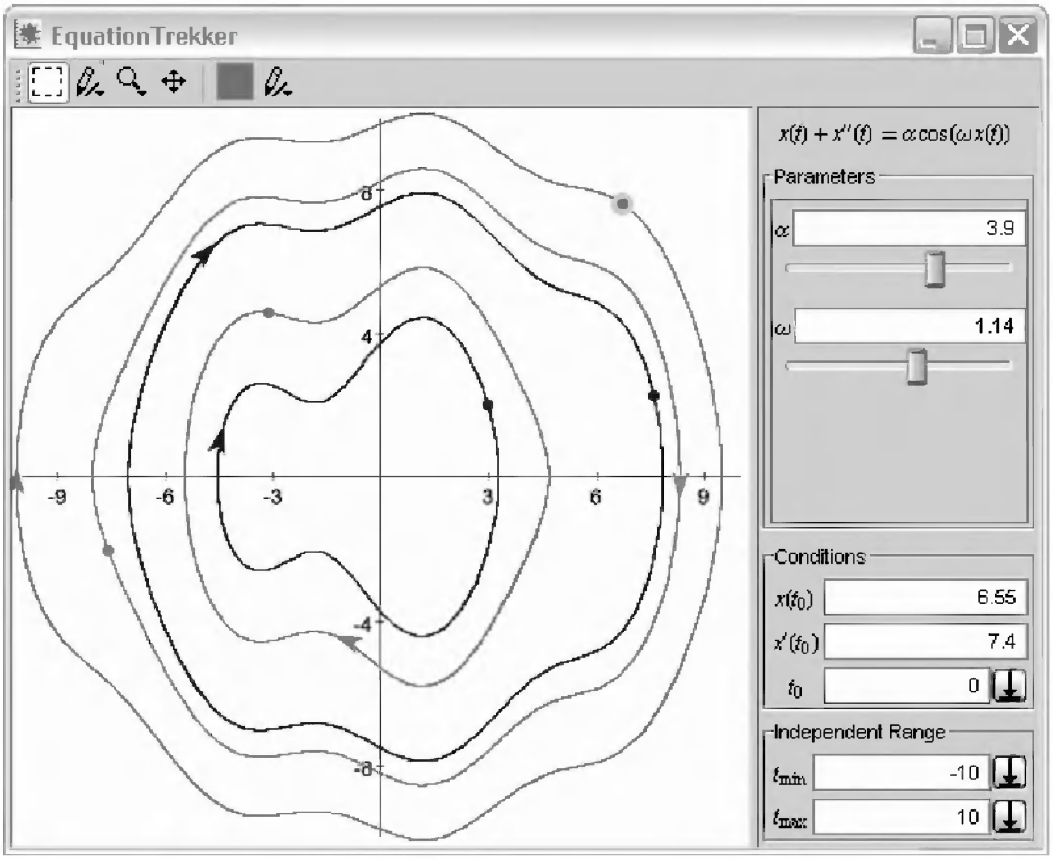


Рис. 11.32. Окно инструмента Equation Trekker системы Mathematica 5.1

зовые портреты решения дифференциального уравнения, представленного в правом верхнем углу. В поле Parameters имеются движковые регуляторы, позволяющие менять параметры уравнения. Под ними есть поля установки начальных условий и пределов изменения времени. Работа с инструментом вполне очевидна.

Нетрудно заметить, что данный инструмент очень напоминает инструмент Maplet системы Maple 9.5. Это говорит о том, что новые средства (в данном случае GUI) постепенно находят дорогу в конкурирующие СКА и СКМ.

11.6.3. Новые средства анимации в Mathematica 5.1

СКМ Mathematica 5.1 пополнилась и новыми эффективными средствами анимации, которые особенно полезны при решении физических задач. К сожалению, решение сложных физических задач требует слишком много места, и подобные задачи в данной книге сознательно не приводятся. Однако стоит продемонстрировать решение одной из таких задач – движение упругого шарика, отскакивающего от сложной неровной наклонной поверхности (рис. 11.33).

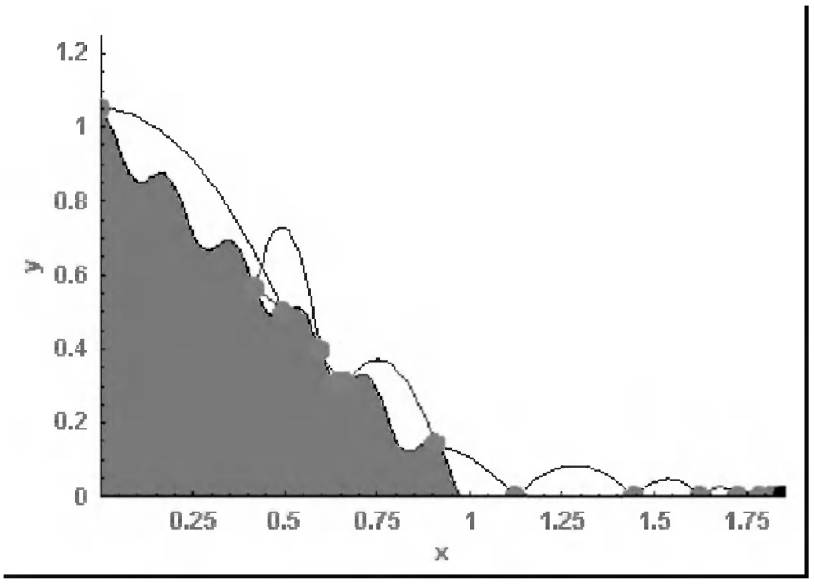


Рис. 11.33. Конечный кадр анимации – отскока упругого шарика от сложной неровной наклонной поверхности

Рисунок 11.33 показывает довольно сложную траекторию шарика. Ее сложность обусловлена тем, что направление отскока шарика может время от времени меняться в зависимости от того, на какой склон того или иного уступа попадает шарик. Например, такой отскок представлен при x, примерно равном 0,5. Рисунки, подобные рис. 11.62, дают очень наглядную картину сложного движения даже такого простого физического объекта, как упругий шарик.

11.6.4. Визуализация скорости вычислений в системе Mathematica 5.1

В некоторых СКМ, например в MATLAB, имеются средства контроля скорости вычислений в заданной компьютерной системе в сравнении со скоростью вычислений в ряде типовых конфигураций компьютера. Это позволяет судить об эффективности работы компьютера и учитывать это при его приобретении.

Аналогичное средство теперь есть и в системе Mathematica 5.1. Часть его окна со сравнительными данными о скорости вычислений разных компьютеров показана на рис. 11.34.

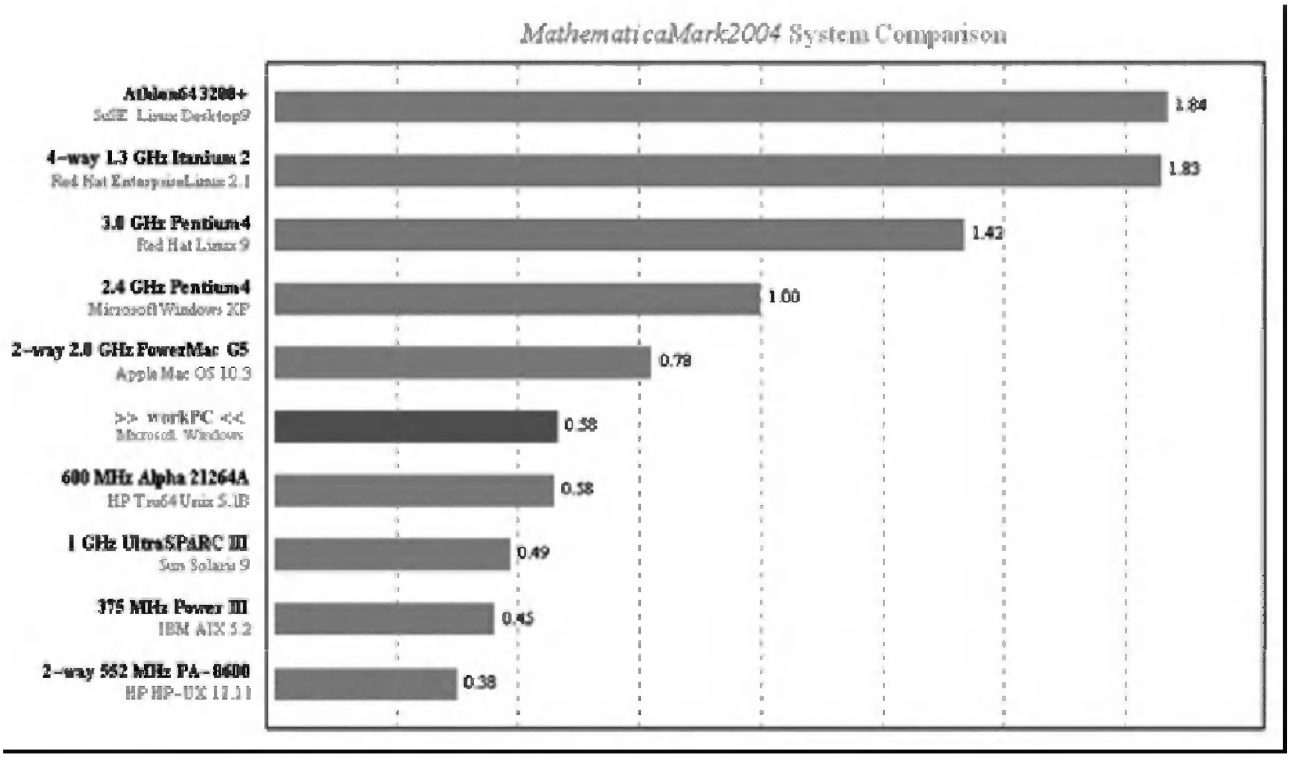


Рис. 11.34. Сравнительные данные о производительности компьютеров разной конфигурации при использовании системы Mathematica 5.1

Помимо описанных, интерес представляют такие средства визуализации, как отображение сложной структуры массивов, поддержка XLS- и AVI-файлов и применение встроенного Web-сервера. Однако эти возможности выходят за рамки основной тематики данной книги. Более подробные данные о средствах визуализации системы Mathematica 5.1 можно найти в справке по этой системе и на интернет-сайте ее разработчика. Там же можно найти сведения о графических возможностях новейшей версии Mathematica 6, которая должна вскоре появиться на нашем рынке.

11.7. Новые средства графики в Mathematica 6

11.7.1. Позиция Graphics меню и графический редактор

При создании сложных ноутбуков в прежних версиях Mathematica явно недоставало средств для подготовки хотя бы простых рисунков и диаграмм, которыми часто сопровождаются математические и научно-технические расчеты. Подобные

рисунки и диаграммы, разумеется, можно создавать средствами программирования систем Mathematica, но это требует много времени и умения хорошо программировать графические задачи.

Учтя это, разработчики Mathematica 6 ввели средство построения с помощью мыши простых рисунков по типу хорошо известного графического редактора Paint. Доступ к нему обеспечен с новой позиции Graphics меню. Она содержит следующие команды:

- **New Graphic** – вывод окна для построения графика;
- **Drawing Tool** – вывод окна графического редактора;
- **Graphics Inspector** – вывод окна инспектора графики;
- **Rendering** – вывод подменю операций рендеринга;
- **Opetations** – вывод подменю дополнительных операций.

Работа с указанными графическими средствами проста и очевидна. Ее иллюстрирует рис. 11.35. На нем показаны окна рисунка, графического редактора и инспектора графики. Отметим, что у графического редактора нет средств для построения незакрашенного эллипса, прямоугольника и полигона. Однако установкой цветов эти фигуры несложно получить.

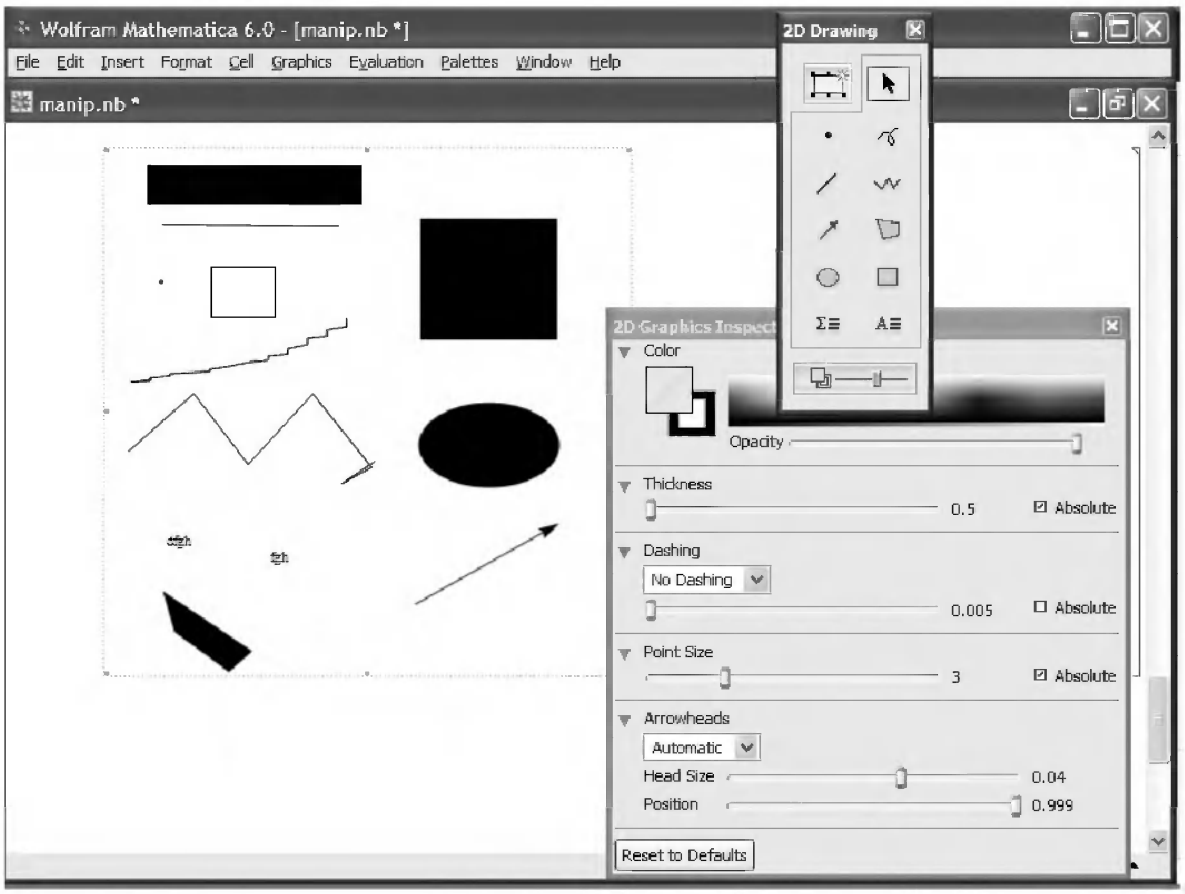


Рис. 11.35. Работа с графическим редактором и инспектором графики в системе Mathematica 6

11.7.2. Расширение возможностей функции Plot

В Mathematica 6 существенно расширены возможности функции построения двумерных графиков **Plot**. Это видно из перечня ее опций, который можно вывести, исполнив команду:

```
Options[Plot]
{AlignmentPoint→Center,AspectRatio→1/GoldenRatio,Axes→True,
AxesLabel→None,AxesOrigin→Automatic,AxesStyle→{},Background→None,
BaselinePosition→Automatic,BaseStyle→{},ClippingStyle→None,
ColorFunction→Automatic,ColorFunctionScaling→True,ColorOutput→Automatic,
ContentSelectable→Automatic,DisplayFunction¦$DisplayFunction,Epilog→{},
Evaluated→Automatic,EvaluationMonitor→None,Exclusions→Automatic,
ExclusionsStyle→None,Filling→None,FillingStyle→Automatic,
FormatType¦TraditionalForm,Frame→False,FrameLabel→None,FrameStyle→{},
FrameTicks→Automatic,FrameTicksStyle→{},GridLines→None,GridLinesStyle→{},
ImageMargins→0.,ImagePadding→All,ImageSize→Automatic,LabelStyle→{},
MaxRecursion→Automatic,Mesh→None,MeshFunctions→{#1&},
MeshShading→None,MeshStyle→Automatic,Method→Automatic,
PerformanceGoal¦$PerformanceGoal,PlotLabel→None,PlotPoints→Automatic,
PlotRange→{Full,Automatic},PlotRangeClipping→True,PlotRangePadding→Automatic,
PlotRegion→Automatic,PlotStyle→Automatic,PreserveImageOptions→Automatic,
Prolog→{},RegionFunction→(True&),RotateLabel→True,Ticks→Automatic,
TicksStyle→{},WorkingPrecision→MachinePrecision}
```

Применение этих опций позволяет легко строить самые разнообразные, а порой просто невероятные по своей выразительности графики. Примеры построения таких графиков можно найти в разделе Options справки по функции Plot, а также в самоучителе Options for Graphics.

11.7.3. Использование опций закраски областей двумерных графиков

Из новых опций функции **Plot** в Mathematica 6 наиболее эффектно выглядят опции закраски областей двумерных графиков **Filling** (Закраска) и **FillingStyle** (Стиль закраска). Первая по умолчанию отключена, вторая имеет значение **Auto** (Автоматический выбор стиля).

Прежде всего продемонстрируем действие опции **Filling** – рис. 11.36. На нем функцией **Plot** строится график функции **Sin[x]/x** в интервале изменения x от -4*Pi до +4*Pi с 4 типами закраски. Они представлены значениями опции **Filling**: **Axis** (окраска идет от каждой точки кривой до оси абсцисс), **Top** (окраска области от вершины окна графика до его кривой), **Bottom** (окраска от кривой до низа окна графика) и 0.5 (окраска от линии графика до горизонтали с вертикальной координатой, равной 0.5).

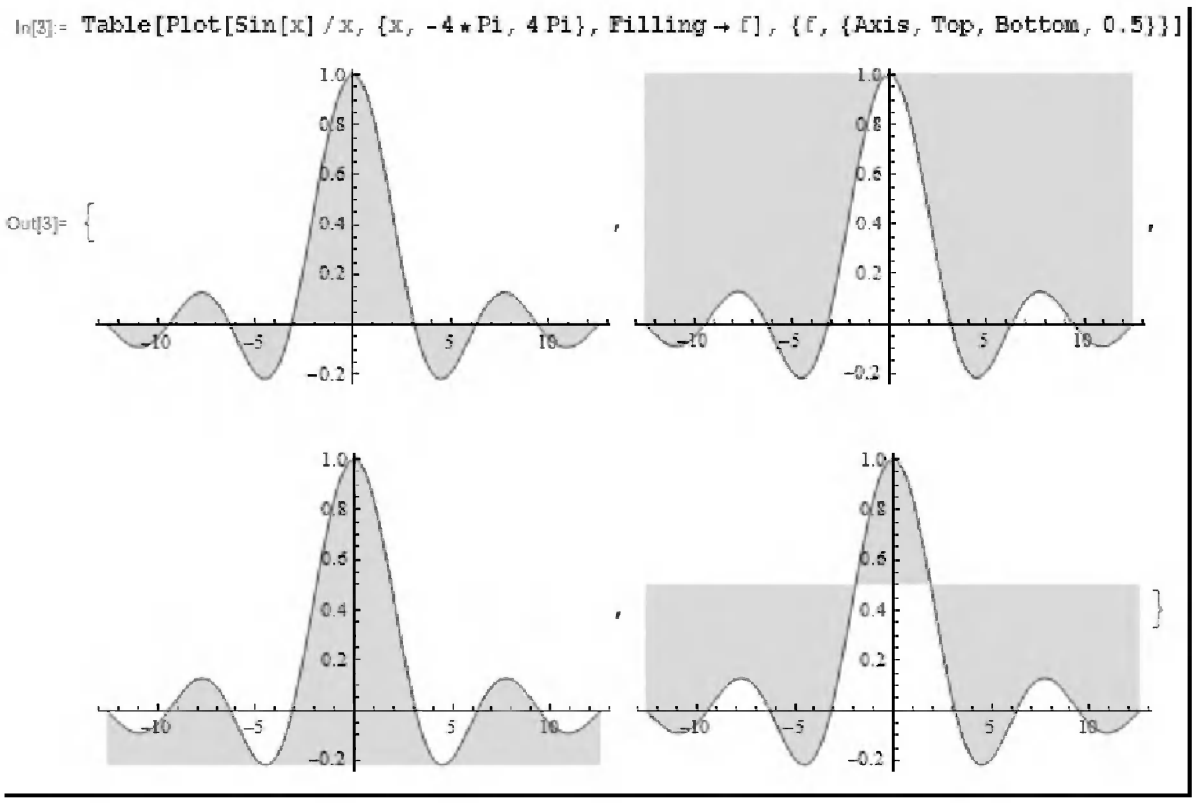


Рис. 11.36. Построение графика sin(x)/x с различными значениями опции окраски

Эта опция может использоваться и с функцией **`ListPlot[list]`**, строящей точки с ординатами, взятыми из списка **`list`**. Она дает возможность построения вертикалей, соединяющих точки графика с осью абсцисс, – рис. 11.37. На этом рисунке показано, как меняется значение простых чисел от их номера. Любопытно, что эта зависимость близка к линейной.

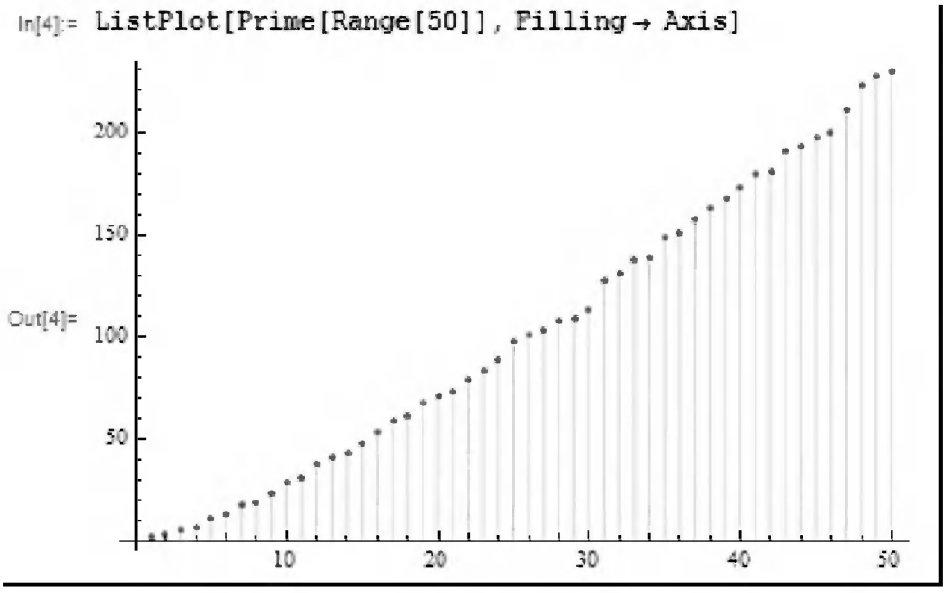


Рис. 11.37. Зависимость значений первых 50 простых чисел от их номера

Подобный вид графиков используется для представления амплитуды гармоник спектров при спектральном Фурье-анализе. Рисунок 11.38 дает пример такого рода. Здесь строится график синусоидального сигнала с интерполяцией между его узловыми точкам. Сигнал выглядит как построенный сплошной линией, но на самом деле он задан как дискретный сигнал. После этого строится график спектра, причем знак гармоник, задающий их фазу, не учитывается. Спектр сигнала симметричный и представлен вертикальными отрезками прямых с точкой над каждым отрезком.

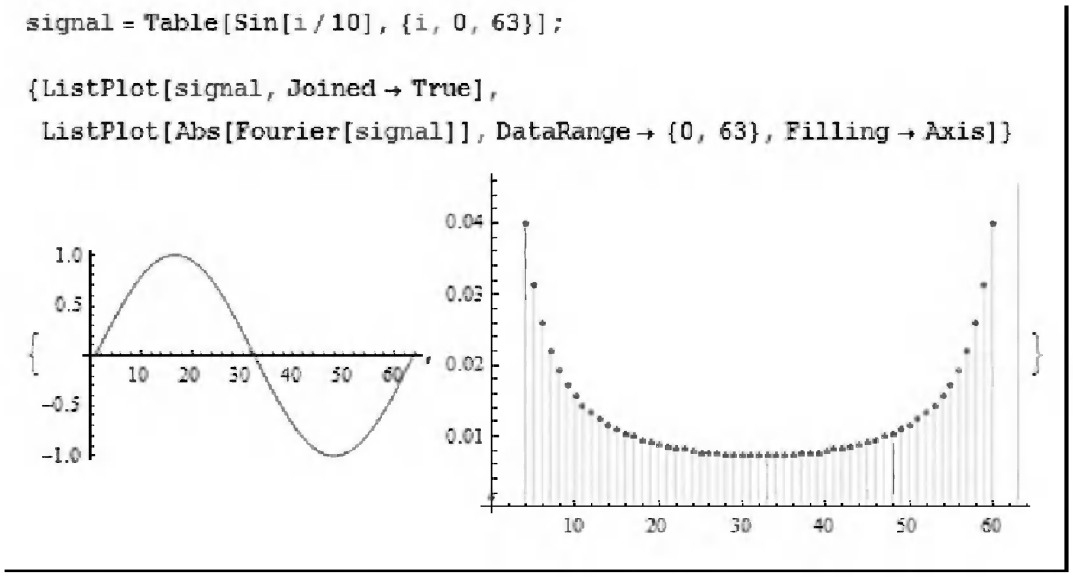


Рис. 11.38. Построение графика синусоидального сигнала и его дискретного прямого преобразования Фурье

Рисунок 11.39 демонстрирует закраску областей между двумя кривыми – синусоидой и косинусоидой. Разумеется, что такая окраска возможна для кривых как периодических, так и непериодических функций.

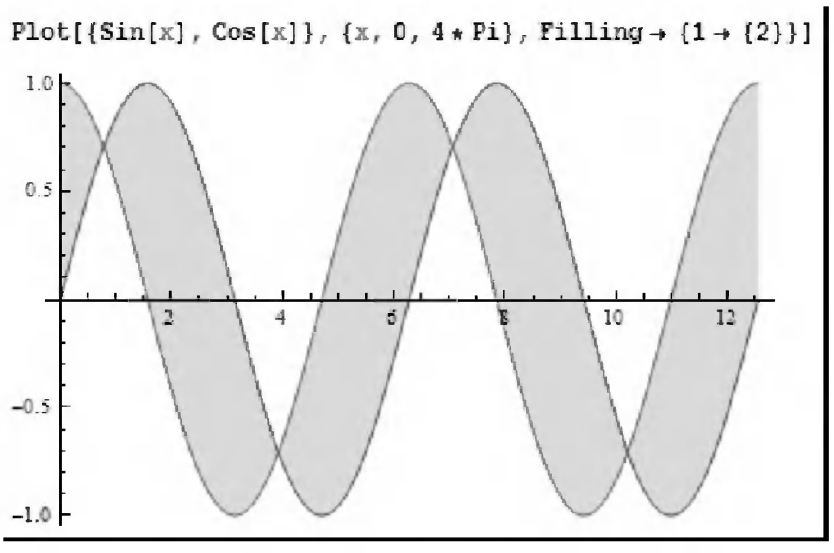


Рис. 11.39. Пример закраски областей между двумя кривыми

Приведенные примеры составляют лишь малую часть примеров применения опций закраски. В справке можно найти многие десятки примеров на построение рисунков с использованием различных видов закраски, в том числе различных областей разными цветами, разными стилями закраски и т. д.

11.7.4. Графические динамические модули в Mathematica 6

Возможность в динамике (в режиме реального времени) изменять те или иные параметры графических объектов и тут же наблюдать вызванные этим их изменения – еще одна замечательная особенность системы Mathematica 6. Она реализуется с помощью функций-модулей **DynamicModule** и **Manipulate**. Ниже представлены характерные примеры их применения в графике.

На рис. 11.40 показан модуль на основе функции **Manipulate**, который реализует основные методы закраски графика функции одной переменной. Функция является суммой двух гармонических колебаний с кратными частотами. Кратность может меняться с помощью двух слайдеров, при этом можно наблюдать множество кривых. Опция **ControlType→SetterBar** задает меню-бар, позволяющее задать и наблюдать 10 вариантов закраски различных областей графика.

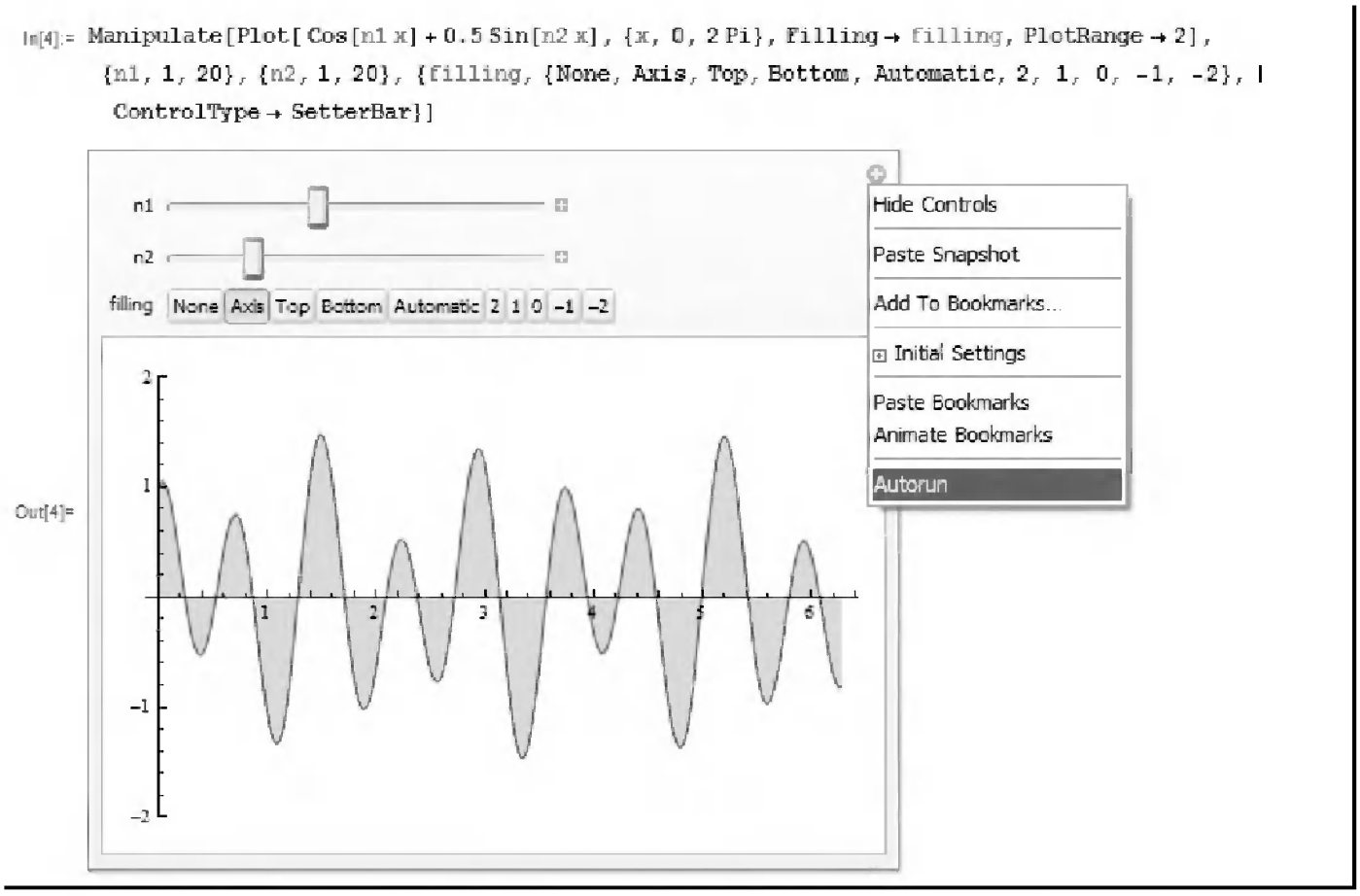


Рис. 11.40. Модуль, иллюстрирующий построение графика суммы двух гармонических колебаний с изменяемой кратностью частот и различные вида закраски графика с помощью меню

Рисунок 11.41 прекрасно иллюстрирует идею получения любого цвета смешением трех базовых цветов: red – красного, green – зеленого и blue – синего. Этот простой модуль строит круг, закрашенный смесью указанных цветов, причем интенсивность каждого цвета задается одним из трех слайдеров. Данный модуль хорошо иллюстрирует суть *RGB-метода*. К сожалению, в этой книге цвета на рисунках (в том числе рис. 11.41) не воспроизводятся и вместо них наблюдаются лишь оттенки серого цвета (тип окраски grayscale).

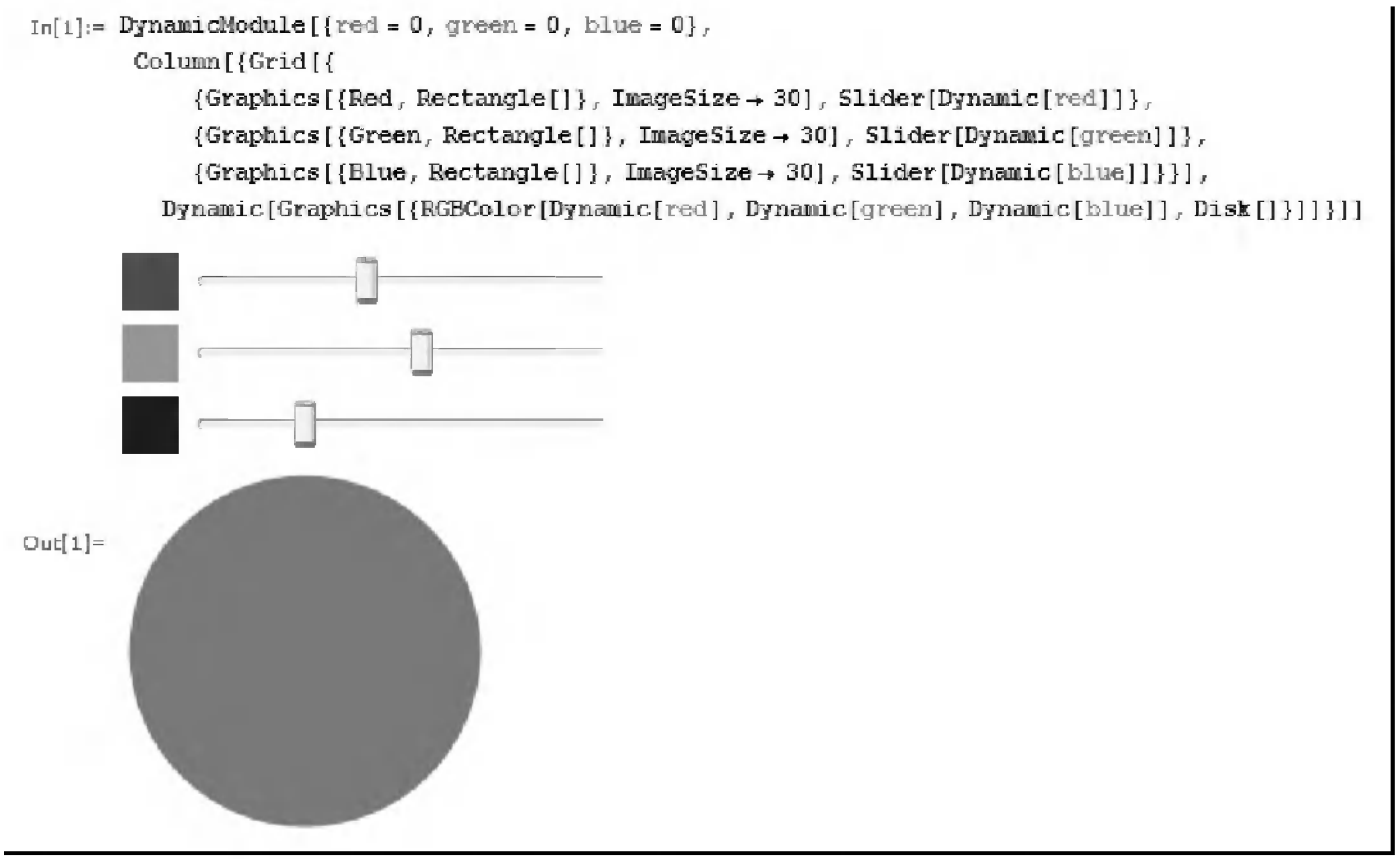


Рис. 11.41. Демонстрация сложения трех цветов: красного, зеленого и синего

Данный модуль построен на основе динамического модуля **DynamicModule**, наполнение которого довольно очевидно и может быть взято пользователем за основу разработки своих подобных модулей.

Следующий модуль, показанный на рис. 11.42, является прекрасной иллюстрацией полиномиальной аппроксимации. Он с помощью локаторов задает 4 точки и осуществляет их классическую аппроксимацию полиномом третьей степени. Сплошной линией строится график полинома, который всегда точно проходит через точки – локаторы. Замечательно тут то, что мышью можно смещать точки с их начального состояния, заданного в модуле, и тут же наблюдать изменение графика полинома. Нетрудно убедиться в том, что точное прохождение графика полинома через узловые точки отнюдь не гарантирует его близость к ним в промежутках между ними. В частности, в этих промежутках возможен значительный выбег графика полинома.

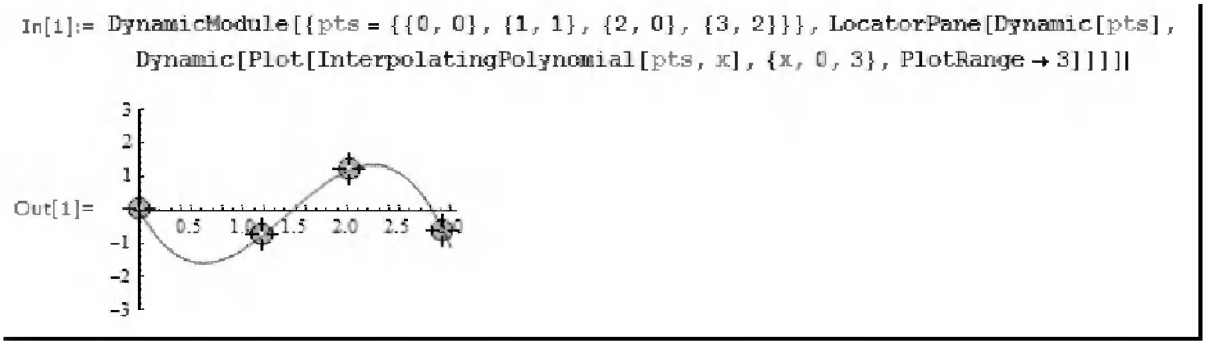


Рис. 11.42. Иллюстрация полиномиальной аппроксимации для четырех точек

Динамический модуль, показанный на рис. 11.43, иллюстрирует построение графика в полярной системе координат с помощью задатчика угла, заданного функцией **`angularSlider`**. Задатчик угла выглядит как окружность, внутри которой размещен радиус-вектор, угловое положение которого задает изменяемый угол. Изменение положения радиус-вектора осуществляется мышью.

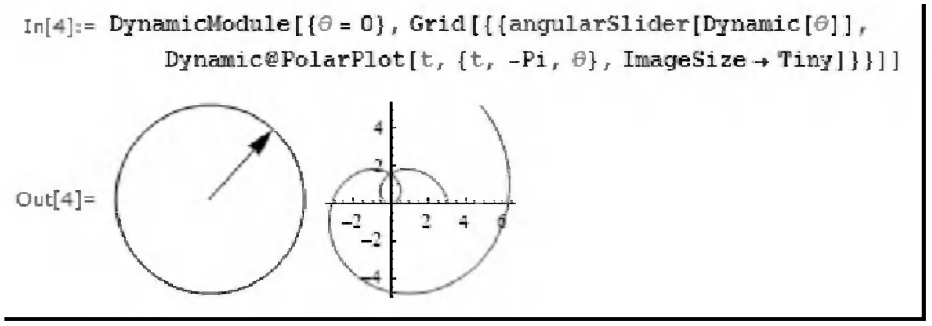


Рис. 11.43. Пример, иллюстрирующий построение графика функции в полярной системе координат при изменении угла поворота радиус-вектора

Наконец, на рис. 11.44 показан модуль, строящий объемную фигуру, которую можно вращать в пространстве с помощью двухкоординатного слайдера. При этом двухкоординатный слайдер изменяет два угла поворота фигуры.

11.7.5. Визуализация данных из списков

В Mathematica 6 существенно расширены и обновлены графические функции для визуализации данных, хранящихся в списках. Так, есть следующие модифицированные функции для построения точек в декартовой системе координат:

- **`ListPlot[{`*y*$_1$**`,`***y*$_2$**`,j}]`** – построение точек списка, заданных ординатами;
- **`ListPlot[{{`*x*$_1$**`,`***y*$_1$**`},{`*x*$_2$**`,`***y*$_2$**`},j}]`** – построение точек списка, заданных координатами;
- **`ListPlot[{`*list*$_1$**`,`***list*$_2$**`,j}]`** – построение нескольких групп точек, заданных списками.

```
In[11]:= DynamicModule[{p = {2 π, 0}},
          {Slider2D[Dynamic[p], {{2 Pi, 0}, {0, Pi}}],
           ParametricPlot3D[{Cos[u] Cos[v], Sin[u] Cos[v], Sin[v]}^3, {u, 0, 2 Pi}, {v, -Pi / 2, Pi / 2},
            Mesh → None, PlotRange → All, Ticks → None, SphericalRegion → True,
            ViewPoint → Dynamic[4 {Cos[p[[1]]] Sin[p[[2]]], Sin[p[[1]]] Sin[p[[2]]], Cos[p[[2]]]}]]]
```

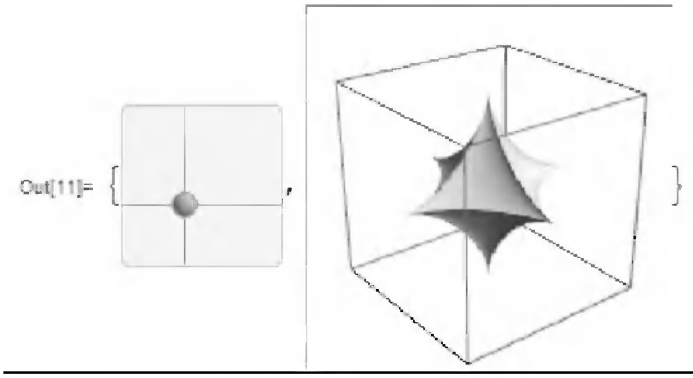
Out[11]= 

Рис. 11.44. Пример, иллюстрирующий вращение трехмерной фигуры в пространстве с помощью двухкоординатного слайдера

Аналогичная по синтаксису записи новая функция **ListLinePlot** строит графики, соединяя точки отрезками линий. На рис. 11.45 показано построение с помощью данной функции звезды.

```
ListLinePlot[Table[{Sin[k 2 Pi / 5], Cos[k 2 Pi / 5]}, {k, 0, 15, 3}], Frame → True,
 Axes → False, Filling → Axis]
```

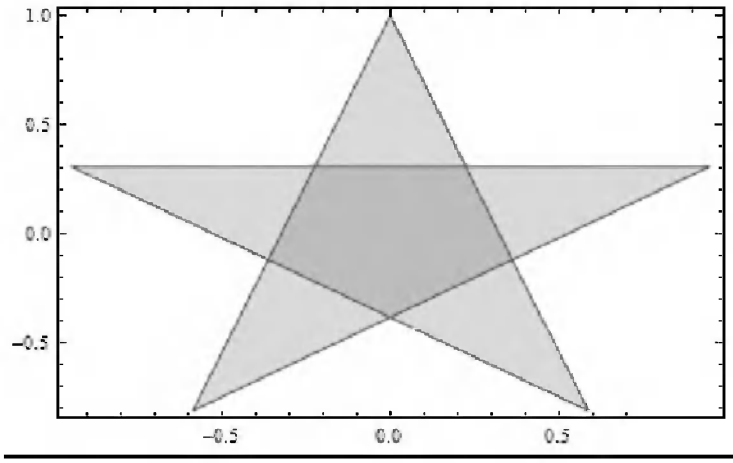


Рис. 11.45. Построение звезды

Еще один пример, показанный на рис. 11.46, показывает график роста 500 случайных чисел благодаря аккумулированию их значений, которые лежат в интервале от –1 до +1. Кривая роста носит случайный характер и меняется при каждом пуске заданной графической функции. Кроме того, тут задается случайный цвет закраски.

В справке можно найти десятки примеров на применение этих функций с разными опциями. Для визуализации функций двух переменных (поверхностей и

```
ListLinePlot[Accumulate[RandomReal[{-1, 1}, 500]],
 ColorFunction → "Rainbow", Filling → 0]
```

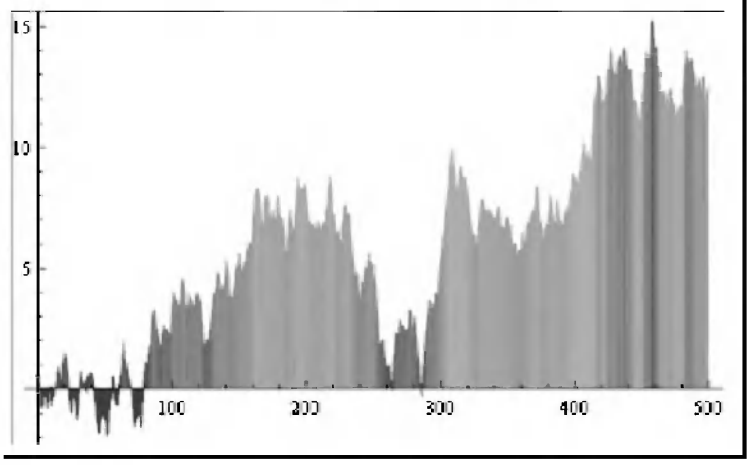


Рис. 11.46. График роста аккумулированного значения 500 случайных чисел со случайным цветом закраски

трехмерных фигур) в виде контурных графиков служат следующие графические функции:

`ListContourPlot[`*array*`]`
`ListContourPlot[{{`x_1, y_1, f_1`}, {`x_2, y_2, f_2`}, ...}]`
`ListContourPlot3D[`*array*`]`
`ListContourPlot3D[{{`x_1, y_1, z_1, f_1`}, {`x_2, y_2, z_2, f_2`}, ...}]`

Ограничимся примером применения функции ListContourPlot для построения контурного графика среза головы человека, данные которого импортируются из файла, – см. рис. 11.47.

```
In[1]:= ListContourPlot[Reverse@Import["ExampleData/head.dcm.gz", "Data"][[1]],
         ColorFunction → "ModifiedSpectral", MaxPlotPoints → 100]
```

Out[1]=

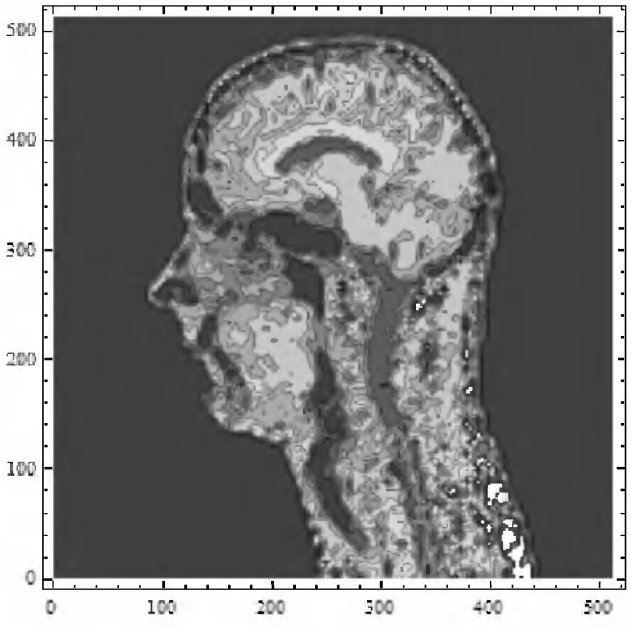


Рис. 11.47. Пример построения контурного графика для среза головы человека

Для построение трехмерных графиков по данным точек служат функции:

```
ListPlot3D[array]
ListPlot3D[{{x1,y1,z1},{x2,y2,z2},...}]
ListPlot3D[{data1,data2,...}]
ListSurfacePlot3D[{{x1,y1,z1},{x2,y2,z2},...}]
RegionPlot3D[pred,{x,xmin,xmax},{y,ymin,ymax},{z,zmin,zmax}]
```

Рисунок 11.48 дает пример построения поверхности по точкам, заданным таблицей для функции $\cos(j^2+i)$. Верхний рисунок получен при отсутствии интерполяции (порядок интерполяции 0), а второй – при применении квадратичной интерполяции (порядок интерполяции 2). Нетрудно заметить, что применение интерполяции позволяет получить весьма реалистическое изображение поверхности. Однако представление поверхности без интерполяции нередко преследует достижение специальных целей – например, наглядного сравнения представлений поверхности в отсутствие и при наличии интерполяции.

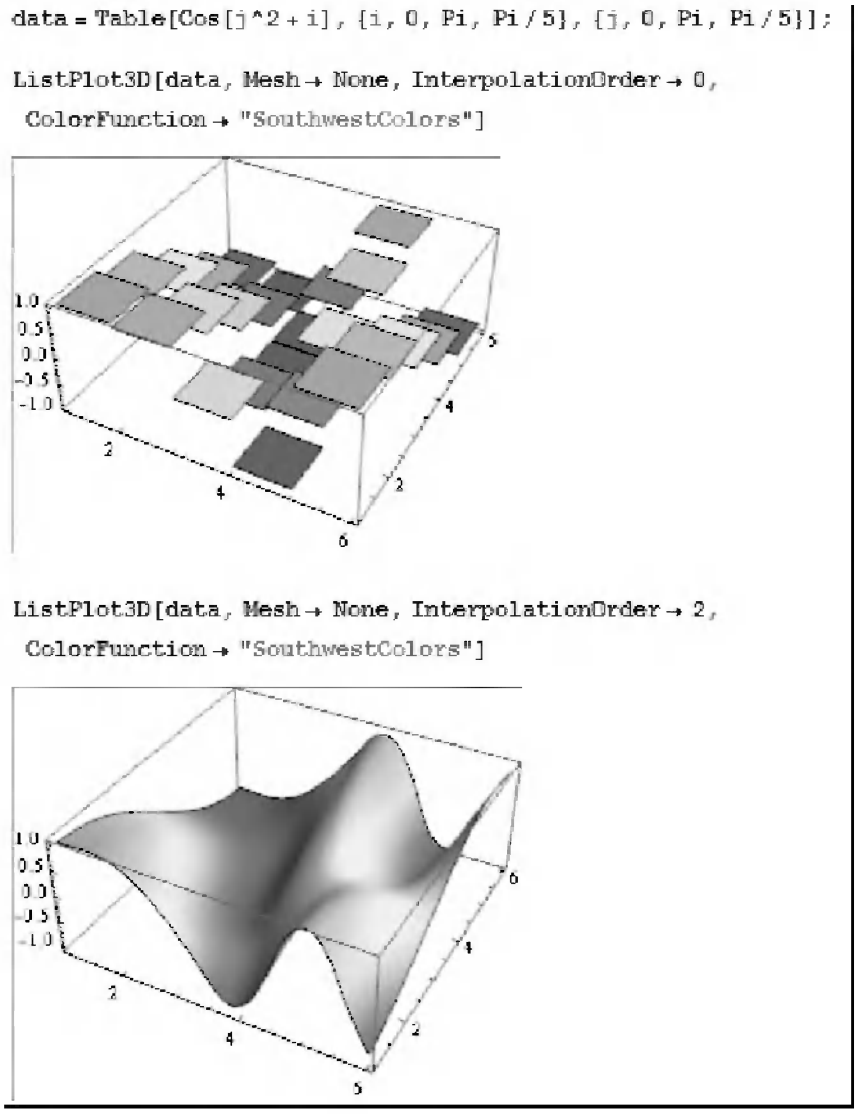


Рис. 11.48. Построение поверхности по точкам без интерполяции (верхний рисунок) и с квадратичной интерполяцией (нижний рисунок)

11.7.6. Рельефная графика

Новая функция системы Mathematica 6 **ReliefPlot [*array*]** служит для построения реалистических графиков рельефа, который задается ординатами точек массива. Примером применения этой функции служит рис. 11.49, на котором построен рельеф поверхности, заданной математической формулой i+cos(i^3+j^3), где i и j меняются с дискретностью 0,03 в интервале от –4 до 4. Вид рельефа зависит от значения опции **ColorFunction**.

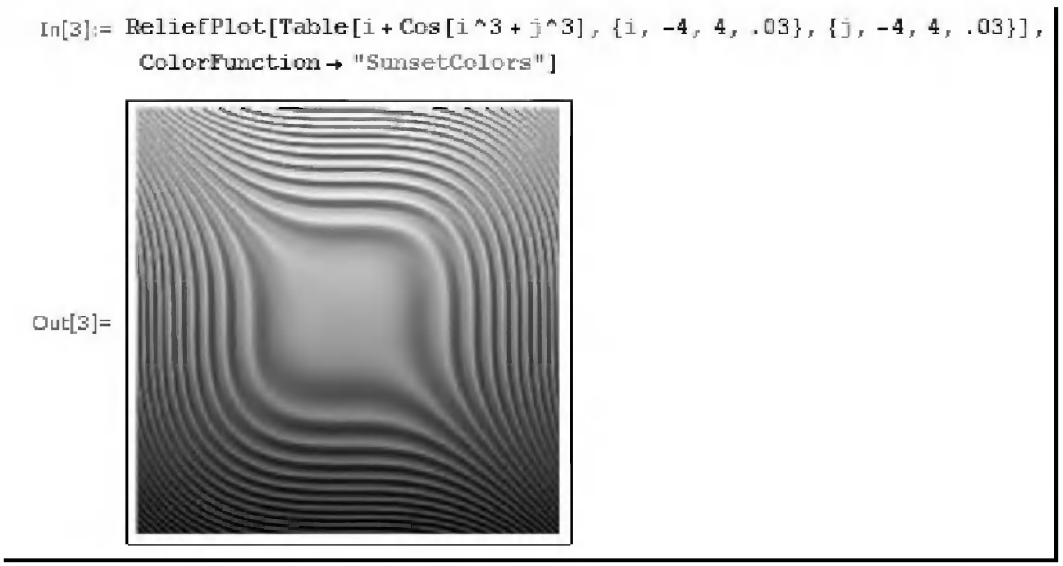


Рис. 11.49. Построение графика рельефа поверхности, заданной формулой i+cos(i^3+j^3)

Еще один пример применения функции **ReliefPlot** представлен на рис. 11.50. Здесь задаются три массива случайных результатов ряда арифметических операций, включающих нормы матриц. Полученные три рельефа являются в значительной мере случайными и меняются от пуска к пуску представленного модуля.

Рисунок 11.51 строит рельеф мнимой части функции sec(i+I*j)^2 для двух значений опции **PlotRange**, равных **All** и **Automatic**. Нетрудно заметить серьезное изменение характера выявления деталей одного и того же рельефа.

Разумеется, массив для этой функции может создаваться не только математическими выражениями, но и любыми другими способами – например, загрузкой массивов рисунков. Множество опций функции **ReliefPlot** позволяет создавать рельефы с разным разрешением, разной окраской и другими особенностями.

11.7.7. Трехмерные объекты, полученные вращением кривых

Довольно широко распространенными являются трехмерные графические объекты, полученные вращением кривых относительно некоторой оси. Например, поворачивая окружность на угол p, можно получить поверхность шара. Меняя преде-

```
Table[
 ReliefPlot[
  Table[
   Evaluate[Sum[Product[Norm[{x, y} - RandomReal[{-3, 3}, {2}]], {i, j}] /
      Product[Norm[{x, y} - RandomReal[{-3, 3}, {2}]], {i, j}], {j, 1, 10}]],
   {x, -5, 5, .05}, {y, -5, 5, .05}], PlotRange → Automatic, ColorFunction → "Rainbow",
  ClippingStyle → Darker[Red]], {3}]
```

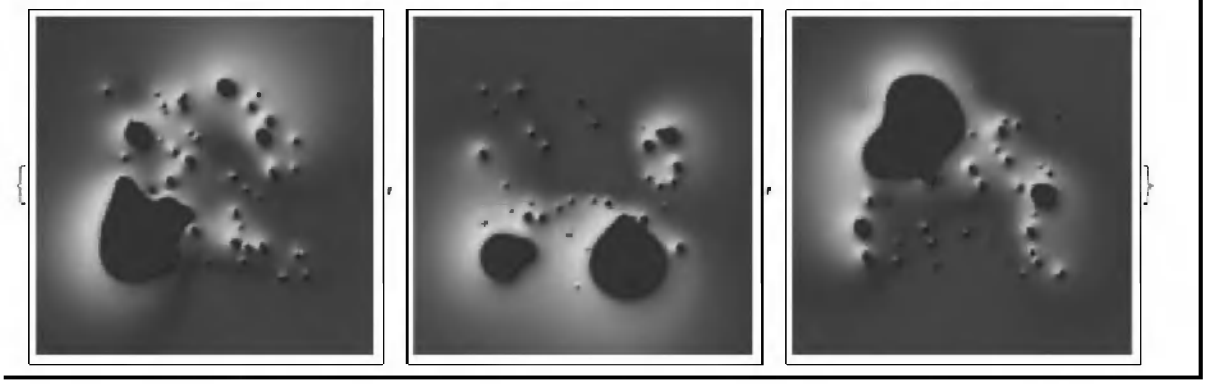


Рис. 11.50. Создание трех случайных рельефов

```
In[7]:= {ReliefPlot[Table[Im[Sec[(i + I j)^2]], {i, -3, 3, .01}, {j, -3, 3, .01}], PlotRange → All],
         ReliefPlot[
          Table[Im[Sec[(i + I j)^2]], {i, -3, 3, .01}, {j, -3, 3, .01}], PlotRange → Automatic]}
```

Out[7]=

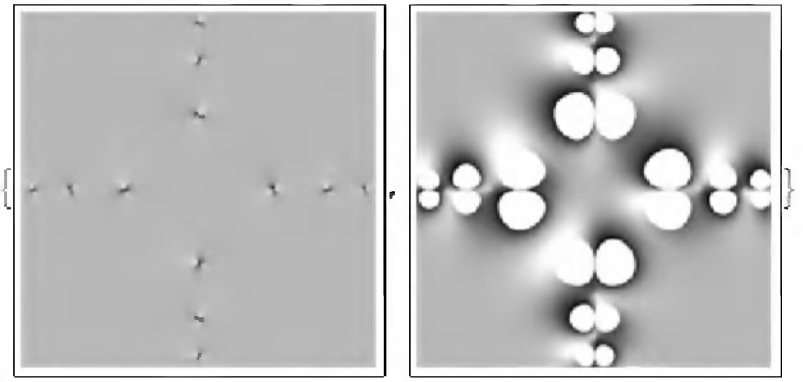


*Рис. 11.51. Рельеф мнимой части функции sec(i+I*j)²*
для двух значений опции PlotRange, равных All и Automatic

лы изменения угла поворота, можно строить замкнутые или незамкнутые фигуры. Для построения таких поверхностей (фигур) в Mathematica 6 служит функция:

RevolutionPlot3D[f_z,{t,t_{min},t_{max}},...]
RevolutionPlot3D[{f_x,f_y,f_z},{t,t_{min},t_{max}},...]

На рис. 11.52 показано применение этой функции для построения объемной фигуры. Здесь параметрами являются два изменяющихся угла, и для вращения используется параметрическая кривая.

Пример применения опций функции **RevolutionPlot3D** представлен на рис. 11.53. Здесь кривая вращения задается с помощью шести неравенств, с уче-

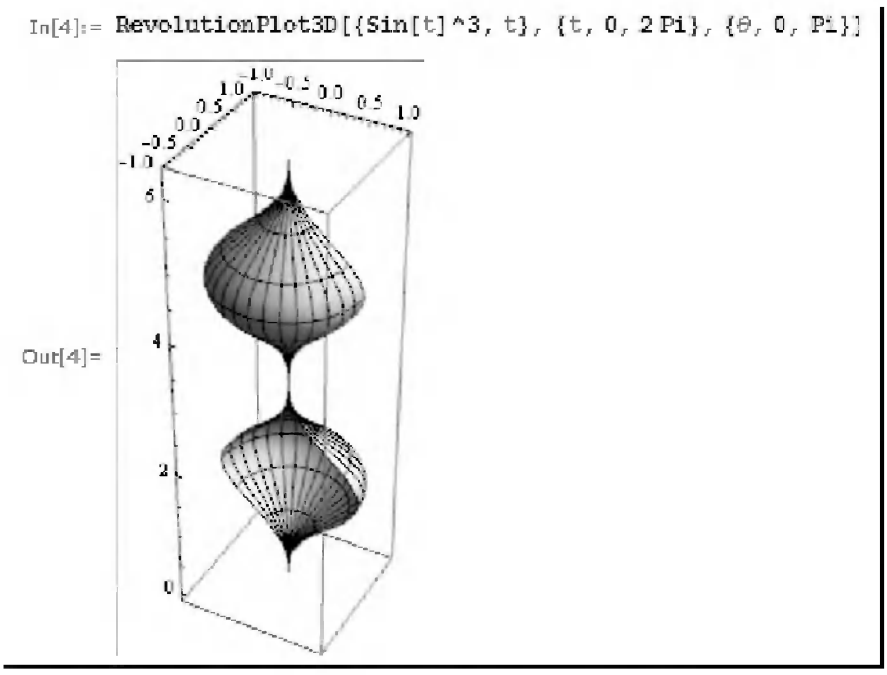


Рис. 11.52. Построение трехмерной фигуры

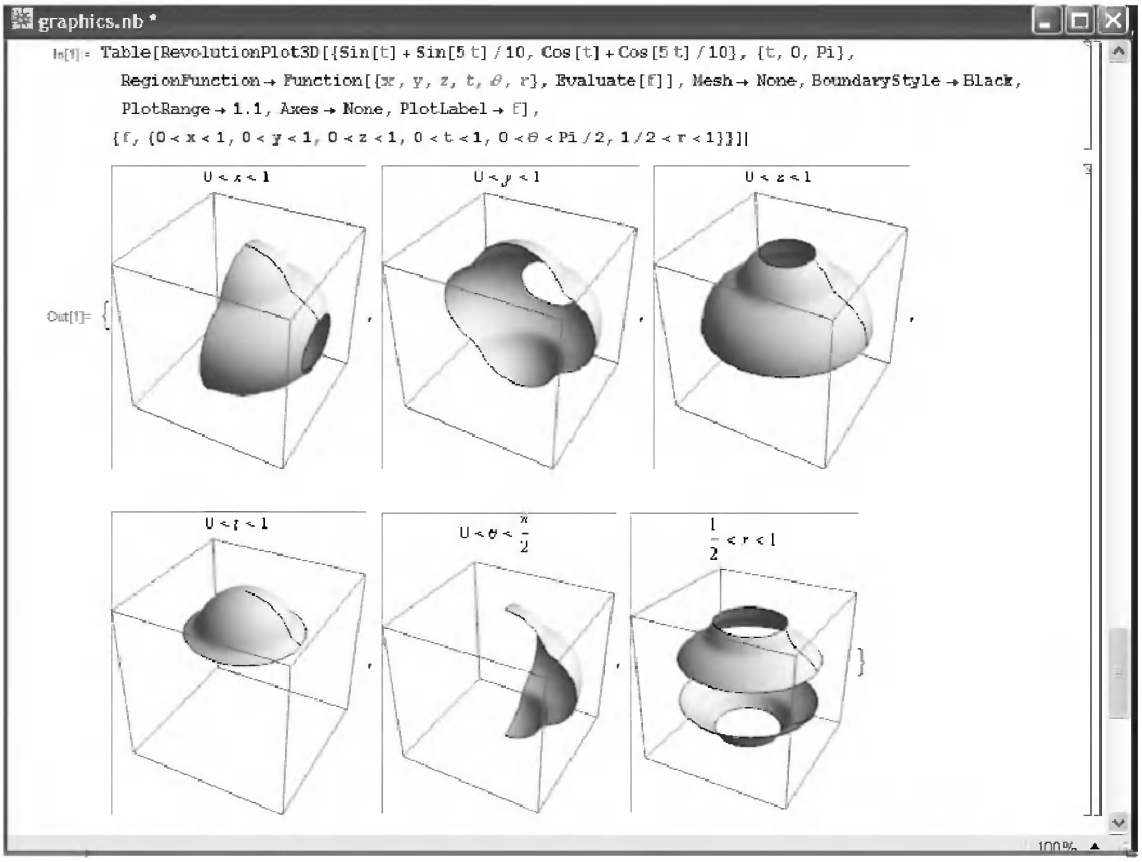


Рис. 11.53. Построение шести вариантов пространственных фигур с применением опций

том которых строятся шесть фигур. К сожалению, как и ранее, цветовая окраска фигур воспроизводится лишь оттенками серого цвета.

Как и другие графические функции, функция **`RevolutionPlot3D`** может строить большое число интересных фигур.

11.7.8. Визуализация работы клеточных автоматов

Комбинируя по определенным алгоритмам на клеточной доске темные и светлые квадраты, можно получить порой очень неожиданные фигуры, напоминающие очереди, фракталы и иные графические объекты с весьма неожиданными математическими и художественными свойствами. Одним из таких свойств является самоподобие фигур и возможность их бесконечного дробления.

Mathematica имеет функцию реализации клеточных автоматов **CellularAutomaton**, общая форма записи которой следующая:

```
CellularAutomaton[rule,init,{t,All,…}]
```

Возможны и упрощенные формы записи. Функция задается спецификацией rule и начальными условиями init. Алгоритм работы функции описан в разделе MORE INFORMATION справки по этой функции. Приведем простой пример ее работы (rule=30, заданы два шага):

```
CellularAutomaton[30,{0,0,0,1,0,0,0},2]
{{0,0,0,1,0,0,0},{0,0,1,1,1,0,0},{0,1,1,0,0,1,0}}
```

О том, насколько разнообразные фигуры может создавать клеточный автомат при небольшом изменении параметров функции **CellularAutomaton**, свидетельствует рис. 11.54 с множеством примеров применения данной функции.

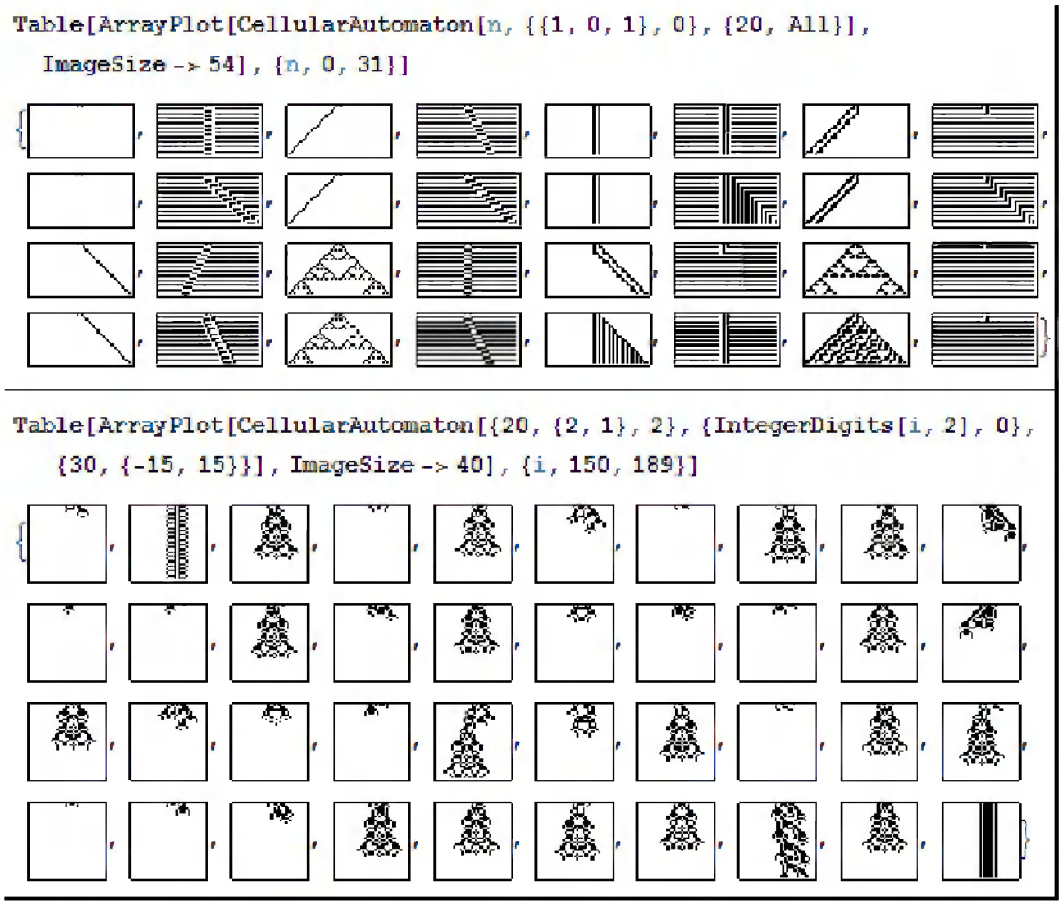


Рис. 11.54. Примеры построения функцией ***CellularAutomaton*** *множества фигур на плоскости*

Еще один рисунок – рис. 11.55 – наглядно показывает, что графика клеточных автоматов не лишена художественной ценности. Некоторые рисунки напоминают художественный орнамент витражей и мозаик.

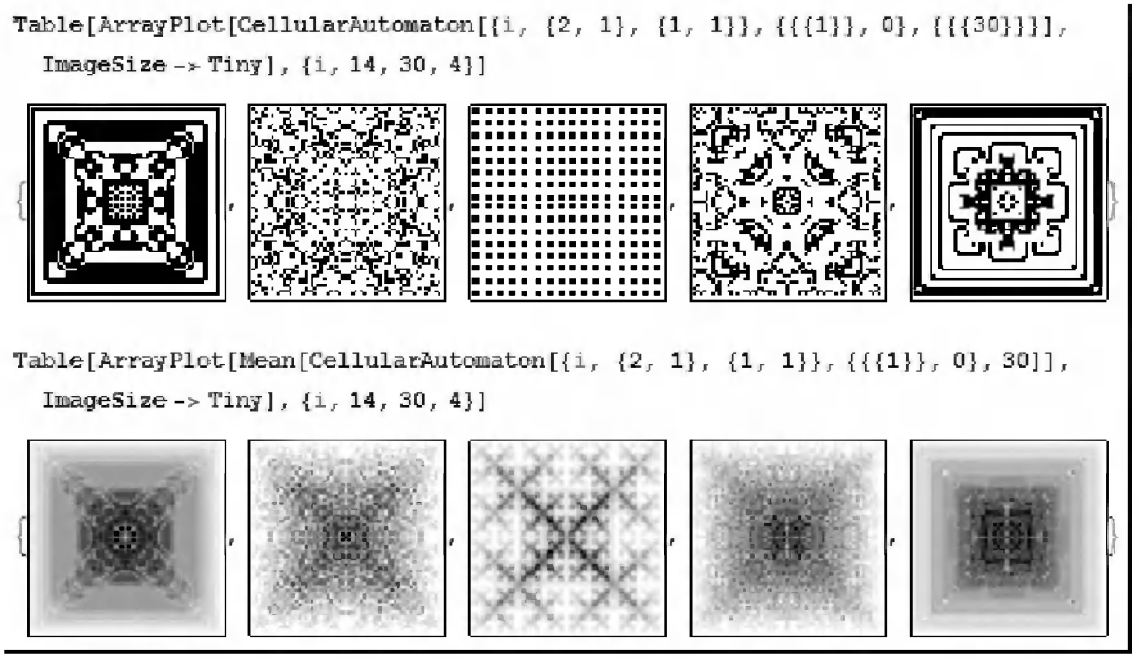


Рис. 11.55. Дополнительные примеры «творчества» клеточных автоматов

Функция **CellularAutomaton** позволяет получать и объемные фигуры. Рисунок 11.56 демонстрирует построение кубической фигуры с помощью функции **Graphics3D** с примитивом **Cuboid**, внутри которой размещены трехмерные фигуры, построенные с помощью функции **CellularAutomaton**. Рисунок 11.56 дает прекрасное представление о комбинированных графиках, которые способна строить система Mathematica 6.

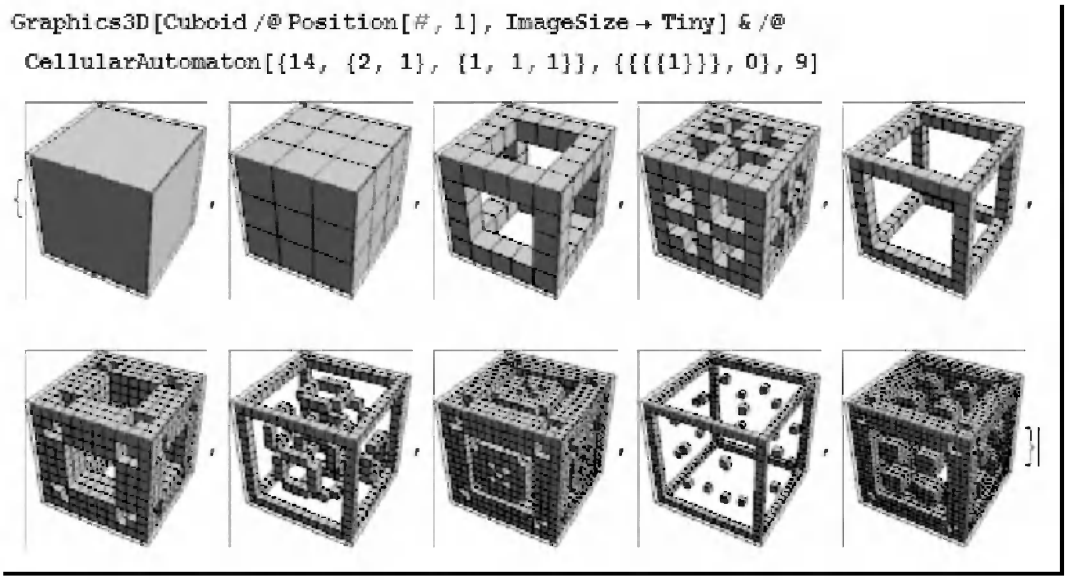


Рис. 11.56. Комбинированные графики – кубы, заполненные фигурами, которые строит клеточный автомат

11.7.9. Графы, деревья и прочее

Графы и цепи являются объектами, хорошо известными в математике и в научно-технических расчетах. Их полноценная поддержка также обеспечена в системе Mathematica. На рис. 11.57 приведены примеры применения функций **GraphPlot** графической поддержки построения двумерных графов и **GraphPlot3D** для поддержки трехмерных графов. С помощью опций этих функций можно обеспечить построение практически любых графов.

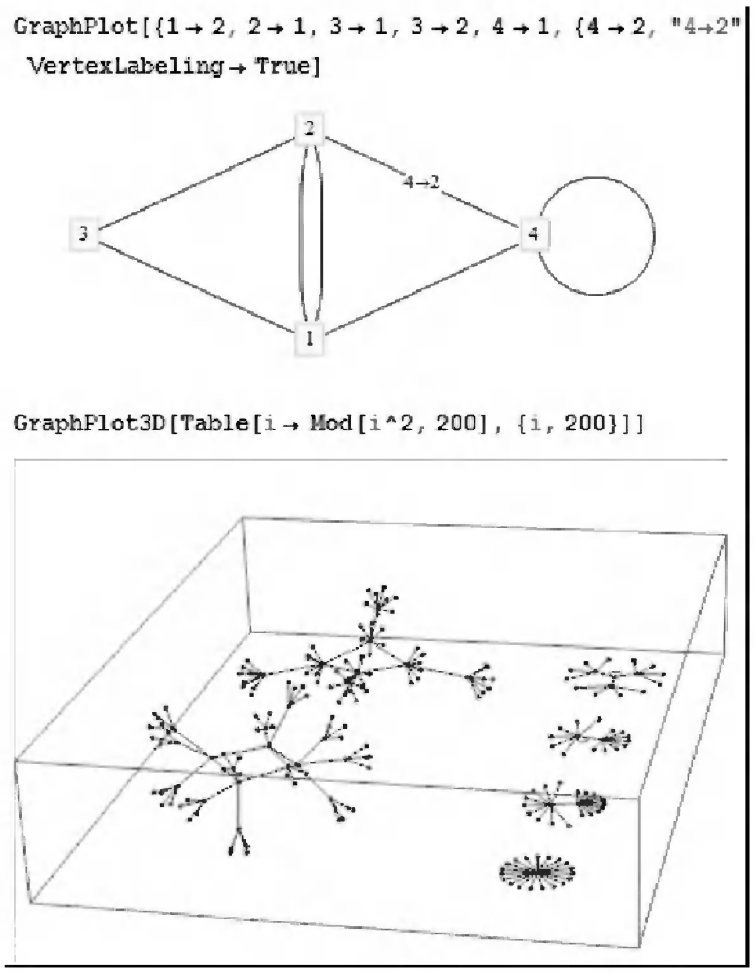


Рис. 11.57. Примеры построения двумерных и трехмерных графов

Для построения множества видов деревьев служит функция **TreePlot**. Примеры ее применения для построения обычных деревьев показаны на рис. 11.58.

Из деревьев могут быть построены довольно занятные и сложные фигуры. Примером этого служат фигуры, показанные на рис. 11.59 и построенные с помощью функции **TreePlot**. В справке по этой функции можно найти много и других примеров ее применения.

```
Table[TreePlot[{1 → 4, 1 → 6, 1 → 8, 2 → 6, 3 → 8, 4 → 5, 7 → 8},
  Automatic, root, VertexLabeling → True], {root, {1, 5}}]
```

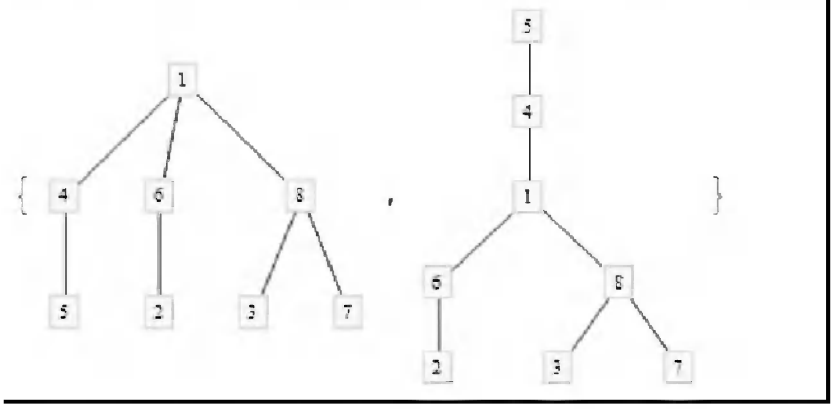


Рис. 11.58. Примеры построения простых деревьев

```
Table[TreePlot[Flatten[Table[{i → 2*i+j-1}, {j, 2}, {i, 2^7-1}]], Center,
  LayerSizeFunction → (RandomReal[{-4, 4}] &)], {6}]
```

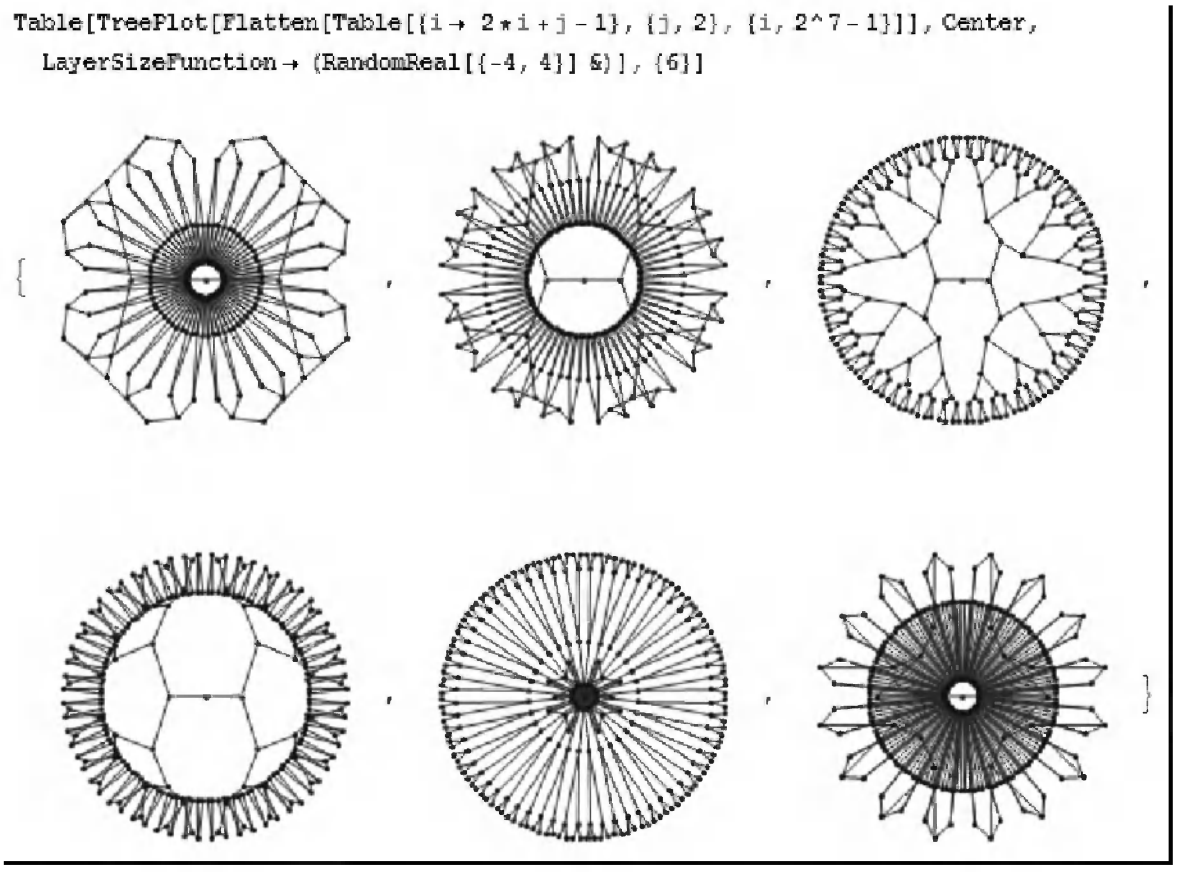


Рис. 11.59. Сложные фигуры, построенные с применением функции ThreePlot

11.7.10. Программирование анимации

В Mathematica 4/5 средства анимации имеются в пакете расширения Graphics. В Mathematica 6 эти средства, созданные на основе графического интерфейса пользователя GUI, включены в ядро и не требуют загрузки никаких пакетов расширения. Они представлены двумя функциями – модулями. Одна из них – это функция Animate, которая имеет следующий синтаксис:

- **Animate[*expr*, {*u*, u_{min}, u_{max}}]** – анимация выражения expr при изменении u в заданных пределах с шагом +1;
- **Animate[*expr*, {*u*, u_{min}, u_{max}, *du*}]** - анимация выражения expr при изменении u в заданных пределах с шагом du;
- **Animate[*expr*, {*u*, {u_1, u_2, ...}}]** – анимация выражения expr при значениях u, выбираемых из списка;
- **Animate[*expr*, {*u*, ...}, {*v*, ...}, ...]** – анимация выражения expr при изменении u, v..., выбираемых из списков.

Анимация с помощью этой функции заключается в выводе ряда кадров, именуемых фреймами и отличающихся параметром управляющей переменной u. Для этого она должна быть введена в состав выражения expr. Функция **Animate** имеет около полутора десятков опций, с помощью которых можно задавать характер и направление анимации. С ними можно познакомиться по примерам применения функции анимации. Для проигрывания анимации служит универсальный проигрыватель, работа с которым очевидна и уже описывалась.

На рис. 11.60 показан пример анимации разложения экспоненты в ряд Тейлора. Выход анимации представлен формулами, изменяющимися в зависимости от изменения управляющей переменной анимации n от 1 до 10 с шагом 1.

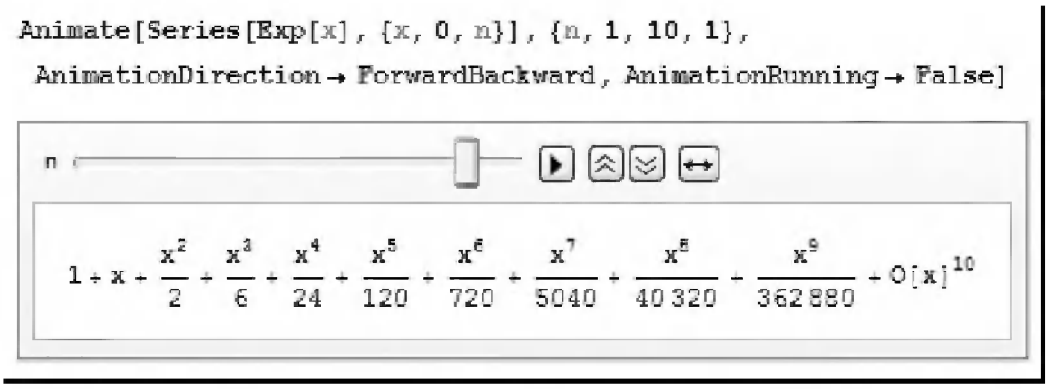


Рис. 11.60. Пример анимации разложения экспоненты в ряд

Следующий пример (рис. 11.61) демонстрирует эффект перемещения по горизонтали кривой графика функции Cos[x+a]^3. Смещение достигается добавлением к аргументу x функции, меняющейся от 0 до 5, управляющей переменной анимации a.

Пример более сложной анимации показан на рис. 11.62. Здесь задается перемещение двух объектов – окружность как бы катится по горизонтальной поверхности, а на ней установлена точка.

Последний пример (рис. 11.63) иллюстрирует трансформацию шара в тор. Вначале заданы функции построения шара sphere и тора torus. Строится (1-λ) sphere + λ torus при изменении λ от 0 до 1. В результате шар постепенно превращается в тор – рис. 11.64.

```
Animate[Plot[Cos[x + a]^3, {x, 0, 10}], {a, 0, 5}]
```

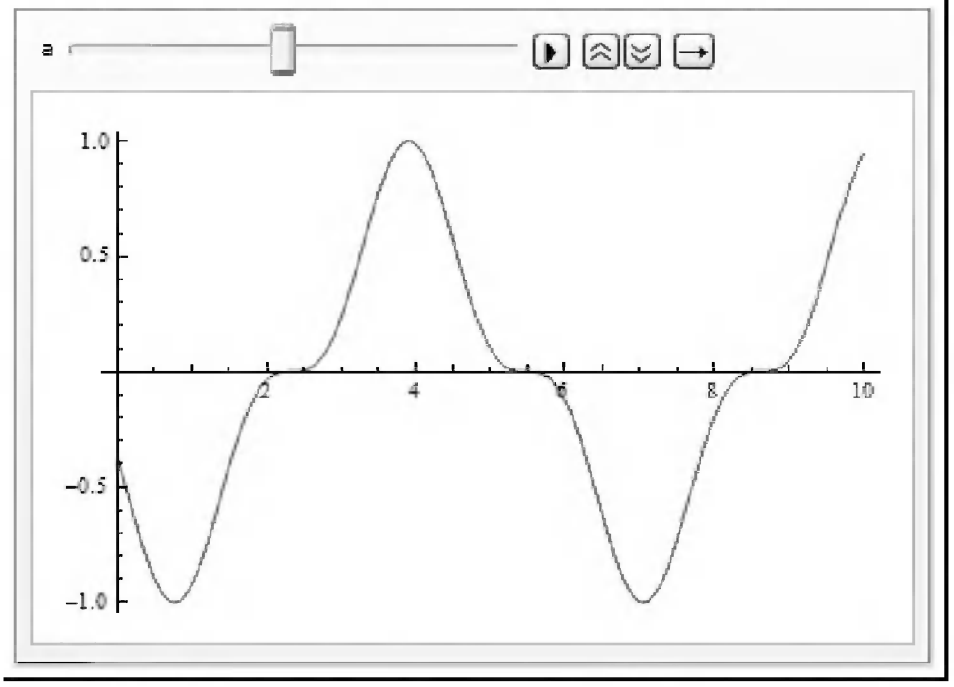


Рис. 11.61. Пример перемещения кривой графика по горизонтали

```
Animate[
 Graphics[Translate[Rotate[{Circle[], Point[{0, 1.1}]}, -t], {t, 0}],
  PlotRange → {{-2, 10}, {-2, 2}}, ImageSize → {Large, Tiny}],
 {t, 0, 2 π}, AnimationRunning → False]
```

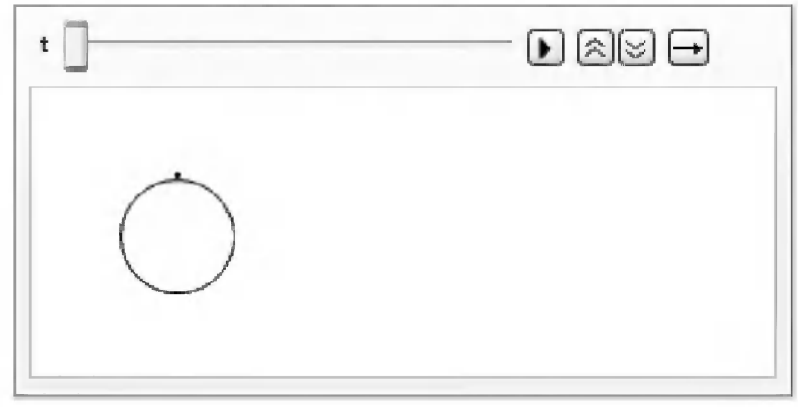


Рис. 11.62. Качение окружности с точкой по горизонтальной поверхности

Еще одна функция анимации имеет две формы записи:

- **`ListAnimate[{`*expr*$_1$**`,`***expr*$_2$**`,...}]`** – анимация путем выбора выражений из списка выражений;
- **`ListAnimate[`*list*`,`*fps*`]`** – анимация путем выбора выражений из списка с заданной скоростью *fps* (фрейм в секунду).

На рис. 11.65 показан пример анимации для функции Factor[x^n+1]. Выход анимации представлен изменяющимися формулами и демонстрирует символь-

```
sphere[θ_, ϕ_] := {Cos[θ] Sin[ϕ / 2], Sin[θ] Sin[ϕ / 2], Cos[ϕ / 2]};

torus[θ_, ϕ_] := {-Sin[ϕ] / 3, (2 - Cos[ϕ]) Sin[θ] / 3, (2 - Cos[ϕ]) Cos[θ] / 3};

Animate[ParametricPlot3D[(1 - λ) sphere[θ, ϕ] + λ torus[θ, ϕ],
  {θ, 0, 2 π}, {ϕ, 0, 2 π}, PlotRange → 1, Mesh → True],
 {λ, 0, 1}, AnimationDirection → ForwardBackward,
 SaveDefinitions → True, AnimationRunning → False]
```

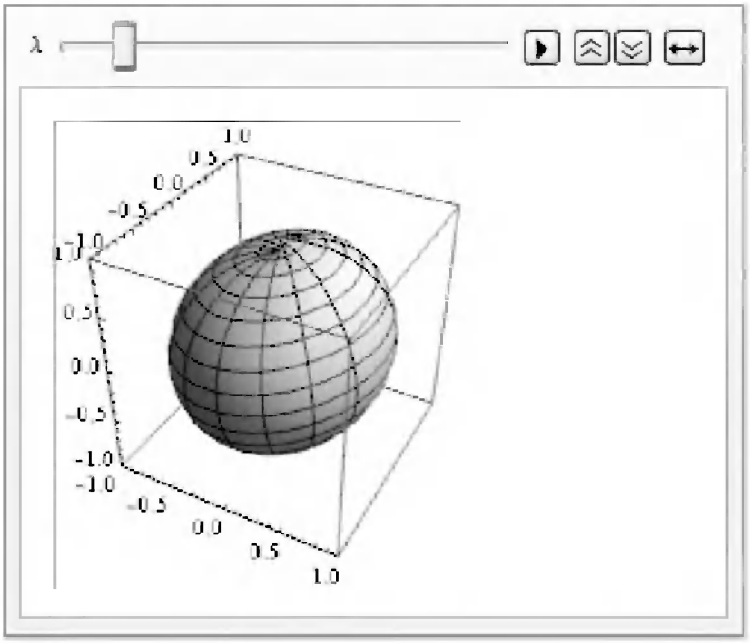


Рис. 11.63. Ноутбук, иллюстрирующий превращение шара в тор (фигура вначале близка к шару)

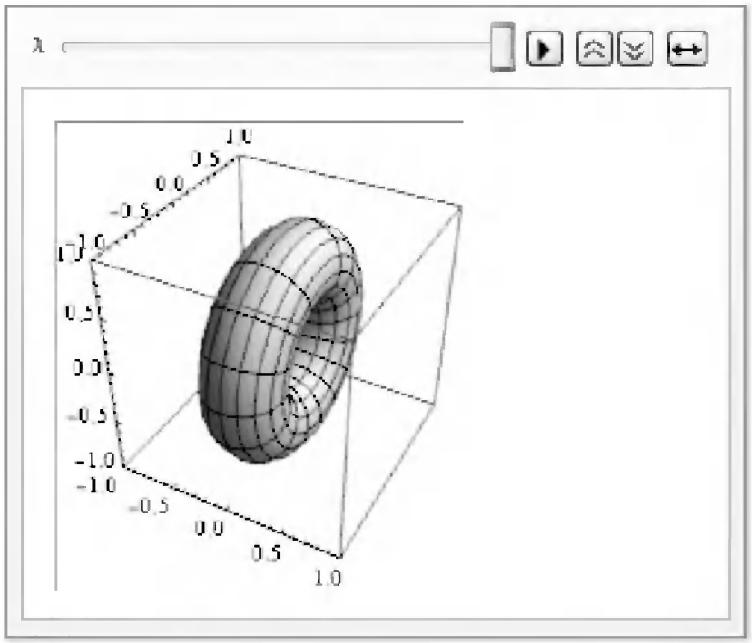


Рис. 11.64. Конечная стадия анимации – фигура на рисунке имеет вид тора

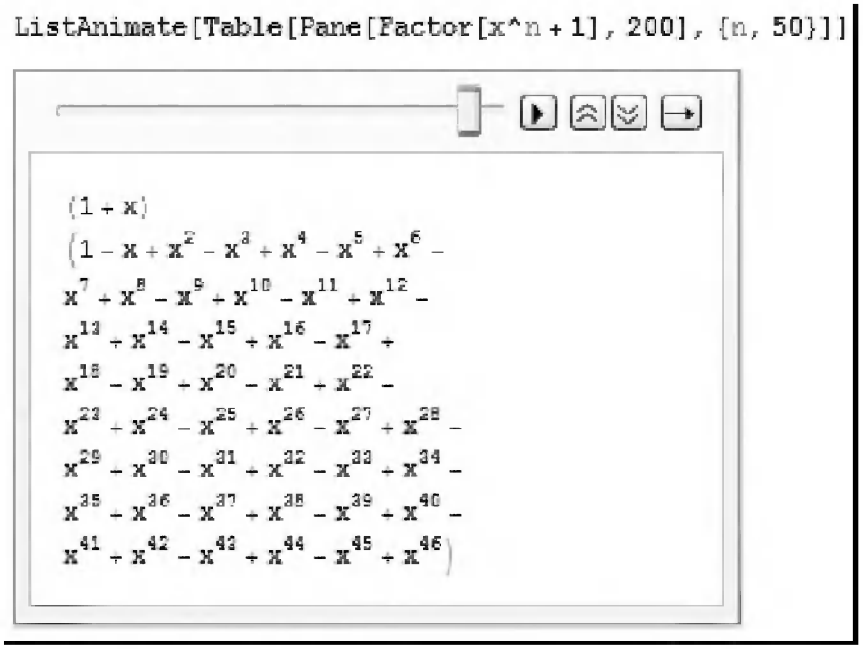


Рис. 11.65. Пример анимации функции Factor[x^n+1] при разных n

ные преобразования (факторизацию) выражения x^n+1. Список выражений формируется с помощью функций Table (создание списка-таблицы) и Pane (ограничение вывода по ширине).

11.7.11. Новые опции управления цветом и визуализация массивов

В Mathematica 6 существенно расширены возможности управления цветом и визуализации массивов. В частности, введена многофункциональная опция ColorFunction. В таблице рис. 11.66 представлен список графических функций, в которых может использоваться эта опция, с указанием переменных, к которым она относится. Опция позволяет задавать зависимости цвета от значений переменных или выражений, в которые она входит.

Таблица значений различных функций может представлять на рисунке орнамент, порой имеющий определенную художественную ценность. Примером может служить пример, показанный на рис. 11.67. Здесь для вывода графического представления массива используется функция ArrayPlot. В оригинале график черно-белый.

Наконец, на рис. 11.68 показан еще один пример применения опции ColorRules для цветовой визуализации матрицы с помощью функции MatrixPlot. При этом опция ColorRules назначает желтый цвет нулевым элементам матрицы и черный – остальным (обратите внимание на то, как это делается).

Raster	a
Plot, ListLinePlot, ListLogPlot, etc.	x, y
ParametricPlot	x, y, u or x, y, u, v
RegionPlot	x, y
ArrayPlot, ReliefPlot	a
ContourPlot and ListContourPlot	f contour levels
DensityPlot and ListDensityPlot	f
ContourPlot3D, ListContourPlot3D	x, y, z, f
Plot3D, ListPlot3D, ListSurfacePlot3D, and ListPointPlot3D	x, y, z
ParametricPlot3D	x, y, z, u or x, y, z, u, v
RegionPlot3D	x, y, z

Рис. 11.66. Таблица функций и переменных для опции ColorFunction

```
ArrayPlot[Table[Cos[x*y]^3, {x, -40, 40}, {y, -40, 40}]]
```

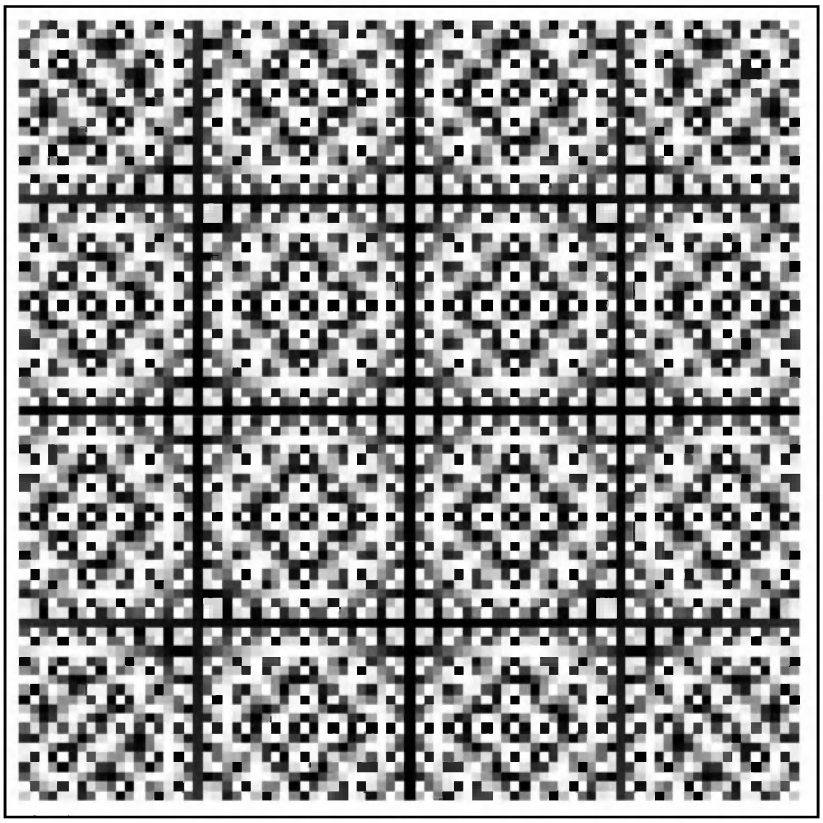


Рис. 11.67. Простейшая визуализация массива зависимости $\cos(xy)^3$

```
MatrixPlot[{{1, 0, 3}, {0, 2, 0}, {2, 0, 4}},
 ColorRules → {0 → Yellow, _ → Black}]
```

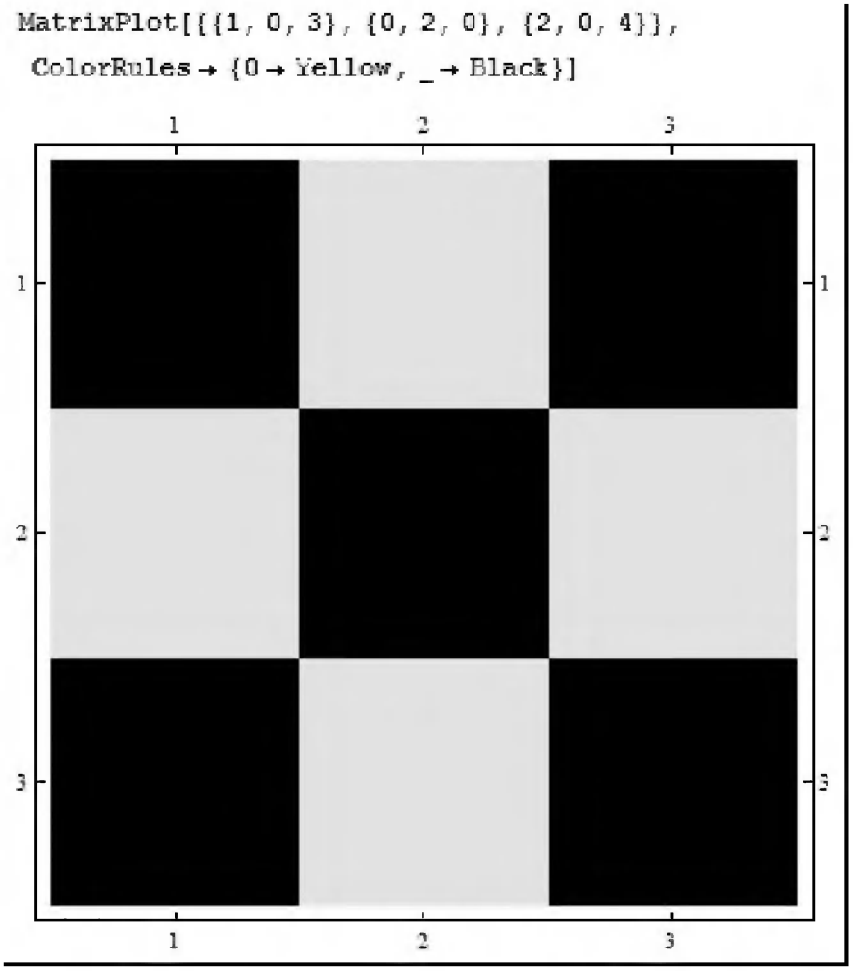


Рис. 11.68. Пример визуализации элементов матрицы

Глава 12

Пакет расширения Graphics системы Mathematica

Пакет расширения Graphics, встроенный в СКМ Mathematica 4/5/5.1/5.2, дает множество мощных дополнительных средств для построения графиков самого изысканного вида. Он является прекрасным инструментом для визуализации задач, допускающих представление результатов в графической форме. Большинство функций этого пакета вошло в ядро Mathematica 6 и представляет интерес для пользователей последнего.

12.1. Создание анимационных графиков

12.1.1. Анимация графика кривой

Фактически техника *анимации* (оживления) графиков уже была описана. Напомним, что она сводится к подготовке отдельных кадров анимационного рисунка, которые специфицируются особой изменяющейся переменной t. Это не обязательно время, возможно, что t задает размеры изображения, его место или иную характеристику. Естественно, что имя переменной можно выбирать произвольно.

Подпакет Animation подключается автоматически и в ряде случаев не требует загрузки (хотя на всякий случай его лучше загрузить при использовании средств анимации). Пакет задает две важнейшие функции:

- **Animate[grcom,{t,tmin,tmax,dt]** – задает построение серии графических объектов grcom при изменении параметра t от tmin до tmax с шагом dt.
- **ShowAnimation[{p1,p2,p3,....}]** – дает анимацию последовательным воспроизведением ранее подготовленных объектов p1,p2,p3,....

Ввиду очевидности применения этих функций их подробное описание нецелесообразно. Для анимации используется проигрыватель, который уже был описан.

12.1.2. Анимация графика, заданного параметрически

Следующий пример иллюстрирует задание анимации графика с параметрическим заданием функции:

```
ShowAnimation[Table[ Graphics[Line[0, 0, Cos[ t], Sin[t]],
PlotRange -> -1, 1, -1, 1], t, 0, 2Pi, Pi/8]]
```

Пустив этот фрагмент программы, вы увидите построение отрезка прямой, вращающегося вокруг одного неподвижного конца. Здесь для анимации вначале строится набор кадров, а затем используется функция **ShowAnimation**.

12.1.3. Анимация трехмерного графика поверхности

Аналогичным образом осуществляется анимация трехмерных графиков поверхностей или фигур. Рисунок 12.1 показывает начало подготовки к анимации сложной поверхности, описываемой функцией Бесселя, аргумент которой меняется от кадра к кадру.

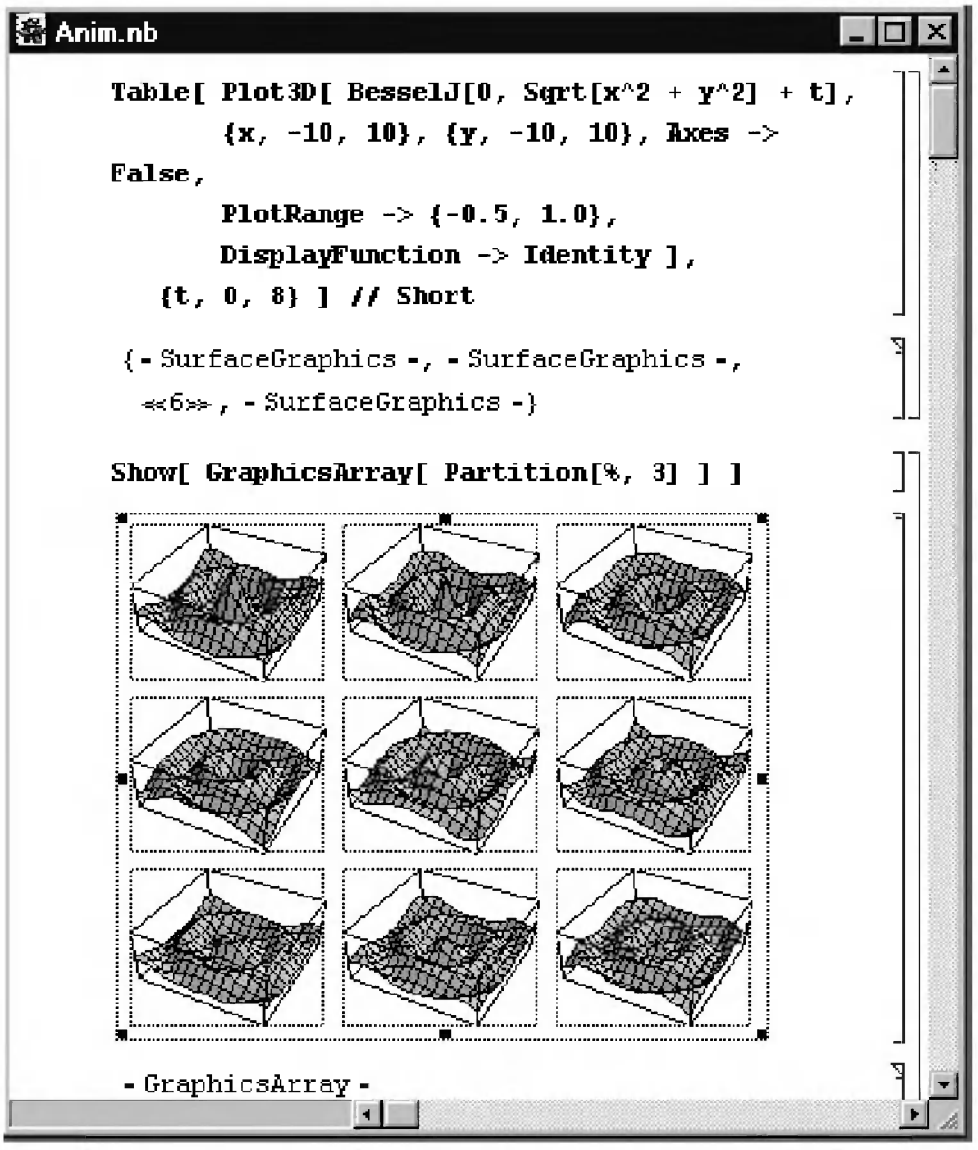


Рис. 12.1. Подготовка к анимации сложной 3D-поверхности

Обратите внимание на применение функций **Show** и **GraphicsArray** для построения на одном графике сразу всех кадров (фаз) анимации. Порой этот набор кадров даже важнее, чем анимация, длящаяся доли секунд или несколько секунд. Для пуска анимации используются команды функции **ShowAnimation**. Как и для

предшествующих примеров, она строит последовательно все рисунки – кадры анимации. При пуске анимации наблюдаются характерные колебания поверхности – пик ее проваливается вниз, образуя впадину, а затем снова выходит вверх.

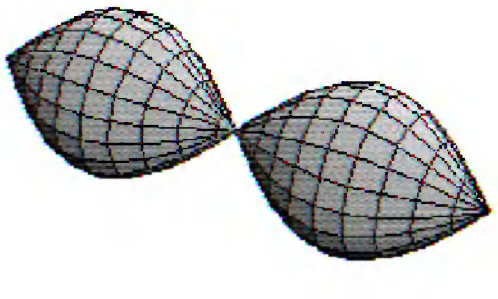
12.1.4. Другие средства анимации

Для упрощения анимации сложных графиков в подпакете Animations задан ряд специализированных функций:

- **MoviePlot[f[x,t],{x,xmin,xmax},{t,tmin,tmax}]** – дает анимацию графика Plot[f[x,t],{x,xmin,xmax}]
- **MoviePlot3D[f[x,y,t],{x,xmin,xmax},{y,ymin,ymax},{t,tmin,tmax}]** – дает анимацию трехмерного графика.
- **MovieDensityPlot[f[x,y,t],{x,xmin,xmax},{y,ymin,ymax},{t,tmin,tmax}]** – дает анимацию трехмерного графика плотности.
- **MovieContourPlot[f[x,y,t],{x,xmin,xmax},{y,emin,ymax},{t,tmin,tmax}]** – дает анимацию контурного графика.
- **MovieParamenticPlot[{f[s,t},{g[s,t]}],{s,smin,smax},{t,tmin,tmax}]** – дает анимацию параметрического графика.
- **SpinShow[graphics]** – дает вращение графического объекта. Функция имеет ряд опций, которые можно просмотреть командой **Options[SpinShow]**.

Пример построения сложной вращающейся в пространстве фигуры, напоминающей гантели, показан на рис. 12.2. В данном случае трехмерная фигура задана в параметрической форме, а для последующей ее анимации используется функция **SpinShow**.

```
g = ParametricPlot3D[
 {x, Cos[t] Sin[x] , Sin[t] Sin[x]},
      {x, -Pi, Pi}, {t, 0, 2Pi},
      Axes -> False, Boxed -> False]
```



```
SpinShow[ g, Frames -> 10,
    SpinRange -> {0 Degree, 180 Degree} ]
```

Рис. 12.2. Построение вращающейся в пространстве фигуры – «гантели»

Последний пример дан в упрощенной форме – без окна с проигрывателем. Разумеется, они появляются при реальном пуске анимации.

12.2. Управление цветом графиков

12.2.1. Установка аргумента цвета

При построении графиков в полярной системе координат полезно указывать цвет комплексной величиной. Для этого в подпакете ArgColor служит функция:

- **ArgColor[z]** – дает цвет, определяемый комплексным аргументом z.
- **ArgShade[z]** – дает уровень серого цвета, определяемый z.

Действие этих функций вполне очевидно.

12.2.2. Установка цветовой системы

Обычно цвета задаются в RGB (Red-Green-Blue) цветовой системе. При этом указывается относительный уровень интенсивности каждого цвета – в виде вещественных чисел с плавающей точкой в интервале от 0 до 1. В подпакете Colors заданы функции установки цвета, заданного в других известных цветовых системах:

- **CMYColor[c,m,y]** – установка цвета по системе CMY (Cyan-Magenta-Yellow).
- **YIQColor[y,i,q]** – установка цвета по системе YIQ (NTSC телевизионная система).
- **HLSColor[h,l,s]** – установка цвета по системе HLS (Hue-Lightness-Saturation).

AllColors – переменная-функция, выводящая список установленных цветов.

Примеры применения функций даны ниже:

```
<<Graphics`Colors`
YIQColor[0.5, -0.1, 0.2]
  RGBColor[0.53,0.4,0.957]
Blue
  RGBColor[0.,0.,1.]
Red
  RGBColor[1.,0.,0.]
```

Кроме этого, в подпакете имеется внушительная таблица англоязычных наименований разных цветов и цветовых оттенков – она выводится функцией **AllColors**. Их можно использовать для задания в качестве аргумента *y* функций, управляющих цветами. Например, шоколадный цвет можно задать следующим образом:

```
Chocolate
  RGBColor[0.823496,0.411802,0.117603]
```

Поскольку система Mathematica постоянно совершенствуется, то данные по компонентам цветов в разных версиях систем могут отличаться.

12.3. Специальные типы графики

12.3.1. Построение стрелок – Arrow

Подпакет Arrow служит для построения стрелок на двумерных графиках (или самих по себе). Для этого предназначена функция:

- **Arrow[start,finish,opts]** – строит стрелку по координатам ее начала start и конца finish. Рекомендуется просмотреть список опций этой функции.

Рисунок 12.3 показывает построение двухсторонних стрелок, посаженных на иглу, стоящую на кресте, – своеобразная модель стрелок компаса.

```
Show[Graphics[{ Arrow[{0,1},{1,0}, HeadScaling -> Relative,
 HeadShape -> {Polygon[{{0,0},{-.2,.05},{-.2,-.05}}],
    Polygon[{{-1,0},{-.8,.05}, {-.8,-.05}}]}],
 Arrow[{.25, .25}, {.75,.75}, HeadShape -> {Line[{{.1,.1},{-.1,-.1}}]},
 Line[{{.1,-.1},{-.1,.1}}]} ]},PlotRange ->All]]
```

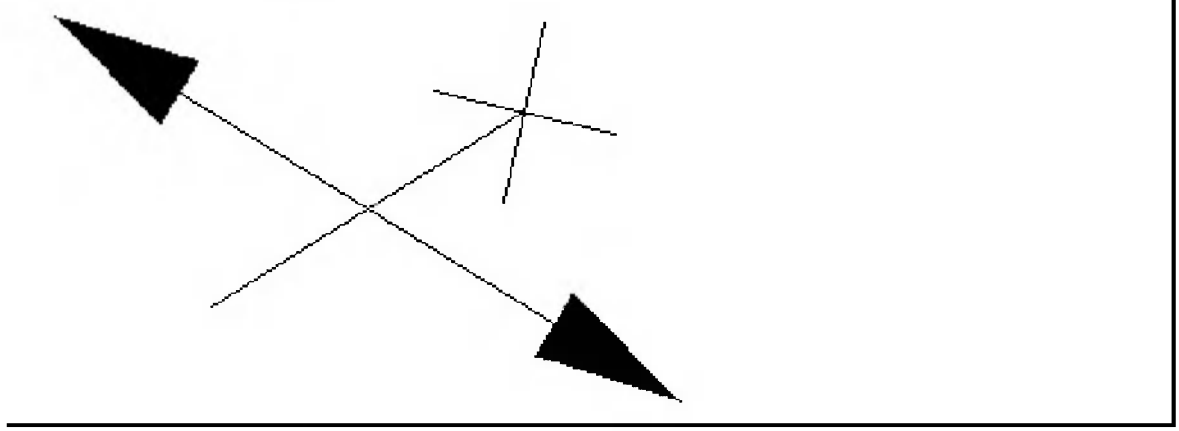


Рис. 12.3. Построение двухсторонних стрелок, посаженных на иглу

Построение стрелок оживляет многие типы графиков. Их можно использовать, к примеру, для указания особых точек на графиках.

12.3.2. Задание картографических систем

Подпакет ComplexMap задает функции для построения графиков в наиболее важных **картографических** системах:

- **CartesianMap[f,{xmin.xmax},{ymin,ymax}]** – строит рисунок f в Картезианской системе координат.
- **PolarMap[f,{rmin,rmax},{thetamin,thetamax}]** – строит рисунок по данным f в полярной системе координат.

Примеры на использование этих довольно простых функций можно найти в справке по пакету. Эти функции безусловно полезны тем пользователям, которые работают в области географии и картографии.

12.3.3. Построение объемных контурных графиков – ContourPlot3D

В подпакете ContourPlot3D заданы две функции, которые строят контурные объемные графики. Напоминаем, что функции ядра **ContourPlot** и **ListContourPlot** строят графики этого типа только двумерные. Для построения **объемных контурных графиков** надо использовать функции:

- **ContourPlot3D[f,{x,xmin,xmax},{y,ymin,ymax},{z,zmin,zmax}]** – строит трехмерный контурный график функции f трех переменных x, y и z.
- **ListContourPlot3D[f,{f_{111},f_{112},...},{f_{121},f_{122},...},...},...}]** – строит контурный график по данным трехмерного массива значений f_{xyz}.

На рис. 12.4 показано построение сферы с отверстием с помощью первой из этих функций. Обратите внимание на то, что никаких усилий по созданию в сфере отверстия не требуется. Оно получено просто усечением ограничительного ящика, в котором размещается сфера. Для этого пределы по оси y заданы как {–1.2,2}, тогда как по остальным осям {–2,2}.

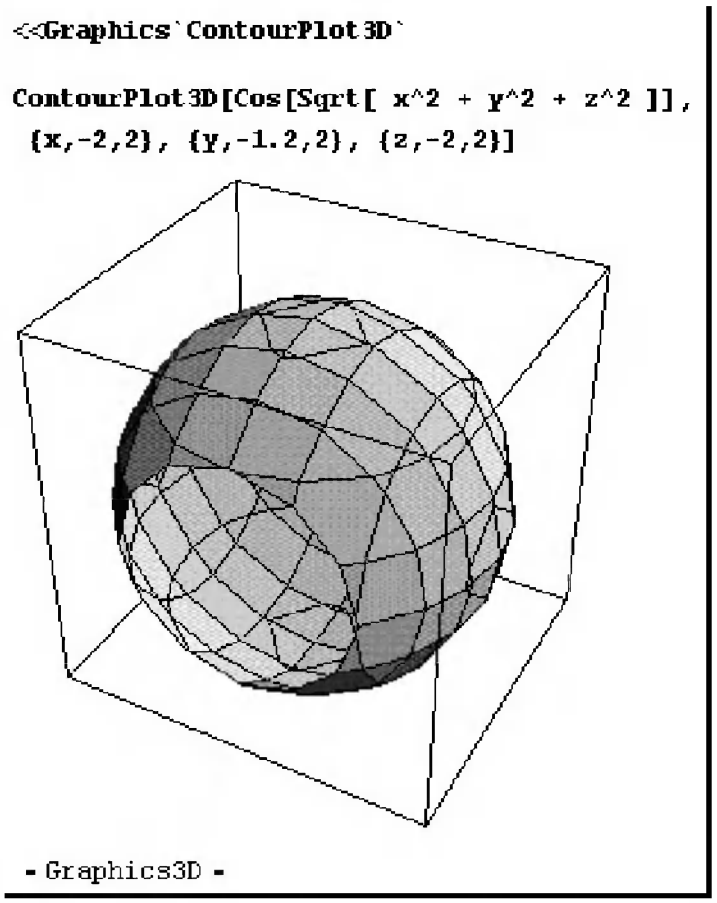


Рис. 12.4. Построение сферы с отверстием

Интересные возможности открывает опция Contours, которая позволяет как бы раздвинуть в пространстве части трехмерной поверхности. Рисунок 12.5 демонстрирует ее действие.

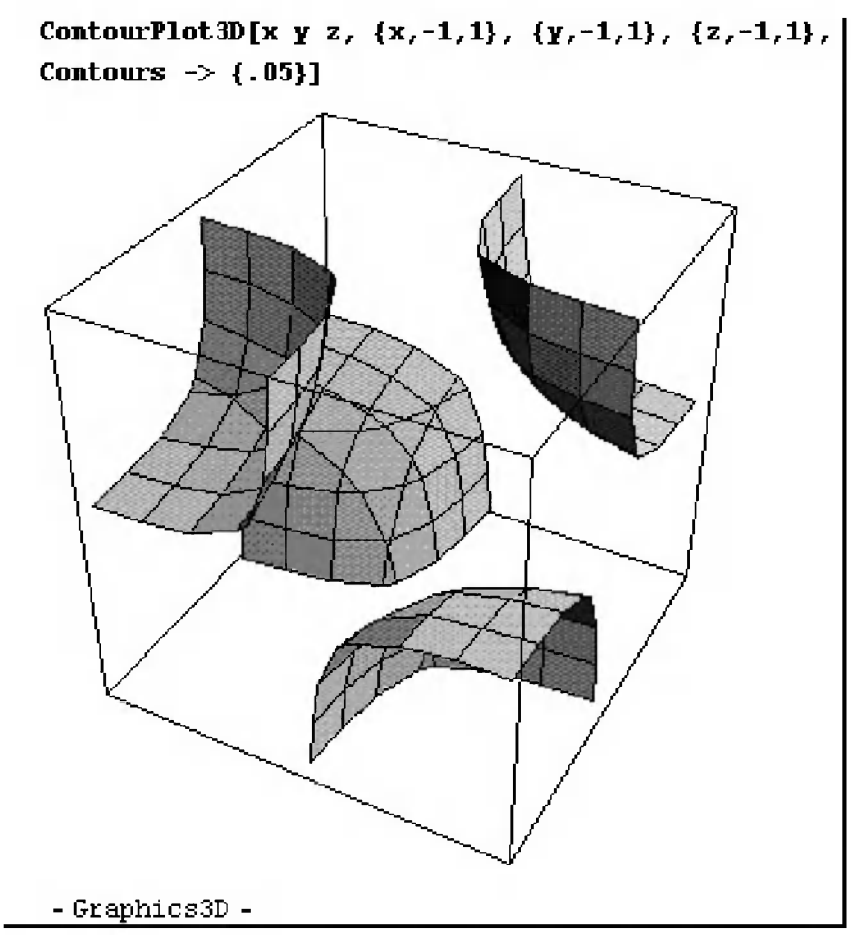


Рис. 12.5. Построение частей сферы в пространстве

Вторая функция – **ListContourPlot3D** позволяет строить ряд фигур или поверхностей в пространстве.

12.3.4. Построение графиков с окраской внутренних областей

Многие графики довольно выигрывают при построении их с *закраской*. Например, чтобы проиллюстрировать значение определенного интеграла от какой-то функции $f(x)$, достаточно просто закрасить ее график в диапазоне изменения x от нижнего предела интегрирования a до верхнего b. Для построения подобных графиков в подпакете FilledPlot имеется ряд полезных функций.

Начнем их описание с основных:

- **FilledPlot[f,{x,xmin,xmax}]** – строит график функции $f(x)$ с окраской площадей, образованных линией функции и горизонтальной осью x. По умолчанию действуют опции Fills->Automatic и Curves->Back (то есть кривые строятся на заднем плане, при значении Front построение фигур задано на переднем плане).
- **FilledPlot[{f1,f2,...},{x,xmin,xmax}]** – строит графики функций с выделением областей между ними разной окраской (цвет задается автоматически).

Поясним применение этих функций рядом примеров. Перед этим надо не забыть загрузить подпакет командой

```
<< Graphics`FilledPlot`
```

Пример построения графиков трех фигур показан на рис. 12.6. Здесь для закраски областей между фигурами и осью абсцисс использованы опция Fills и директива GrayLevel (окраска серым цветом заданной плотности).

```
FilledPlot[{x^2/20-0.5, -Cos[x], Sin[x]/x}, {x, 0, 2 Pi},
 Fills -> {{{1, Axis}, GrayLevel[.8]},
 {{2, 3}, GrayLevel[.6]}}, Curves -> Front]
```

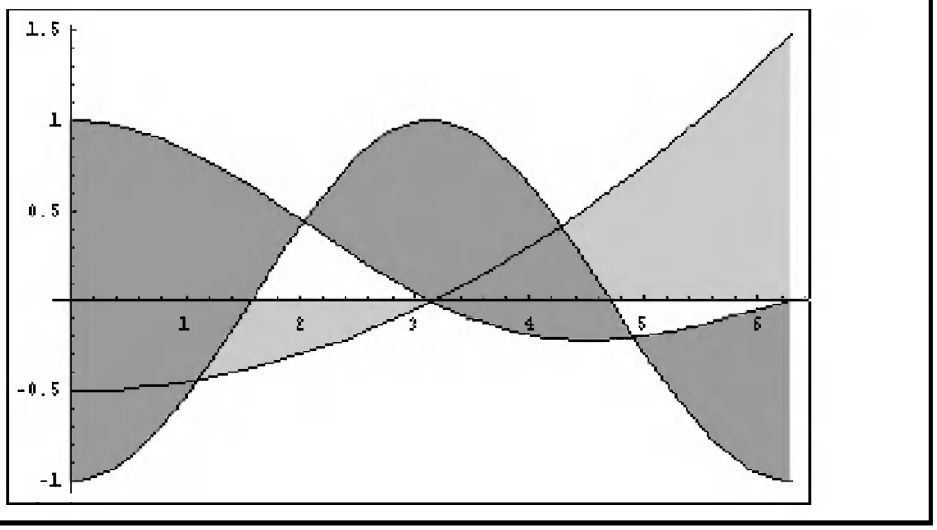


Рис. 12.6. Построение трех кривых с закраской серым цветом

Имеется также ряд специальных функций для построения кривых с окраской образуемых ими областей:

- **FilledListPlot[{y1,y2,...}]** – строит графики с окраской, меняющейся между кривыми {1,y1), {2,y2) и осью абсцисс *x*.
- **FilledListPlot[{{x1,y1},{x2,y2},...}]** – строит графики ряда кривых с окраской, заданной {xi,yi) и осью абсцисс *x*.
- **FilledListPlot[data1, data2,...}]** – строит графики ряда кривых с закраской областей, специфицированных данными datai.

Иногда важное значение может иметь опция AxesFront->Значение. При значении этой опции False область закраски закрывает соответствующую часть осей, а при значении True оси выводятся поверх закраски – рис. 12.7. Сам по себе общий вывод осей задается опцией Axes->True – оси выводятся и Axes->False – они вообще не выводятся.

В приведенном примере область окраски ограничена треугольником, который строится как полигон. Если установить опцию AxesFront->False, то часть осей (внутри треугольника) будет не видна.

```
Show[Graphics[{GrayLevel[.8],
 Polygon[{{0,1/2},{1,0},{1,1},{0,1/2}}]},
 AxesFront -> True, Axes -> True,
 AxesOrigin -> {.5,.5}]]
```

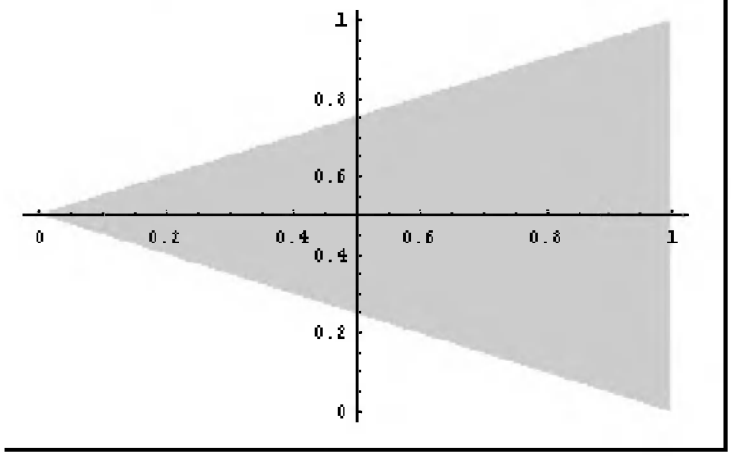


Рис. 12.7. Пример вывода осей поверх закрашенной области

12.3.5. Графики логарифмические и полулогарифмические

Подпакет Graphics задает ряд функций для построения специальных графиков, например с логарифмическими и полулогарифмическими масштабами, с нанесенными на кривые точки, графики в виде гистограмм и т. д. Такие графики широко применяются для визуализации не только математических и физических, но и финансовых и экономических расчетов.

Для построения логарифмических и полулогарифмических графиков служат следующие функции:

- **LogPlot[f,{x,xmin,xmax}]** – строит линейно-логарифмический график f(x) при изменении x от xmin до xmax.
- **LogLinearPlot[f,{x,xmin,xmax}]** – строит логарифмически-линейный график $f(x)$.
- **LogLogrPlot[f,{x,xmin,xmax}]** – строит логарифмический (по обеим осям) график $f(x)$.
- **LogListPlot[{{x1,y1},{x2,y2},...}]** – строит линейно-логарифмический график точек.
- **LogLinearListPlot[{{x1,y1},{x2,y2},...}]** – строит логарифмически-линейный график точек.
- **LogLogListPlot[{{x1,y1},{x2,y2},...}]** – строит логарифмический (по обеим осям) график точек.

Ввиду очевидности этих функций ограничимся одним примером – построением экспоненциальной функции в полулогарифмическом масштабе (по оси y – ло-

гарифмический, а по оси x – линейный). К примеру, график экспоненты в полулогарифмическом масштабе дает прямую линию (проверьте!).

Группа функций

LogListPlot[{y1,y2,...}] **LogLinearListPlot[{y1,y2,...}]** и **LogLogListPlot[{y1,y2,...}]**

дает те же построения, что предшествующие три точки, с той разницей, что ординаты абсцисс точек x равны 1,2,3,... и т. д. Это иногда упрощает задание графиков.

12.3.6. Графики в полярной системе координат

Для построения графиков функций в полярной системе координат заданы следующие функции:

- **PolarPlot[f,{t,tmin,tmax}]** – строит график функции в полярной системе координат как положение конца радиус-вектора f при изменении угла от tmin до tmax.
- **PolarPlot[{f1,f2,...}]** – строит графики ряда функций f1, f2,... в полярной системе координат.
- **PolarListPlot[{r1,r2,...}]** – строит графики ряда функций по их радиус-векторам.

Применение этих функций вполне очевидно.

12.3.7. Построение столбиковых диаграмм

В ряде случаев удобно представление данных в виде столбиковых и круговых диаграмм. Для построения *столбиковых диаграмм* служат функции, описанные ниже.

BarChar[datakist1,datalist2,...] – строит столбиковую диаграмму по данным листов, располагая столбики рядом и обеспечивая их автоматическую закраску цветом (см. рис. 12.8).

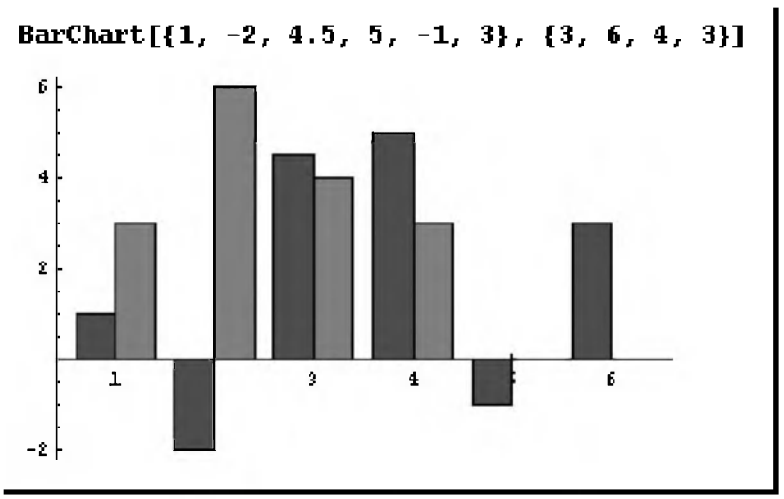


Рис. 12.8. Построение столбиковых диаграмм для двух списков данных с расположением столбцов рядом друг с другом.

Здесь любопытно отметить, что списки данных могут иметь разную длину. Число столбцов задается большим списком. Отсутствующие данные у списков меньшей длины считаются нулевыми, и для них столбцы не строятся. Данные, представленные отрицательными числами, строятся как столбцы, обращенные вниз.

StackedBarChar[datakist1,datalist2,...] – строит столбиковую диаграмму, располагая столбцы одних данных над столбцами других данных с автоматическим выбором цветов для каждого набора данных.

В этой столбиковой диаграмме вначале строятся столбцы, представляющие данные для первого списка, над ними надстраиваются столбцы новых данных, так что общая высота столбцов пропорциональна сумме численных значений i-х элементов списков.

PercentileBarChar[datakist1,datalist2,...] – строит столбиковую диаграмму, отображающую нормированные данные в процентах.

GeneralizeBarChar[datakist1,datalist2,...] – строит столбиковую диаграмму с заданной высотой и шириной столбцов.

В этом случае имеется возможность в списках данных указать позицию по оси x, высоту столбца и его ширину. Ширина задается в относительных единицах: при 1 столбцы сливаются (но выделяются цветом), при величине меньше 1 они разделяются пустыми промежутками, а при величине больше 1 столбцы перекрываются.

Все указанные функции имеют опции, существенно влияющие на вид диаграмм. Рекомендуется просмотреть их с помощью функции **Options**.

Ограничимся тремя наглядными примерами на применение опций. Рисунок 12.9 показывает построение столбиковой диаграммы с горизонтальным расположением столбцов и произвольным выбором цвета для каждого набора.

```
PercentileBarChart[{1,3,-4,5,3.5,-3}, {-3,-2,5,3},
 BarStyle -> {RGBColor[0,1,0],
 RGBColor[1,1,0]}, BarOrientation -> Horizontal,
 Axes -> False, Frame -> True]
```

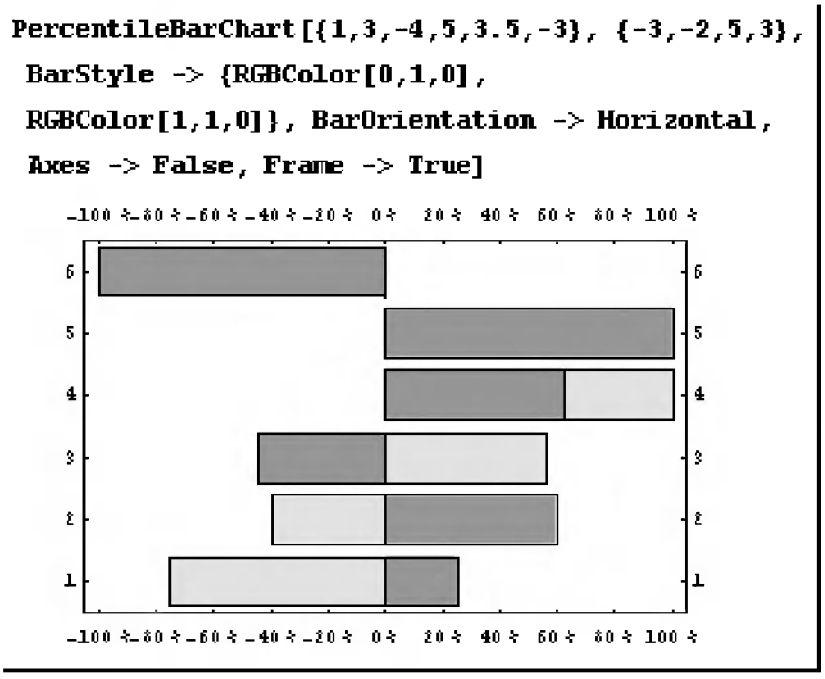


Рис. 12.9. Построение столбиковой диаграммы с горизонтально расположенными столбцами и произвольным выбором цвета для каждого комплекта данных

На рис. 12.10 показан более сложный пример построения столбиковых диаграмм. Наряду с цветовыми эффектами задается обвод пунктирной линией столбцов одного из комплектов данных и – главное – вывод надписей (названия месяцев) под наборами столбцов. Обратите внимание на то, что надписи могут быть на русском языке, несмотря на выбор набора шрифтов по умолчанию.

```
BarChart[{1, 3, 4, 4.5, 3.5, 3}, {3, 2, 5, 3},
 BarSpacing -> -.3, BarGroupSpacing -> .5,
 BarStyle -> {GrayLevel[.6], Hue[0]},
 BarEdgeStyle -> {{Dashing[{.01}],Hue[0]},GrayLevel[0]},
 BarLabels -> {"Апрель","Май","Июнь","Июль","Август","Сентябрь"},
 PlotLabel -> "Projected and Current Profit,
 Tourist Season", DefaultFont -> {"Helvetica", 9} ]
```

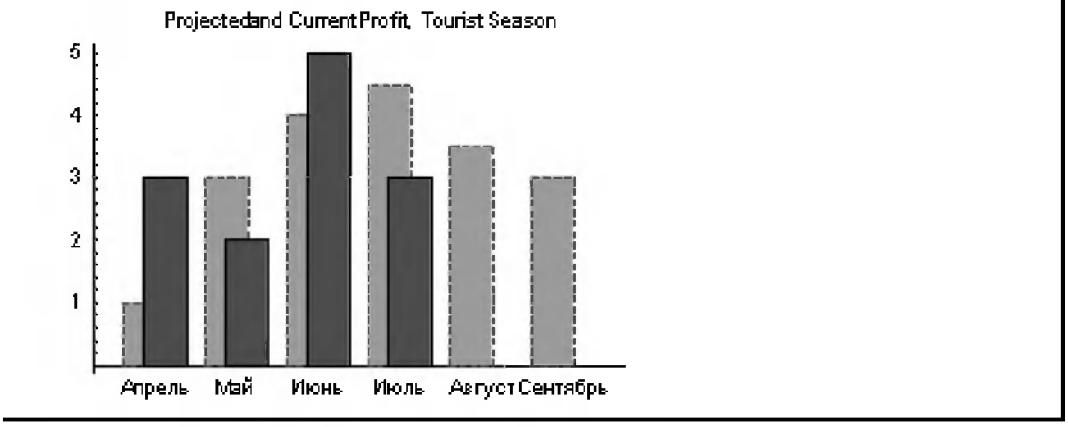


Рис. 12.10. Комплексное построение столбиковых диаграмм

12.3.8. Построение круговых диаграмм

Круговые диаграммы наиболее удобны, когда оцениваются относительные величины, при этом сумма данных соответствует площади круга. Следующая функция позволяет стоить такие диаграммы:

- **PieChart[data]** – строит круговые диаграммы по данным data (рис. 12.11). Тип диаграммы задается опциями, список которых и значения по умолчанию можно получить командой **Options[PieChart]**. Одна из опций PieExploded позволяет отделить заданный сектор от диаграммы, что порой повышает наглядность представления данных.

Здесь используется также следующая функция:

- **DisplayTogether[plot1,plot2,...,opts]** – строит комбинированный графический объект.
- **DisplayTogetherArray[plot1,plot2,...,opts]** – строит комбинированный графический объект.

Возможность разнообразить круговые диаграммы иллюстрирует рис. 12.12. Здесь показано, что в секторы можно заносить небольшие надписи, например имена людей, наименование лет и месяцев и т. д.

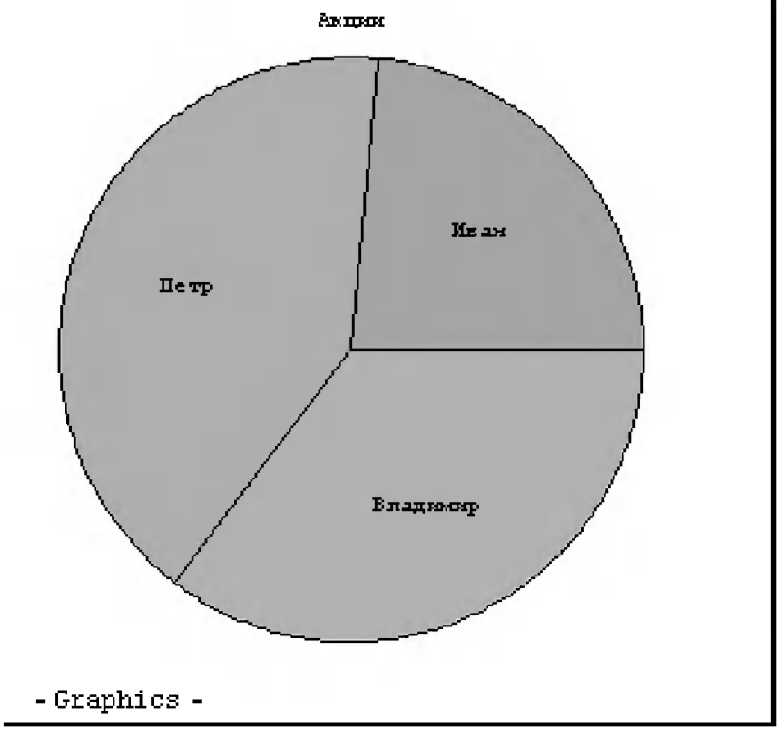
```
DisplayTogetherArray[
    PieChart[{.2,.25,.1}],
    PieChart[{.2,.23,.1},
        PieExploded->All],
    PieChart[{.2,.23,.1},
        PieExploded->{{3,.2}}]
]
```

- GraphicsArray -

Рис. 12.11. Построение набора круговых диаграмм

```
<< Graphics`Graphics`

PieChart[{12, 21, 18},
    PieLabels -> {"Иван", "Петр", "Владимир"},
    PieChart -> "Акции",
    PieStyle->{Hue[.42],Hue[.48],Hue[.5]}]
```



- Graphics -

Рис. 12.12. Построение круговой диаграммы с надписями внутри секторов

12.3.9. Объединение графиков различного типа

Функция **DisplayTogether** позволяет объединять графики разного типа. На рис. 12.13 показано построение графика экспериментальных точек и линий параболической и кубической регрессий.

```
<< Graphics`Graphics`

data = Table[{n/15, (n/15)^2 + 2 + Random[Real,{-.3,.3}]},
         {n, 15}];

fit = Fit[data, {1, x, x^2}, x]

2.11662 - 0.712792 x + 1.7508 x^2

altfit = Fit[data, {1,x^3}, x]

2.05351 + 1.14684 x^3

DisplayTogether[  Plot[altfit, {x,0,1},
 PlotStyle -> Hue[.6]],  ListPlot[data,
    PlotStyle -> {Hue[0], PointSize[.03]}],
    Plot[fit, {x,0,1},    PlotStyle -> {GrayLevel[0],
 Dashing[{.03}]}]]
```

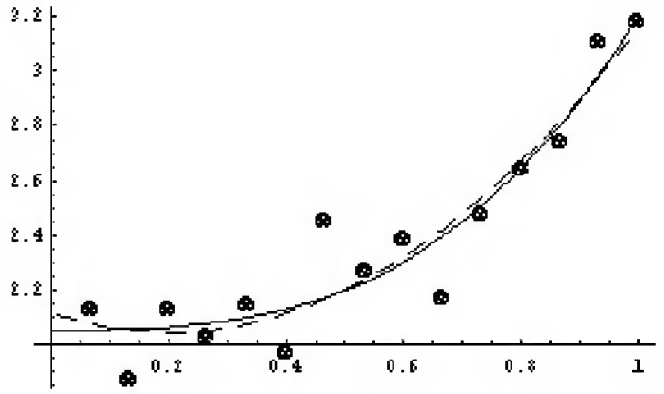


Рис. 12.13. Совместное построение исходных точек данных и линий параболической и кубической регрессий

Иногда нужно использовать в качестве точек графика цифры. Это реализует следующая функция, имеющая три формы:

- **TextListPlot[{y1,y2,...}]** – построение точек с ординатами yi и абсциссами 1,2,... с представлением их числами 1,2,....
- **TextListPlot[{{x1,y1},{x2,y2},...}]** – построение точек с координатами {xi,yi} и представлением их числами 1,2,....
- **TextListPlot[{{x1,y1,expr1},{x2,y2,expr2},...}]** – построение точек с координатами {xi,yi} и представлением их числами, заданными выражениями expri.

Еще одна функция, также имеющая три формы

LabelListPlot[{y1,y2,...}]
LabelListPlot[{{x1,y1},{x2,y2},...}]
LabelListPlot[{{x1,y1,expr1},{x2,y2,expr2},...}]

дает тот же результат, но дополнительно строит и сами точки.

Наконец, есть еще одна функция

- **ErrorListPlot[{{y1,d1},{y2,d2},...}]** – построение графика точек yi с зонами ошибок di (они строятся отрезками прямых).

Таким образом, подпакет Graphics обеспечивает построение наиболее распространенных типов графиков, используемых в научно-технической и финансово-экономической литературе.

12.4. Создание трехмерных графиков

12.4.1. Трехмерные столбиковые диаграммы

В подпакете Graphics3D, загружаемом командой

```
<<Graphics`Graphics3D`,
```

имеется ряд программ для простого построения трехмерных графиков. Они описаны ниже с примерами.

- **BarChart3D[{{z_{11},z_{12},...},{z_{21},z_{22}},....}]** – строит трехмерную столбиковую диаграмму по наборам данных высот столбцов z_{11},z_{12},... (рис. 12.14).
- **BarChart3D[{{{z_{11},$stile_{11}$},{z_{21},$stile_{21}$},....}]** – строит трехмерную столбиковую диаграмму по наборам данных высот столбцов z_{11},z_{12},... с указанием спецификации стиля для каждого столбца.

Нетрудно заметить, что функция **BarChart3D** автоматически задает стиль и цвет построения столбцов диаграммы. Эта функция имеет массу опций, с помощью которых можно менять вид диаграммы. Как обычно, перечень опций можно вывести с помощью команды **Options[BarChart3D]**.

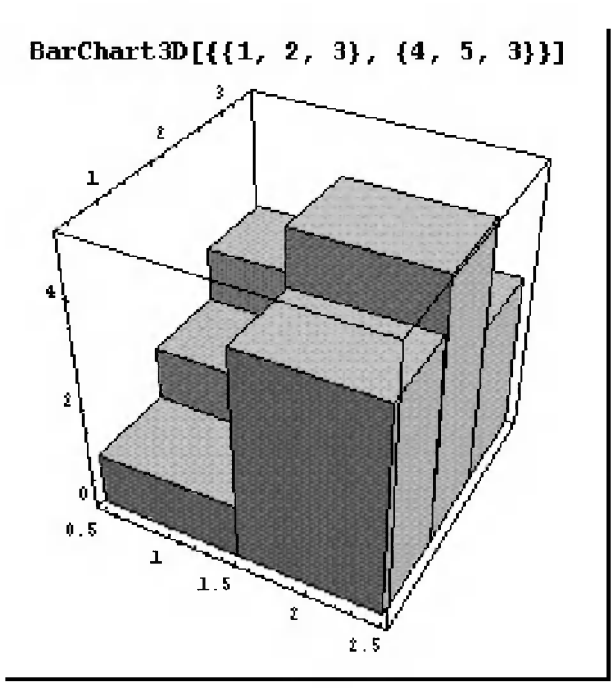


Рис. 12.14. Построение трехмерной столбиковой диаграммы

12.4.2. Построение точек и кривых в пространстве

Иногда возникает необходимость в построении точек и кривых в пространстве. Для этого служит функция:

- **ScatterPlot3D[{{x_1,y_1,z_1},{x_2,y_2,z_2},...}]** – строит точки в пространстве по их заданным координатам.

При использовании опции PlotJoined->True точки соединяются отрезками прямых и строится линия в пространстве.

12.4.3. Построение графиков поверхности и ее проекций

Для построения трехмерного графика поверхности служит функция:

- **ListSurfacePlot3D[{{{x_{11},y_{11},z_{11}},{x_{12},y_{12},z_{13}},...}}}]** – строит поверхность по координатам ее точек.

Обычно список координат точек также задается функцией **Table**.

- **ShadowPlot3D[f,{x,xmin,xmax},{y,ymin,ymax}]** – строит график поверхности f(x,y) с ее проекцией на опорную плоскость (рис. 12.15).
- **ListShadowPlot3D[{{{x_{11},y_{11},z_{11}},{x_{12},y_{12},z_{13}},...}}}]** – строит график поверхности z(x,y) с ее проекцией на опорную плоскость по координатам точек поверхности.

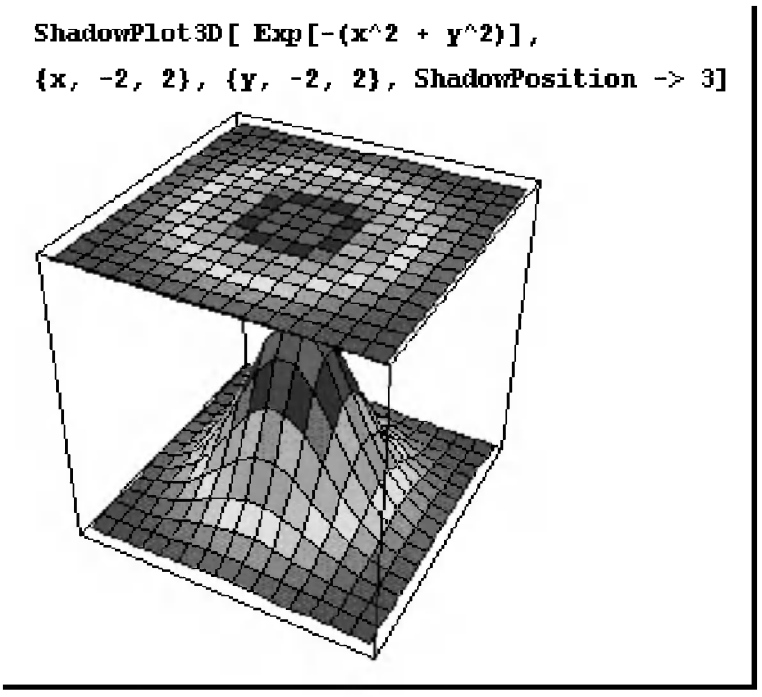


Рис. 12.15. Построение графика 3D-поверхности и ее проекции

С помощью функции **Shadow[go]**, где go – графический объект, представляющий трехмерную фигуру, можно построить и более сложные рисунки – например, график объемной фигуры и сразу всех трех ее проекций на взаимно перпендикулярные плоскости. Такое построение иллюстрируется документом, показанным на рис. 12.16.

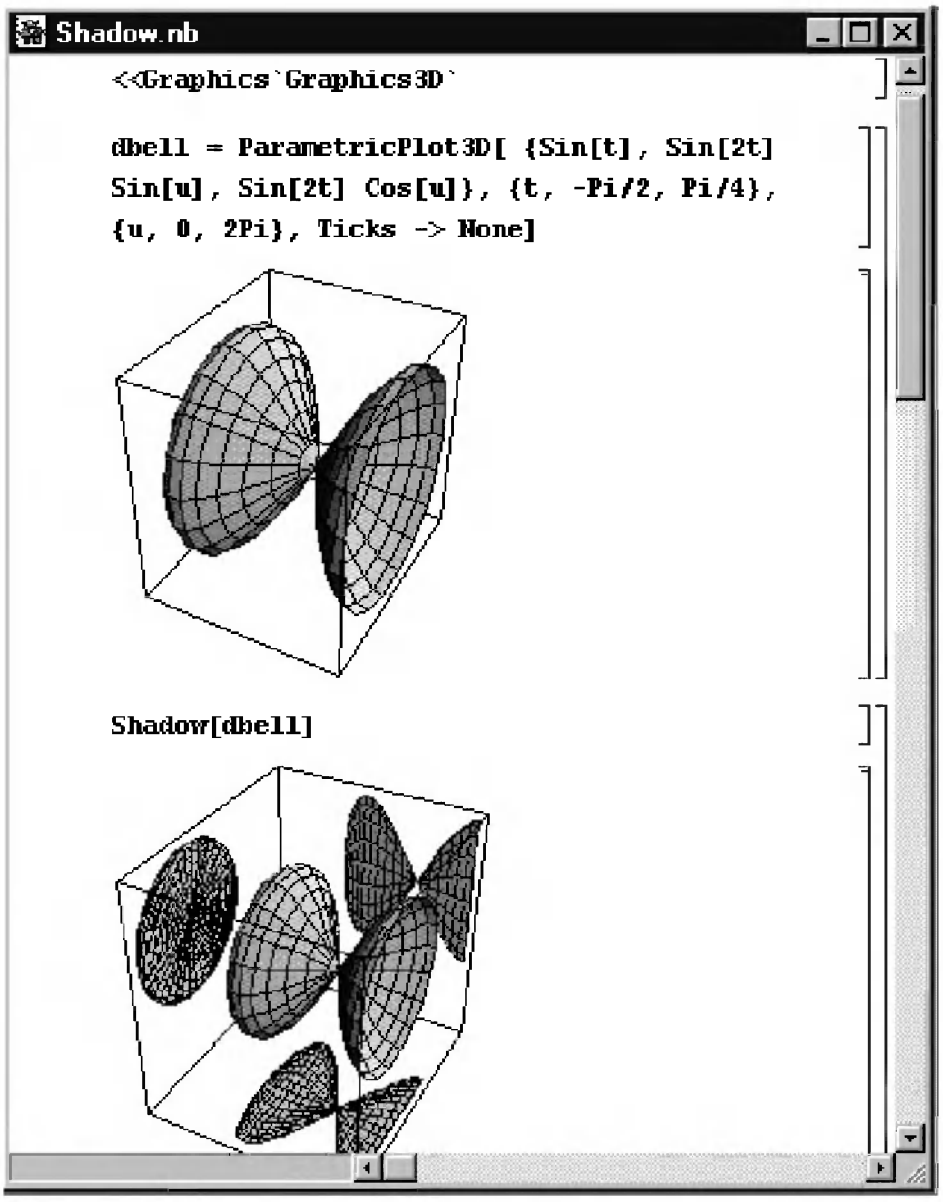


Рис. 12.16. Построение объемной фигуры и всех трех ее объектов

С функцией **Shadow** можно использовать различные опции. Отметим наиболее существенные: X, Y и Zshadow. Например, задав Zshadow->False, можно удалить одну из проекций, плоскость которой перпендикулярна оси z.

Для получения проекций на заданную плоскость, расположенную в пространстве, служат функции:

- **Project[g,pt]** – дает проекцию объекта g на диагональную плоскость, заданную списком из трех элементов pt. Например, список {1,1,0} даст проекцию на диагональную плоскость.
- **Project[g,{e1,e2},pt]** – дает проекцию объекта g в плоскости, определенной списком векторов {e1,e2} и списком pt.
- **Project[g,{e1,e2},pt,origin]** – дает проекцию объекта g в плоскости, определенной списком векторов {e1,e2} и списками pt и origin.

12.4.4. Построение трехмерных графиков с каскадным расположением

В конце подпакета определена следующая функция:

- **StackGraphics[{g1,g2,...}]** – строит в пространстве графические объекты, располагая их каскадно (рис. 12.17).

```
gtab = Table[Plot[Sin[x*n], {x, -Pi, Pi},
DisplayFunction -> Identity], {n, 3}]

{-Graphics-, -Graphics-, -Graphics-}

Show[StackGraphics[gtab]]
```

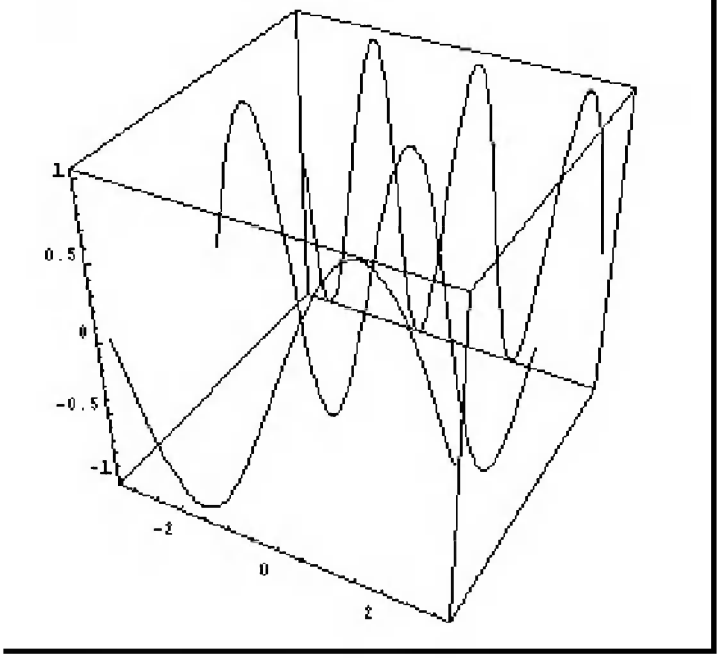


Рис. 12.17. Пример построения трех синусоид, расположенных каскадно

Обратите внимание на то, что здесь каждая синусоида расположена на своей плоскости, а последние расположены каскадно, то есть друг за другом.

12.5. Специальные средства построения графиков

12.5.1. Построение графиков неявных функций

Пакет ImplicitPlot задает три варианта функции для построения графиков неявно заданных функций:

- **ImplicitPlot[eqn,{x,xmin,xmax}]** – построение неявно заданной уравнением eqn функции при x, меняющемся от xmin до xmax.
- **ImplicitPlot[eqn,{x,xmin,m1,m2,...,xmax}]** – построение неявно заданной уравнением eqn функции при x, меняющемся от xmin до xmax, с исключением точек m1, m2,...
- **ImplicitPlot[{eqn1,eqn2,...},ranges,options]** – построение неявно заданных уравнениями eqn функции при x, меняющемся в пределах ranges, и при задании опций options.

На рис. 12.18 показано применение третьей формы функции **ImplicitPlot** с использованием опции PlotStyle и директивы Dashing.

```
ImplicitPlot[{(x^2 + y^2)^2 == (x^2 - y^2),
 (x^2 + y^2)^2 == 2 x y}, {x,-2,2},
 PlotStyle->{GrayLevel[0],Dashing[{.03}]}]
```

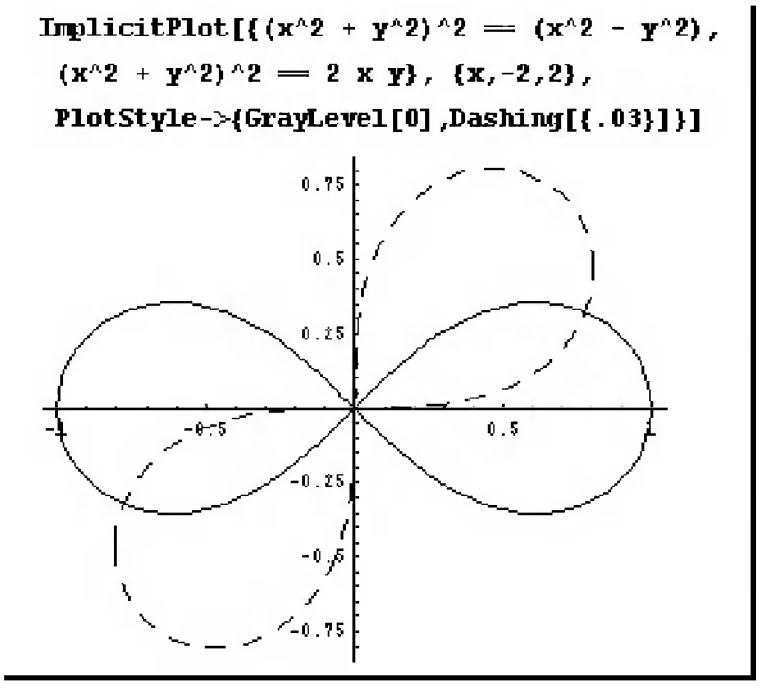


Рис. 12.18. Пример применения функции ImplicitPlot с опцией

12.5.2. Вывод обозначений кривых – легенд

Наглядность графиков, особенно имеющих несколько кривых, повышается при выводе обозначений кривых – так называемой легенды. В подпакете Legend для этого заданы следующие средства:

- **PlotLegend->{text1,text2,...}** – опция для функции Plot, устанавливающая легенду в виде последовательных текстовых надписей (рис. 12.19).
- **ShowLegend[graphic,legend1,legend2,...]** – устанавливает легенду в имеющийся график graphic.
- **{{{box1,text1},..},opts}** – спецификация легенды с цветными примитивами или графиками для рамок boxi с текстами texti.
- **{colorfunction,n,ninstring,maxstring,opts}** – спецификация легенды с n рамками и указанием цветовой спецификации с помощью строк, размещаемых в рамках.

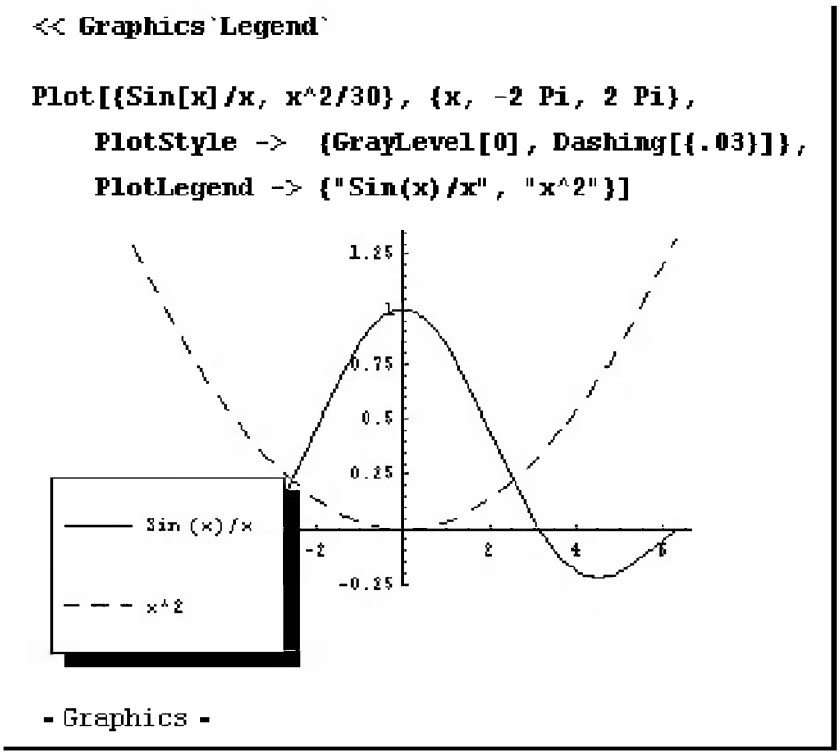


Рис. 12.19. Пример построения графиков двух функций с выводом легенды

Обратите внимание на то, что среди многочисленных опций функции **Plot** имеется ряд относящихся к параметрам легенды: LegendPosition->{-1,1} – установка позиции легенды, LegendSize->Automatic – установка размера легенды, LegendShadow->Automatic – установка автоматического заполнения легенды, LegendOrientation->Vertical – ориентация рамки легенды, LegendLabel->None – отметка легенды и LegendTextDirection->Automatic – направление текста. С помощью этих опций можно существенно влиять на вид легенды.

В заключение отметим еще две функции подпакета Legend:

- **Legend[legendargs,opts]** – создает графический примитив для задания индивидуальной легенды.
- **ShadowBox[pos,size,opts]** – создает графический примитив в виде рамки для легенды.

К примеру, функция

```
ShadowBox[{0, 0}, {1, 1},ShadowBackground -> GrayLevel[.8]]
```

создает графический примитив в виде пустого ящика для легенды с тенью. Для его просмотра можно использовать команду:

```
Show[Graphics[%]]
```

Применение функции **Graphics** здесь связано с тем, что **ShadowBox** порождает графический примитив, а не законченный графический объект.

12.5.3. Построение графиков со множеством объектов – MultipleListPlot

В подпакете MultipleListPlot задан расширенный вариант встроенной функции **ListPlot**:

- **MultipleListPlot[list1,list2,...]** – строит множество графических объектов по данным списков. Списки могут быть представлены ординатами y, координатами точек {x,y}, в виде {point,ErrorBar[{negerr,poserr}] и т. д.

Особое значение имеет опция PlotJoined. Если она используется в виде PlotJoined->True, то это означает соединение точек на графиках отрезками линий разного стиля, выбираемого автоматически.

Эта опция может быть представлена и со значением списком. Например, ее применение в виде PlotJoined->{True,False,False} означает, что точки первой кривой соединяются линиями, тогда как точки второй и третьей кривых линиями не соединяются. Рисунок 12.20 поясняет этот способ построения графиков.

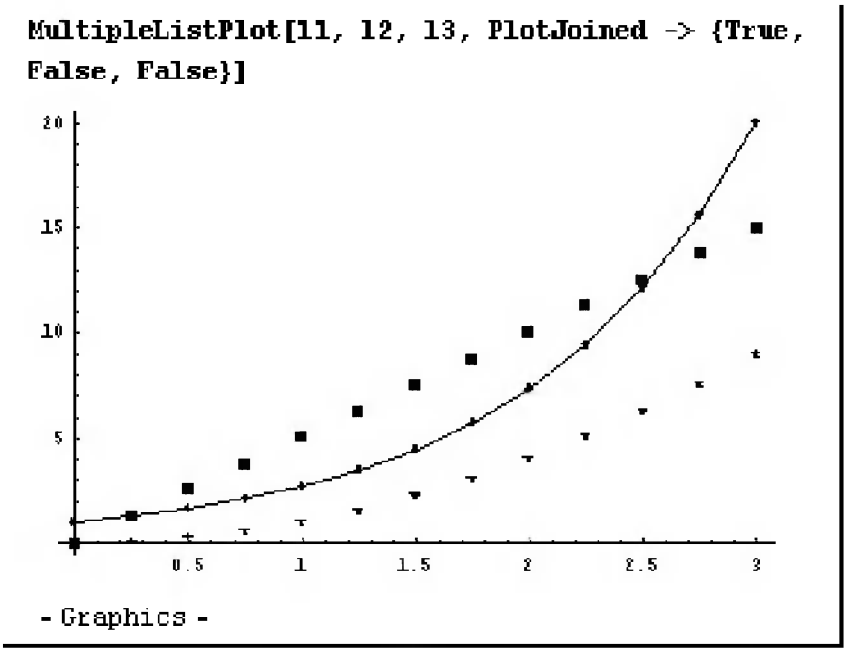


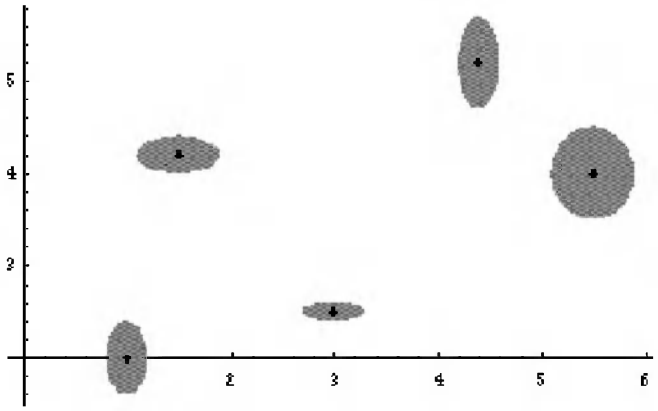
Рис. 12.20. Построение двух графиков по точкам и третьего с точками, соединенными отрезками прямых

12.5.4. Построение графиков с зонами ошибок

Функция **MultipleListPlot** может использовать в списках указания на построение точки с зоной ошибок (ErrorBar). Интересный случай построения точек с двусторонними зонами ошибок и зонами в виде окружностей или эллипсов демонстрирует рис. 12.21.

```
mybarfunc[pt_, ErrorBar[xerr_, yerr_]] :=
{GrayLevel[0.5],Disk[pt, {Max[Abs[xerr]],
Max[Abs[yerr]]}]}

MultipleListPlot[ {{{1, 2}, ErrorBar[0.2, 0.4]},
{{1.5, 4.2}, ErrorBar[0.4, 0.2]},
{{3, 2.5},   ErrorBar[0.3, 0.1]},
{{4.4, 5.2}, ErrorBar[.2, {-0.5, 0.3}]},
{{5.5, 4}, ErrorBar[{-0.4, 0.3}, {-0.2, 0.5}]}},
ErrorBarFunction -> mybarfunc]
```



- Graphics -

Рис. 12.21. Построение точек с зонами погрешности разного типа

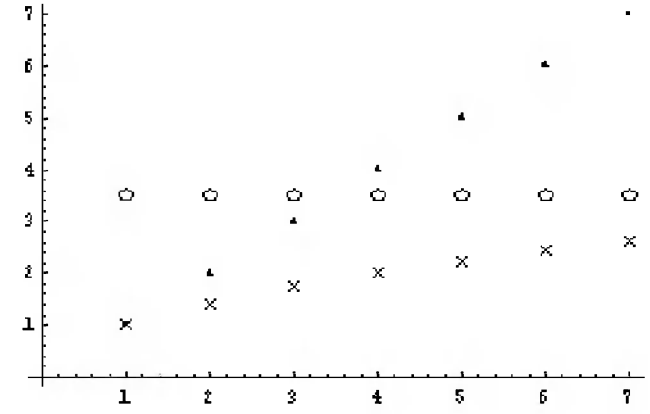
12.5.5. Построение графиков с примитивами

Следующие функции служат для вывода в качестве точек символов:

- **PlotSymbol[type]** – задает тип символа (type: Box, Diamond, Star или Triangle). Возможно применение опции Filled->False.
- **PlotSymbol[type,size]** – задает тип символа и его размер size.
- **MakeSymbol[ptimitives]** – задает вывод символа, создаваемого графическим примитивом.

Рисунок 12.22 показывает построение точек с применением *графических примитивов*. Этот путь позволяет обозначать точки графиков практически любыми фигурами.

```
MultipleListPlot[ Range[7], Table[3.5, {7}],
   Sqrt[Range[7]], SymbolShape ->
   {PlotSymbol[Triangle],
    MakeSymbol[RegularPolygon[5, 3]],
    MakeSymbol[{Line[{{2, 2}, {-2, -2}}],
       Line[{{-2, 2}, {2, -2}}]}]}]
```



- Graphics -

Рис. 12.22. Построение точек с применением графических примитивов

12.5.6. Построение графиков с примитивами – полигонами

Для создания примитивов в виде регулярных многоугольников (полигонов) задана директива:

- **RegularPolygon[n]** – регулярный n-угольный полигон.
- **RegularPolygon[n,rad]** – регулярный n-угольный полигон с заданным радиусом описанной окружности rad.
- **RegularPolygon[n,rad,ctr]** – то же с заданным центром ctr.
- **RegularPolygon[n,rad,ctr,tilt]** – то же с углом поворота фигуры на tilt градусов.
- **RegularPolygon[n,rad,ctr,titl,k]** – соединение линиями через k вершин.

Рисунок 12.23 показывает построение полигона – семиугольника с радиусом 3, центром в точке {0,0}, углом поворота 20 градусов и соединением через три вершины (попробуйте задать этот параметр другим целым числом и убедитесь, насколько меняется форма фигуры).

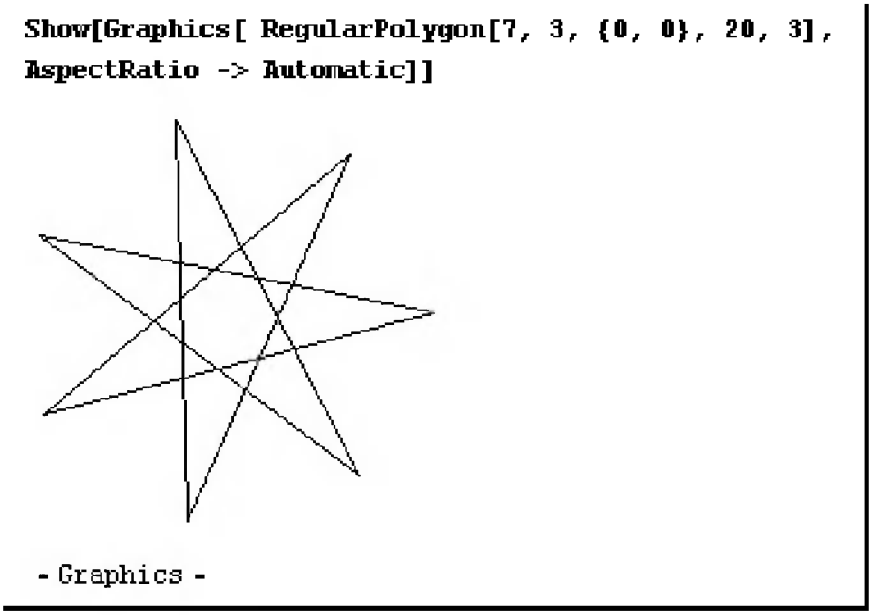


Рис. 12.23. Построение полигона

12.5.7. Задание спецификаций линий графиков

Директива

```
Dashing[{Dot,Dash,LongDash,...}],AbsoluteDashing[{Dot,...}]
```

служит для спецификации типа линий графиков.

Применение функций подпакета MultipleListPlot наиболее ценно при визуализации математических расчетов, где преобладают графики того типа, которые создаются этими функциями.

12.5.8. Построение трехмерных графиков функций, заданных параметрически

Трехмерные графики параметрически заданных функций относятся к числу наиболее сложных, но в то же время весьма эффектных. В подпакете ParametricPlot3D определены функции, упрощающие подготовку таких графиков:

- **ParametricPlot3D[{fx,fy,fz},{u,u0,u1,du},{v,c0,v1.dv}]** – строит 3D-поверхность, заданную параметрически функциями fx, fy и fz от переменных u и v с заданными диапазонами изменения и приращениями du и dv (рис. 12.24).
- **PointParametricPlot3D[{fx,fy,fz},{u,u0,u1,du}]** – строит точками 3D-поверхность, заданную параметрически функциями fx, fy и fz от одной переменной u с заданным диапазоном изменения и приращением du.

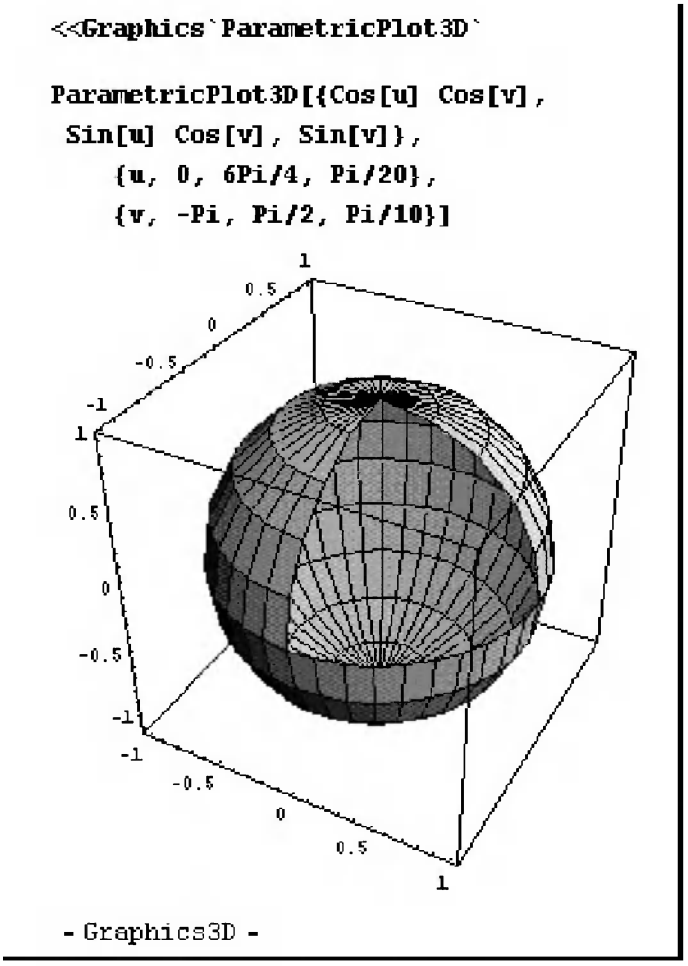


Рис. 12.24. Пример построения сферы с вырезом с помощью функции ***ParametricPlot3D***

- **PointParametricPlot3D[{fx,fy,fz},{u,u0,u1,du},{v,c0,v1.dv}]** – строит точками 3D-поверхность, заданную параметрически функциями fx, fy и fz от переменных u и x с заданными диапазонами изменения и приращениями du и dv.

Обратите внимание на то, что выбором диапазона изменения углов можно получить вырез сферы. Окраска поверхности осуществляется автоматически.

12.5.9. Трехмерные графики в сферической и цилиндрической системах координат

Для построения трехмерных поверхностей в сферической и цилиндрической системах координат служат функции:

- **SphericalPlot3D[r,{t,tmin,tmax},{p,pmin,pmax}]** – построение графика в сферической системе координат.
- **CylindricallPlot3D[z,{t,tmin,tmax},{p,pmin,pmax}]** – построение графика в цилиндрической системе координат.

На рис. 12.25 показано построение усеченной сверху сферы с помощью функции **SphericalPlot3D**. Нетрудно заметить, что применение данной функции дает самый простой способ построения сферы. Это естественно, поскольку система координат сферическая.

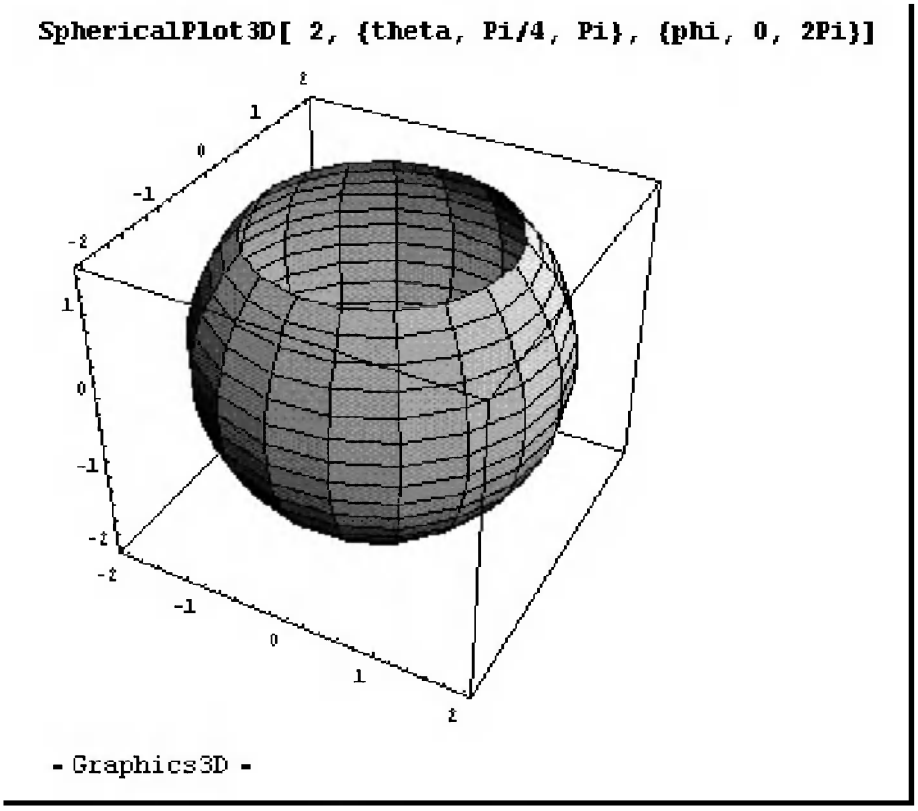


Рис. 12.25. Пример построения сферы с помощью функции SphericalPlot3D

12.5.9. Изменение точки обзора трехмерных графиков

С помощью опции ViewPoint можно изменять положение точки, с которой рассматривается фигура. Это существенно меняет ее вид – см. рис. 12.26.

Еще раз напоминаем, что интерфейс Mathematica предусматривает изменение точки просмотра уже построенной фигуры. Рекомендуется просмотреть список опций данных функций, позволяющих в широких пределах менять вид и стиль построения графиков.

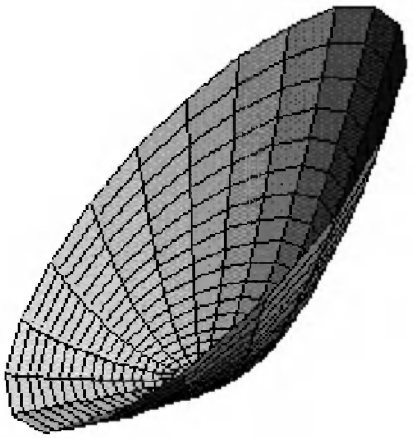
12.6. Построение графиков полей

12.6.1. Представление полей на плоскости

В подпакете PlotField имеются функции, позволяющие строить стрелками графики полей:

- **PlotVectorField[{fx,fy},{x,xmin,xmax},{y,ymin,ymax}]** – строит плоскость из векторов (стрелок), ограниченную пределами изменения x и y.
- **PlotGradientField[f,{x,xmin,xmax},{y,ymin,ymax}]** – строит плоскость из векторов (стрелок) градиента, ограниченной пределами изменения x и y.

```
CylindricalPlot3D[ (1 + Sin[phi]) r^2,
  {r, 0, 1}, {phi, 0, 2Pi},
    Boxed -> False, Axes -> False,
      ViewPoint ->  {1.5, -1, .2}]
```



```
- Graphics3D -
```

Рис. 12.26. Пример построения фигуры, видимой из заданной точки просмотра

- **PlotHamiltonianField[f,{x,xmin,xmax},{y,ymin,ymax}]** – строит плоскость полей Гамильтона, ограниченную пределами изменения x и y.
- **PlotPolyFueld[f,{x,xmin,xmax},{y,ymin,ymax}]** – представляет график комплексной функции f(x,y).

Рисунок 12.27 показывает применение функции **PlotVectorField** для построения графика градиента поля с помощью функции **PlotGradientField**.

Указанные функции имеют множество опций. Отметим основные из них:

- **ScaleFactor->Automatic** – устанавливает размер векторов (стрелок).
- **ScaleFunction->None** – устанавливает функцию, вычисляющую размер стрелок.
- **MaxArrowLenght->None** – устанавливает ограничение по длине стрелок.
- **ColorFunction->None** – задает функцию цвета.
- **PlotPoints->15** – задает число точек по координатам для построения стрелок.

Применение опций позволяет строить самые разнообразные графики различных полей – тепловых, гравитационных, электрических и др.

Есть еще одна функция, представляемая в двух формах:

- **ListPlotVectorField[{{vect11,vect12,...},{vect21,vect22,...},...}]** – строит график векторного поля прямоугольного массива векторов $\mathbf{(vect)}_{xy}$.
- **ListPlotVectorField[{{pt1,vect1,...},{pt2,vect2,...},...}]** – строит график векторного поля прямоугольного массива векторов $\mathbf{(vect)}_{xy}$.

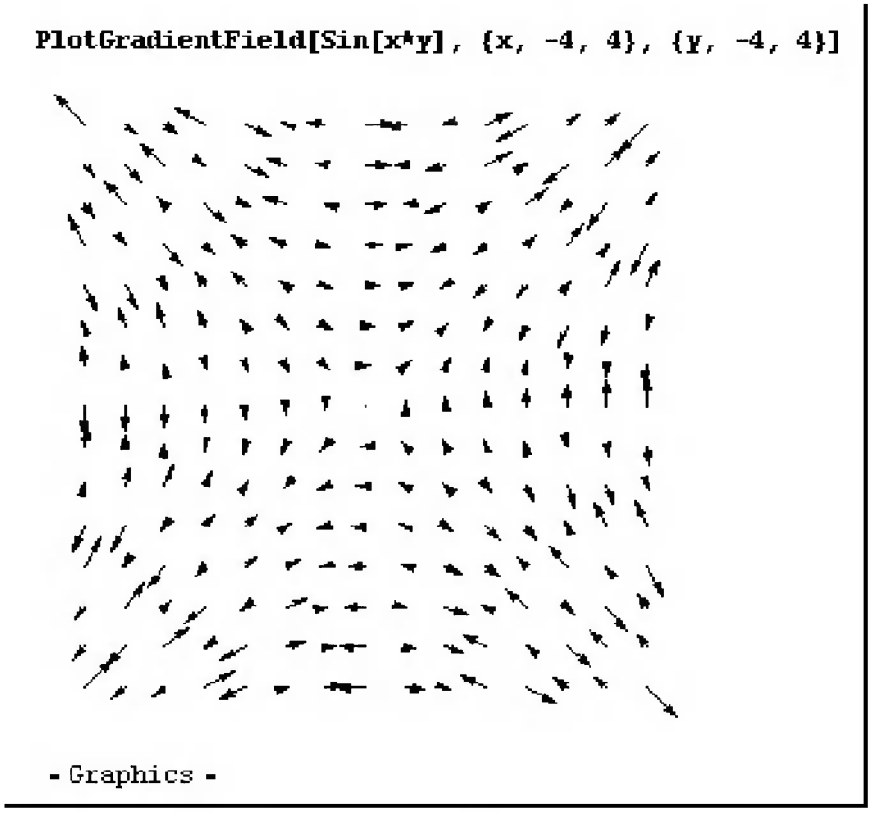


Рис. 12.27. График градиента поля

12.6.2. Представление полей в пространстве – PlotField3D

Для представления векторных полей в пространстве служат функции подпакета PlotField3D:

- **PlotVectorField3D[{fx,fy,fz},{x,xmin,xmax},{y,ymin,ymax},{z,zmin, zmax}]** – строит график векторного поля параметрически заданной 3D-фигуры.
- **PlotGradientField3D[{fx,fy,fz},{x,xmin,xmax},{y,ymin,ymax},{z,zmin, zmax}]** – строит график градиента векторного поля параметрически заданной 3D-фигуры.

Эти функции подобны описанным в предшествующем разделе, но используются для построения векторных полей не на плоскости, а в пространстве. Рисунок 12.28 показывает пример такого построения.

Имеется еще одна функция:

- **ListPlotVectorField3D[{{vect$_1$,pt$_1$,...},{vect$_2$,pt$_2$,...},...}]** – строит график векторного поля в пространстве по данным векторов **vect$_i$**.

При большом числе векторов в пространстве графики этого типа теряют наглядность. Рекомендуется тщательно отлаживать их, используя весь набор опций (как его получить, описывалось неоднократно).

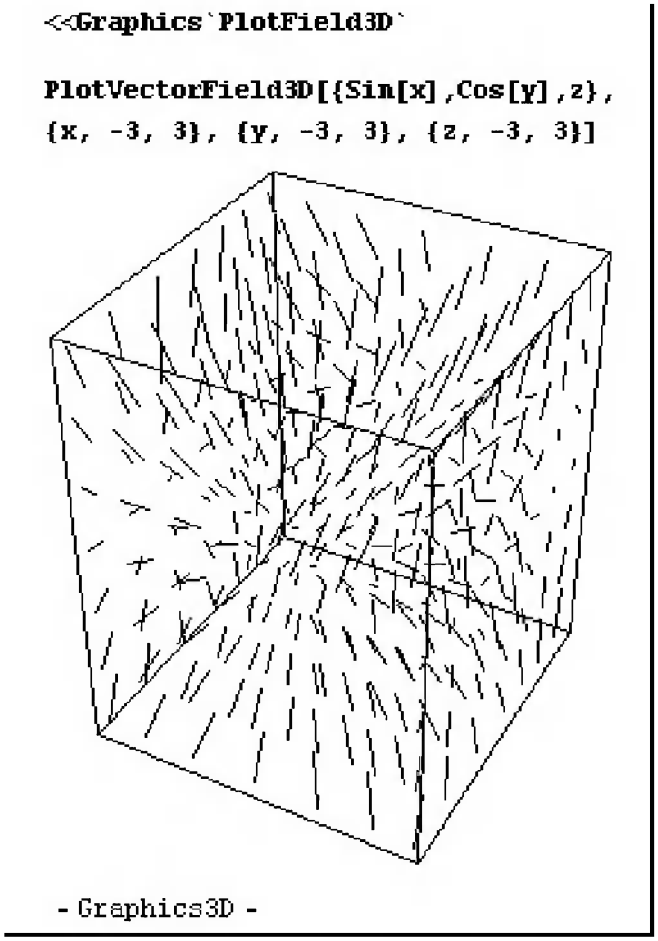


Рис. 12.28. Пример построения графика векторного поля в пространстве отрезками прямых

12.7. Построение пространственных фигур стереометрии

12.7.1. Построение полиэдров

Подпакет Polyhendra служит для создания регулярных пространственных фигур – полиэдров [88]. Они задаются как графические примитивы и выводятся функцией:

- **Show[Polyhedron[polyname]]** – строит полиэдр с именем polyname в центре графика.
- **Show[Polyhedron[polyname,{x,y,z},scale]]** – строит полиэдр с именем polyname с центром в точке {x,y,z} и параметром масштаба scale.

Возможно задание следующих имен полиэдров: Tetrahedron, Cube, Octahedron, Dodecahedron, Icosahedron, Hexahedron, GreatDodecahedron, SmallStellatedDodecahedron, GreateStellatedDodecahedron и GreatIcosahedron.

Возможность вывода с помощью функции Show двух полиэдров иллюстрирует рис. 12.29.

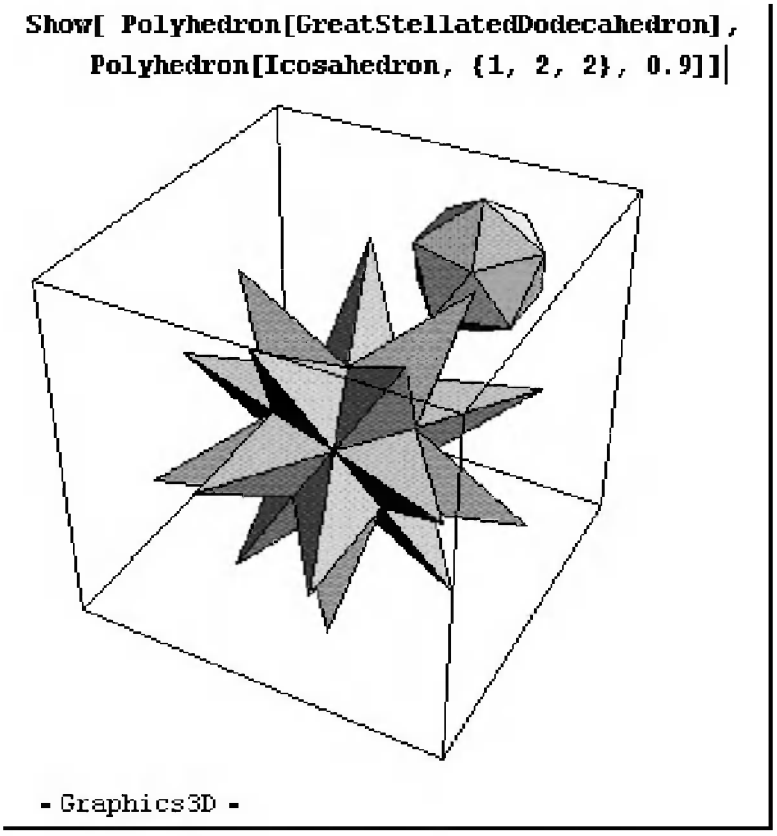


Рис. 12.29. Вывод функцией Show двух полиэдров

Для вывода полиэдров служит также ряд описанных ниже функций. Так, для построения звездообразных полиэдров предназначена функция:

- **Show[Stellate[Polyhedron[polyname]]** – построение звездообразных полиэдров.
- **Show[Stellate[Polyhedron[polyname], ratio]** – построение звездообразных полиэдров с заданным отношением описанной и вписанной сфер ratio.

Рисунок 12.30 показывает построение звездообразного (или игольчатого) полиэдра. Представленная фигура напоминает некоторых морских животных – ежей и звезд.

Полиэдры, применяемые в геодезии, можно получить с помощью следующей функции:

- **Show[Geodesate[Polyhedron[polyname], n]** – построение полиэдров, состоящих их регулярных n-угольных многоугольников, образующих сферу.
- **Show[Geodesate[Polyhedron[polyname], n,{x,y,z},radius]** – построение полиэдров, состоящих их регулярных n-угольных многоугольников, образующих сферу с заданным положением центра {x,y,z} и радиуса radius.

12.7.2. Построение усеченных полиэдров

Для построения усеченных полиэдров предназначены функции:

- **Show[Truncate[Polyhedron[polyname]]]** – построение усеченных полиэдров.
- **Show[Truncate[Polyhedron[polyname], ratio]** – построение усеченных полиэдров с заданным радиусом.

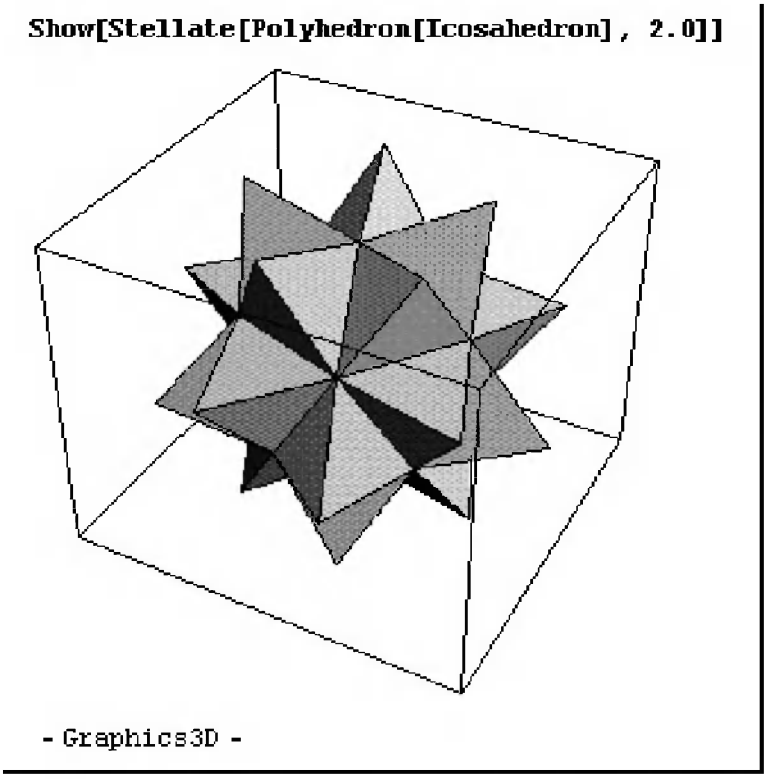


Рис. 12.30. Построение звездчатого полиэдра

- **Show[OpenTruncate[Polyhedron[polyname]]]** – построение полиэдров с открытым усечением.
- **Show[OpenTruncate[Polyhedron[polyname], ratio]** – построение полиэдров с открытым усечением и заданным отношением ratio (до 0.5).

Усечение может быть открытым – реализуется функцией со словом Open. В этом случае фигура выглядит так, будто она склеена из тонкого картона (рис. 12.31). При этом в местах усечения фигура прозрачна.

В заключение этого раздела отметим следующие функции:

- **First[Polyhedron[polyname]]** – возвращает список полигонов для указанного полиэдра.
- **Vertices[polyname]** – возвращает список координат вершин полиэдра.
- **Faces[polyname]** – возвращает список вершин, ассоциированных с каждой поверхностью.

Эти функции ничего не строят, а лишь возвращают специфические параметры полиэдров. Их можно использовать на занятиях стереометрии.

12.7.3. Создание графических форм

Нередко желательно придать трехмерным объектам определенную форму, например кольца или бублика. Некоторые возможности для этого дают функции подпакета Shapes. Основной из них является функция:

- **Show[Graphics3D[shape]]** – отображение формы со спецификацией shape.

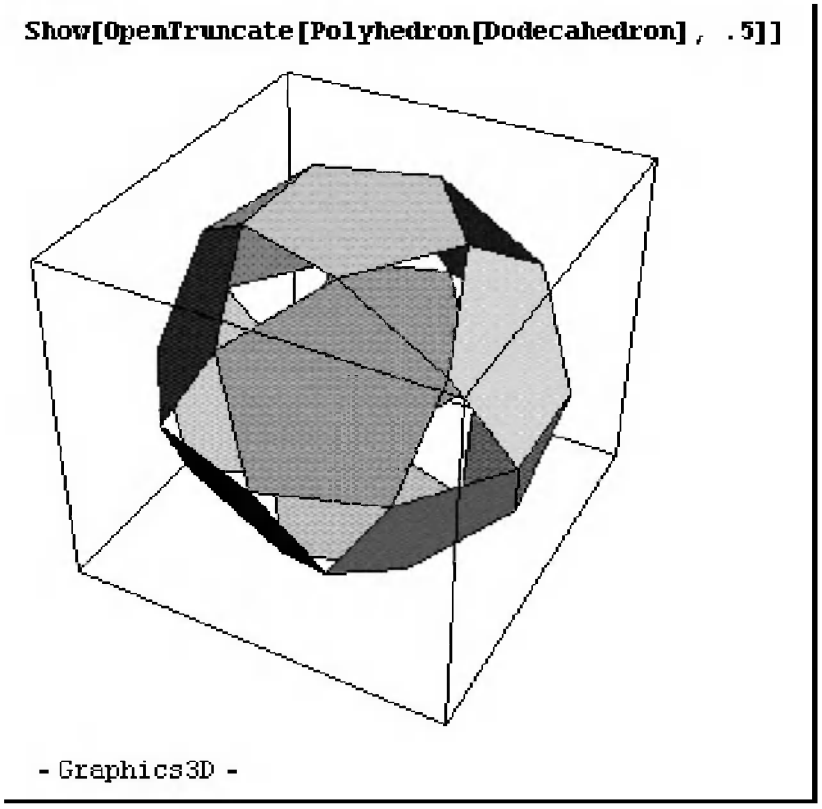


Рис. 12.31. Построение усеченного полиэдра с открытыми местами усечения

С ней могут использоваться графические примитивы:

- **Cone[r,h,n]** – конус с основанием радиуса r и высотой h на основе n-стороннего полигона в основании.
- **Cylinder[r,h,n]** – цилиндр радиуса r и высотой h на основе n-стороннего полигона.
- **Torus[r1,r2,n,m]** – объемное кольцо с внешним и внутренним радиусами r1 и r2 и числом сторон каркаса n и m.
- **Sphere[r,n,m]** – сфера радиуса r, составленная из многоугольников с параметрами n и m и числом сторон n(m-2)+2.
- **MoebiusStrip[r1,r2,n]** – кольцо Мебиуса с радиусами r1 и r2, построенное на основе полигона с 2n сторонами.
- **Helix[r,h,m,n]** – плоская спираль радиуса r и высоты h на основе поверхности, разбитой на n и m четырехугольников.
- **DoubleHelix[r,h,m,n]** – плоская двойная спираль радиуса r и высоты h на основе поверхности, разбитой на n и m четырехугольников.

Возможно указание фигур без параметров. Это означает, что они выбираются по умолчанию следующими:

Cone[1, 1, 20] **Cylinder[1, 1, 20]** **Helix[1, 0.5, 2, 20]**
DoubleHelix[1 ,0.5, 2, 20] **MoebiusStrip[1, 0.5, 20]** **Sphere[1, 20, 15]**
Torus[1, 0.5, 20, 10]

На рис. 12.32 показан пример построения фигуры **DoubleHelix** без указания ее параметров с помощью функций **Show** и **Graphics**.

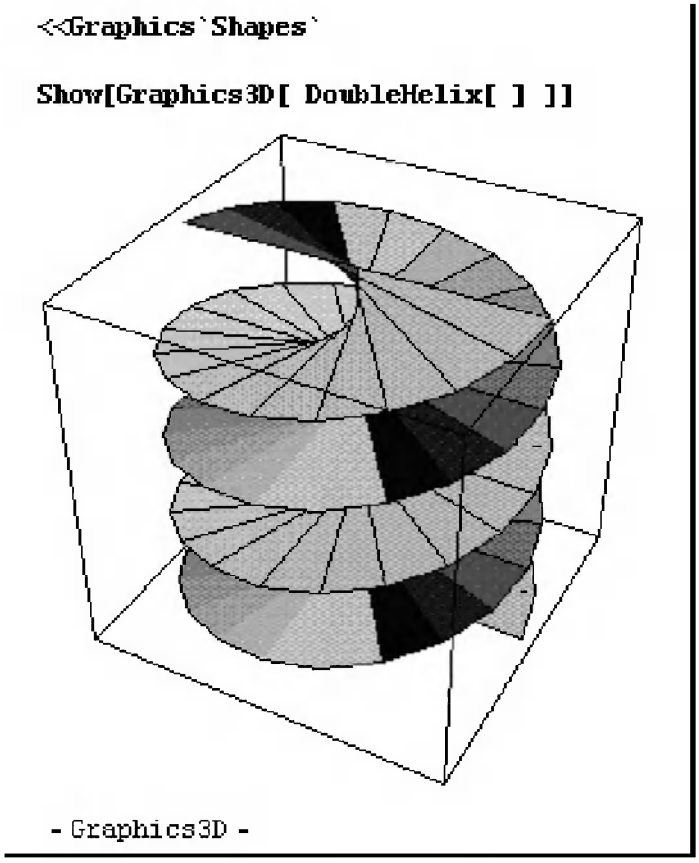


Рис. 12.32. Пример построения фигуры без указания параметров

12.7.4. Преобразования пространственных фигур

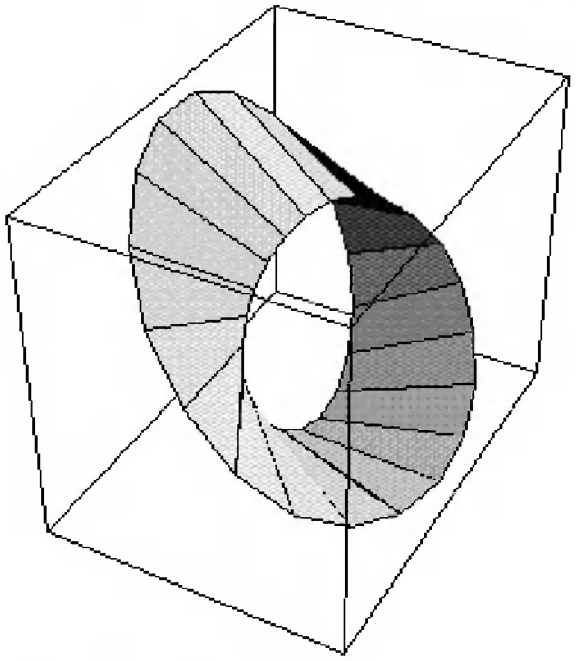
Имеются еще и следующие функции:

- **RotateShape[g,phi,theta,psi]** – поворот графического объекта на углы phi, theta и psi (рис. 12.33).
- **TranslateShape[g,{x,y,z}]** – преобразование графического объекта в представляющий его вектор.
- **AffineShape[g,{scale1,svale2,scale3}]** – умножение всех координат объекта g на указанные множители.

12.7.5. Построение фигур, пересекающихся в пространстве

Функции **Show** и **Graphics** позволяют строить трехмерные фигуры, которые пересекаются в пространстве. Пример такого построения приведен на рис. 12.34. Нетрудно заметить, что линии пересечения строятся с точностью до одной ячейки – полигона. Поэтому для получения качественных фигур надо увеличивать число полигонов, из которых фигуры синтезируются. Это, однако, увеличивает время построения фигур.

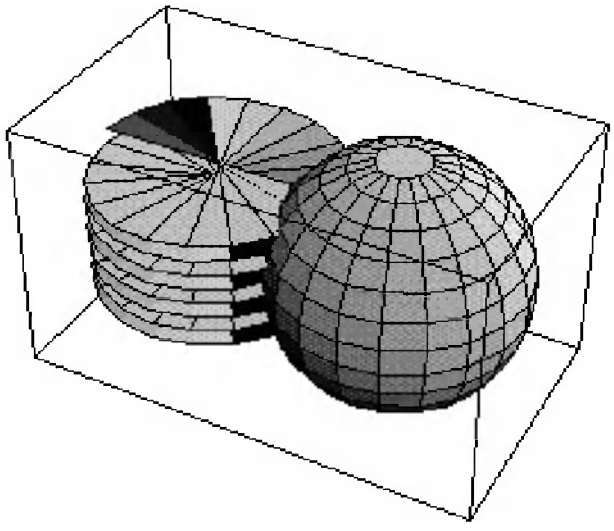
```
Show[RotateShape[ Graphics3D[MoebiusStrip[ ] ],
                  Pi/4, Pi/3, Pi/2]]
```



```
- Graphics3D -
```

Рис. 12.33. Построение повернутого кольца Мебиуса с заданными углами поворота

```
Show[TranslateShape[ Graphics3D[
Sphere[1, 20, 15] ], {1.5, 0, 0}],
Graphics3D[ DoubleHelix[1, 0.5, 4, 20] ]]
```



```
- Graphics3D -
```

Рис. 12.34. Построение двух фигур, пересекающихся в пространстве

12.8. Другие возможности пакета

12.8.1. Применение сплайнов

Подпакет Spline вместе с уже описанным подпакетом SplineFit (сплайновая регрессия) обеспечивает представление данных с помощью сплайна. В подпакете Spline определена единственная функция:

- **Spline[points,type]** – создает графический примирив, представляющий сплайн-кривую типа type (Cubic, Bezier или CompoziteBezier – см. описание подпакета NumericalMath'SplineFit').

Среди ее опций важно отметить следующие (значения, как и ранее, даны по умолчанию): **SplineDots->None**, **SplinePoinrs->25**, **MaxBemd->10.0** и **SplineDivision->20.0**.

Рисунок 12.35 показывает возможность построения сплайн-функции вместе с точками, через которые она проходит.

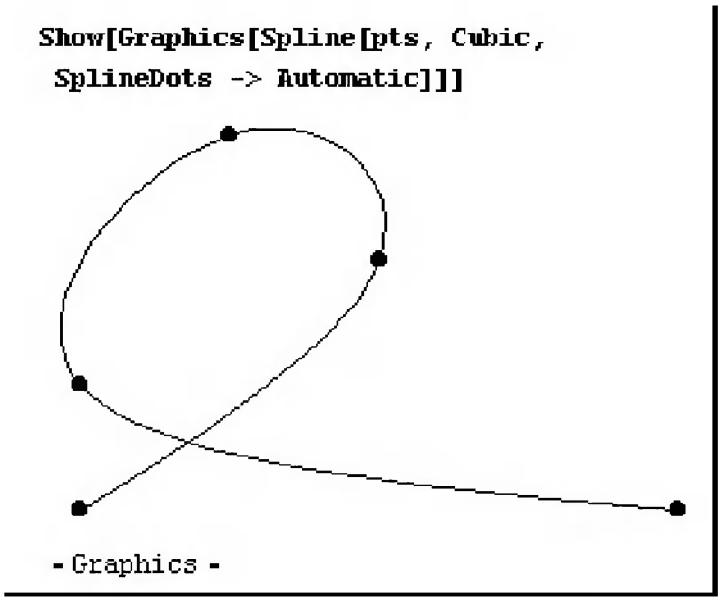


Рис. 12.35. Построение исходных точек и проходящей через них сплайн-функции

12.8.2. Фигуры вращения

Одна из задач компьютерной графики – создание поверхностей вращения. Средства для этого дает подпакет SurfaceOfRevolution. Они представлены следующими функциями:

- **SurfaceOfRevolution[f,{x,xmin,xmax}]** – строит поверхность, образованную вращением кривой, описанной функцией f при изменении x от xmin и xmax, в плоскости x-z .
- **SurfaceOfRevolution[{f_x,f_y},{t,tmin,tmax}]** – строит поверхность, образованную вращением кривой, описываемой параметрически заданной на

плоскости функцией $\{f_x,f_y\}$, в плоскости x-z при изменении параметра t от tmin и tmax.

- **SurfaceOfRevolution[{f_x,f_y,f_z},{t,tmin,tmax}]** – строит поверхность, образованную вращением кривой, описываемой параметрически заданной в пространстве функцией **{f_x,f_y,f_z}**, в плоскости x-z между **xmin** и **xmax**.
- **SurfaceOfRevolution[f,{{x,xmin,xmax},{theta,thetamin,thetamax}}]** – строит поверхность вращения кривой, описываемой функцией f, при угле theta, меняющемся от **thetamin** и **thetamax**.

Рисунок 12.36 дает пример построения поверхности, образованной линией cos(x) при изменении x от 0 до 2π, вращающейся вокруг оси вращения x–z. Построение задано функцией **SurfaceOfRevolution[f,{x,xmin,xmax}]**. В этом случае линия вращается в пределах угла от 0 до 2π, поэтому поверхность получается круговой. Опция ViewVertical задает угол обзора фигуры.

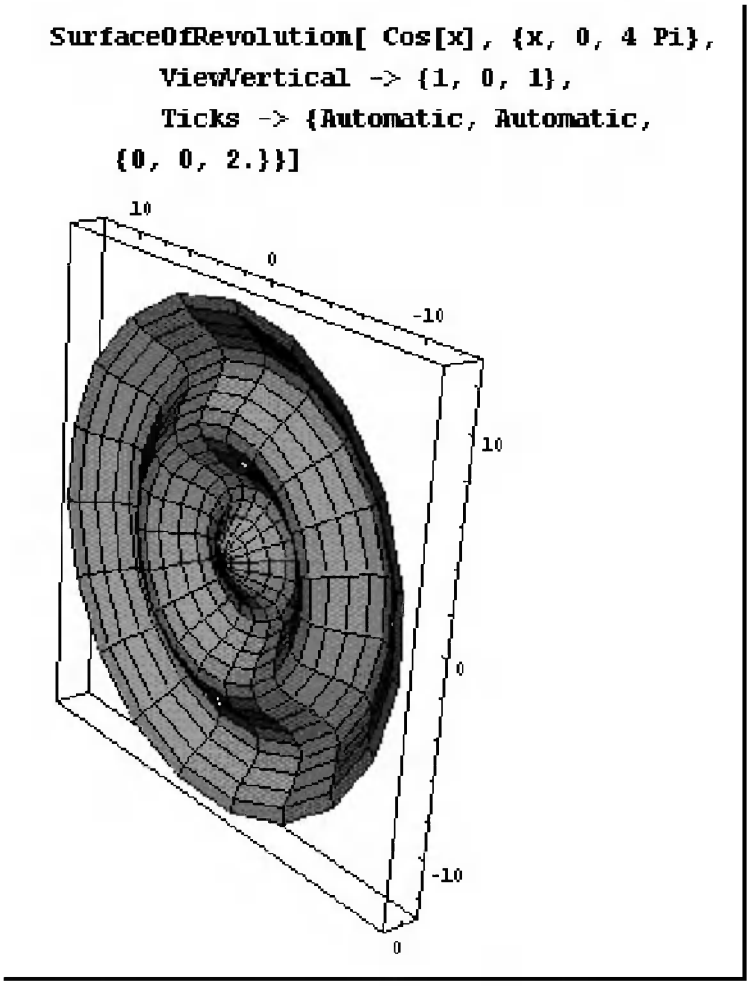


Рис. 12.36. Фигура, образованная вращением линии cos(x)

Пример на применение функции

SurfaceOfRevolution[{f_x,f_y},{t,tmin,tmax}]

представлен на рис. 12.37. Формируется этакое декоративное яйцо на подставке. Заменив в определении функции **Cos[u]** на **Sin[u]**, можно получить изображение рюмки. Рисунок 12.37 демонстрирует также возможность построения объемной

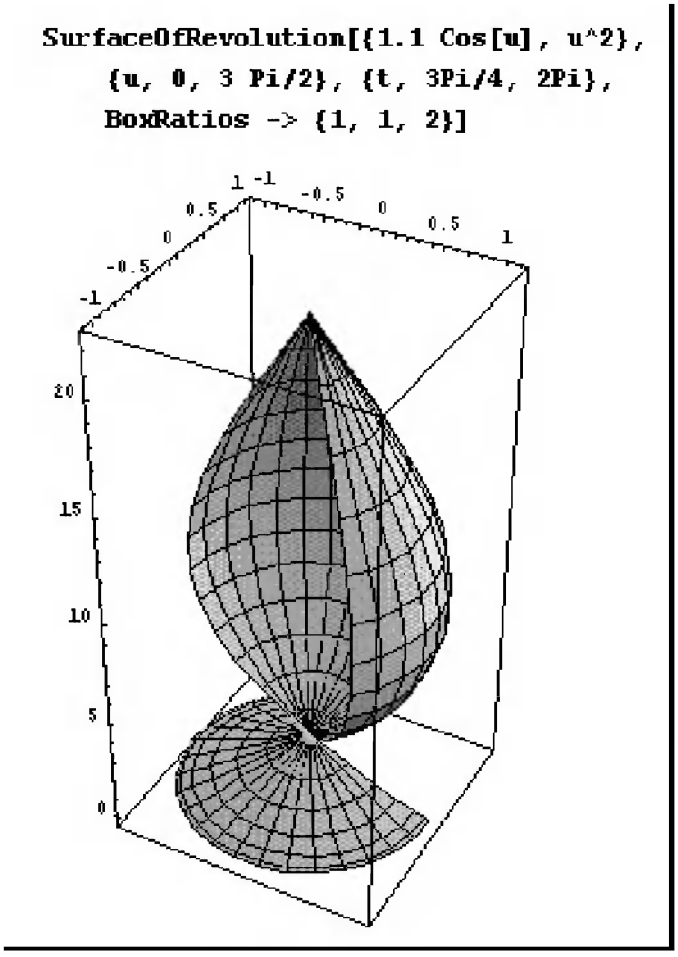


Рис. 12.37. Построение декоративного яйца с вырезом на подставке

фигуры с вырезами. Все, что для этого надо, – удачно выбрать диапазон изменения угла вращения. Если он будет от 0 до 2π, то фигура окажется сплошной и не будет содержать вырезов.

Для поворота фигуры вращения служат опции:

- **RevolutionAxes->{x,y}** – задает поворот в плоскости x–z.
- **RevolutionAxes->{x,y,z}** – задает поворот в пространстве.

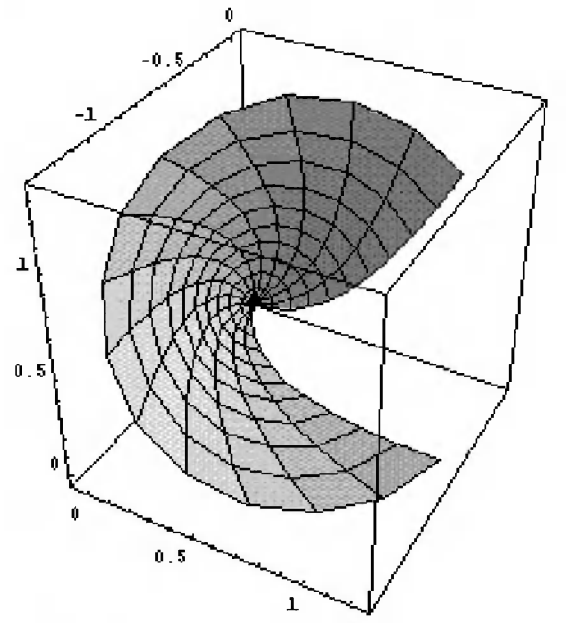
Следующая функция позволяет построить фигуру вращения, образующая линия которой задается массивом точек:

- **ListSurfaceOfRevolution[{point1,point2,...}]** – создает поверхность вращения, заданную массивом точек point1, point2,....
- **ListSurfaceOfRevolution[{point1,point2,...},{theta,thetamin,thetamax}]** – создает поверхность вращения, заданную массивом точек и углом вращения theta от thetamin до thetamax.

Рисунок 12.38 показывает задание массива точек с помощью функции Table и фигуру вращения, полученную при использовании функции **ListSurfaceOfRevolution** во второй форме.

```
dat = Table[{n, n^2}, {n, 0, 1, .1}];

ListSurfaceOfRevolution[dat, {t, 0, 12Pi/8},
RevolutionAxis -> {1, -1, 1}, PlotRange -> All]
```




```
- Graphics3D -
```

Рис. 12.38. Пример построения фигуры вращения с образующей, заданной массивом точек

12.8.3. Что еще в пакете расширения Graphics?

Помимо уже рассмотренных подпакетов, пакет расширения Graphics содержит подпакеты TreeScript и Common. Подпакет содержит функцию преобразования графических объектов в программный код, например

```
<<Graphics`ThreeScript`
obj = Graphics3D[Polygon[0,0,0, 0,1,0, 0,1,1]]
 …Graphics3D…
ThreeScript[ "object.ts", obj]
 object.ts
!!object.ts
% Graphics3D objects
boundingbox
  0 0 0
  0 1 1
viewpoint
  1.3 -2.399999999999999 2.
```

```
ambientlight
  0 0 0
lightsources
  1. 0. 1.
  1  0  0
  1. 1. 1.
  0  1  0
  0. 1. 1.
  0  0  1
polygon
  0  0  0
  0  1  0
  0  1  1
```

Подпакет Common содержит просто перечень системных символов (точнее, слов), которые приняты во всех подпакетах пакета Graphics. Вот этот список: Horizontal, MaxArrowLength, Scale Factor, ScaleFunction и Vertical.

Глава 13

Визуализация в других системах

Другие системы также имеют достаточно развитые средства для графической визуализации вычислений. В этой главе описана уникальная графика самой массовой СКМ Mathcad и современных СКА начального уровня Derive 5/6 и MuPAD 2.5/3.

13.1. Построение двумерных графиков в Mathcad

13.1.1. Вставка шаблона графика

Графики в системе Mathcad являются такими же объектами, как константы, переменные, формулы и т. д. Графики вставляются в документ с помощью *шаблонов*. В позиции **Graph** подменю вставок **Insert** (или в панели графических шаблонов – рис. 13.1) имеется список следующих шаблонов графики:

- **X-Y Plot** (Декартов график);
- **Polar Plot** (Полярный график);
- **3D Plot Wizard** (Мастер трехмерной графики);
- **Surface Plot** (График поверхности);
- **Contour Plot** (Контурный график);
- **3D Scatter Plot** (Точечный график);
- **3D Bar Plot** (Трехмерная гистограмма);
- **Vector Field Plot** (Векторное поле).

Рассмотрим эти виды графиков более подробно. Это рассмотрение относится к любой версии системы Mathcad – от Mathcad 12 и выше.

13.1.2. Вставка шаблона двумерного графика

Графики в системе Mathcad могут имеют окна и изменяемые размеры и перемещаться в окне редактирования документа. Для вывода шаблона двумерной графики в декартовой системе координат служит команда **X-Y Plot** (Декартов график) или клавиша ввода символа `@`. Они выводят в текущем положении курсора шаблон двумерного графика.

Незаполненный шаблон 2D-графика представляет собой большой пустой прямоугольник с шаблонами и местами ввода данных в виде темных маленьких прямоугольников, расположенных около осей абсцисс и ординат будущего графика, – такой шаблон показан на рис. 13.1 в левом верхнем углу окна документа. В них необходимо ввести выражения, задающие координаты точек графика по осям X (горизонтальная ось) и Y (вертикальная ось). В общем случае это могут быть функции некоторой переменной `x`.

Начиная с Mathcad 12 появилась возможность построения 2D-графиков с двумя осями Y. Для построения шаблона такого графика достаточно построить

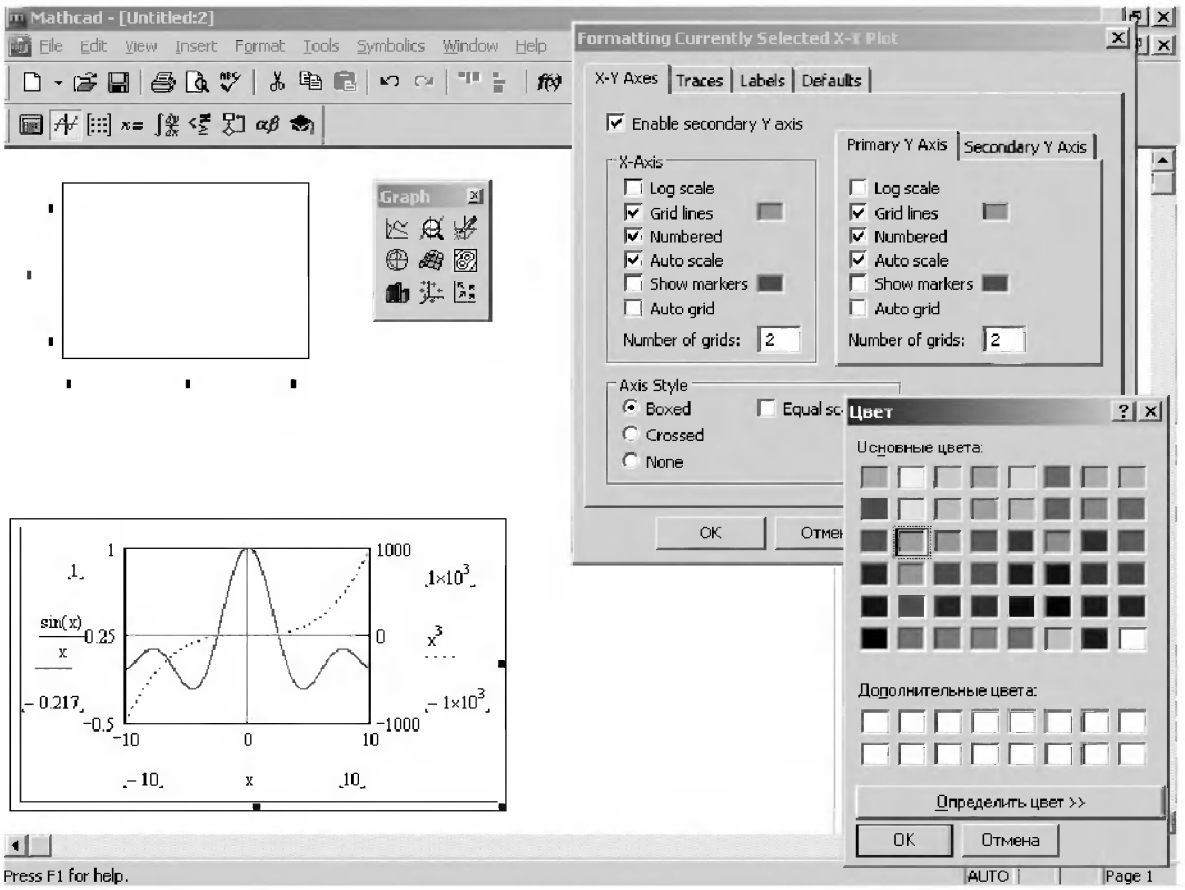


Рис. 13.1. Окно системы Mathcad 12 со средствами построения двумерных графиков

обычный шаблон двумерного графика, поместить в него маркер мыши и дважды щелкнуть левой клавишей мыши. Появится окно форматирования двумерного графика, показанное в окне документа рис. 13.1 справа с открытой вкладкой **XY Axis**.

На этой вкладке в Mathcad 12 есть три приятные мелочи, отсутствующие в предшествующих реализациях Mathcad, – опция **Enable Secondary Y axis** (установка двойных Y-осей) и два прямоугольника с цветным фоном для установки цветов масштабной сетки **Grids** и показа маркеров **Show Markers**. При активизации прямоугольников мышью появляется панель смены цветов, видная на рис. 13.1 под окном форматирования двумерного графика. Кроме того, если задействована опция установки двух осей Y, то на вкладке появляются двойная внутренняя вкладка для установки опций для графиков каждой оси. График двух функций с двумя осями Y показан на рис. 13.1 в левом нижнем углу окна документа. Шаблон обычного графика с одной осью Y показан над этим графиком.

Если строятся графики нескольких функций в одном шаблоне, то функции следует разделять запятыми – см. график на рис. 13.1 для двух функций (sin(x)/x и x^3). Для выделения и вставки нужных данных в шаблоне удобно использовать клавиши перемещения курсора. Можно также выделить данные и места ввода в шаблоне с помощью мыши. В Mathcad 14 можно менять формат чисел у осей графиков.

Если график уже построен, то при его выделении появляются крайние места ввода с автоматически введенными числами, которые служат для указания предельных значений абсцисс и ординат, то есть задают масштаб графика. В принципе, можно не заполнять эти места ввода самостоятельно, но тогда масштаб, скорее всего, окажется не вполне удобным, например будет представлен неокругленными десятичными числами, обеспечивающими максимальный размер графика. Тем не менее рекомендуется всегда вначале использовать автоматическое масштабирование и лишь затем выбирать более подходящий масштаб.

Для наиболее распространенных графиков в декартовой системе координат Mathcad предусматривает два способа построения графиков функций одной переменной `f(x)`:

- упрощенный способ без задания ранжированной переменной `x` (пределы изменения `x` автоматически задаются от –10 до 10) и вводом выражения (функции) в сам график – рис. 13.1;
- обычный способ с заданием ранжированной переменной `x` (рис. 13.2).

Начинающие пользователи обычно задают ранжированную переменную `x` целочисленной, например `x:=-15..15`. При этом они забывают, что в данном случае график задается небольшим числом точных целочисленных значений `x`: –15, –14, –13, ..., –1, 0, 1, 2, 3, ..., 14, 15. В некоторых случаях (рис. 13.2 сверху) это ведет к грубому искажению формы графиков.

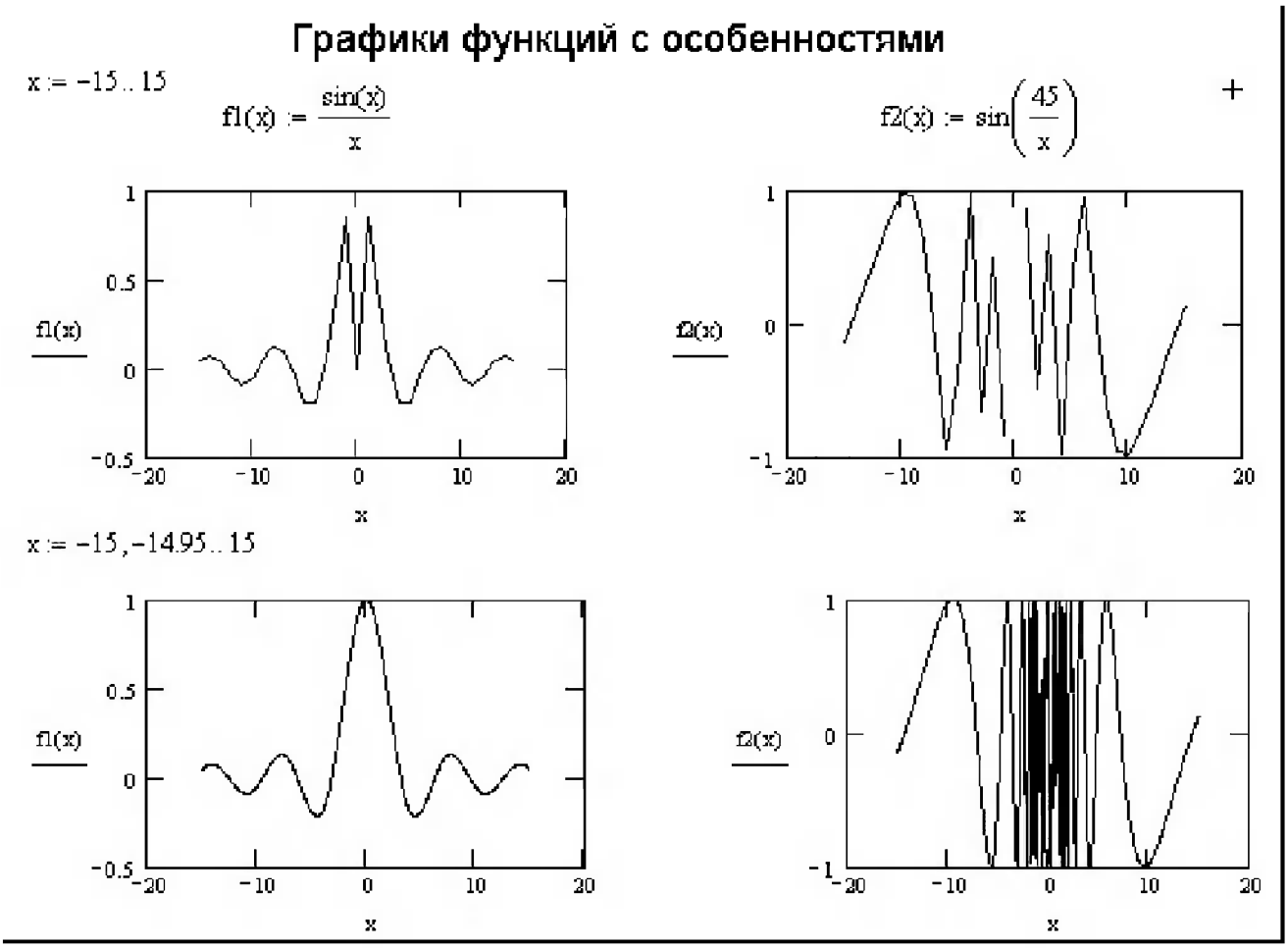


Рис. 13.2. Построение графиков функций обычным способом

В левой части рисунка построены графики функции sin(x)/x, имеющей устранимую особенность типа 0/0→1 в точке x = 0. Однако система Mathcad ведет себя в данном случае как типичная система для численных расчетов – вне зависимости от значения знаменателя дроби она считает значение дроби равным нулю, если числитель равен нулю. При указанном задании переменной x в точке x = 0 получаем значение функции f1(x) = 0, что и дает провал первого графика в точке x = 0. Интересно отметить, что Mathcad 11/12 введена новая функция sinc(z) =sin(*z*)/*z*, которая учитывает устранимую особенность в точке z=0 и может использоваться с комплексным аргументом.

Графики функции sin(45/x), показанные в правой части рисунка, при x→0 дают резкое повышение частоты синусоиды. При недостаточно малом шаге изменения x, равном 1, особенность этой функции практически незаметна.

Простейшим способом избавиться от указанных недостатков графиков является их построение при более мелком шаге изменения x. Например, если задать x как x:=-15, -14.95..15, шаг изменения x будет равен 0.05. Указание в числе -14.95 разделительной точки означает переход от целочисленного представления x к представлению в виде принципиально приближенных вещественных чисел с плавающей точкой. В результате числитель выражения sin(x)/x хоть немного, но отличается от нуля в точке примерного равенства x нулю. Результат налицо – нижние графики на рисунке уже дают хорошее представление о форме и особенностях функций.

13.1.3. Простейшие приемы форматирования двумерных графиков

Чтобы построить график в автоматическом режиме вычислений, достаточно вывести указатель мыши за пределы графического объекта и щелкнуть левой кнопкой. В «ручном» режиме вычислений для этого нужно еще нажать клавишу **F9**.

Если что-либо в построенном графике не вполне удовлетворяет пользователя, можно применить команды изменения формата графиков. Эти команды позволяют изменять заданные по умолчанию параметры графиков. Заметим, что окно задания форматов графиков появляется, если дважды щелкнуть мышью на графике либо щелкнуть один раз, если график выделен. Это окно было показано на рис. 13.1 при открытой вкладке **XY-Axes**. На ней можно установить тип и осей и установить идентичность масштаба по осям (опция **Equal Scales**). Ряд установок на этой вкладке был описан выше.

Графики можно перемещать по полю окна документа и изменять в размерах. Для этого надо выделить график, щелкнув на нем мышью. Можно поступить и по-иному – поместить указатель мыши вблизи графика и, нажав левую кнопку мыши, перемещать указатель в направлении графика, захватывая его рамкой выделения в виде прямоугольника из черных пунктирных линий. Как только график окажется внутри пунктирного прямоугольника, надо отпустить кнопку мыши. График будет выделен. Теперь можно растягивать или сжимать графики или перемещать их по экрану мышью с нажатой левой клавишей.

Если при выделенном рисунке нажать клавишу **F3**, рисунок будет перенесен в буфер обмена. Переместив курсор в новое место и нажав клавишу **F4**, можно вставить рисунок на новое место. Кроме того, ряд команд форматирования графиков имеется в контекстном меню. Оно появляется при щелчке на графике правой кнопкой мыши.

Полезно отметить, что, помимо команд для операций с буфером обмена, в контекстном меню имеются команды трассировки графиков **Trace** (Трассировка), изменения масштаба выделенной части графика **Zoom** (Масштаб) и вывода графиков или текстовых надписей поверх изображения **Bring to Front** (На передний план) или под ним **Send to Back** (На задний план). Это позволяет создавать сложные графики с поясняющими надписями, которые невозможно задать при использовании обычных команд форматирования. Эти возможности мы проиллюстрируем по мере описания графических средств системы Mathcad.

13.1.4. Графики с параметрическим заданием функций

Допускается строить двумерные графики с параметрическим заданием функций по осям координат. При этом в местах ввода могут стоять произвольные функции одной переменной x. На рис. 13.3 показаны 4 графика, наглядно иллюстрирую-

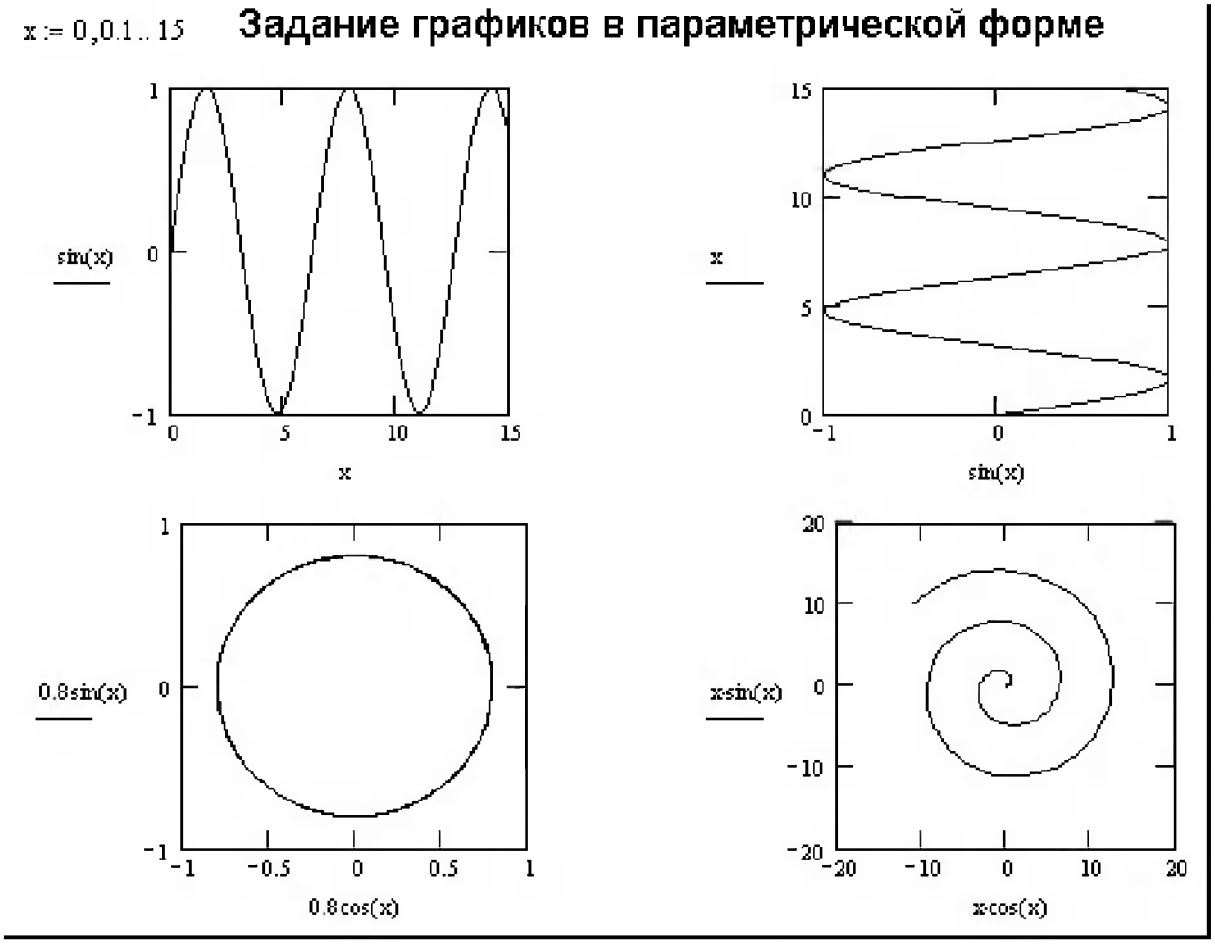


Рис. 13.3. Задание графиков в параметрической форме

щие, что, в сущности, двумерная графика – это графика с параметрическим заданием функций по осям координат.

Два первых графика на рисунке иллюстрируют эффект от перемены функций, заданных по осям X и Y. В результате функция `sin(x)` оказывается повернутой на угол 90°. На третьем графике показано построение параметрически заданной окружности по формулам `R·cos(x)` и `R·sin(x)`, где `R = 0.8` – радиус окружности. На последнем графике показано построение спирали по формулам `x·cos(x)` и `x·sin(x)`.

13.1.5. Построение графиков ряда функций на одном рисунке

Как уже отмечалось, допускается построение графиков ряда функций с перечислением их через запятую в месте ввода шаблона по оси Y. Однако допускается задание различных функций не только по оси Y, но и по оси X. Все это означает возможность построения нескольких графиков разного типа на одном рисунке, что и иллюстрирует рис. 13.4.

Параметрически заданная функция на рисунке образует так называемую фигуру Лиссажу, которую наблюдают на экране осциллографа, подавая на отклоняю-

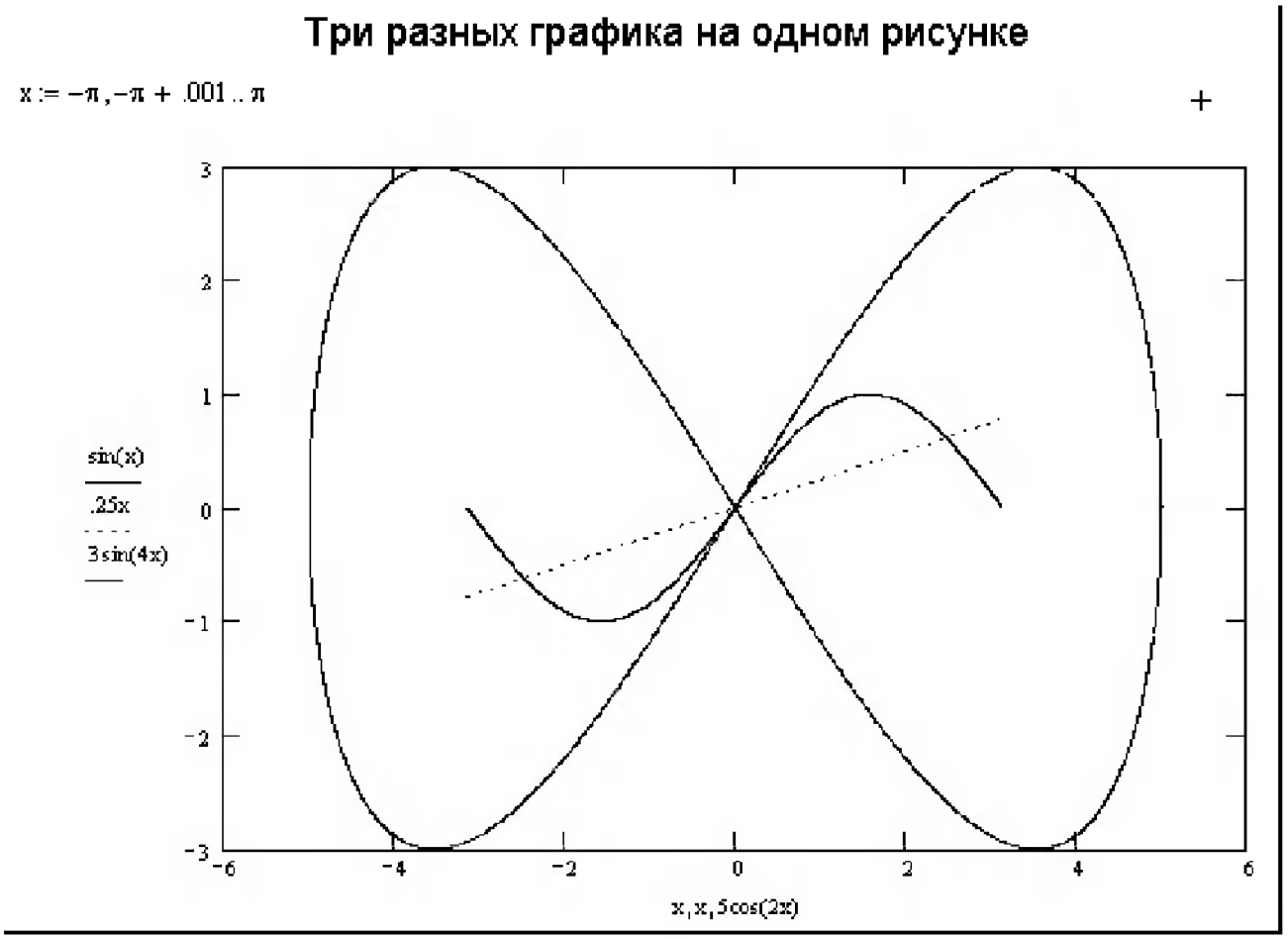


Рис. 13.4. Графики трех функций на одном рисунке – одна из них задана параметрически

щие пластины X и Y синусоидальные сигналы с кратными частотами (в нашем случае кратность равна 2/4 и фигура напоминает знак бесконечности).

13.1.6. Полулогарифмические и логарифмические графики

В ряде случаев графики в линейном масштабе, установленном по умолчанию, бывают недостаточно представительными и не выявляют математических закономерностей построения той или иной функции. Например, график логарифмической функции в линейном масштабе имеет резко нелинейный вид с крутым участком при малых значениях x и довольно плоским при больших значениях x (рис. 13.5).

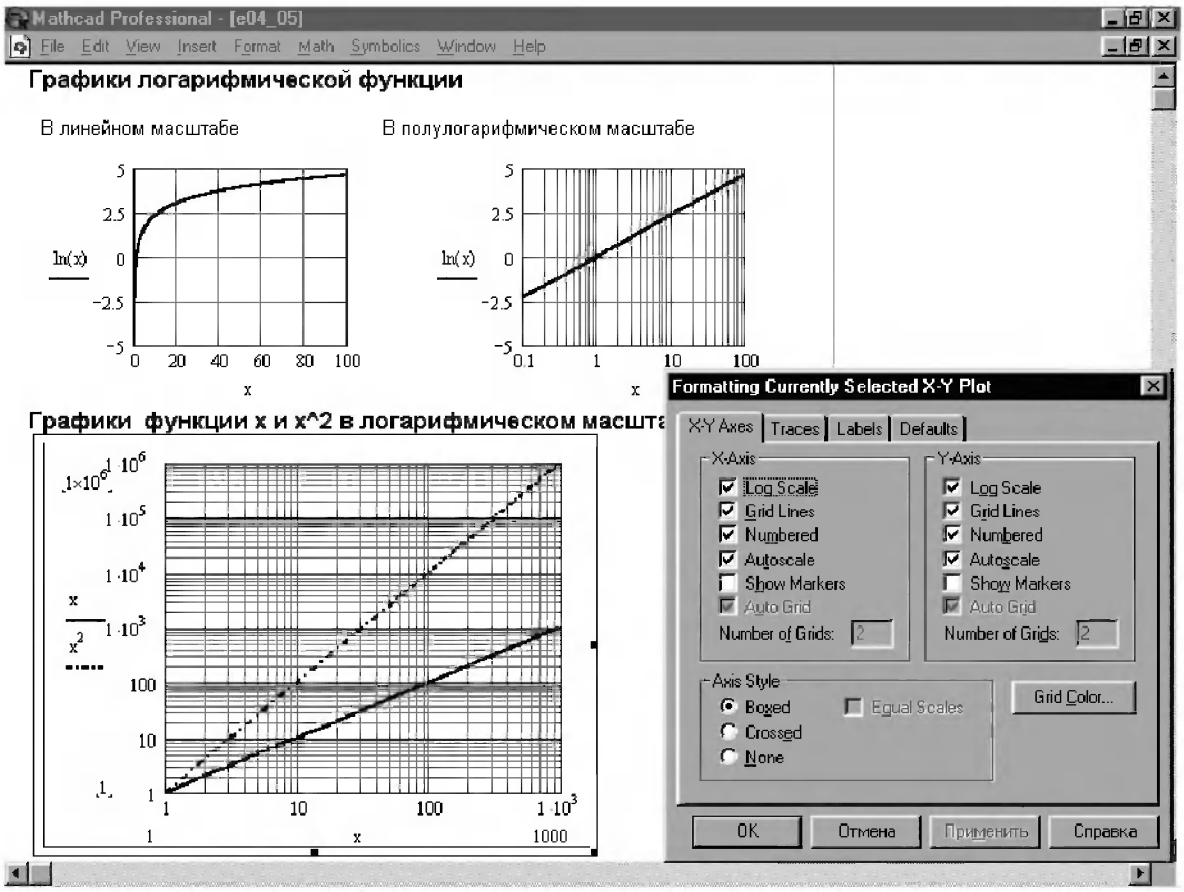


Рис. 13.5. Графики с разным масштабом

При построении полулогарифмического графика (масштаб по оси y – линейный, а по оси x – логарифмический) график логарифмической функции превращается в прямую. На рисунке снизу представлены также графики функций x и x^2 в логарифмическом масштабе по обеим осям – иногда его называют двойным логарифмическим масштабом. Обратите внимание на то, что оба графика имеют вид прямых, но с разными углами наклона.

Графики в логарифмическом и полулогарифмическом масштабах строятся как обычные графики. Однако их надо форматировать с помощью окна форматирова-

ния. На рисунке это окно показано в правом нижнем углу окна документа. Для его вызова достаточно установить указатель мыши на графике и дважды щелкнуть левой кнопкой мыши. На вкладке **X-Y Axes** (Оси X–Y) окна форматирования двумерных графиков нужно установить флажок **Log Scale** (Логарифмический масштаб).

13.1.7. Построение графиков в полярной системе координат

В полярной системе координат каждая точка задается углом `W` и длиной его радиус-вектора `R(W)`. График функции обычно строится при изменении угла `W` в определенных пределах, чаще всего от 0 до 2π. Выбор команды **Polar Plot** (Полярный график) в подменю **Graph** (График) меню **Insert** (Вставка) или нажатие комбинации клавиш **Ctrl+7** выводит шаблон таких графиков, показанный на рис. 13.6. Этот шаблон имеет форму окружности и содержит места ввода данных. На рисунке также показаны заполненный шаблон графика и окно форматирования графика с открытой вкладкой Traces, в которой можно установить тип, цвет и толщину линии графика (или ряда линий).

Перед построением таких графиков надо задать пределы изменения ранжированной переменной `W` (она может иметь и другое имя). После вывода шаблона следует ввести `W` в место ввода снизу и функцию `R(W)` в место ввода слева, а также

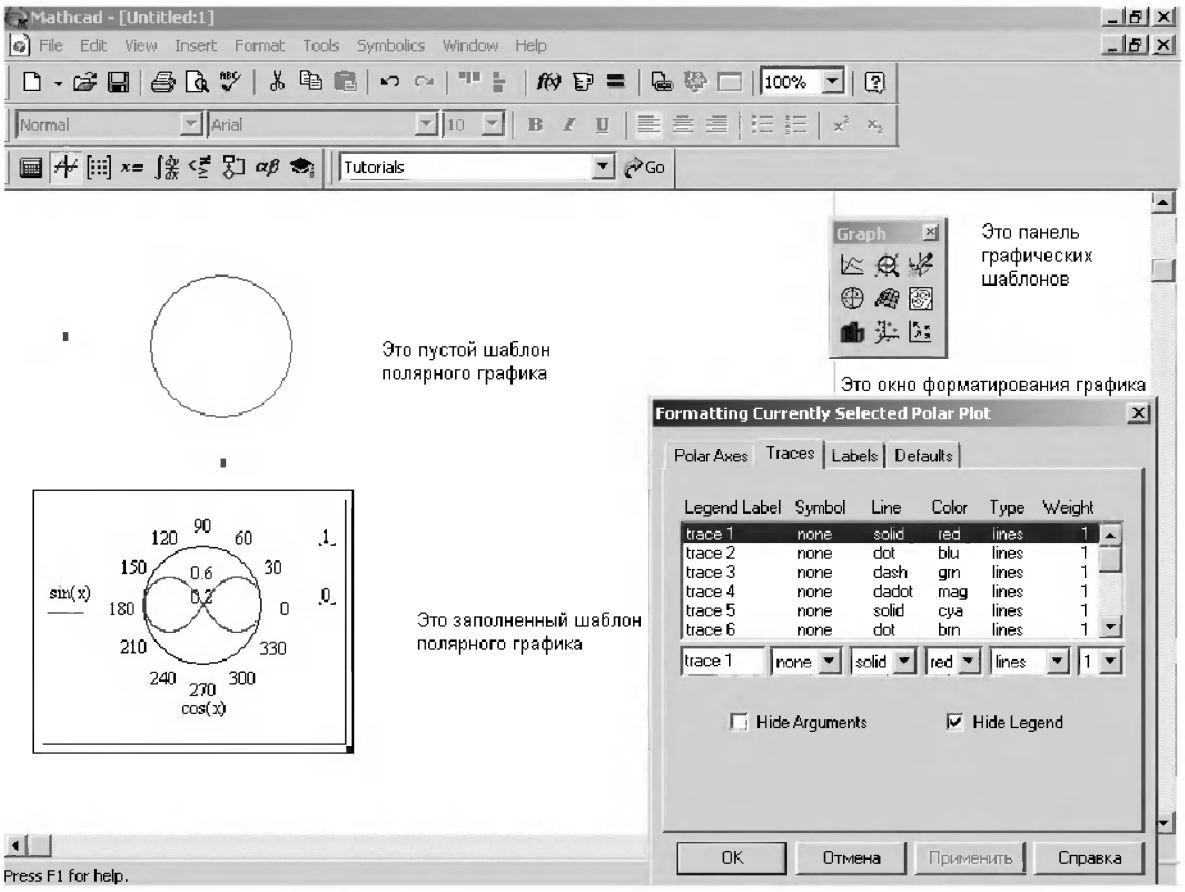


Рис. 13.6. Шаблон полярного графика

указать нижний предел изменения длины радиус-вектора `R(W)` – `Rmin` – в месте ввода справа снизу и верхний предел – `Rmax` – в месте ввода справа сверху. Эти места ввода становятся видимыми при выделении графика.

На рис. 13.7 показано построение ряда графиков в полярной системе координат – от одиночной окружности радиуса 0.8 до трех кривых на одном графике.

Рис. 13.7. Примеры построения графиков функций в полярной системе координат

Надо также отметить возможность прямого построения графиков функций в полярной системе координат без определения диапазона изменения независимой переменной `x` (в этом случае в качестве независимой переменной надо указывать именно `x`, а не `W`).

При прямом построении графика достаточно просто заполнить место ввода функции. Саму функцию надо описать ее уравнениями, которые вписываются в соответствующие места ввода. Можно также задать построение графиков нескольких функций в одном шаблоне (рис. 13.8). Стоит щелкнуть мышью вне области графика, как последний будет построен.

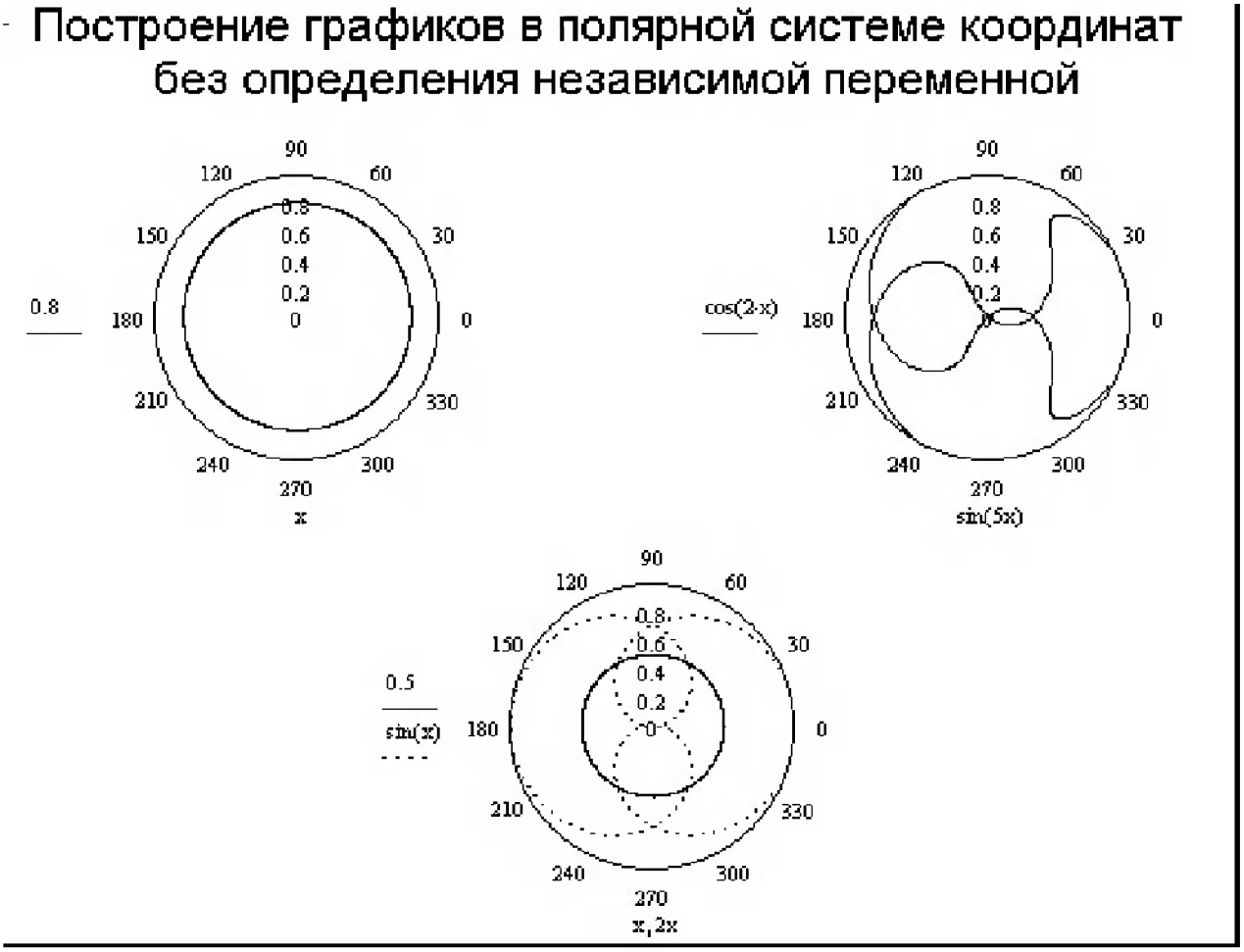


Рис. 13.8. Пример прямого построения графиков функций в полярной системе координат без определения независимой переменной

13.2. Построение трехмерных графиков в Mathcad

Трехмерная графика системы Mathcad при правильном ее применении способна поразить воображение многих пользователей. Она дает прекрасное представление о сложных поверхностях и объемных фигурах, расположенных в пространстве. Особенно впечатляет трехмерная графика с функциональной окраской поверхностей и фигур и с построением ряда пересекающихся объектов в пространстве. Вид графика в Mathcad, в отличие от других СКМ, задается из окна форматирования, что отвечает идее визуально-ориентированного программирования графических операций. Форматы запоминаются вместе с документом, в состав которых входят графики.

13.2.1. Построение поверхностей по матрице аппликат их точек

До появления Mathcad 2000 при построении графика поверхности, представленной функцией z(x,y) двух переменных, приходилось предварительно определять матрицу M аппликат (высот z) ее точек. Разумеется, этот способ возможен и в последующих версиях системы Mathcad.

Поскольку элементы матрицы M – индексированные переменные с целочисленными индексами, то перед созданием матрицы требуется задать индексы в виде ранжированных переменных с целочисленными значениями, а затем уже из них формировать сетку значений x и y – координат для аппликат z(x,y). Значения x и y при этом обычно должны быть вещественными числами, нередко как положительными, так и отрицательными.

Все это приводит к усложнению алгоритма подготовки данных для построения трехмерного графика поверхности, особенно большие трудности возникают у школьников и студентов. Они не всегда понимают, зачем все эти сложности и как задать область изменения x и y, чтобы была построена нужная часть поверхности.

После выполнения указанных выше определений командой **Surface Plot** (График поверхности) подменю **Graph** (График) меню **Insert** (Вставка) вводится шаблон графика, левый верхний угол которого помещается в место расположения курсора. Шаблон, в свою очередь, содержит единственное место ввода – темный прямоугольник у левого нижнего угла основного шаблона. В него надо занести имя матрицы аппликат поверхности. После этого надо установить указатель мыши в стороне от графического блока и щелкнуть левой кнопкой. На рис. 13.9 показан пример такого построения.

В данном случае построена поверхность в виде «проволочного каркаса» со всеми видимыми линиями.

Наглядность представления поверхностей и трехмерных фигур зависит от множества факторов: масштаба построений, углов поворота фигуры относительно осей, применения алгоритма удаления невидимых линий или отказа от него, использования функциональной окраски и т. д. Для изменения этих параметров следует отформатировать график с помощью окна форматирования (см. рис. 1.66). Для этого, в частности, можно использовать контекстное меню, также показанное на рис. 13.9.

Фрагмент документа Mathcad, показанный на рис. 13.10, иллюстрирует применение алгоритма удаления невидимых линий и задание функциональной окраски поверхности для описанного выше графика.

Нетрудно заметить, что применение алгоритма удаления невидимых линий делает рисунок поверхности намного более наглядным. Дальнейшее повышение наглядности восприятия трехмерных графиков обеспечивается применением функциональной окраски. По существу, она дает дополнительную информацию о третьем измерении, в нашем случае чем ниже расположена поверхность фигуры, тем она темнее.

Рис. 13.9. Задание и построение поверхности без удаления невидимых линий

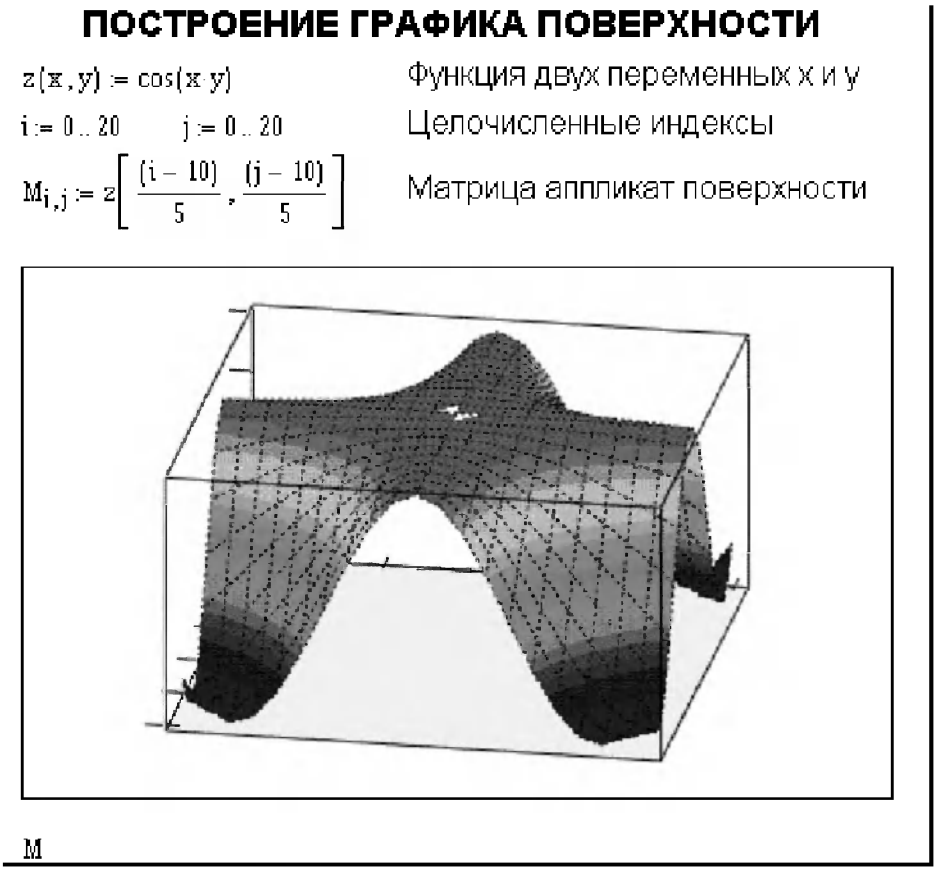


Рис. 13.10. Задание и построение поверхности с удалением невидимых линий и использованием функциональной окраски

С помощью команд изменения формата можно получить множество разновидностей трехмерной графики. В частности, возможен вывод координатных осей, «параллелепипеда», обрамляющего фигуру, и иных деталей подобных графиков, например титульных надписей.

Поскольку график строится на основе матрицы, содержащей только координаты высот фигуры, то истинные масштабы по осям X и Y неизвестны и на рисунках не проставляются. Возможно, впрочем, выводить порядковые номера элементов матриц в заданном направлении (по X и по Y). Необходимо следить за тем, как сформировать векторы `X` и `Y`, чтобы фигура выглядела естественно и была видна нужная часть фигуры в пространстве. Все это несколько затрудняет быстрое создание графиков трехмерных поверхностей нужного вида.

13.2.2. Построение параметрически заданных поверхностей

Большие возможности дает несколько иной способ задания поверхностей – в параметрическом виде. При этом приходится формировать три матрицы `X`, `Y` и `Z` и указывать их в шаблоне в виде `(X,Y,Z)`. Скобки необходимы, поскольку в противном случае Mathcad попытается построить три поверхности по данным матриц `X`, `Y` и `Z`.

На рис. 13.11 показаны построенные таким способом сферы: одна – при параметрах форматирования, заданных по умолчанию, другая (см. главу 5) – после простого форматирования путем введения обрамляющего параллелепипеда, применения алгоритма удаления невидимых линий и использования функциональной окраски, зависящей от значений координаты `x`.

Подобный способ построения пространственных фигур открывает новые возможности для наглядной визуализации трехмерных объектов различной формы.

13.2.3. Построение трехмерных фигур с вырезом

Параметрическая форма задания трехмерных фигур открывает еще одну возможность – представление объемных фигур с вырезом. Такие фигуры отличаются повышенной наглядностью, ибо в вырезе видна внутренняя структура фигур. Все, что надо для такого построения, – ограничить диапазон изменения параметрических углов, сделав его меньше обычного значения 2π. Этот прием иллюстрирует рис. 13.12.

Рекомендуется внимательно сравнить рис. 13.11 и рис. 13.12. Вся разница (в аналитических выражениях) заключается в уменьшении диапазона изменения индекса `j` – на рис. 13.12 он меняется от `0` до `N-4`, а на рис. 13.11 – от `0` до `N`.

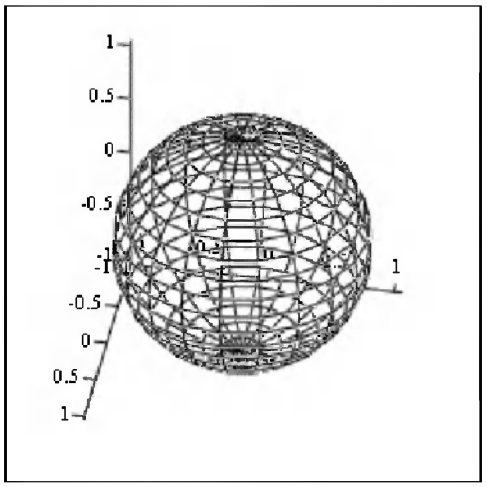
ПОСТРОЕНИЕ СФЕРЫ

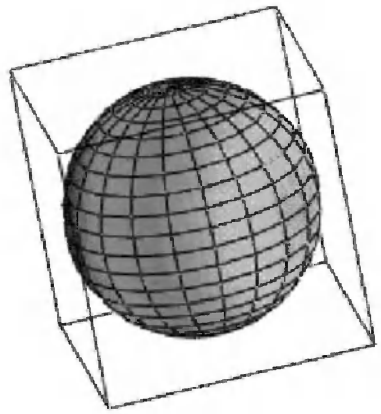
$N := 20$ Число вертикальных линий деления

$i := 0 .. N \qquad \phi_i := i \cdot \frac{\pi}{N} \qquad j := 0 .. N \qquad \theta_j := j \cdot 2 \cdot \frac{\pi}{N}$

$X_{i,j} := \sin(\phi_i) \cdot \cos(\theta_j) \qquad Y_{i,j} := \sin(\phi_i) \cdot \sin(\theta_j) \qquad Z_{i,j} := \cos(\phi_i)$

Графическое изображение сферы





(X, Y, Z) (X, Y, Z)

Рис. 13.11. Построение сферы, заданной параметрически

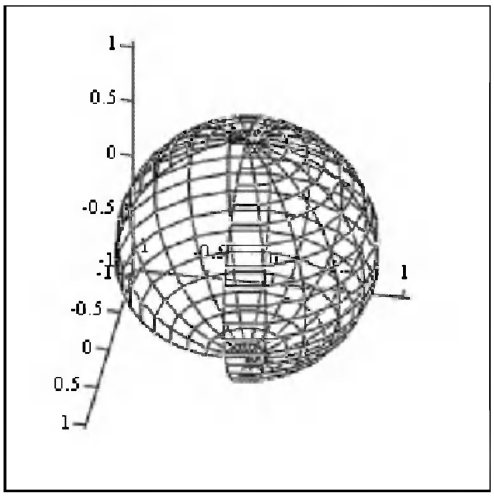
ПОСТРОЕНИЕ СФЕРЫ

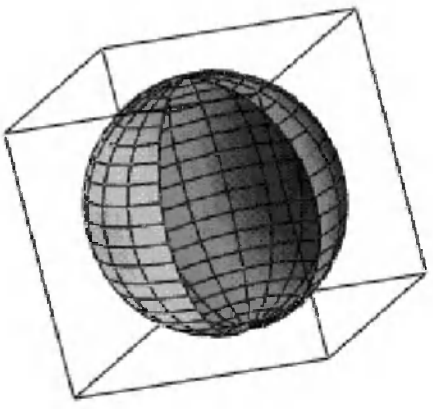
$N := 20$ Число вертикальных линий деления

$i := 0 .. N \qquad \phi_i := i \cdot \frac{\pi}{N} \qquad j := 0 .. N - 4 \qquad \theta_j := j \cdot 2 \cdot \frac{\pi}{N}$

$X_{i,j} := \sin(\phi_i) \cdot \cos(\theta_j) \qquad Y_{i,j} := \sin(\phi_i) \cdot \sin(\theta_j) \qquad Z_{i,j} := \cos(\phi_i)$

Графическое изображение сферы





(X, Y, Z) (X, Y, Z)

Рис. 13.12. Построение сферы с вырезом

13.2.4. Построение трехмерных графиков без задания матрицы

Mathcad 2001i/11/12/13/14 обладает принципиально новой возможностью – допускается построение трехмерных графиков без задания матрицы аппликат поверхностей. В результате построение графиков поверхностей выполняется столь же просто, как и построение двумерных графиков. Фрагмент документа Mathcad, показанный на рис. 13.13, иллюстрирует эту возможность.

Рис. 13.13. Построение графика поверхности в системе Mathcad 2001 без задания матрицы

В данном случае для построения оказалось достаточным задать функцию двух переменных z(x,y) := cos(x·y) и указать ее имя z в месте ввода шаблона графики. Единственным недостатком такого упрощенного метода построения поверхностей является неопределенность в масштабировании, поэтому для получения приемлемого вида графиков требуется форматирование. Впрочем, в любом случае получение графика в достаточно эффектном виде всегда требует его форматирования.

13.2.5. Построение графика поверхности, заданной в векторной параметрической форме

Описанный выше новый метод быстрого построения поверхности может иметь множество вариантов. Один из них – задание поверхности в векторной параметрической форме. Пример такого построения иллюстрирует рис. 13.14. Строится фигура, напоминающая бублик (тор).

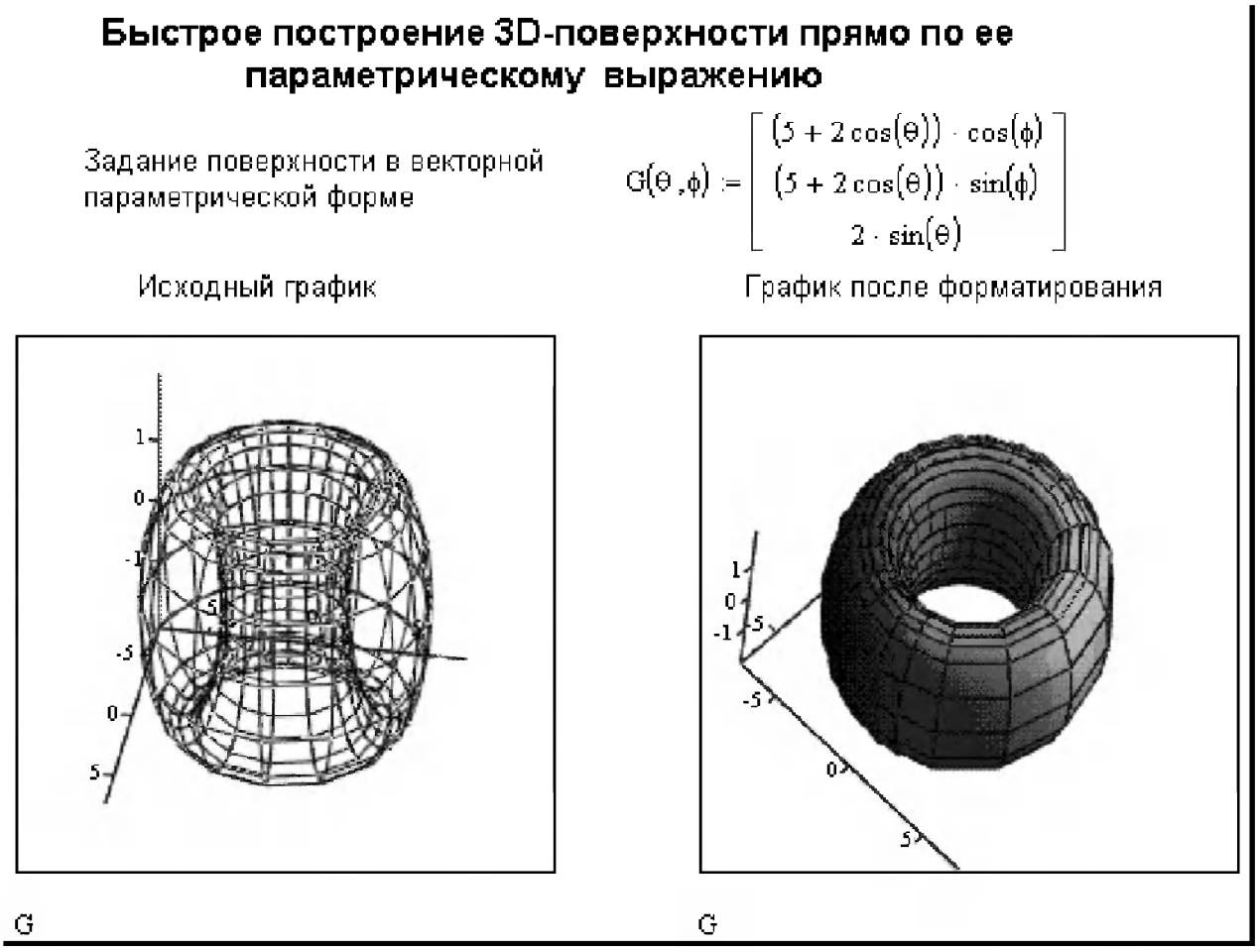


Рис. 13.14. Построение графика поверхности при задании ее в векторной параметрической форме

Обратите внимание на особую наглядность задания поверхности в такой форме с помощью единственной формулы и простоту построения графика. В данном случае для создания исходного графика (на рисунке слева) не требуется никаких промежуточных операций. Вид графика (на рисунке справа) можно улучшить путем форматирования и поворота графика мышью.

13.2.6. Применение новой функции CreateMesh

Mathcad 2001i/11 поддерживает новую графическую функцию для задания поверхностей:

```
CreateMesh(F,s0,s1,t0,t1,sgrid,tgrid,fmap)
```

Эта функция возвращает массив из трех матриц, представляющих координаты `x`, `y` и `z` для функции `F`, определенной в векторной параметрической форме в качестве функции двух параметров `sgrid` и `tgrid`. Аргументы `s0`, `s1`, `t0` и `t1` определяют пределы изменения переменных `sgrid` и `tgrid`. Аргумент `fmap` – трехэлементный вектор значений, задающих число линий в сетке изображаемой функции. Создаваемый функцией `CreateMesh` массив можно использовать для ввода в шаблон трехмерной графики поверхности. В качестве примера рассмотрим построение поверхности, заданной функцией `sin(u·v)`.

1. Задайте функцию двух переменных следующим образом:

   ```
   H(u,v):=sin(u·v)
   ```

2. Задайте пределы изменения переменных:

   ```
   u0 := -2
   u1 := 2
   v0 := -2
   v1 := 2
   ```

3. Используя функцию `CreateMesh`, создайте матрицу аппликат поверхности:

   ```
   C := CreateMesh(H,u0,u1,v0,v1)
   ```

4. Выберите команду **Insert** (Вставка) ⇒ **Graph** (График) ⇒ **Surface Plot** (График поверхности).
5. В единственное место ввода появившегося шаблона трехмерного графика введите имя матрицы аппликат поверхности, созданной на шаге 3.
6. Щелкните в документе вне области графика. Поверхность будет построена.
7. При необходимости отформатируйте график.

Построение поверхности с применением функции `CreateMesh` иллюстрирует фрагмент документа Mathcad, показанный на рис. 13.15.

Рисунок иллюстрирует технологию применения этой функции. Построение слева дано при форматировании по умолчанию, а справа – после ввода функциональной окраски и поворота фигуры мышью.

13.2.7. Построение объемной фигуры, образованной вращением кривой

Еще один полезный пример применения функции `CreateMesh` – построение объемной фигуры, которая получается вращением кривой, заданной функцией `f(x)`, вокруг оси X или Y (рис. 13.16).

Применение графической функции CreateMesh

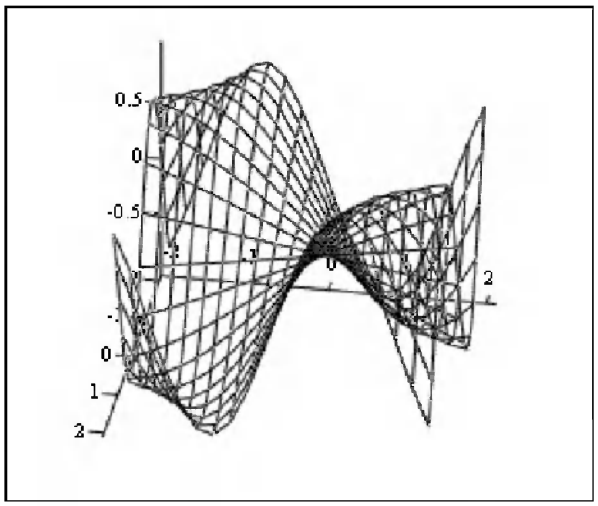
1. Зададим функцию двух переменных $H(u,v) := \sin(u \cdot v)$

2. Зададим пределы изменения переменных $u0 := -2$ $u1 := 2$ $v0 := -2$ $v1 := 2$

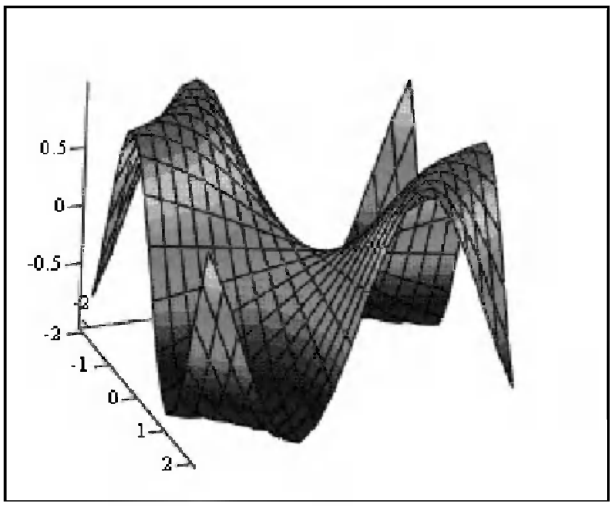
3. Используя функцию **CreateMesh** создадим матрицу поверхности

$C := \text{CreateMesh}(H, u0, u1, v0, v1)$

4. Задав построение графика типа **Surface Plot** получим:



C



C

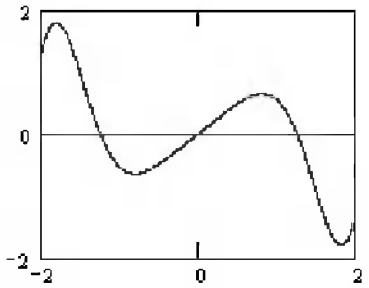
Рис. 13.15. Пример построения графика поверхности с применением функции CreateMesh

ПОСТРОЕНИЕ ФИГУРЫ, ПОЛУЧЕННОЙ ВРАЩЕНИЕМ КРИВОЙ ВОКРУГ ОСИ X

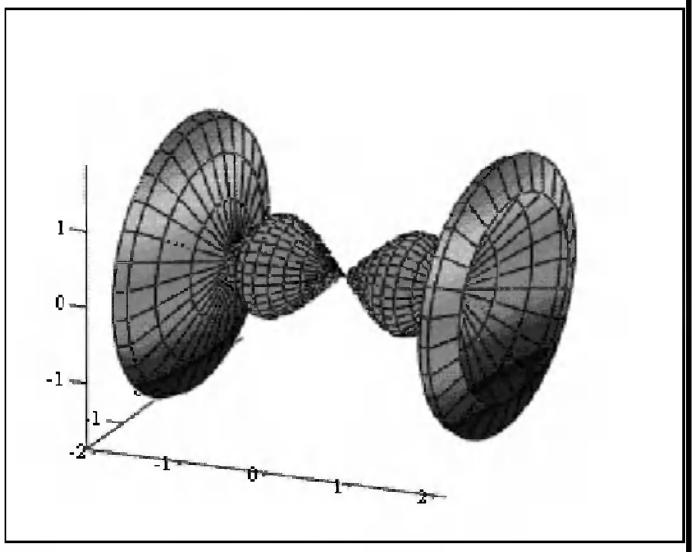
$f(x) := x \cdot \cos(x^2)$ $a := -2$ $b := 2$ $mesh := 30$

$F(u,v) := u$ $G(u,v) := f(u) \cdot \cos(v)$ $H(u,v) := f(u) \cdot \sin(v)$ $S := \text{CreateMesh}(F, G, H, a, b, 0, 2\pi, mesh)$

График функции f(x)



Поверхность вращения графика f(x)



S

Рис. 13.16. Построение объемной фигуры, полученной вращением кривой

Слева на рисунке показана исходная кривая, заданная функцией f(x), а справа – объемная фигура, построенная с применением форматирования для повышения наглядности графика (подробно команды форматирования описаны в главе 5).

13.2.8. Построение полиэдров

В Mathcad 2000 появилась функция для построения объемных фигур полиэдров – Polyhedron(name), где name – имя фигуры. Имя ряда фигур можно задавать явно, например Polyhedron("cube"), или в виде номера (см. ниже). Рисунок 13.17 иллюстрирует применение данной функции для построения октаэдра с его заданием в явном виде. В данном случае переменной name присваивается значение "octahedron":

```
name := "octahedron"
```

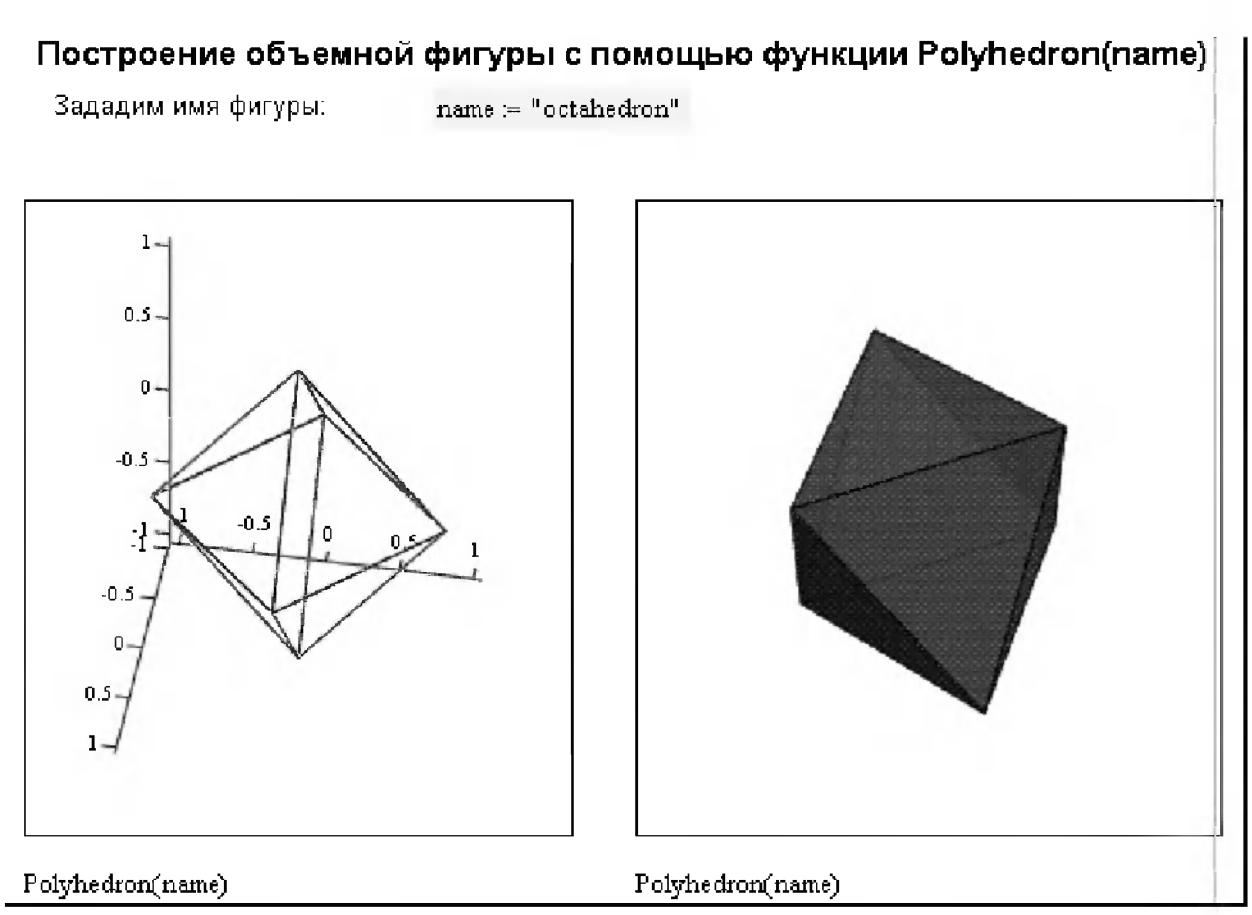


Рис. 13.17. Построение объемной фигуры – октаэдра

Далее в место ввода шаблона трехмерной поверхности вводится функция Polyhedron(name).

Построенная фигура может форматироваться, как и другие графики трехмерных поверхностей (подробно команды форматирования будут описаны в главе 5), а также поворачиваться и масштабироваться с помощью мыши.

Как уже упоминалось, помимо явного задания имени фигуры, можно задавать ее в виде параметра `"#N"`, где `N` – номер фигуры. Для получения имени фигуры по ее номеру можно использовать функцию `PolyLookup`. Например, как показано на рис. 13.18, вызов функции `PolyLookup("#12")` позволил получить следующие данные:

- `"cuboctahedron"` – первое имя фигуры;
- `"rhombic dodecahedron"` – второе имя фигуры;
- `"2|3 4"` – описатель фигуры.

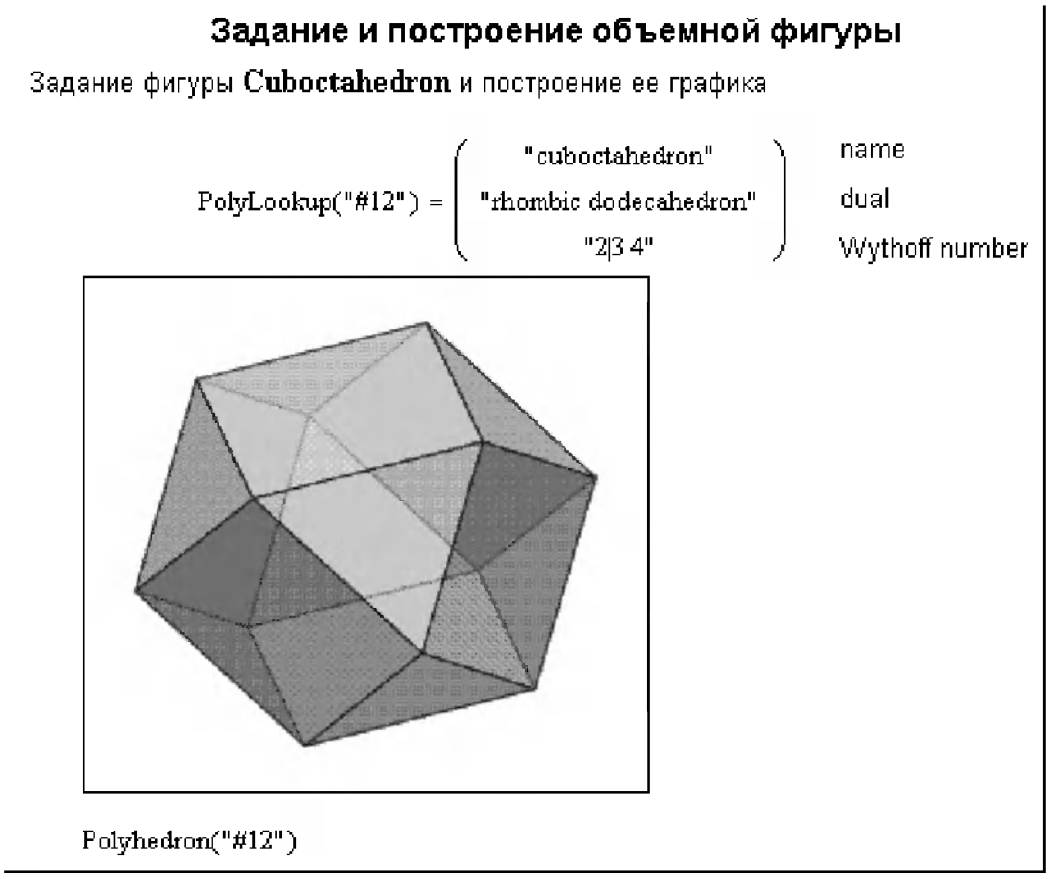


Рис. 13.18. Получение имени фигуры по ее номеру

Большое число примеров применения функции `Polyhedron` можно найти в справочной системе Mathcad – раздел **Polyhedron** (Полиэдры) таблиц **Reference Tables** (Таблицы ссылок). На рис. 13.19 показана страница этого раздела, посвященная регулярным и квазирегулярным полиэдрам. Всего в таблицах имеется около 60 полиэдров.

Наличие превосходных средств построения большого числа полиэдров позволяет эффектно использовать Mathcad в преподавании аналитической геометрии в пространстве и стереометрии.

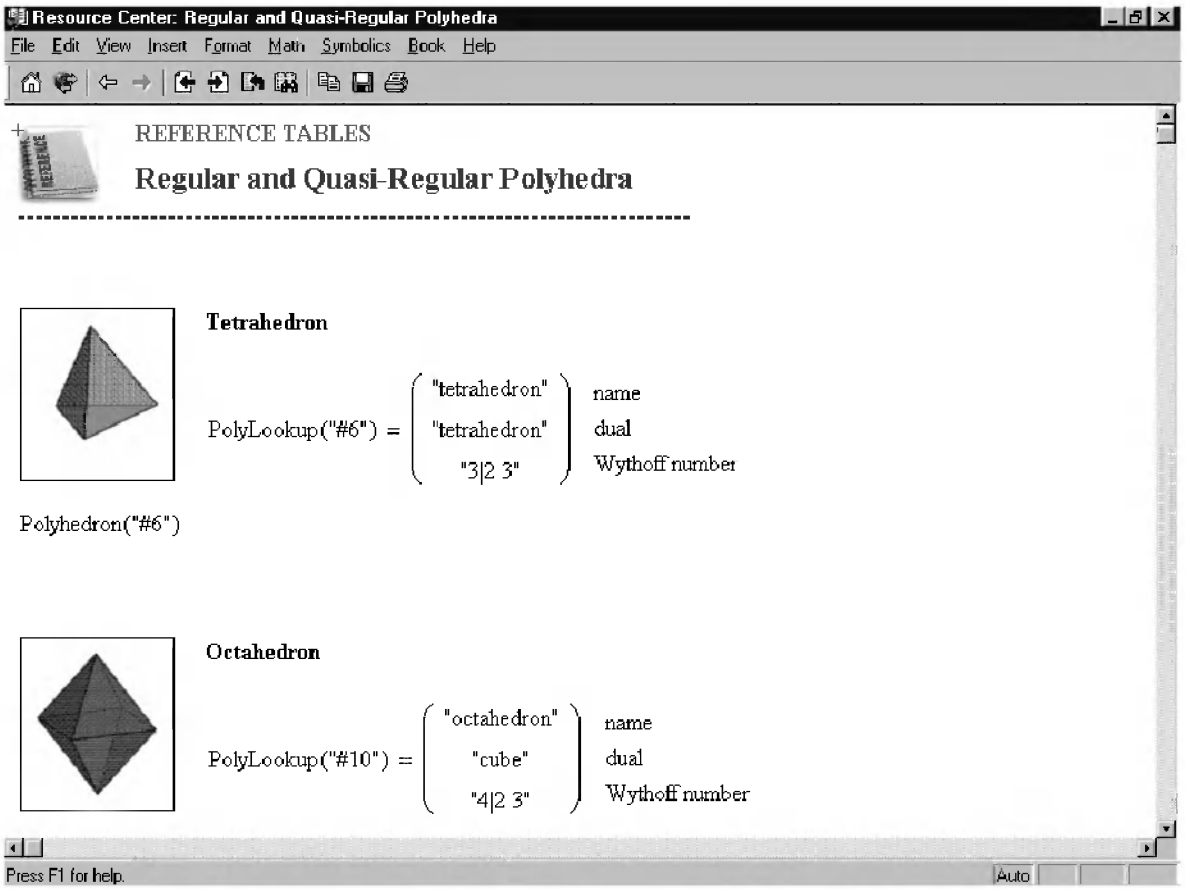


Рис. 13.19. Страница справочных таблиц, посвященная регулярным и квазирегулярным полиэдрам

13.2.9. Построение на одном графике нескольких трехмерных объектов

Новые версии Mathcad предоставляют возможность построения нескольких пересекающихся или непересекающихся поверхностей на одном графике. Для этого достаточно перечислить функции задания поверхностей в местах ввода шаблона графика через запятую.

На рис. 13.20 показано построение октаэдра (см. выше подраздел «Построение полиэдров»), как бы находящегося внутри поверхности, заданной функцией `z(x,y) := -cos(x·y)`.

Место ввода шаблона трехмерного графика должно быть заполнено следующим образом:

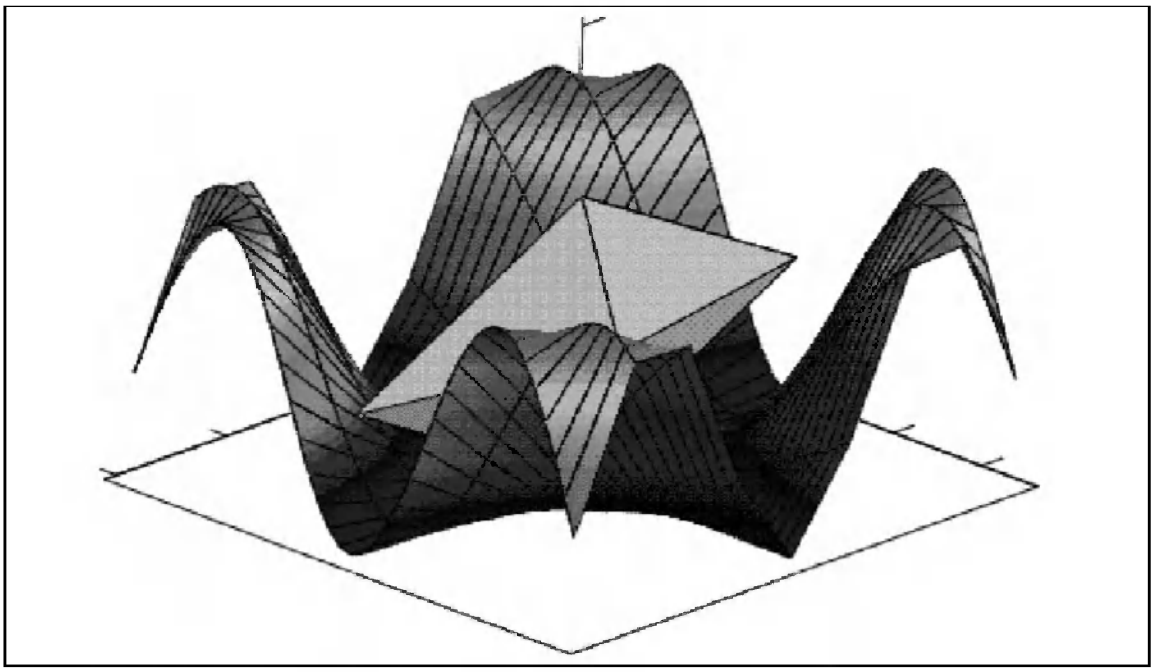
```
z,Polyhedron("Octahedron")
```

Рисунок 13.21 иллюстрирует построение на одном рисунке трех пересекающихся поверхностей. Рисунок отформатирован с применением функциональной окраски. В данном случае поверхности заданы следующими функциями: z1, z2 и z3. Их определение дано над рисунком. Место ввода шаблона трехмерного графика должно быть заполнено следующим образом:

```
z1,z2,z3
```

ПОСТРОЕНИЕ ОКТАЭДРА ВНУТРИ ПОВЕРХНОСТИ

$z(x,y) := -\cos(x \cdot y)$ Функция двух переменных x и y

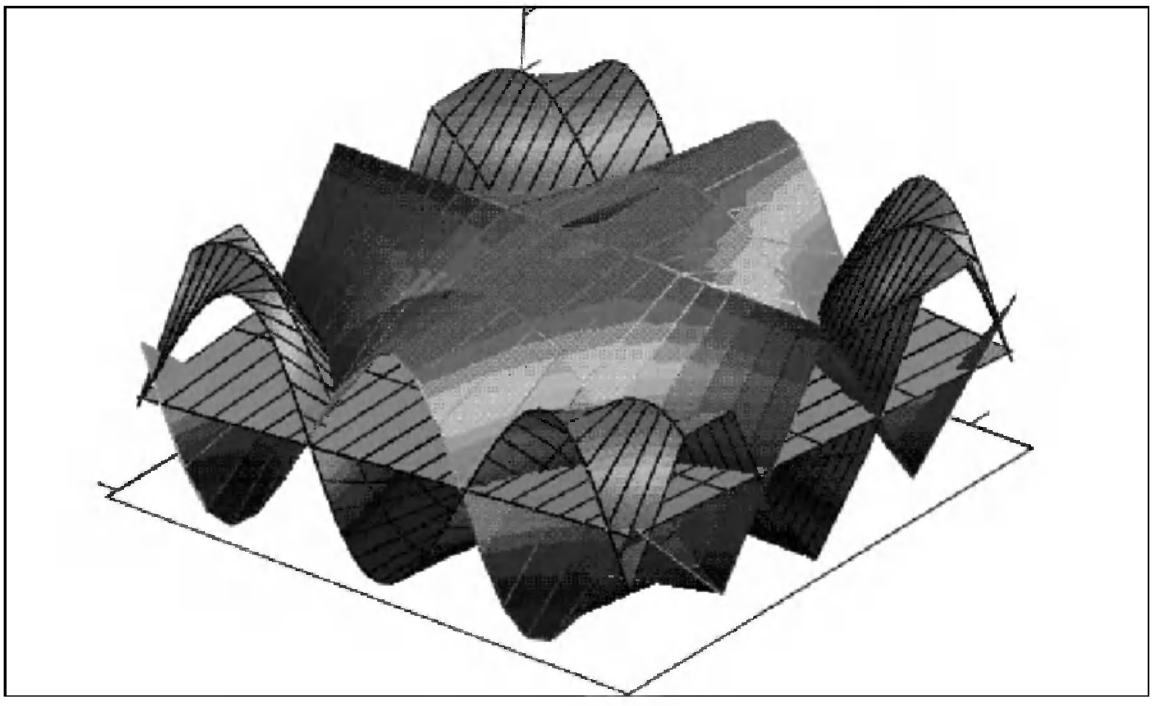


z, Polyhedron("Octahedron")

Рис. 13.20. Построение октаэдра внутри поверхности

ПОСТРОЕНИЕ ГРАФИКОВ ТРЕХ ПОВЕРХНОСТЕЙ

$z1(x,y) := -\cos(x \cdot y)$ $z2(x,y) := \cos(x \cdot y)$ $z3(x,y) := x \cdot \frac{y}{25}$



z1, z2, z3

Рис. 13.21. Построение на одном рисунке трех пересекающихся поверхностей

Возможно построение на одном графике поверхностей с разными способами описания. Пример такого построения дан на рис. 13.22. Единственное место ввода шаблона трехмерного графика должно быть заполнено следующим образом: 1.25G, Z. Множитель 1.25 увеличивает размеры фигуры G.

Рис. 13.22. Построение на одном графике трехмерных поверхностей с разными способами задания

Приведенными примерами возможности трехмерной графики далеко не исчерпываются. Ниже рассматривается ряд других типов таких графиков. Кроме того, форматированием можно обеспечить дополнительные свойства графиков, например их представление в перспективе, в цилиндрической и сферической системах координат и др.

13.2.10. Стандартный способ построения контурных графиков

Рисунок 13.23 иллюстрирует стандартный способ (с заданием матрицы аппликат) построения контурного графика в виде линий равного уровня в системе Mathcad 2001.

Обычно функциональная окраска такого графика, вводимая из окна форматирования, существенно улучшает вид контурных графиков. Но при необходимости ее можно убрать и оставить только контурные линии.

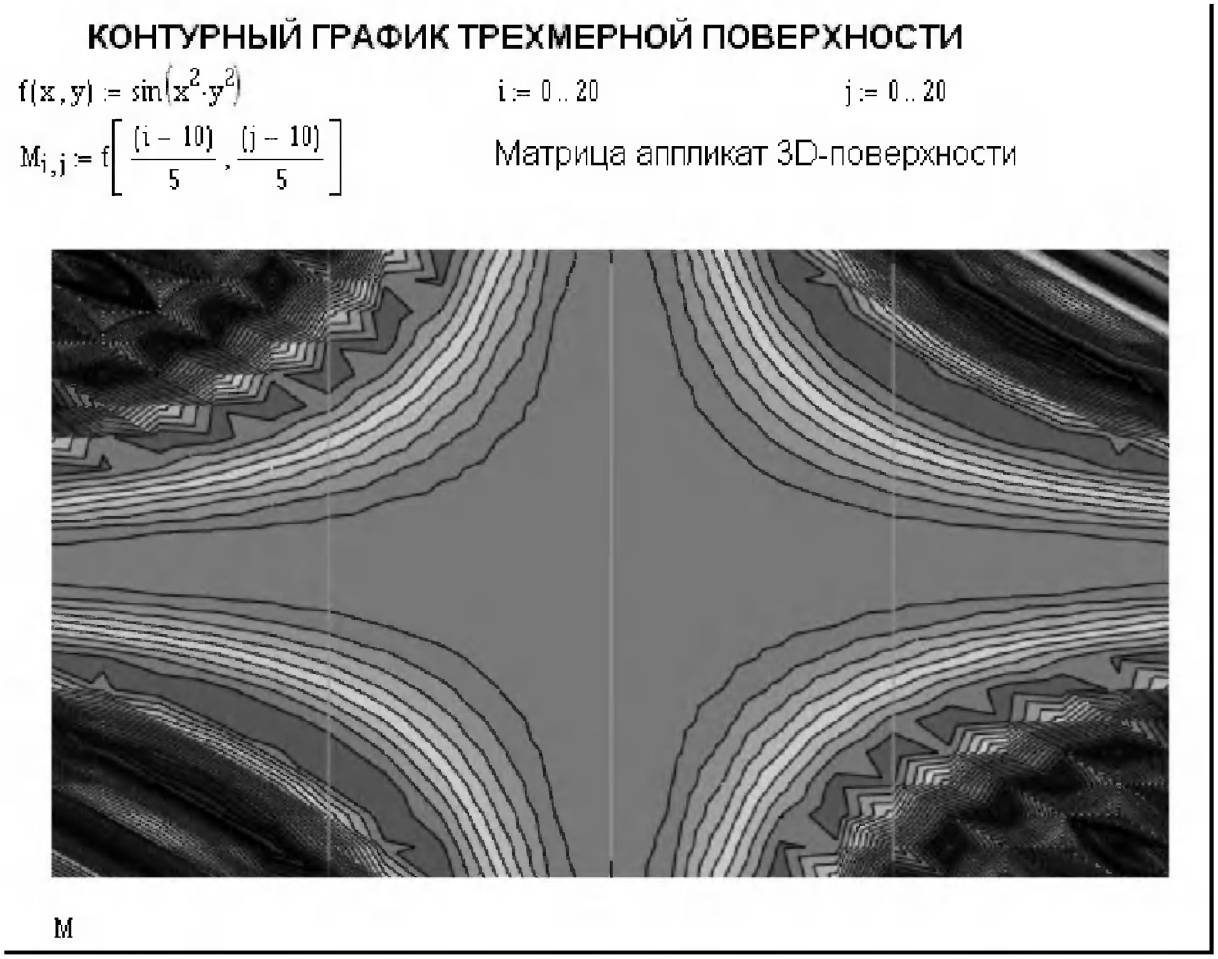


Рис. 13.23. Контурный график поверхности, построенный по ее матрице аппликат

13.2.11. Построение контурных графиков без явного задания матрицы

В Mathcad 2001i/11/12 есть более удобный способ построения контурных графиков – без задания в явном виде матрицы узловых точек, используемых для построения кривых. В этом случае (рис. 13.24) достаточно в месте ввода указать имя функции.

График отформатирован с оцифровкой линий равного уровня (подробно команды форматирования описаны в главе 5). В место ввода шаблона контурного графика должно быть введено имя функции (`f`). Представление графика с оцифровкой линий удобно для количественных оценок. Однако в большинстве случаев указание числовых значений уровней загромождает график.

Иногда контурные графики получаются даже более информативными, чем просто трехмерные поверхности. У последних одни части поверхности нередко закрывают другие. Например, пик на переднем плане может закрыть меньшие пики или впадины на заднем плане. У контурных графиков такого эффекта нет, и на них легко обнаруживаются все пики и впадины (правда, при достаточно большом числе линий равного уровня и малом расстоянии между ними).

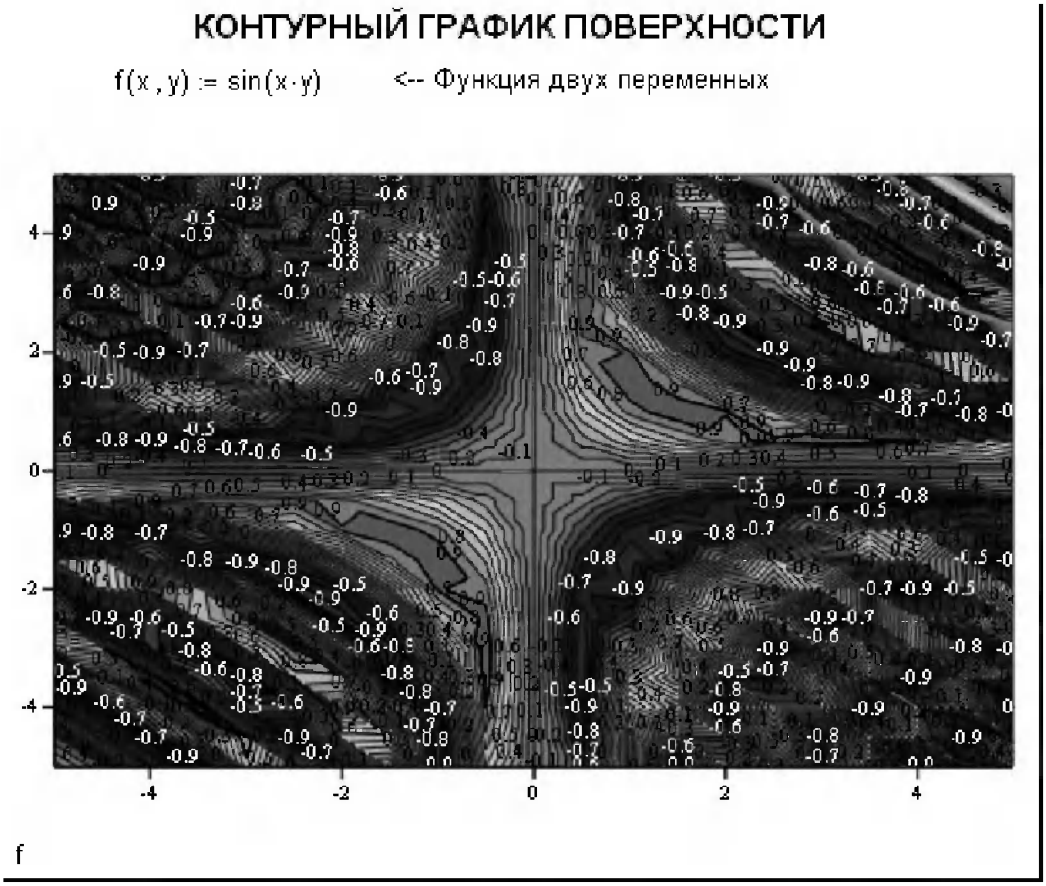


Рис. 13.24. Контурный график поверхности, построенный без задания матрицы

13.3. Точечный трехмерный график

13.3.1. Определение точечного графика

Нередко трехмерные поверхности представляют в виде находящихся в пространстве точек, кружочков или иных фигур. Каждая из этих фигур несет информацию о геометрическом положении ее центра в трехмерном пространстве. Такой график создается командой **3D Scatter Plot** (Точечный график) подменю **Graph** (График) меню **Insert** (Вставка).

Размеры точек, их вид и окраску можно изменять с помощью команд форматирования трехмерного графика (см. главу 5). Обычно неплохо выглядят графики с малыми фигурами, расположенными внутри параллелепипеда. Однако в целом наглядность таких графиков не очень высока. Их имеет смысл использовать не для показа трехмерных поверхностей, а лишь для размещения на поверхностях небольшого числа объектов. Объекты могут быть представлены не только в виде точек, но и в виде иных фигур – крестиков, окружностей, квадратов и др.

13.3.2. Построение точечного графика с заданием матрицы аппликат точек

Сначала рассмотрим пример построения точечного графика при явном задании аппликат точек с помощью матрицы M. Благодаря явному заданию матрицы аппликат точек этот способ пригоден для любых версий Mathcad. В место ввода шаблона точечного графика должно быть введено имя матрицы аппликат поверхности (M). Вид графика показан на рис. 13.25.

Графики этого типа имеют ограниченное применение.

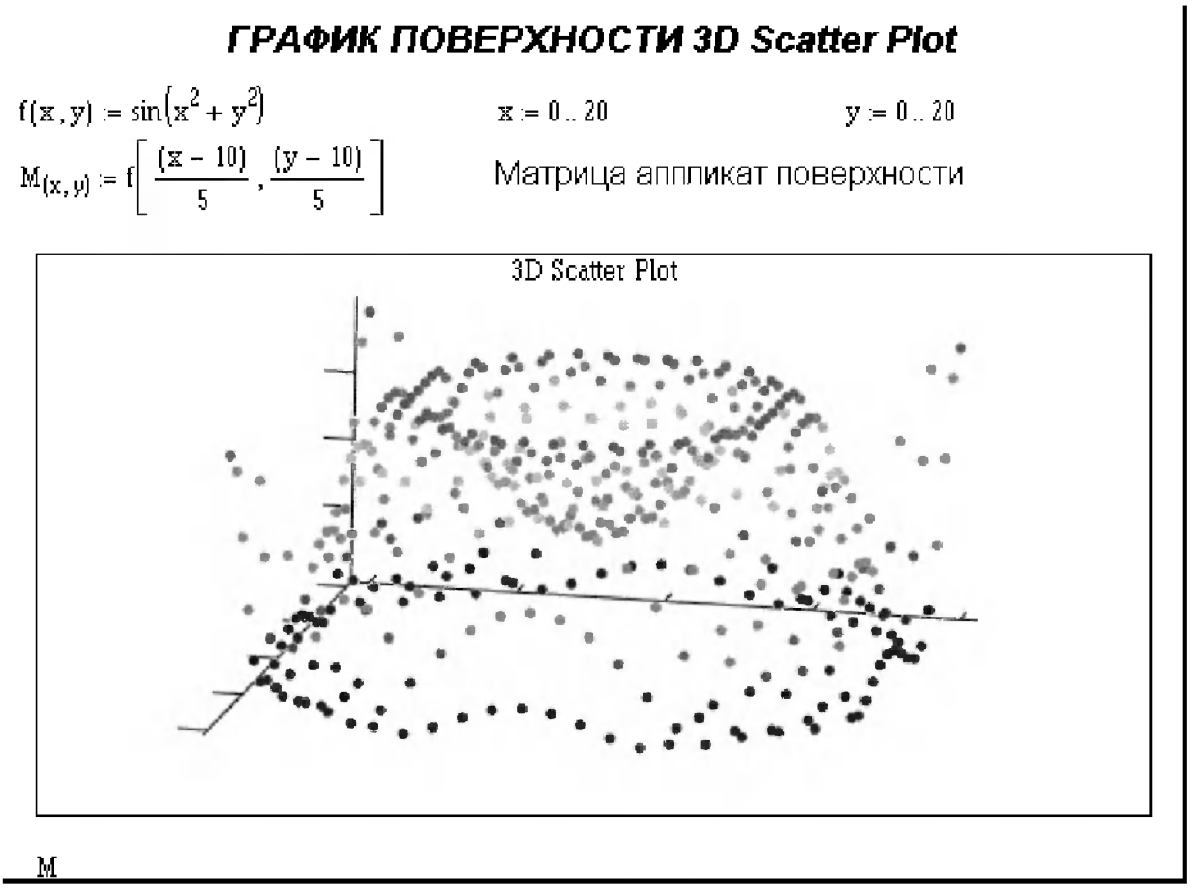


Рис. 13.25. График трехмерной поверхности в виде разбросанных в пространстве точек

13.3.3. Построение точечного графика с заданием только функции поверхности

Mathcad 2001 позволяет строить точечные графики без явного задания матрицы аппликат отображаемых точек. Достаточно указать в шаблоне графика имя функции двух переменных, задающей аппликаты точек (рис. 13.26). В место ввода шаблона точечного графика должно быть введено имя функции f.

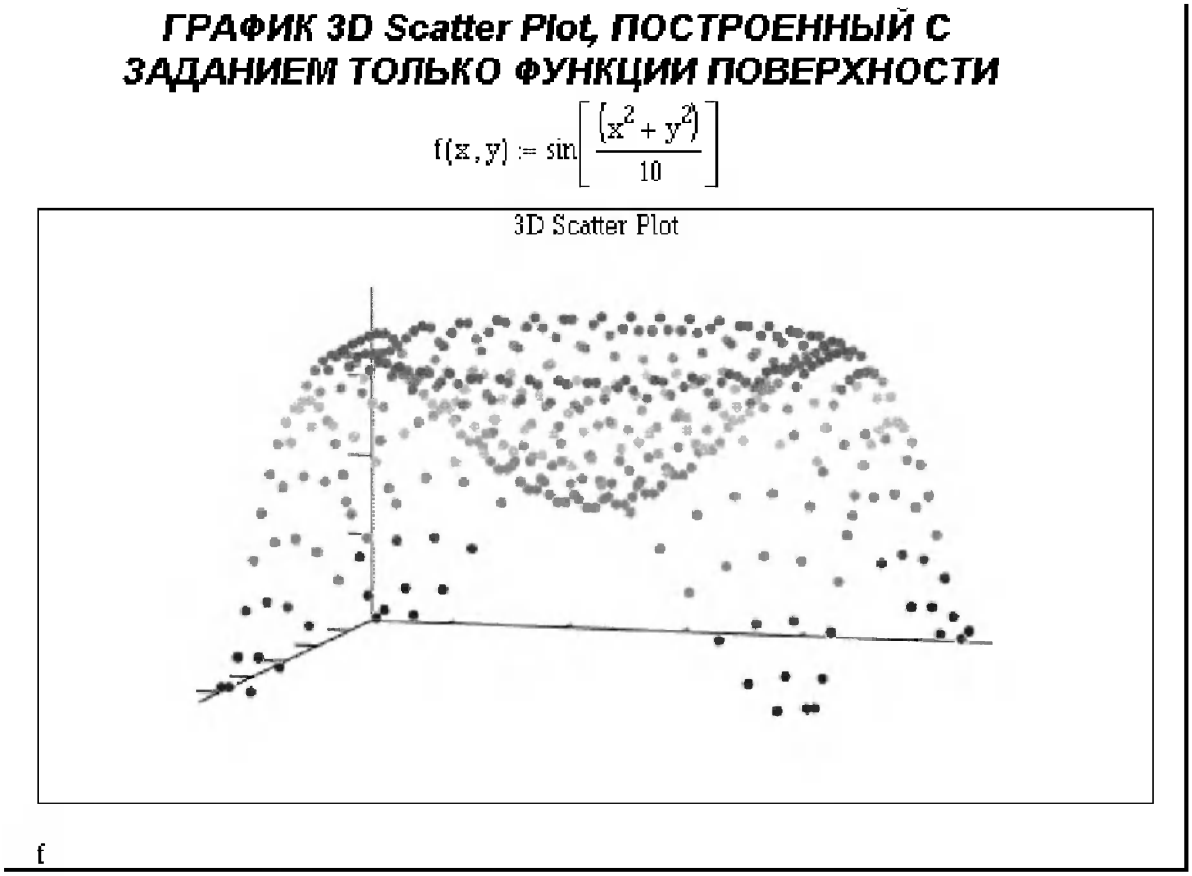


Рис. 13.26. Построение точечного графика с заданием только функции поверхности

Этот способ имеет то неудобство, что масштабы осей, установленные по умолчанию, не всегда приемлемы, поэтому требуется либо нормировать функцию, либо переформатировать график, изменив масштабы осей. Разумеется, формирование графика позволяет получить и иные эффекты, например можно придать точкам функциональную окраску, вывести титульную надпись и др.

13.3.4. Применение функции CreateSpace

Новые версии Mathcad имеют еще одну новую графическую функцию:

```
CreateSpace(F,t0,t1,tgrid,fmap)
```

Эта функция отличается от функции `CreateMesh` (см. подраздел «Применение новой функции CreateMesh» в разделе «Трехмерный график поверхности») только тем, что заданная в векторном виде функция `F` представляет собой набор функций одной переменной `t`, причем параметры `t0` и `t1` определяют пределы ее изменения, а переменная `tgrid` – число линий сетки. Необязательный параметр `fmap` в виде списка трех переменных позволяет задать тип координатной системы, например прямоугольная система (cartesian) задается по умолчанию следующим образом:

```
fmap_default(e1,e2,e3) := c(e1,e2,e3)
```

Вместо первого параметра функции `CreateMesh` (вектора `F`) можно задать список из трех функций `f1`, `f2` и `f3` переменной `t`.

С помощью функции `CreateSpace` удобно строить точечные графики в виде пространственных спиралей и иных подобных геометрических образов. Для примера рассмотрим алгоритм использования функции `CreateSpace` для построения точечного графика пространственной спирали.

1. Введите в векторной форме функцию `H(t)` одной переменной `t`, как показано ниже:

$$H(t) := \begin{pmatrix} t \cdot \sin(t) \\ t \cdot \cos(t) \\ t \end{pmatrix}$$

2. Задайте пределы изменения переменных и число точек графика:

```
t0 := 0
t1 := 16
tgrid := 160
```

3. С помощью функции `CreateSpace` создайте матрицу аппликат:

```
C := CreateSpace(H, t0, t1, tgrid)
```

4. Выберите команду **Insert** (Вставка) ⇒ **Graph** (График) ⇒ **3D Scatter Plot** (Точечный график).
5. В единственное место ввода появившегося шаблона точечного графика введите имя матрицы аппликат, созданной на шаге 3.
6. Щелкните в документе вне области графика. График будет построен.
7. При необходимости отформатируйте график (подробно команды форматирования описаны в главе 5).

График, показанный на рис. 13.27 слева, получен без форматирования, а справа – после форматирования.

13.4. Трехмерная гистограмма

Весьма распространенной формой представления поверхностей является ряд трехмерных столбиков, высота которых определяется значением координаты `z(x,y)`. Для этого используется команда **3D Bar Plot** (Трехмерная гистограмма) подменю **Graph** (График) меню **Insert** (Вставка).

13.4.1. Обычное построение гистограмм

Рисунок 13.28 иллюстрирует стандартную технологию построения таких графиков – с заданием матрицы аппликат поверхности, представляемой столбиками. В место ввода шаблона трехмерной гистограммы должно быть введено имя матрицы аппликат поверхности (`M`).

Подобные графики широко применяются при представлении сложных статистических данных. В отношении трехмерной гистограммы возможны все операции форматирования, включая поворот графиков мышью.

Применение графической функции CreateSpace

1. Зададим в векторной форме функции одной переменной t:

$$H(t) := \begin{pmatrix} t \cdot \sin(t) \\ t \cdot \cos(t) \\ t \end{pmatrix}$$

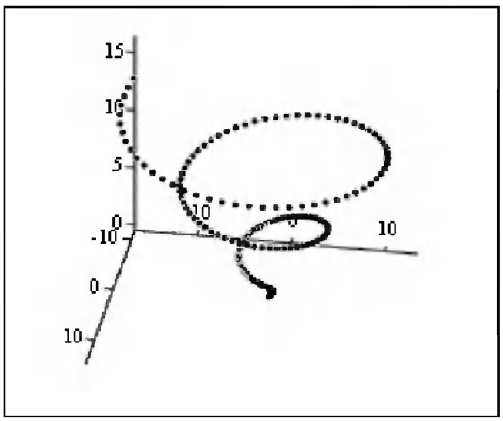
2. Определим пределы t и число точек графика grid:

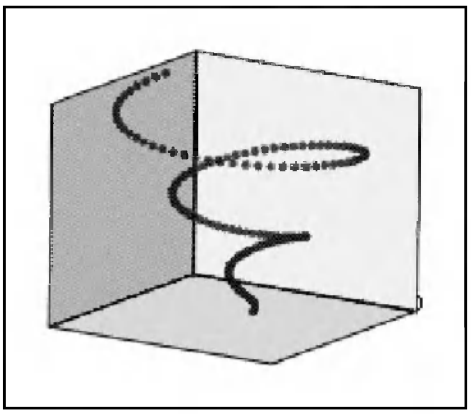
t0 := 0 t1 := 16 tgrid := 160

3. Используя функцию **CreateSpace** создадим матрицу

C := CreateSpace(H,t0,t1,tgrid)

4. Используя построение графика типа **Scatter Plot** получим





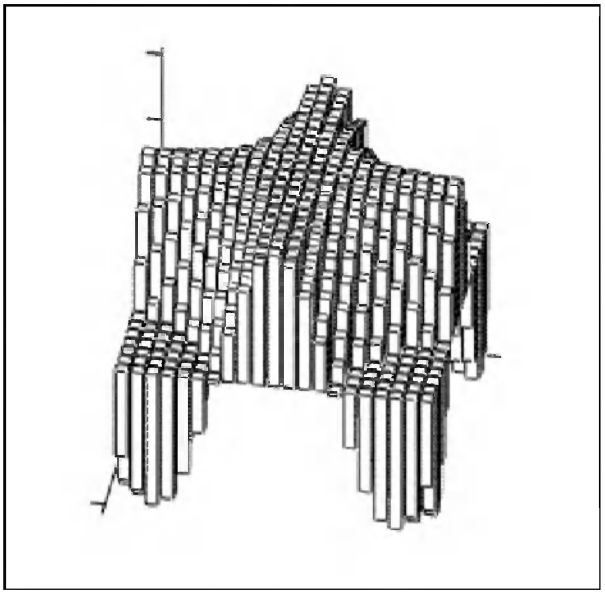
C C

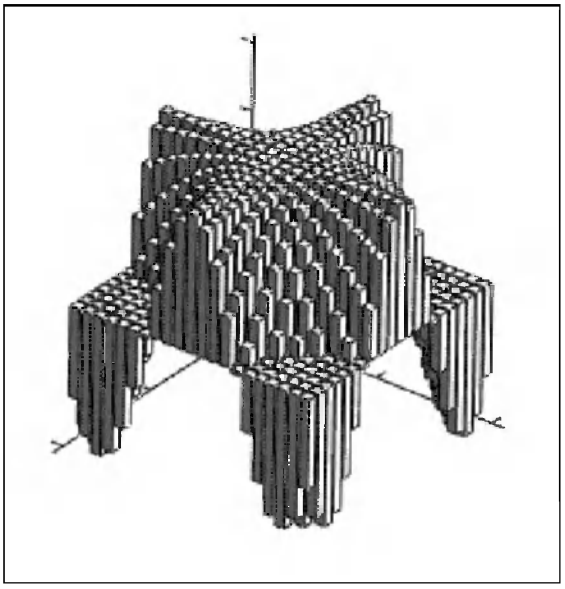
Рис. 13.27. Построение точечного графика пространственной спирали

ГРАФИК ПОВЕРХНОСТИ ТИПА 3D Bar Plot

$f(x,y) := \cos(x \cdot y)$ $i := 0..20$ $j := 0..20$

$M_{i,j} := f\left[\frac{(i-10)}{5}, \frac{(j-10)}{5}\right]$ Матрица аппликат 3D-поверхности





M M

Рис. 13.28. Представление поверхности трехмерными столбиками

13.4.2. Построение трехмерных гистограмм с заданием только функции поверхности

Mathcad 2001 дает возможность упростить построение трехмерных гистограмм за счет указания в месте ввода графика только имени функции. Пример такого построения дан на рис. 13.29.

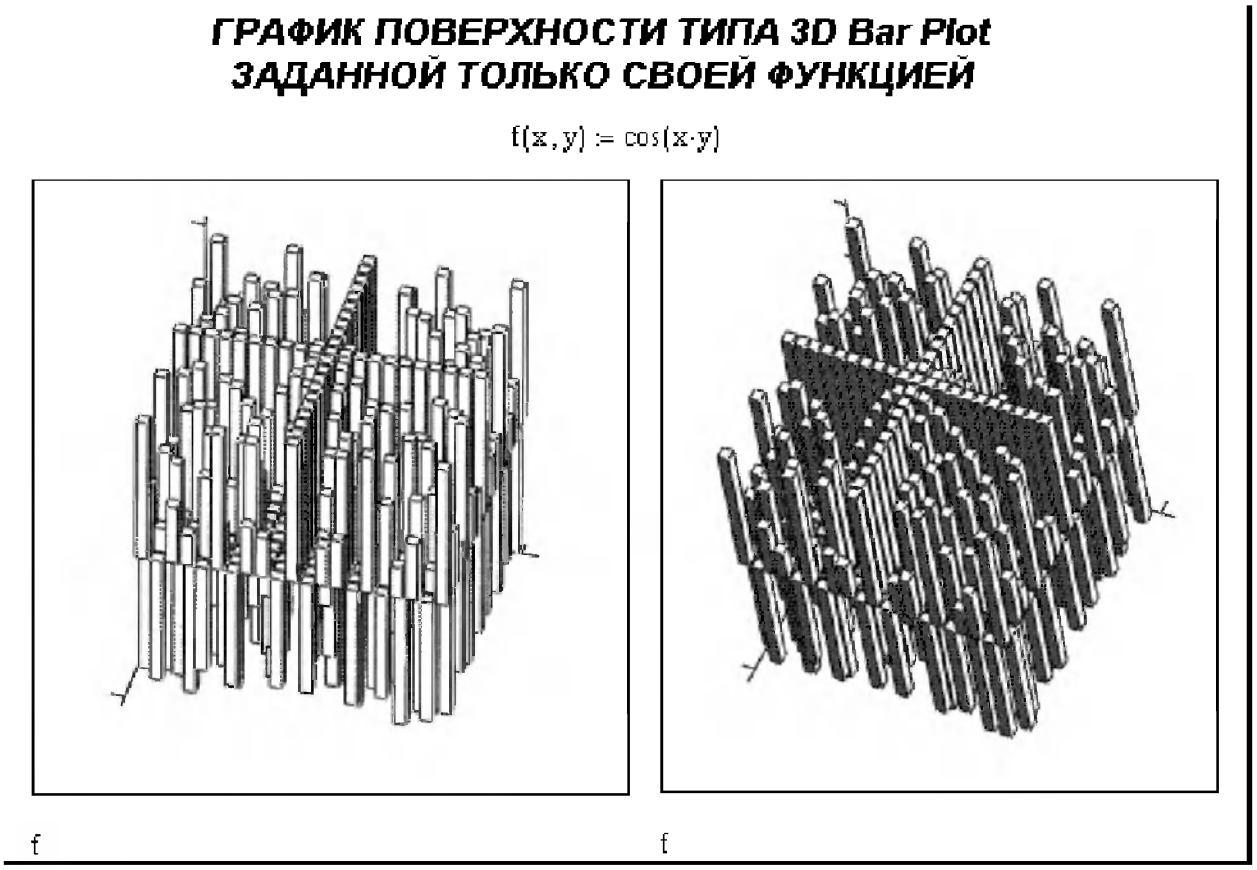


Рис. 13.29. Пример трехмерной гистограммы с заданием только функции поверхности

13.5. Трехмерный график в векторном представлении

Еще один вид представления поверхности – векторное представление. Оно задается набором коротких стрелочек – векторов. Стрелка обращена острием в сторону нарастания высоты поверхности, а плотность расположения стрелок зависит от скорости этого нарастания. Для построения используется команда **Vector Field Plot** (Векторное поле) подменю **Graph** (График) меню **Insert** (Вставка).

13.5.1. Обычное построение графиков векторного поля

На рис. 13.30 показан пример графика векторного поля. В место ввода шаблона векторного поля должно быть введено имя матрицы аппликат поверхности (M).

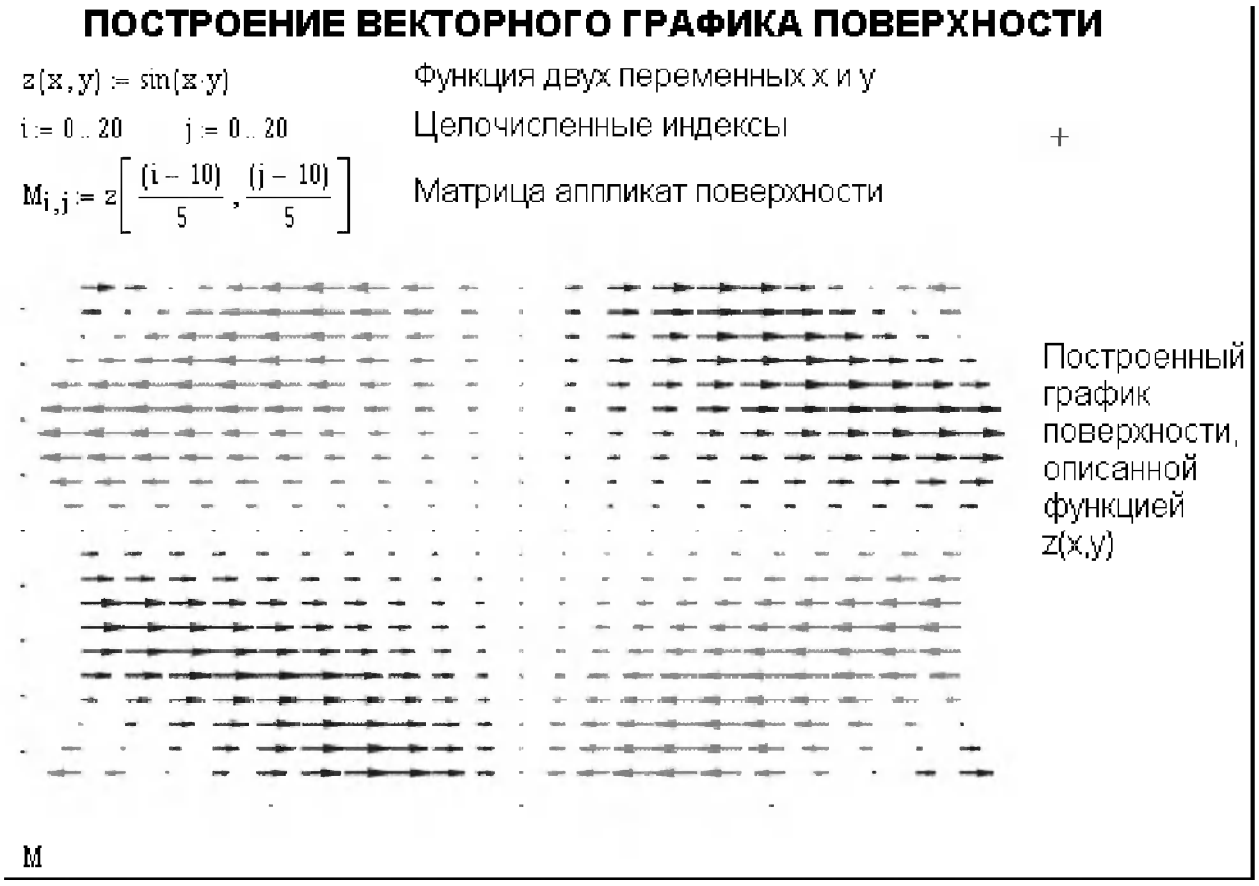


Рис. 13.30. Представление поверхности векторами

Эти графики применяются редко из-за трудности построения множества стрелок, для каждой из которых надо рассчитывать градиент поля. Но Mathcad обеспечивает возможность широкого применения графиков этого вида. Они особенно удобны для представления электромагнитных, тепловых, гравитационных и иных полей.

13.5.2. Построение графика векторного поля, заданного в параметрической форме

Еще один способ построения графиков векторных полей заключается в задании поверхности в параметрической форме. Этот случай представлен на рис. 13.31. В данном случае в место ввода шаблона графика векторного поля вводятся в скобках имена матриц M и N, а исходная формула для построения выглядит следующим образом:

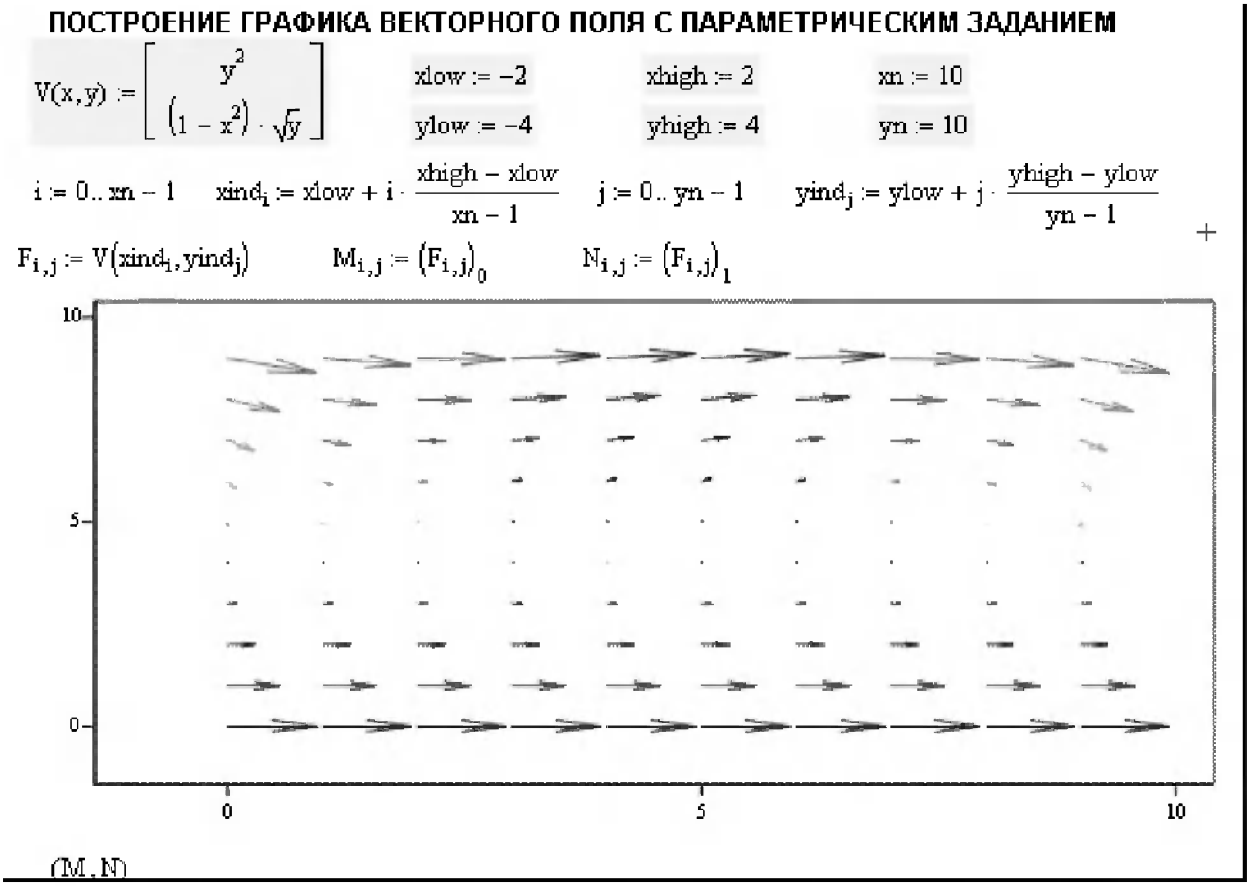


Рис. 13.31. График векторного поля поверхности, заданной в параметрическом виде

$$V(x,y):=\begin{bmatrix} y^2 \\ \left(1-x^2\right)\cdot\sqrt{y} \end{bmatrix}.$$

Место ввода шаблона векторного поля должно быть заполнено следующим образом: (M,N).

Ряд примеров на построение графиков различного типа можно найти в быстрых «шпаргалках» QuickSheets.

13.6. Специальные приемы построения трехмерных графиков

13.6.1. Построение трехмерных графиков мастером

Форматирование трехмерных графиков в Mathcad 2000/2001 – довольно сложный процесс, поскольку число параметров форматирования достигает многих десятков (см. главу 5). Чтобы упростить создание трехмерных графиков, использу-

ется специальный мастер, разбивающий процедуру построения и форматирования графика на несколько вполне очевидных этапов.

Для создания трехмерного графика с помощью мастера вначале надо ввести функцию или матрицу поверхности, например так:

```
f(x,y) := sin(0.2x·y)
```

Далее требуется выбрать команду **Insert** (Вставка) ⇒ **Graph** (График) ⇒ **3D Plot Wizard** (Мастер трехмерной графики). В окне документа появится первое окно мастера – **Plot Type** (Тип графика), показанное на рис. 13.32.

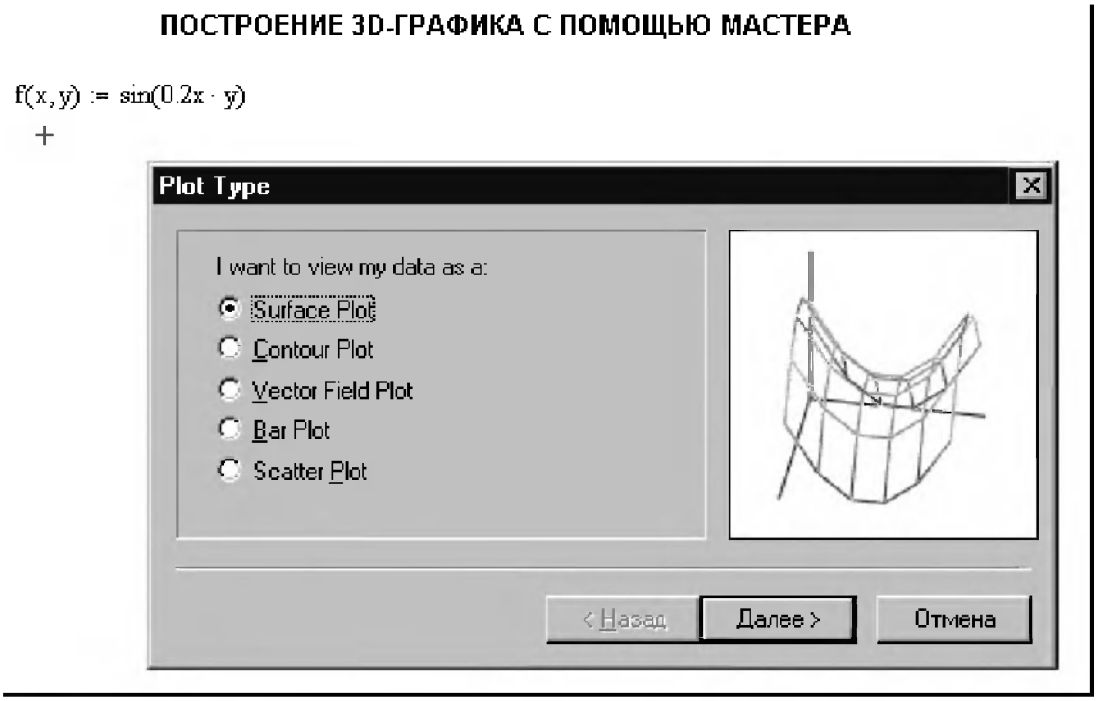


Рис. 13.32. Окно Plot Type мастера создания трехмерных графиков

В этом окне с помощью переключателей требуется выбрать необходимый тип графика. В нашем случае установите переключатель **Surface Plot** (График поверхности) и щелкните на кнопке **Next** (Далее), чтобы перейти к следующему окну мастера – **Appearance** (Вид), показанному на рис. 13.33. В нем установите переключатель **Fill Surface and Draw Lines** (Окрашенная поверхность с линиями).

Щелкнув на кнопке **Next** (Далее), можно вывести последнее окно мастера – **Coloring** (Задание цветовой гаммы), показанное на рис. 13.34. Здесь установите переключатель **Color using lighting** (Цвет определяется источником света). Вместо кнопки **Next** (Далее) в этом окне появляется кнопка **Finish** (Готово), что указывает на завершение операций с мастером. Щелкните на ней, если считаете правильными введенные параметры. Кнопка **Back** (Назад) позволяет вернуться к предыдущему окну, если вы решите изменить введенные ранее параметры. А если вы вообще передумали строить график, щелкните на кнопке **Cancel** (Отмена).

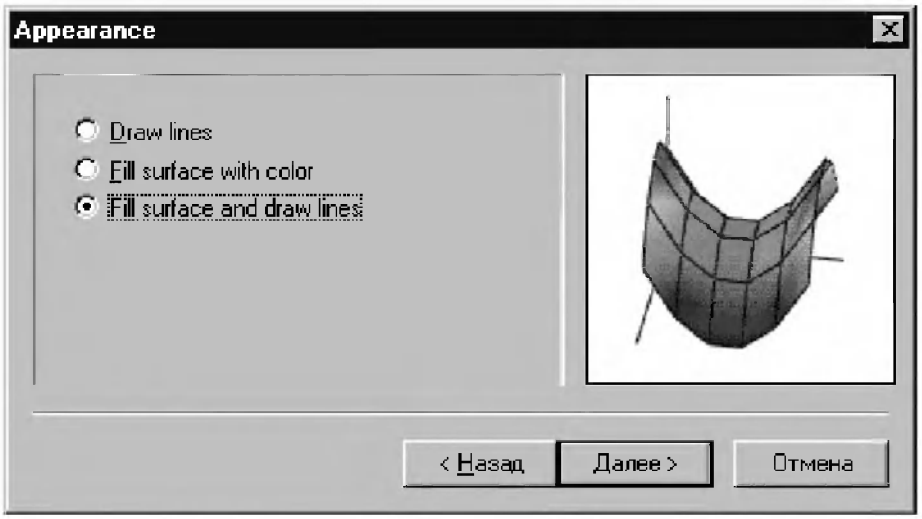


Рис. 13.33. Окно Appearance мастера создания трехмерных графиков

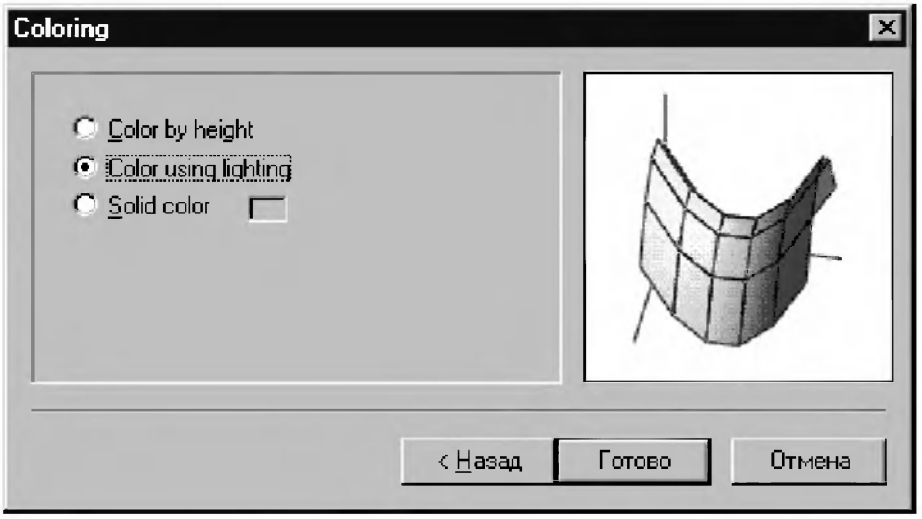


Рис. 13.34. Завершающее окно мастера – Coloring

При работе с мастером все вводимые параметры наглядно отображаются в области предварительного просмотра. Однако это не ваш график. Для демонстрации параметров форматирования используется не относящийся к делу рисунок параболической поверхности. По завершении работы с мастером на месте курсора ввода (красный крестик) появится шаблон трехмерного графика, но пока без самого графика. Для его построения надо в месте ввода шаблона графика указать имя той матрицы или функции, для которой должен быть построен график (в нашем случае это `f`). Затем надо щелкнуть левой кнопкой мыши вне области графика – график будет тут же построен, что и иллюстрирует рис. 13.35.

Разумеется, построенный график можно подвергнуть добавочному форматированию всеми возможными способами. К примеру, график, показанный на рисунке, увеличен в размере.

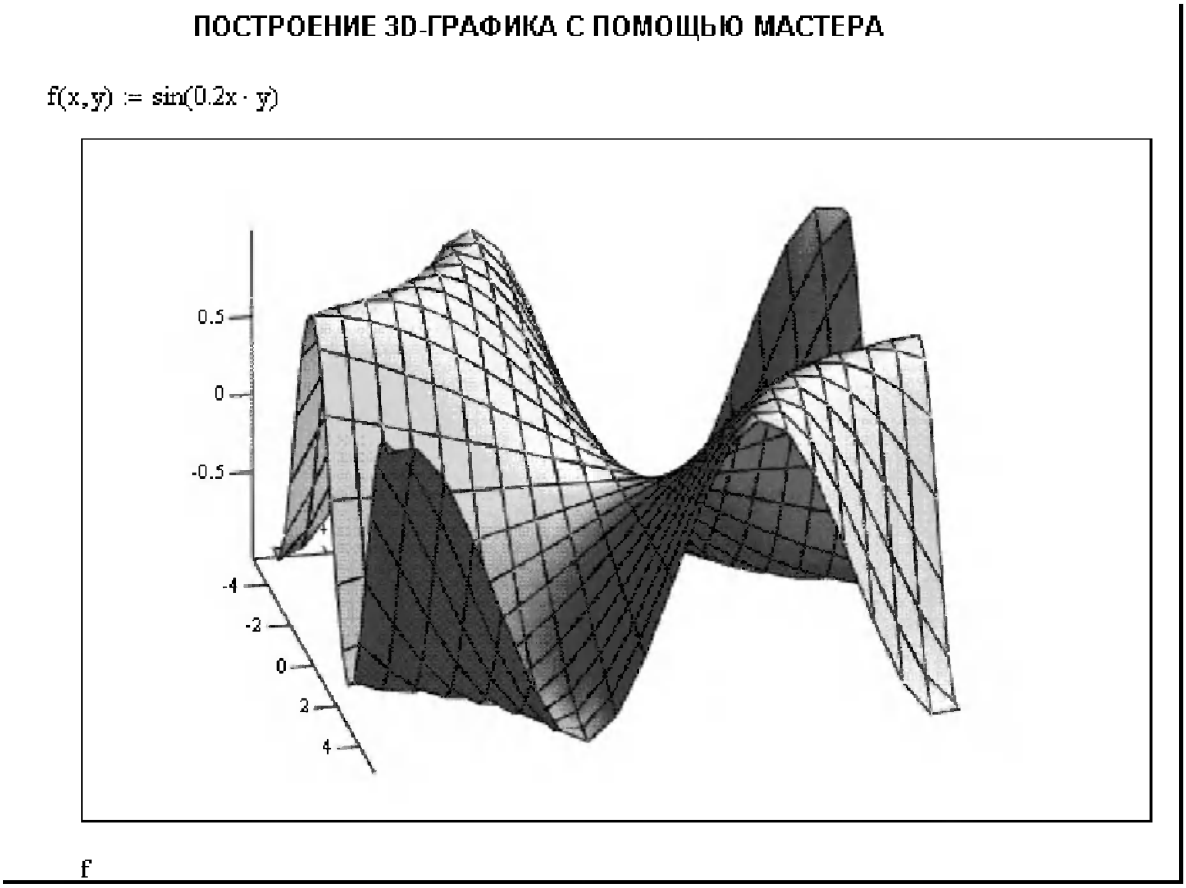


Рис. 13.35. Готовый график, построенный с помощью мастера

Высокая наглядность процесса создания трехмерных графиков с помощью мастера делает его применение предпочтительным, особенно для пользователей, впервые приступающих к освоению премудростей системы Mathcad. Однако следует отметить, что мастер реализует не все возможности форматирования. Например, он не позволяет задать тип координатных осей, установить обрамление графика параллелепипедом, окрасить плоскости и т. д.

13.6.2. Трехмерный «лоскутный» график

Хотя в подменю **Graph** (График) меню **Insert** (Вставка) нет соответствующей команды, существует возможность построения еще одного типа графика – так называемого «лоскутного» графика. Для его построения нужно построить трехмерный график любого типа, а затем дважды щелкнуть мышью в области графика. В открывшемся окне форматирования перейдите на вкладку **General** (Общие), установите переключатель **Patch Plot** («Лоскутный» график) и щелкните на кнопке **OK**. График будет тут же перестроен, а окно форматирования закрыто. Например, на рис. 13.36 показан «лоскутный» график, построенный на основе графика поверхности, созданного в разделе «Трехмерный график поверхности» (см. рис. 13.9).

Как нетрудно заметить, «лоскутный» график представляет собой поверхность в виде прямоугольников или иных простых фигур.

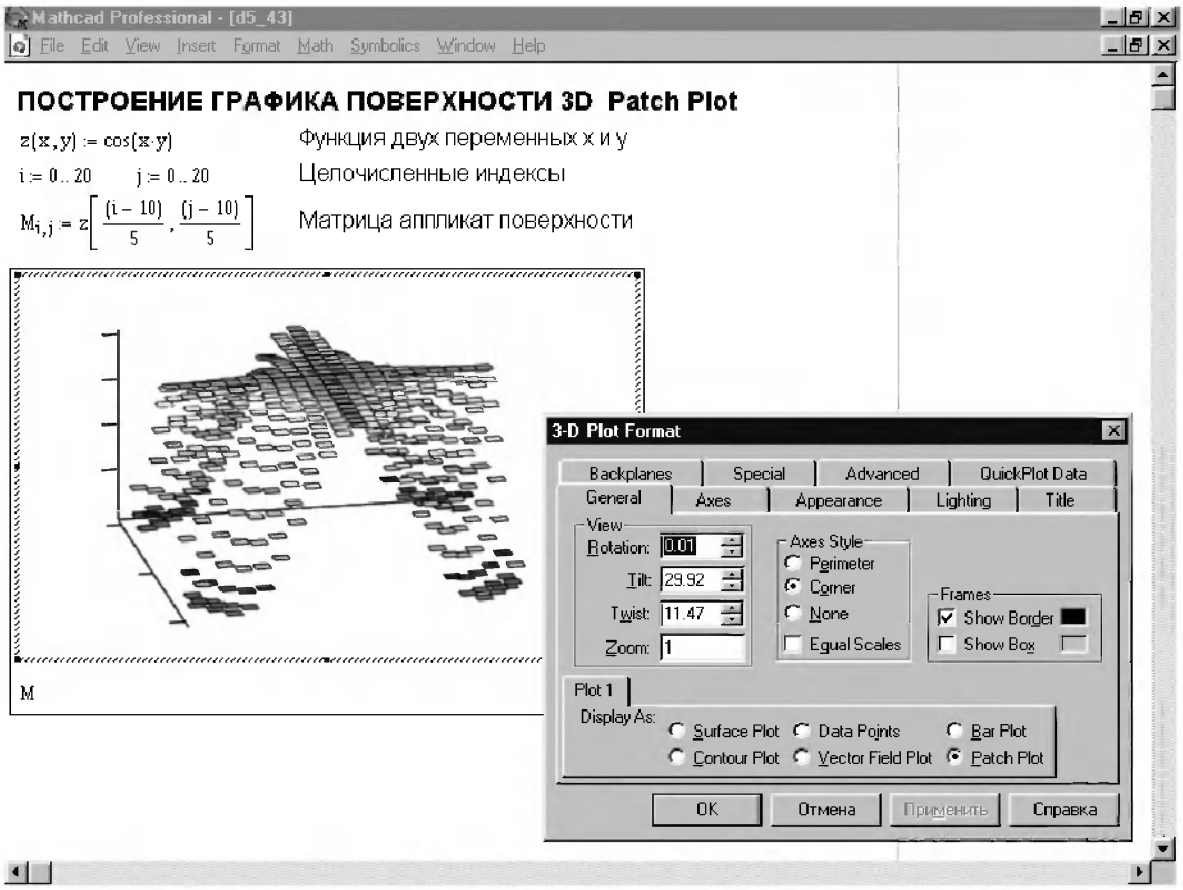


Рис. 13.36. Пример построения и форматирования графика типа Patch Plot

13.6.3. Два пересекающихся в пространстве тора

Рисунок 13.37 иллюстрирует построение двух пересекающихся в пространстве торов. Здесь интересно то, что математически задается один тор с параметрическим заданием построения, но в месте ввода данные для построения тора указываются дважды. В итоге получается весьма реалистичное изображение двух пересекающихся торов.

Еще одна примечательная деталь показанного на рисунке графика – форматирование с использованием световых эффектов, имитирующих блики света при освещении торов от внешнего источника света. К сожалению, на черно-белых рисунках книги не видна игра цветов, но блики света отчетливо заметны.

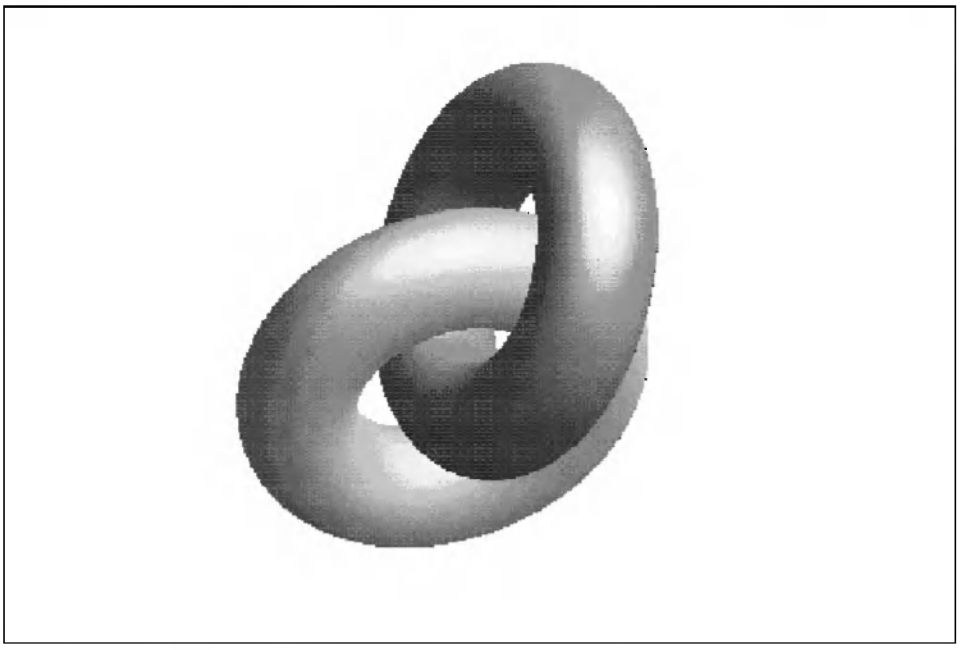
13.6.4. Тор «в тряпках» и с обмоткой

Следующий фрагмент документа (рис. 13.38) иллюстрирует построение двух объектов – тора и некоторой поверхности. Она отформатирована так, что создает весьма необычный эффект – тор как бы лежит в куче тряпок. Место ввода шаблона трехмерного графика должно быть заполнено следующим образом: (x,y,z), (X,Y,Z).

ДВА "БУБЛИКА" С ИМИТАЦИЕЙ БЛИКОВ СВЕТА

$N := 60 \qquad i := 0..N \qquad \phi_i := i \cdot 2 \cdot \frac{\pi}{N} \qquad j := 0..N \qquad \theta_j := j \cdot 2 \cdot \frac{\pi}{N}$

$x_{i,j} := (5 + 2\cos(\phi_i))\cos(\theta_j) \qquad y_{i,j} := (5 + 2\cos(\phi_i))\sin(\theta_j) \qquad z_{i,j} := 2\sin(\phi_i)$



(x,y + 6,z),(z,y + 1,x)

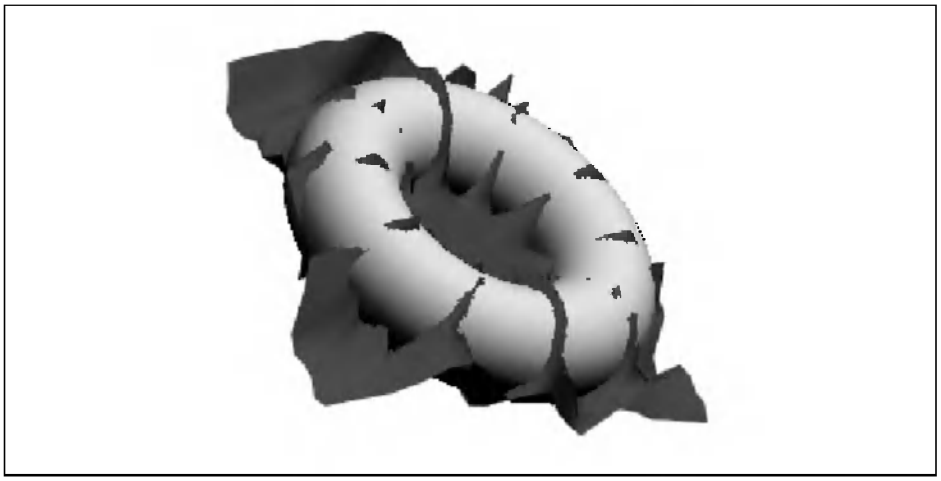
Рис. 13.37. Построение двух пересекающихся в пространстве торов

ТОР "В ТРЯПКАХ"

$N := 40 \qquad i := 0..N \qquad \phi_i := i \cdot 2 \cdot \frac{\pi}{N} \qquad j := 0..N \qquad \theta_j := j \cdot 2 \cdot \frac{\pi}{N} \qquad t := 0..1000$

$x_{i,j} := (5 + 2\cos(\phi_i))\cos(\theta_j) \qquad y_{i,j} := (5 + 2\cos(\phi_i))\sin(\theta_j) \qquad z_{i,j} := 2\sin(\phi_i)$

$X_t := \left(5 + 2.5\cos\left(\frac{t \cdot \pi}{50}\right)\right)\cos\left(\frac{t \cdot \pi}{500}\right) \qquad Y_t := \left(5 + 2.5\cos\left(\frac{t \cdot \pi}{50}\right)\right)\sin\left(\frac{t \cdot \pi}{500}\right) \qquad Z_t := 2.5\sin\left(\frac{t \cdot \pi}{50}\right)$



(x,y,z),(X,Y,Z)

Рис. 13.38. Построение тора «в тряпках» и тора с обмоткой

Еще одно построение торов представлено на рис. 13.39. Слева построен просто тор, а справа – тор с обмоткой.

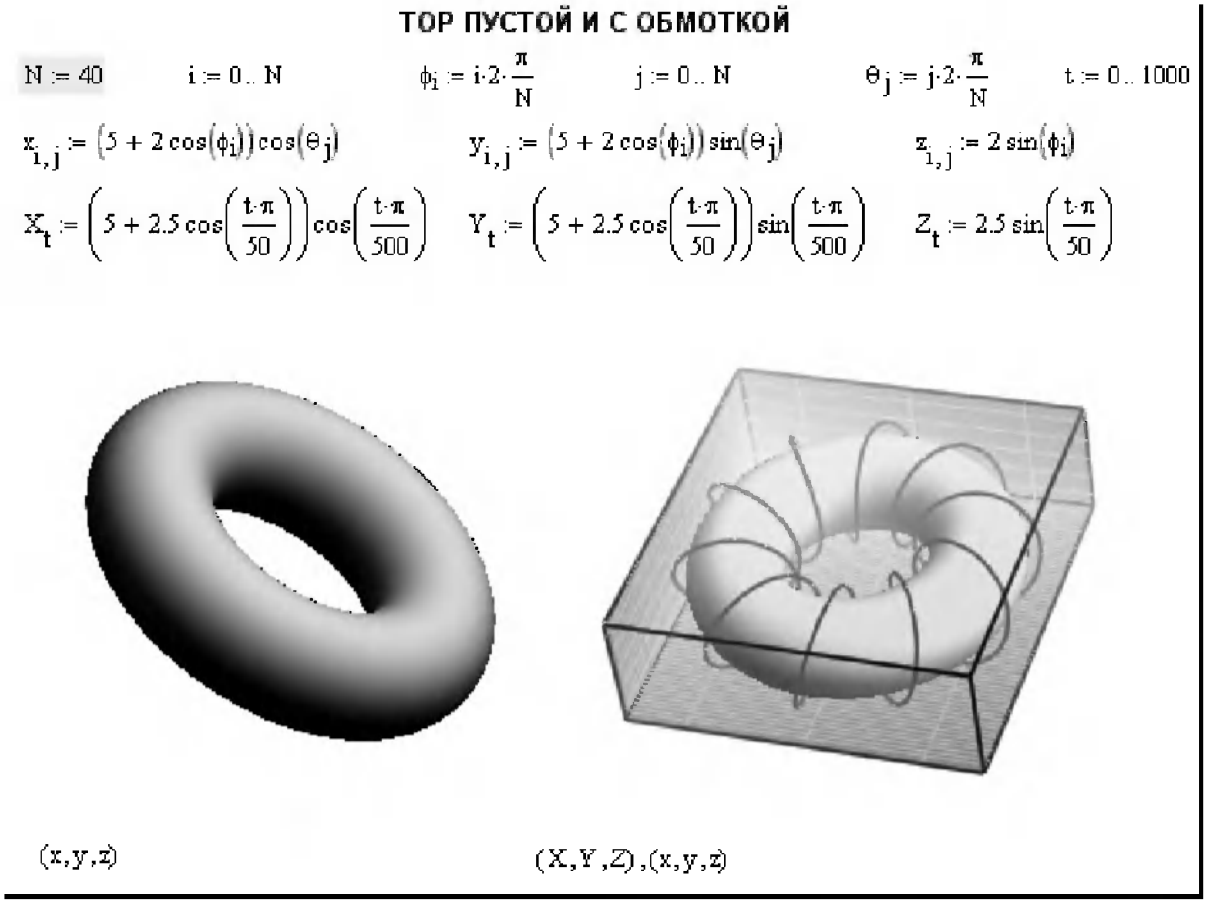


Рис. 13.39. Построение отдельного тора и тора с «обмоткой»

Вид подобных графиков сильно зависит от того, как они отформатированы. К сожалению, документы Mathcad в явном виде не содержат указаний на то, как форматируются графики. О параметрах форматирования уже построенных графиков можно узнать из окна форматирования, которое выводится при двойном щелчке мыши на графике.

13.6.5. Поверхность в поверхности

Интересную картину создает график, показанный на рис. 13.40. Здесь строится поверхность полупрозрачной полусферы, внутри которой расположен не то городской, не то скальный пейзаж. Он создается в виде поверхности, генерируемой с помощью генератора случайных чисел. В цветном варианте получается очень реалистическая картина.

Для получения представленной на рис. 13.40 картинки также придется изрядно потрудиться с форматированием рисунка.

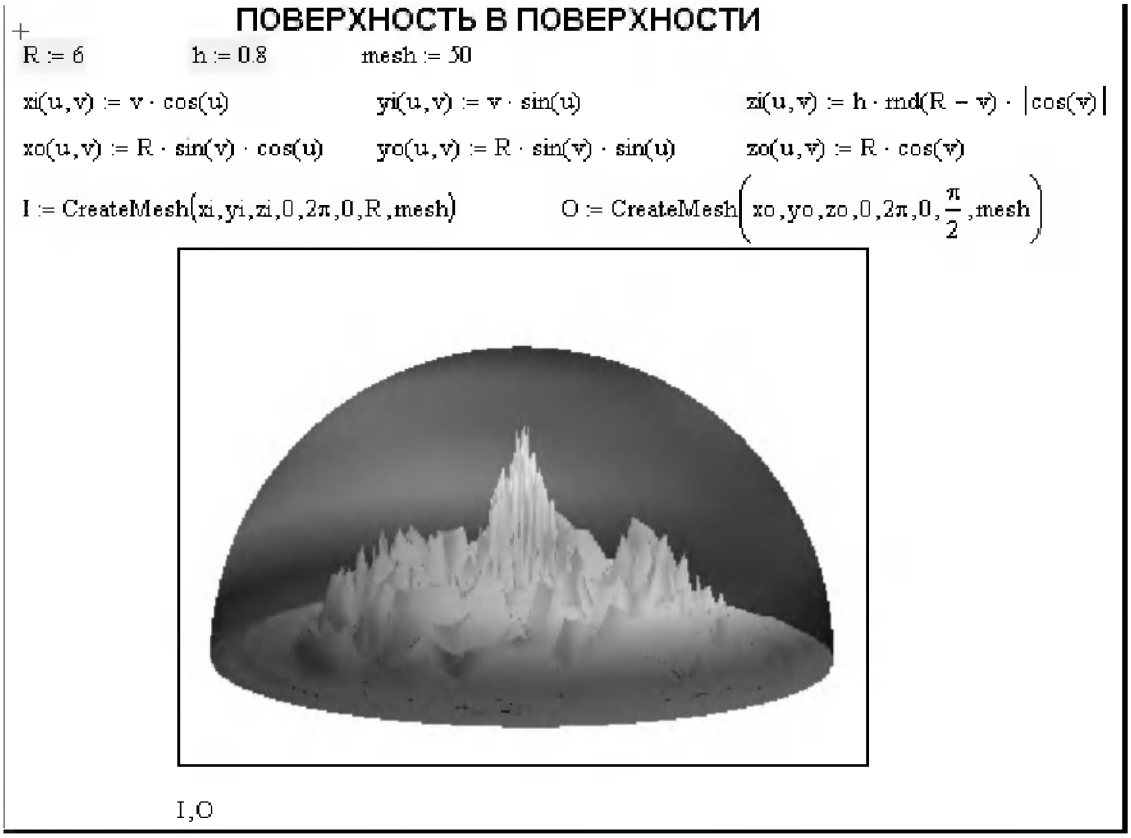


Рис. 13.40. Построение поверхности в поверхности

13.6.6. Поверхность, нанизанная на столбики

Рисунок 13.41 иллюстрирует построение на одном графике ряда объемных столбиков, на которые как бы натянута поверхность (причем натянута так, что столбики как бы прорывают эту поверхность).

13.6.7. Цилиндры, пересекающиеся в пространстве

На рис. 13.42 показаны три цилиндра, пересекающихся в пространстве. При этом один из цилиндров (меньшего диаметра) находится внутри другого цилиндра (большего диаметра).

Полезно обратить внимание на отображение эффекта прозрачности, благодаря чему один из цилиндров (с малым диаметром) отчетливо виден внутри другого цилиндра (с большим диаметром).

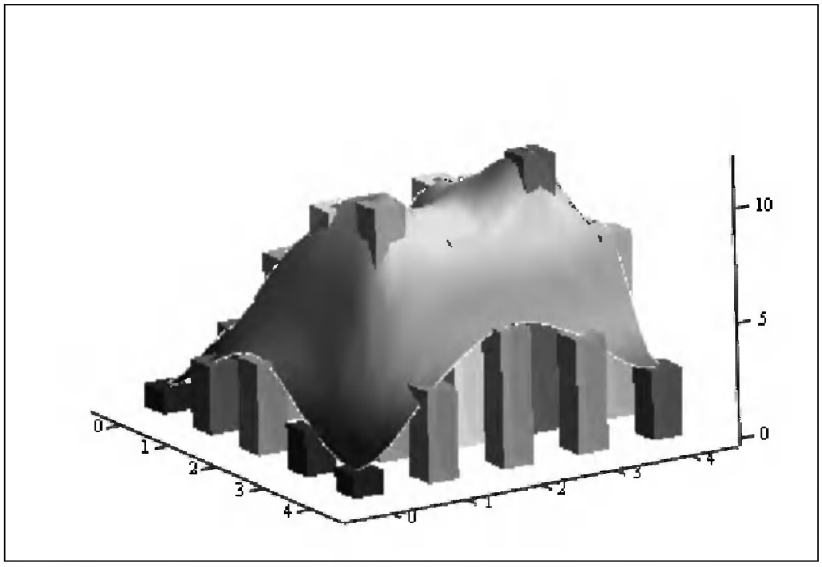
13.6.8. Конусы, пересекаемые плоскостью

Еще один пример комбинированного построения трехмерных графиков представлен на рис. 13.43. На этот раз строятся два конуса с общей вершиной и наклонная плоскость, пересекающая их.

ПОВЕРХНОСТЬ, НАНИЗАННАЯ НА СТОЛБИКИ

$$M := \begin{pmatrix} 1 & 3 & 1 & 5 & 2 \\ 3 & 7 & 4 & 9 & 2 \\ 4 & 10 & 9 & 10 & 6 \\ 2 & 11 & 9 & 12 & 8 \\ 1 & 4 & 6 & 5 & 3 \end{pmatrix} \qquad i := 0..4 \qquad j := 0..4$$

$$X_{i+5\cdot j} := i \qquad Y_{i+5\cdot j} := j \qquad Z_{i+5\cdot j} := M_{i,j}$$



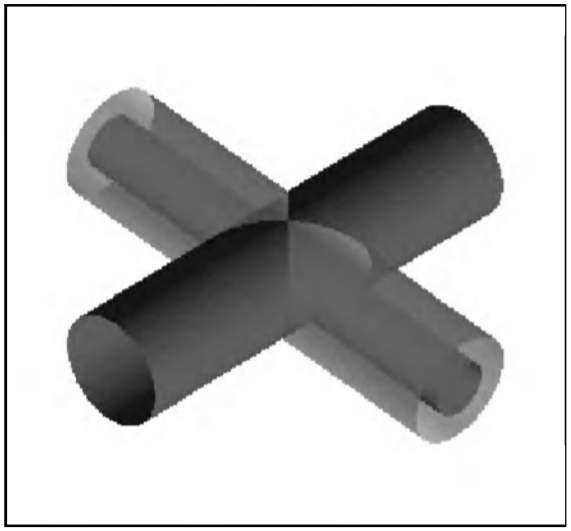
(X,Y,Z),M

Рис. 13.41. Поверхность, нанизанная на столбики

ЦИЛИНДРЫ, ПЕРЕСЕКАЮЩИЕСЯ В ПРОСТРАНСТВЕ

$i := 0..40 \qquad j := 0..40$

$$X_{i,j} := \cos\left(\frac{2\cdot\pi\cdot i}{40}\right) \qquad Y_{i,j} := \sin\left(\frac{2\cdot\pi\cdot i}{40}\right) \qquad Z_{i,j} := \frac{j-20}{5}$$



$$(X,Y,Z),(Z,Y,X),\left(Z,\frac{Y}{2},\frac{X}{2}\right)$$

Рис. 13.42. Цилиндры, пересекающиеся в пространстве

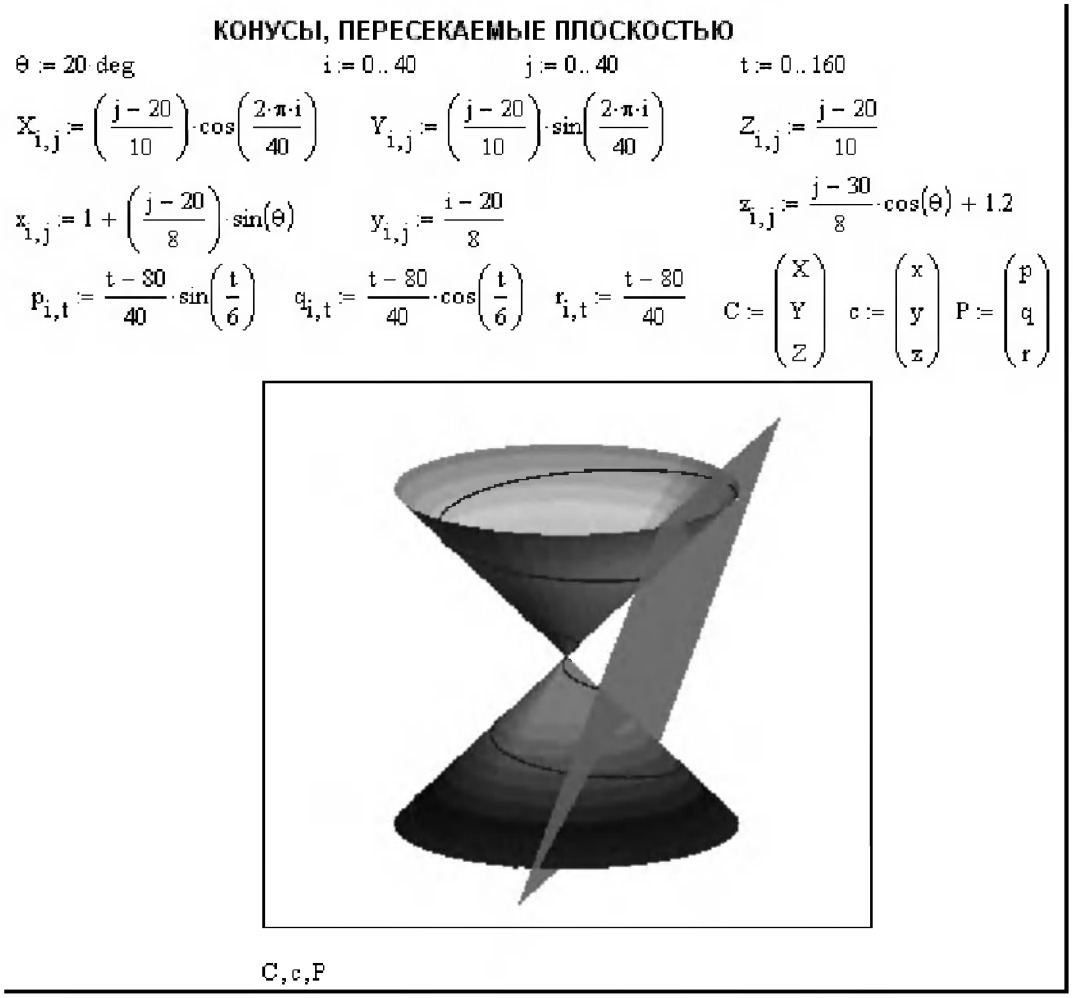


Рис. 13.43. Конусы, пересекаемые плоскостью

13.6.9. Лента Мебиуса

Лента Мебиуса – геометрический объект, давно волнующий воображение начинающих математиков. Если представить себе бумажную полоску, то, перевернув ее на 180 градусов и соединив в кольцо, мы и получим ленту Мебиуса. Рисунок 13.44 показывает построение такой ленты средствами системы Mathcad.

Замечательное свойство ленты Мебиуса – в том, что поверхности полоски становятся неразличимыми. Проведя карандашом линию внутри полоски, мы обнаружим, что линия с одной поверхности плавно переходит в линию на другой поверхности.

13.6.10. Пирамида

Мы уже приводили примеры построения множества геометрических объектов. Дополним их набор построением пирамиды – рис. 13.45.

В данном случае для задания ступенек пирамиды используется встроенная функция ceil. Это не единственный, но интересный прием построения данной фигуры.

ЛЕНТА МЕБИУСА

$v := 0.5$ Ширина кольца

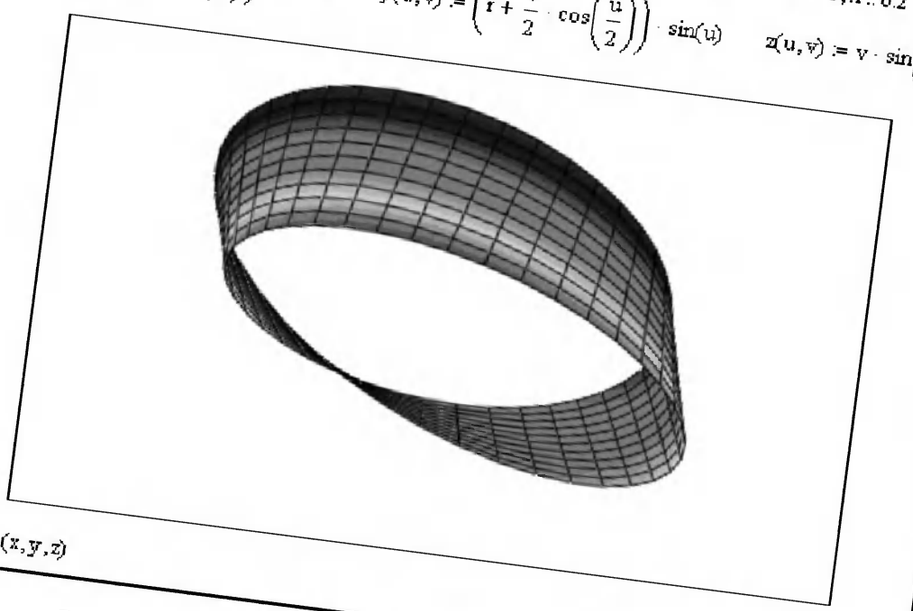
$r := 3$ "Радиус" кольца

$$x(u,v) := \left(r + \frac{v}{2}\cdot\cos\left(\frac{u}{2}\right)\right)\cdot\cos(u)$$

$$y(u,v) := \left(r + \frac{v}{2}\cdot\cos\left(\frac{u}{2}\right)\right)\cdot\sin(u)$$

$u := 0,.1..6.2$

$$z(u,v) := v\cdot\sin\left(\frac{u}{2}\right)$$



(x,y,z)

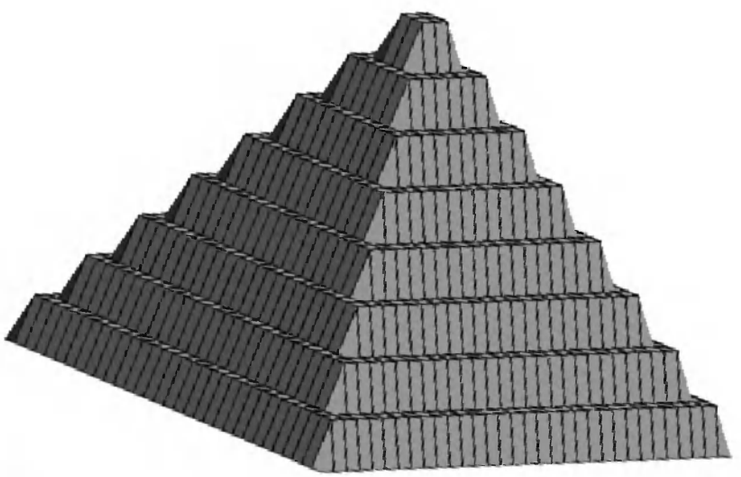
Рис. 13.44. Лента Мебиуса с одним переворотом

ПОСТРОЕНИЕ ПИРАМИДЫ

$N := 8$ Число ступеней пирамиды

$P_{4N,4N} := 0$ $i := 1..4N-1$ $j := 1..4N-1$

$$P_{i,j} := \text{ceil}\left[\frac{1}{2}\cdot\min\left(\begin{pmatrix} i \\ 4N-i \\ j \\ 4N-j \end{pmatrix}\right)\right]$$



P

Рис. 13.45. Пирамида

13.6.11. Пространственные спирали

Ниже представлены исходные данные для довольно сложного алгоритма построения двух пространственных толстых спиралей, образующих фигуру, напоминающую цепь. В отличие от предыдущих примеров, здесь в параметрической форме задается вся фигура. Построение иллюстрирует рис. 13.46 (напоминаем, что формульные блоки в Mathcad обрабатываются слева направо и сверху вниз, поэтому взаимное расположение блоков имеет важнейшее значение).

Рис. 13.46. Фрагмент документа с исходными данными для построения пространственных спиралей

Построенные фигуры, причудливо обвивающие друг друга в пространстве, представлены на рис. 13.47.

Заметим, что все представленные в этом разделе фигуры можно вращать мышью, добиваясь наиболее ясного и наглядного их представления. Приведенные приме-

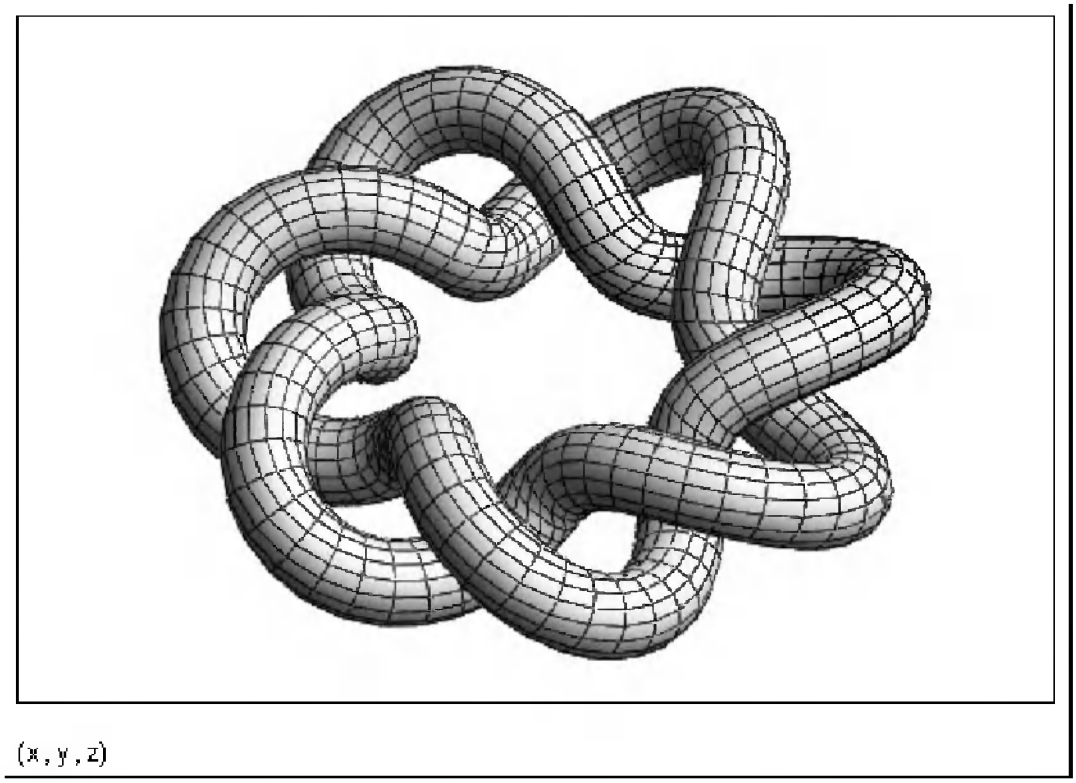


Рис. 13.47. Пространственные спирали, построенные по данным рис. 13.46

ры свидетельствуют о весьма высоком качестве визуализации, обеспечиваемом средствами трехмерной графики Mathcad.

13.6.12. Представление функций двух переменных графиками векторного поля

Иногда желательно представление функций двух переменных графиками векторного поля, составленными из стрелок. Есть два распространенных вида таких графиков: в виде градиентного поля и поля Гамильтона. Их реализация в виде программных модулей представлена на рис. 13.48.

Читателю, желающему разобраться с представленными программными модулями, надо обратиться к главе 10. На рис. 13.49 показаны графики, построенные с помощью модулей и тестовых функций, представленных на рис. 13.48.

Средства построения графиков градиентного и Гамильнонового полей

$$\text{GFPlot}(f,x1,x2,m,y1,y2,n) := \left| \begin{array}{l} dx \leftarrow \dfrac{x2-x1}{m} \\ dy \leftarrow \dfrac{y2-y1}{n} \\ \text{for } i \in 0..m \\ \quad \text{for } j \in 0..n \\ \quad\quad tx \leftarrow x1 + i\cdot dx \\ \quad\quad ty \leftarrow y1 + j\cdot dy \\ \quad\quad M_{i,j} \leftarrow \dfrac{d}{dtx} f(tx,ty) + i\cdot\dfrac{d}{dty} f(tx,ty) \\ M \end{array} \right.$$

$$\text{HFPlot}(f,x1,x2,m,y1,y2,n) := \left| \begin{array}{l} dx \leftarrow \dfrac{x2-x1}{m} \\ dy \leftarrow \dfrac{y2-y1}{n} \\ \text{for } i \in 0..m \\ \quad \text{for } j \in 0..n \\ \quad\quad tx \leftarrow x1 + i\cdot dx \\ \quad\quad ty \leftarrow y1 + j\cdot dy \\ \quad\quad M_{i,j} \leftarrow \dfrac{d}{dty} f(tx,ty) - i\cdot\dfrac{d}{dtx} f(tx,ty) \\ M \end{array} \right.$$

$$f1(x,y) := \sqrt{x^2+y^2}$$

$$f2(x,y) := \sin\left(\sqrt{x^2+\frac{3}{2}\cdot y^2}\right)$$

Рис. 13.48. Программные модели для построения графиков функций двух переменных в виде градиентного поля и поля Гамильтона

13.7. Техника анимации (оживления) графиков

13.7.1. Принципы анимации графиков

Анимация (или «оживление») графиков – возможность, также присущая Mathcad. Принцип анимации достаточно прост. В системе имеется встроенная переменная FRAME, принимающая целочисленные значения (по умолчанию она меняется от 0 до 9 с шагом 1). Любая функция, график которой планируется наблюдать в развитии, должна быть функцией этой переменной, идентифицирующей, по существу, просто номер текущего кадра. Диапазон изменения переменной FRAME задается в диалоговом окне команды **Animate** (Анимация) меню **View** (Вид). Не следует пытаться присваивать переменной FRAME значения иным путем.

При создании анимационных рисунков все кадры строятся с одинаковыми координатами углов и, следовательно, с одинаковыми размерами и одинаковым по-

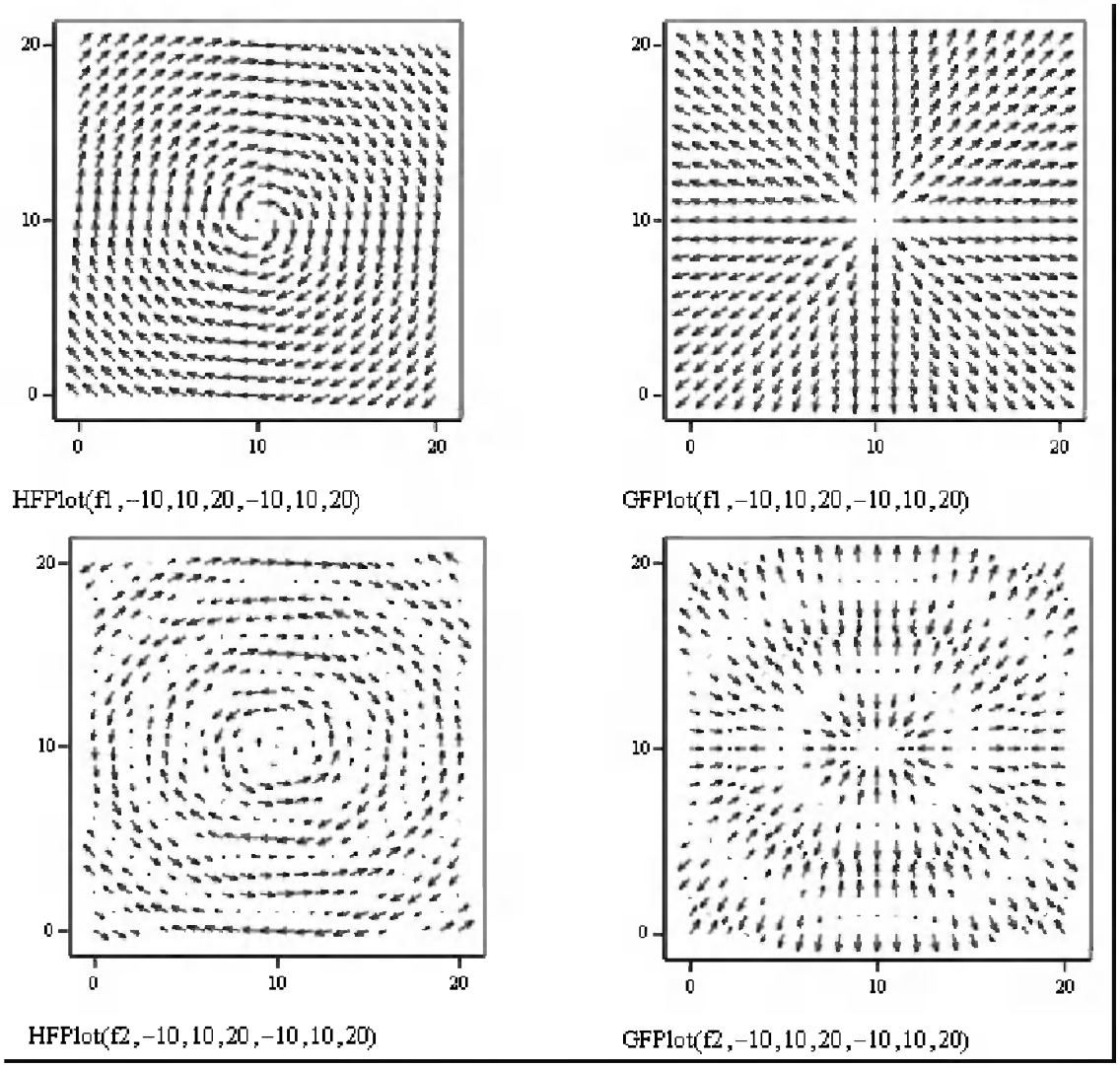


Рис. 13.49. Примеры построения графиков функций двух переменных в виде градиентного поля и поля Гамильтона

ложением на экране. Их вывод один за другим с заданной скоростью (по умолчанию 10 кадров в секунду) и создает «живую» картинку. Просмотр анимации (созданной последовательности кадров) осуществляется с помощью специального проигрывателя.

13.7.2. Подготовка к анимации

Для построения анимационного графика вначале системная переменная задается FRAME как ранжированная, затем задается функция, у которой переменная FRAME должна определять вид каждого кадра анимированного графика. После этого в специальном диалоговом окне задаются три основных параметра: начальное значение переменной FRAME, ее конечное значение и частота смены кадров. Для примера давайте анимируем контурный график некоторой сложной функции. Процедура подготовки к анимации графика выглядит следующим образом.

1. Задайте системную переменную FRAME как ранжированную, например так:

```
g := 2·FRAME
```

2. Задайте функцию, одним из параметров которой (определяющим вид каждого кадра анимации) будет заданная на первом шаге переменная. Пусть это будет следующая функция:

$$A(u,x) := \sin\left(\frac{u}{5}\right)^2 \cdot \cos\left(\frac{u+v}{8}\right)^2 + \exp\left[-\frac{(u-5-g)^2+(v-10-g)^2}{50}\right].$$

3. Выбором команды **Contour Plot** (Контурный график) в подменю **Graph** (График) меню **Insert** (Вставка) выведите шаблон контурного графика (подробно процедура построения контурного графика описана в разделе «Контурный трехмерный график» главы 4).
4. В единственное место ввода шаблона контурного графика введите имя функции, заданной на шаге 2 (это имя `A`).
5. Выбором команды **Animate** (Анимация) в меню **View** (Вид) выведите диалоговое окно для задания параметров анимации.
6. Задайте следующие параметры анимации (как показано на рис. 13.50):
   - начальное значение переменной `FRAME` – 0;
   - конечное значение переменной `FRAME` – 9;
   - частота смены кадров – 10 кадров в секунду.
7. Выделите мышью нужный фрагмент изображения. Можно выделить любую часть графика и даже расположенные около него объекты, например

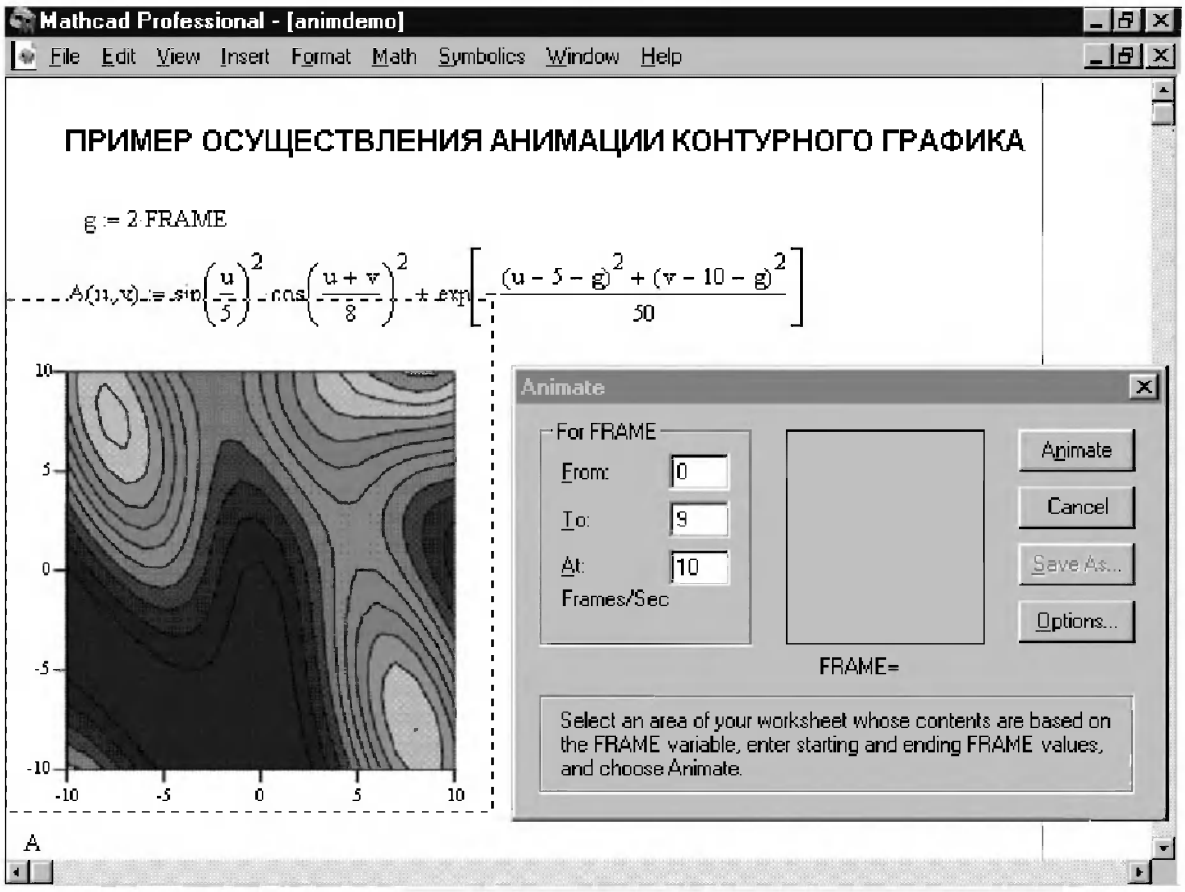


Рис. 13.50. Подготовка к построению анимированного графика

формулы. В этом случае они тоже будут отображаться при воспроизведении анимации.

На этом подготовительные операции к созданию анимации заканчиваются.

13.7.3. Создание кадров изображения

Щелчок на кнопке **Animate** (Анимировать) окна параметров анимации приводит к созданию последовательности анимационных кадров. При этом эти кадры будут видны в области просмотра окна, а под этой областью можно наблюдать изменение переменной FRAME.

С помощью кнопки **Options** (Параметры) можно выбрать формат сжатия видеофайлов и систему работы с ними. Кроме Microsoft Video 1.1, возможна работа и с рядом других видеосистем, разумеется, если они установлены. Чем больше конечное значение переменной FRAME и выше частота кадров, тем более плавно происходит считывание, но увеличивается объем AVI-файлов.

По окончании создания серии кадров для анимации в окне появится проигрыватель анимационных кадров (рис. 13.51).

На этом создание анимационного рисунка заканчивается, и можно приступить к его просмотру.

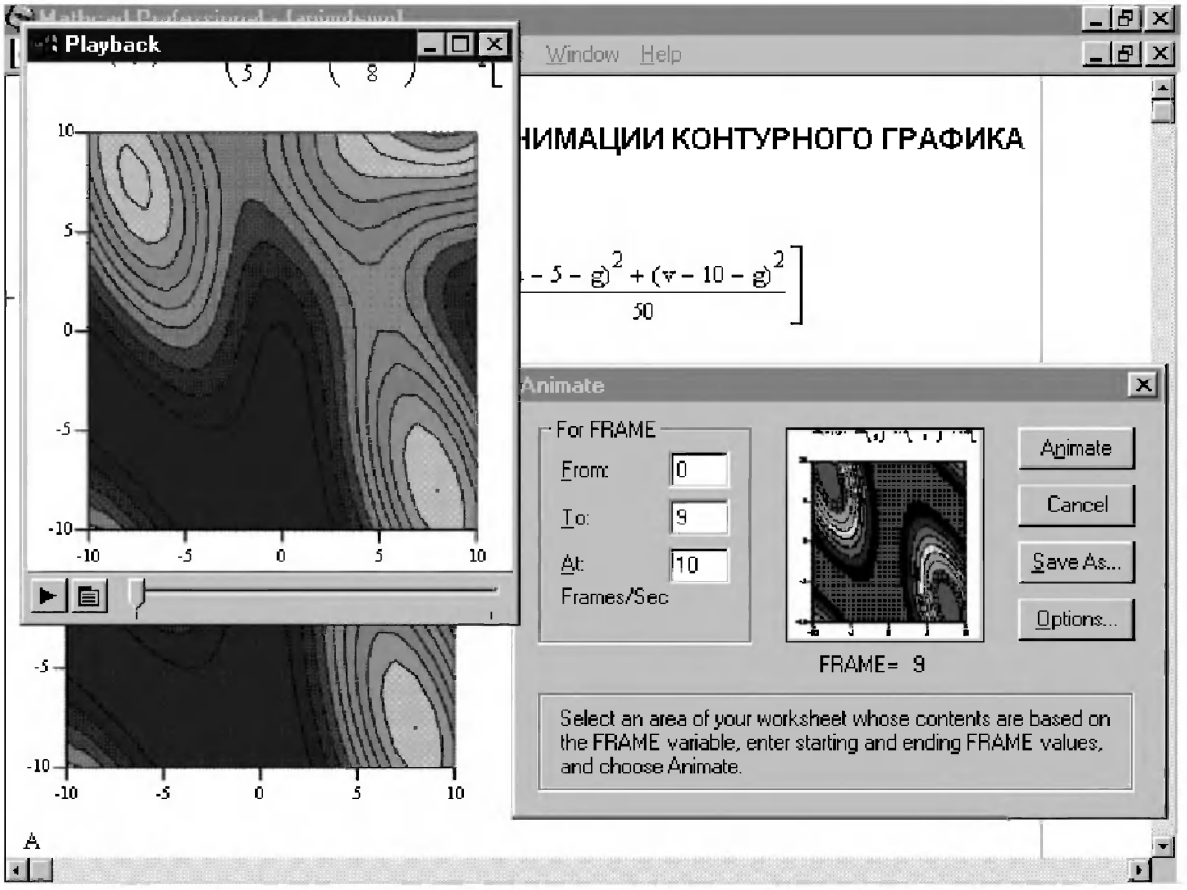


Рис. 13.51. Просмотр анимационных кадров с помощью проигрывателя

13.7.4. Воспроизведение анимированного рисунка

Для просмотра анимированных рисунков используется специальный проигрыватель Playback. Щелкнув в окне проигрывателя на кнопке с изображением треугольника, можно наблюдать изменение графика во времени.

Щелкнув на кнопке **Save As** (Сохранить как) в диалоговом окне **Animate** (Анимация), можно вызвать стандартное окно записи файлов на диск. В этом окне можно найти папку, в которую будет помещен записываемый файл. Файл записывается с расширением `.avi`, принятым для файлов программной видеосистемы Microsoft Video for Windiows. До записи в файл кадры видеоролика хранятся в оперативной памяти ПК, что ограничивает их число.

При необходимости размер окна проигрывателя можно уменьшить в два раза или, напротив, увеличить в два раза. Это делается с помощью меню, появляющегося при щелчке в окне проигрывателя на кнопке с изображением экрана (вторая слева).

При воспроизведении анимированных рисунков рекомендуется отключить все параметры автоматического масштабирования графиков. Автоматическое изменение масштаба подчас ведет к скачкообразному изменению размеров графика, хотя на деле он должен меняться без скачков, точнее говоря, со скачками, но определяемыми только изменением переменной `FRAME` = 0, 1, 2, 3 и т. д. Скачки размеров фигур в ходе воспроизведения анимации порой могут озадачить даже опытных пользователей, но куда хуже, если не очень опытные пользователи принимают их «за чистую монету».

13.7.5. Вызов проигрывателя

Теперь перейдем к описанию самой «крутой» возможности графики системы Mathcad – просмотру произвольных видеофильмов. Это может быть боевик, который позволяет немного скрасить выполнение нудных вычислений, или специально подобранная вставка для пояснения сути электронного урока.

Любой видеофайл с расширением `.avi` (не обязательно с описанной выше анимированной графикой) может быть просмотрен с помощью проигрывателя, запускаемого командой **Playback** (Воспроизвести) меню **View** (Вид). При этом проигрыватель появляется в окне Mathcad вначале в уменьшенном виде (рис. 13.52).

Как уже отмечалось, проигрыватель управляется всего двумя кнопками. Первая (с изображением треугольника) запускает проигрыватель. Однако для воспроизведения какого-либо видеоклипа его файл должен быть загружен. Для загрузки файлов используется вторая кнопка. Она выводит на экран меню, показанное на рис. 13.53.

Ниже перечислены пункты этого меню:

- **View** (Вид) – подменю управления размером окна просмотра;
- **Volume** (Громкость) – ползунок (регулятор) управления громкостью (если AVI-файл содержит звук);

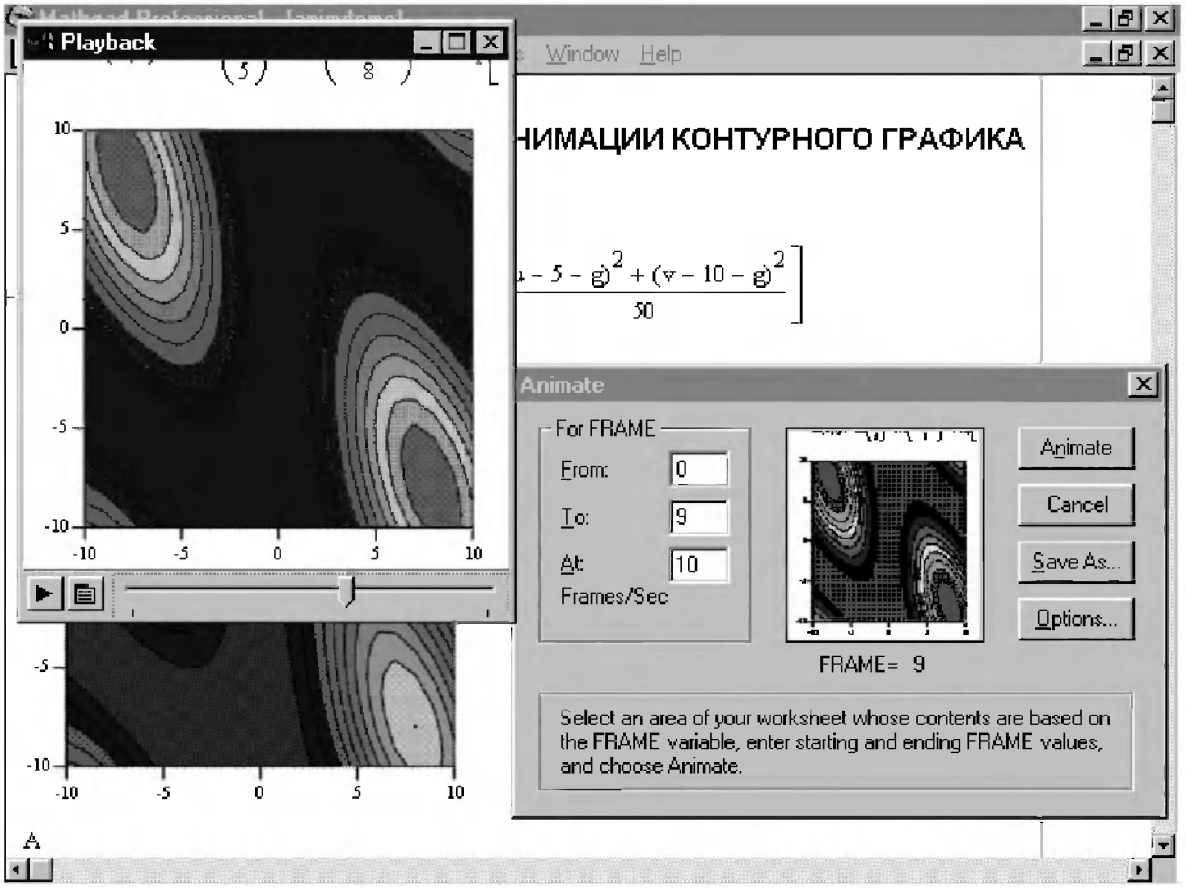


Рис. 13.52. Проигрыватель после вызова командой Playback

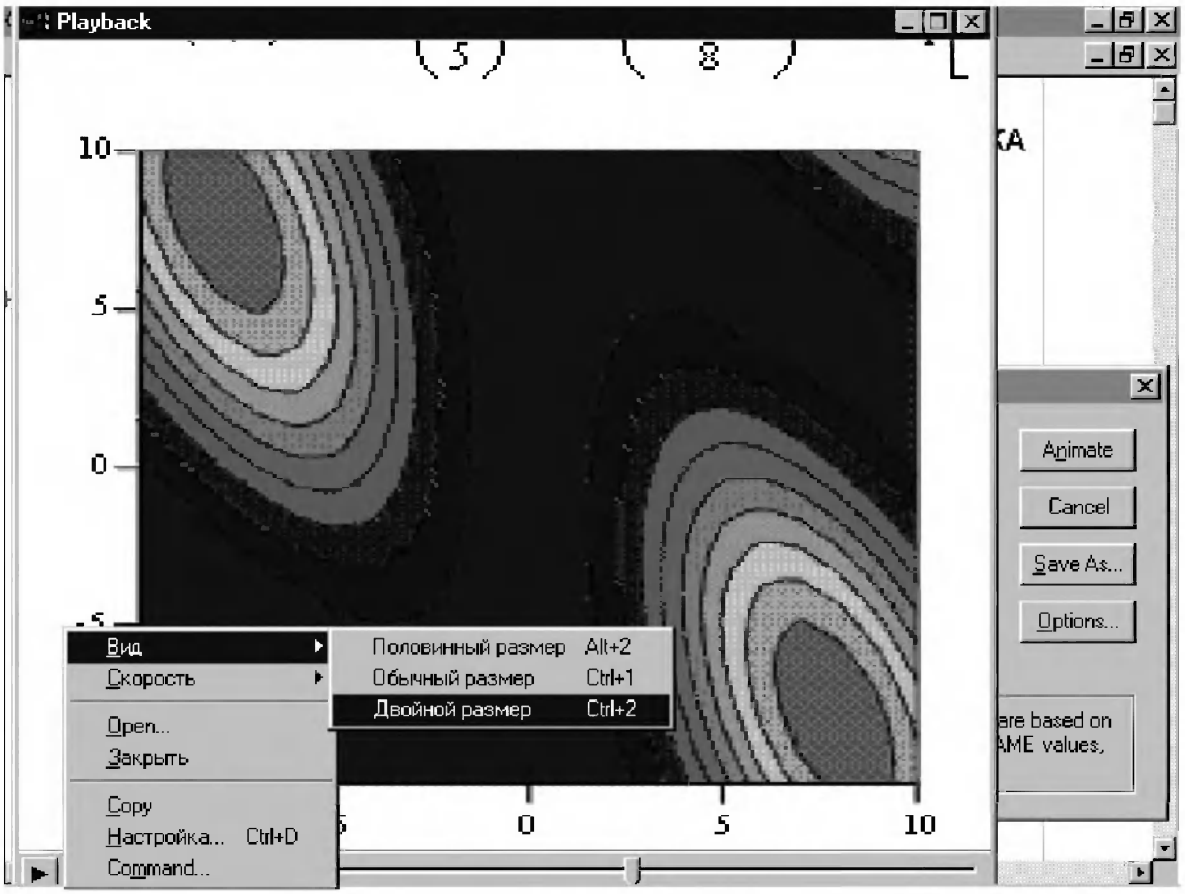


Рис. 13.53. Воспроизведение видеофильма проигрывателем Playback в среде MathCAD

- **Speed** (Скорость) – ползунок (регулятор) управления скоростью воспроизведения;
- **Open** (Открыть) – вывод стандартного диалогового окна загрузки файлов;
- **Close** (Закрыть) – закрытие текущего окна проигрывателя;
- **Copy** (Копировать) – копирование фильма в буфер обмена;
- **Configure** (Настройка) – вывод диалогового окна настройки;
- **Command** (Команда) – управление из командной строки.

Поскольку проигрыватель Playback не относится к системе Mathcad, имена команд зависят от его версии. В данном случае приводятся команды как для локализованной, так и для нелокализованной версий проигрывателя. Некоторые из команд (например, команда отображения регулятора громкости) появляются только в том случае, когда они необходимы.

Выбор команды **Open** (Открыть) позволяет открыть стандартное диалоговое окно для загрузки файлов. Кроме кнопок, в окне проигрывателя расположен индикатор воспроизведения в виде шкалы с ползунком.

На рис. 13.54 показано окно **Video Properties** (Свойства видео), появляющееся при выборе команды **Configure** (Настройка) или при нажатии комбинации клавиш **Ctrl+D**. Это окно позволяет с помощью переключателей задать видеорежим:

- **Video Mode** (В окне) – обычный для Windows оконный режим (размер окна можно выбрать в раскрывающемся списке);
- **Full Screen** (Во весь экран) – изображение занимает весь экран.

Рис. 13.54. Диалоговое окно свойств видео

Применение проигрывателя заметно облегчает построение эффектных обучающих программ, в частности подготовку видеороликов по работе с самой системой. К сожалению, AVI-файлы требуют достаточно большого объема памяти на жестком диске, поэтому приходится делать их довольно короткими. В то же время множество готовых видеофильмов можно найти на компакт-дисках и в Интернете.

13.8. Графика системы Derive

13.8.1. Позиция меню Windows

В Derive for Windows 5/6 графики строятся в отдельных окнах с помощью команд главного меню или панели инструментов. Специальных функций или операторов для их построения просто нет. Для открытия новых окон в подменю позиции `Window` (Окно) главного меню Derive 6 расположены основные команды управления окнами и элементами пользовательского интерфейса:

- **New 2D-plot Window** – создание нового окна для двумерной графики;
- **New 3D-plot Window** – создание нового окна для трехмерной графики;
- **Customize** – вывод окна установок пользователя.

Имеется также ряд типовых команд для установки взаимного расположения окон и вывода панели инструментов и статуса:

- **Cascade Ctrl+Shift+C** – каскадное расположение окон;
- **Tile Horisontally Ctrl+Shift+H** – расположение окон по горизонтали;
- **Tile Vertically Ctrl+Shift+V** – расположение окон по вертикали;
- **Display Tabs** – вывод панели с вкладками окон.

Набор этих команд у предшествующих реализаций Derive несколько иной. Рассмотрим возможности задания и вывода окон графики, поскольку работа с окном выражений уже описывалась в главе 1.

13.8.2. Построение и форматирование двумерного графика

Для построения двумерного графика достаточно ввести выражение для графика в панели ввода выражений (снизу рис. 13.55), затем командой **New 2D-plot Window**, или соответствующей кнопкой панели инструментов, открыть окно двумерной графики и, наконец, активировать кнопку построения двумерного графика.

Окно двумерного графика имеет свою инструментальную панель (на месте панели ввода команд) и свои кнопки для свертывания окна в бирку, управления размерами и закрытия. Имеет оно и свое главное меню, несколько отличное по набору позиций и команд от главного меню окна выражений.

Окно графики имеет графический маркер в виде крестика, который может перемещаться клавишами управления курсора или мышью. Перемещения будут плавными и медленными. Для быстрых, но резких перемещений надо нажимать клавиши управления курсором одновременно с клавишей **Ctrl**. Маркер служит

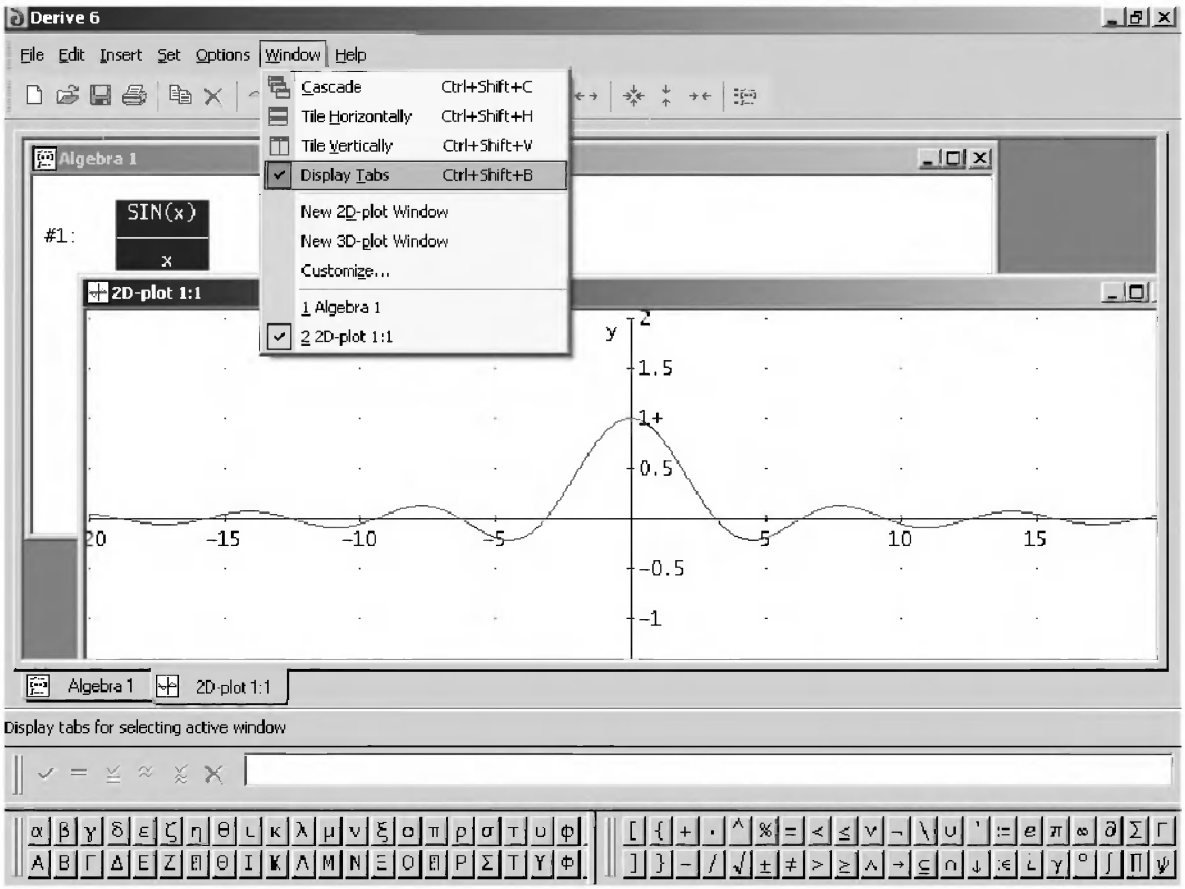


Рис. 13.55. Пример построения двумерного графика функции sin(x)/x

для приближенного определения координат характерных точек графика и может иметь различные режимы перемещения. Для вывода панели просмотра координат маркера и центра графика служит опция **Follow Cross** в позиции меню **Options**.

Особенно интересен режим **Trace Mode** – при этом графический курсор превращается из крестика в маленький квадратик и устанавливается на кривую графика, соответствующую выделенному выражению. При этом он мышкой или клавишами курсора перемещается всегда точно по кривой. Это повышает точность анализа особых точек кривых.

Перемещение маркера мышью, к сожалению, нельзя сделать плавным, поскольку фиксация положения маркера производится в момент щелчка левой клавишей мышки. Однако клавишами управления курсором можно обеспечить плавное движение маркера (при нажатой клавише **Ctrl** оно ускоряется). Графический маркер удобно использовать также для центровки графика – достаточно поместить его в точку графика, которую желательно сделать центром и нажать кнопку центрирования, как график тут же будет перестроен.

Таким образом можно приближенно найти экстремумы, или нули функции. Их затем можно использовать для уточнения численными методами.

Панель инструментов окна двумерной графики имеет ряд кнопок. С их назначением несложно ознакомиться, наведя на нужную кнопку курсор мыши, нажав ее левую кнопку, и, держа ее нажатой, начать перемещать мышь – появится всплывающая подсказка о назначении кнопки. Отжав левую кнопку мыши,

можно увидеть появление окна уточнения размера выделенного фрагмента изображения – рис. 13.56.

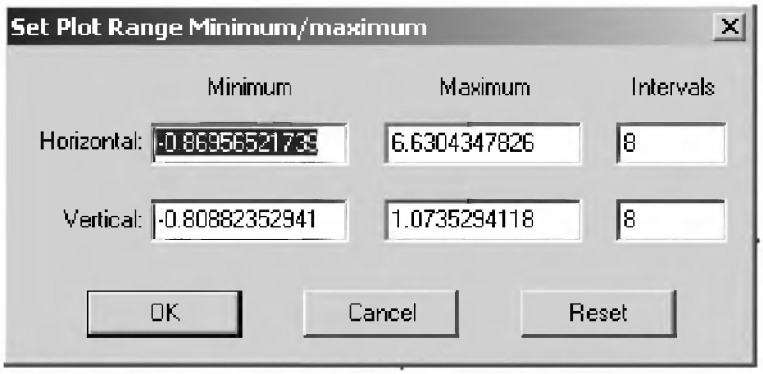


Рис. 13.56. Уточнение размера выделенного фрагмента рисунка

Уточнение размера выделенной части рисунка полезно для получения удобных значений чисел у отметок координатных осей. Если этого не сделать, то у таких отметок будут проставляться довольно длинные и неудобные для применения числа, как это показано на рис. 13.57.

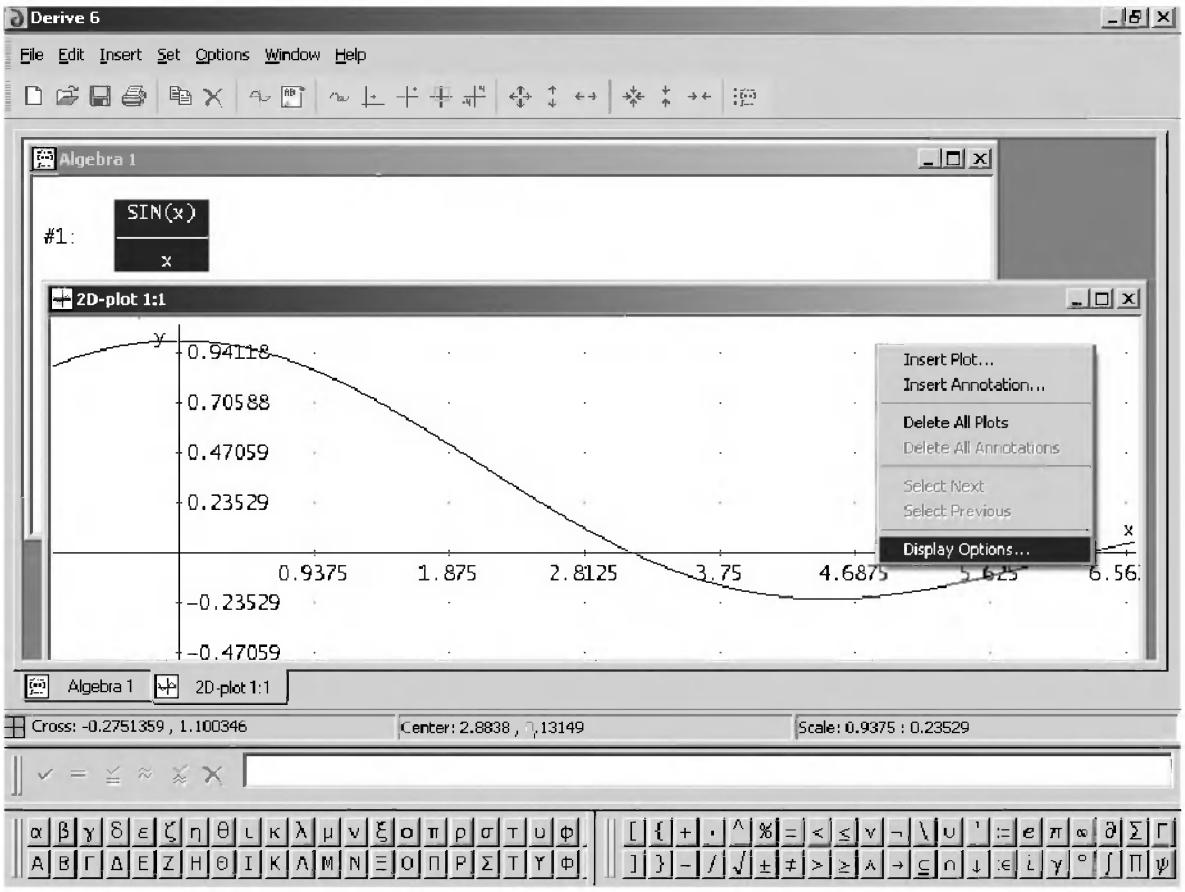


Рис. 13.57. График выделенного фрагмента

Фактически данная возможность реализует графическую «лупу», позволяющую просматривать небольшие фрагменты графиков. Это очень полезно для выявления и изучения их тонких особенностей.

На рис. 13.57 показано также контекстное меню правой клавиши мыши. Стоит обратить внимание на команду **Display Options...**. Эта команда есть и в позиции **Options** меню. Она выводит окно форматирования двумерных графиков, показанное на рис. 13.58 с открытой вкладкой **Axes**.

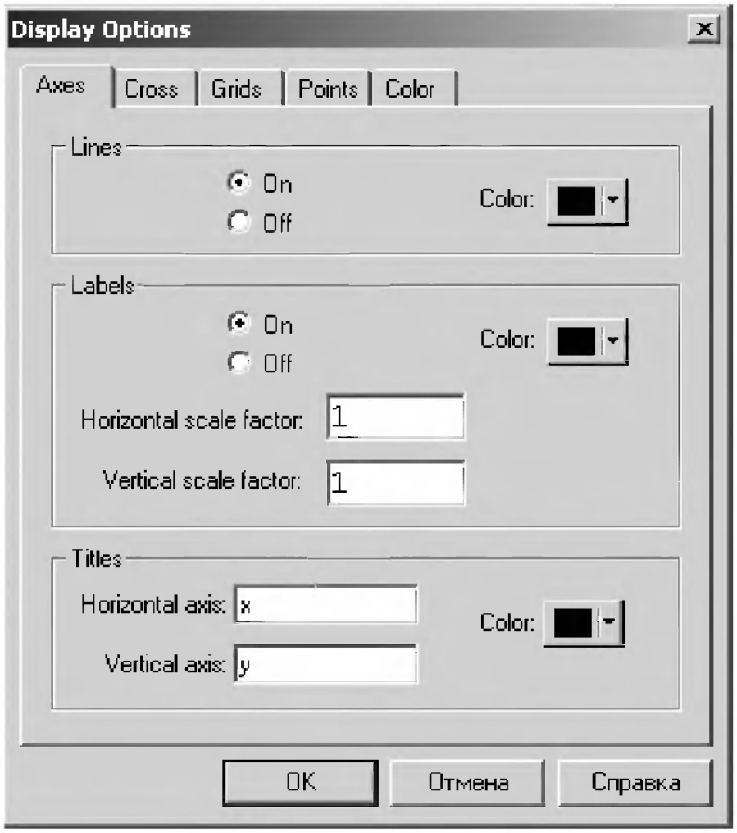


Рис. 13.58. Окно форматирования двумерных графиков

Окно форматирования имеет следующие вкладки:

- **Axes** – установки осей (ввод/вывод их линий, меток и надписей по осям);
- **Cross** – включение/выключение крестообразного маркера;
- **Grids** – включение/выключение масштабной сетки;
- **Points** – установка соединений между точками и размеров точек;
- **Color** – установка цвета точек и линий, а также цвета фона (возможна и установка рисунка для фона).

13.8.3. Графики функций в полярной системе координат

При построении графика в полярной системе координат надо задать функцию $r(\vartheta)$ и открыть окно двумерной графики. Кроме того, надо в окне **Coordinate System** (позиция **Set** меню) установить полярную систему координат (Polar). При

пуске построения графика откроется окно **Parametric Plot** Parameter, показанное на рис. 13.59, по которому можно уточнить параметры построения (если надо).

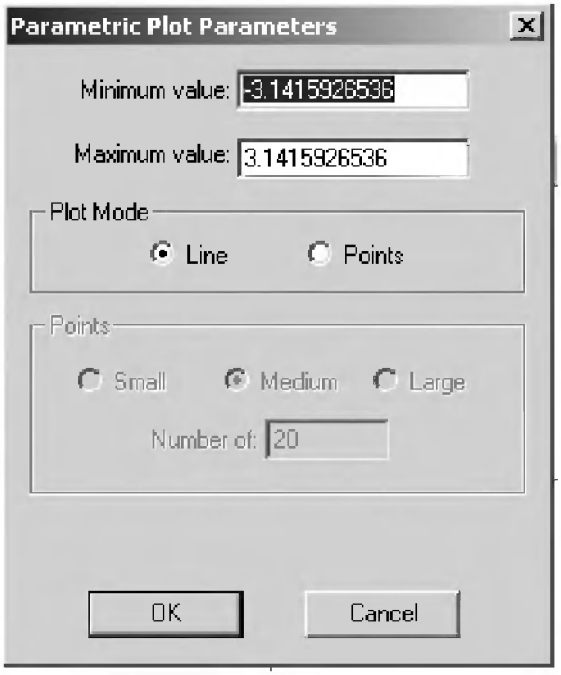


Рис. 13.59. Окно Parametric Plot

Нажатие кнопки **OK** строит график – рис. 13.60. Назначение кнопок панели инструментов и команд главного меню в данном случае такое же, как для двумерных графиков.

Графики в полярной системе координат можно строить также при параметрическом задании функций в виде [f1(x), f2(y)]. Пример такого построения дан на рис. 13.61.

13.8.4. Построение трехмерных графиков

Для построения графиков поверхностей надо задать функцию двух переменных вида *z(x,y)* или *f(x,y)*. После выделения строки с такой функцией надо использовать команду **New 3D-plot**. Она создает новое окно для трехмерной графики со своим главным меню и инструментальной панелью.

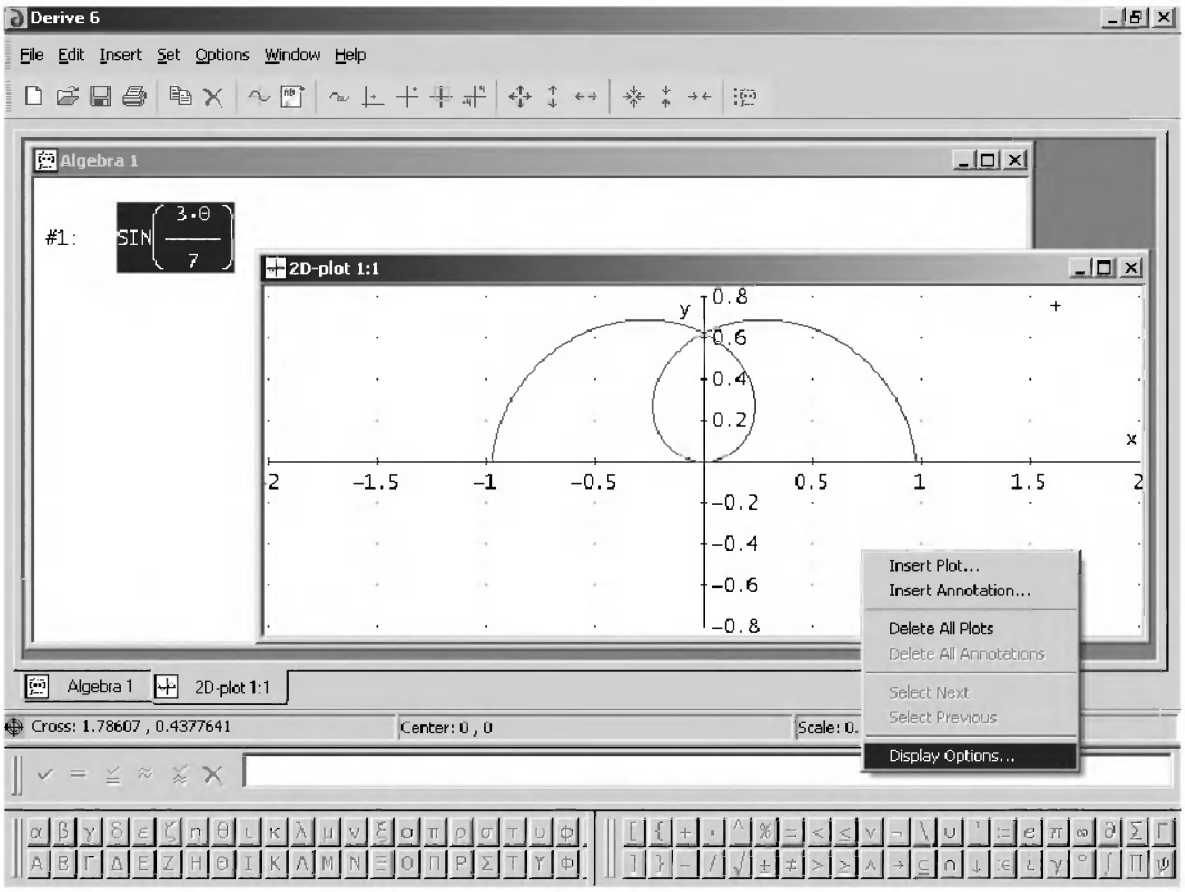


Рис. 13.60. Пример построения графика в полярной системе координат

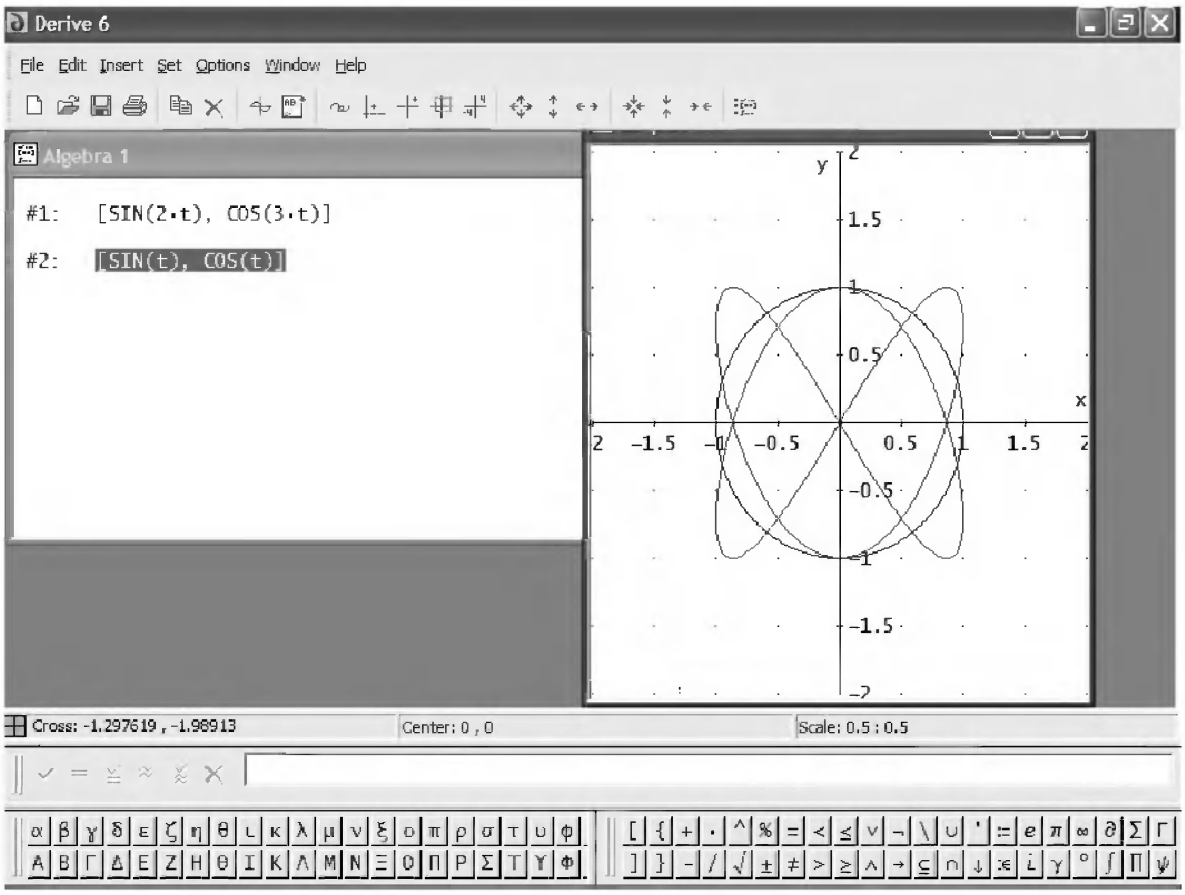


Рис. 13.61. Пример построения графика параметрически заданных функций

Эту же команду можно исполнить, нажав кнопку в панели инструментов окна выражений с изображением трехмерной системы координат.

После вывода окна графика надо в его инструментальной панели нажать аналогичную кнопку и график можно построить. Для наглядности можно рядом или поверх него вывести окно с исходным выражением. Осуществив все эти операции, можно получить график, представленный на рис. 13.62.

График поверхности имеет вид *проволочного каркаса*, но невидимые линии его устранены алгоритмом удаления невидимых линий. Этот алгоритм включен по умолчанию, поскольку обеспечивает лучший вид графиков, чем у графиков со всеми видимыми линиями. При необходимости, однако, алгоритм удаления невидимых линий может быть отключен.

В позиции **Edit** меню окна графики предусмотрены новые команды стирания всех графиков и всех аннотаций. Новыми возможностями Derive 5/6 стали выделение аннотации прямоугольником и их плавное перемещение мышью в любое место, а также редактирование аннотации.

Команда **Create Annotation** (создание аннотации) из подменю **Edit** удалена. Она перешла в подменю **Insert** меню графического окна. В это подменю переведены и позиции построения 2D- и 3D-графиков, а также вставки объектов.

Трехмерная графика Derive 5/6 существенно продвинута по сравнению с предшествующими версиями и обеспечивает все основные возможности такой

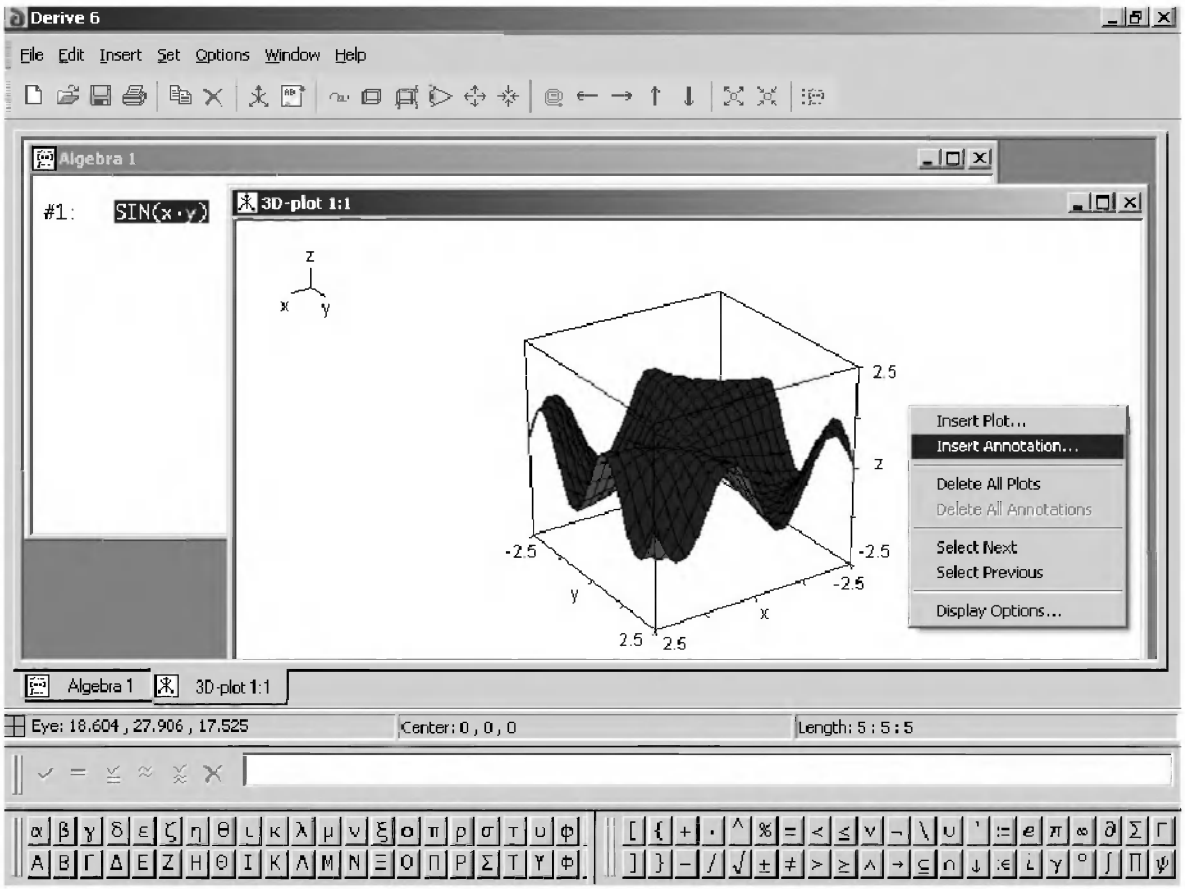


Рис. 13.62. Пример построения трехмерного графика

графики, применяемые в современных системах компьютерной математики. В частности, введена возможность функциональной окраски трехмерных фигур и поверхностей.

Заметные улучшения введены в управление средствами трехмерной графики. Панель инструментов окна 3D-графики пополнилась новыми кнопками для поворота объектов вокруг осей x и y, приближения и удаления объекта и анимации путем вращения объекта вокруг оси z. Все это дает удобные и легкие в освоении средства для манипуляции с 3D-графиками.

Из новаций в меню окон 3D-графики надо отметить расширение установок в позиции **Set**. Наиболее важным и новым тут стала возможность выбора одной из трех координатных систем. Это следующие системы: **Rectangular** (Прямоугольная), **Spherical** (Сферическая) и **Cylindrical** (Цилиндрическая). Таким образом, есть возможность перестройки графика в любой из этих координатных систем.

Заметной модификации подверглось меню опций 3D-графики **Options**. Новыми опциями являются опции анимации (вращения) 3D-объекта, упрощения до построения графика и аппроксимации (вычисления в численном виде) до построения графика.

В целом графика Derive 5/6 представляет пользователю (как педагогу, так и учащемуся) поистине неограниченные возможности в графической визуализации сложных математических понятий и образов.

Внимание! *Обычная графика системы Derive удовлетворяет минимальным потребностям в визуализации математических понятий. Представлены основные типы графиков: функции одной переменной, параметрически заданных функций двух переменных и графика поверхностей.*

13.8.5. Построение 3D-фигур с функциональной окраской и параметрическим заданием

Интересной является возможность построения 3D-графиков при параметрическом задании их списком из трех функций. Каждая из них определяет координату x, y и z поверхности как функцию от длины радиус вектора и углов s и t – см. пример на рис. 13.63.

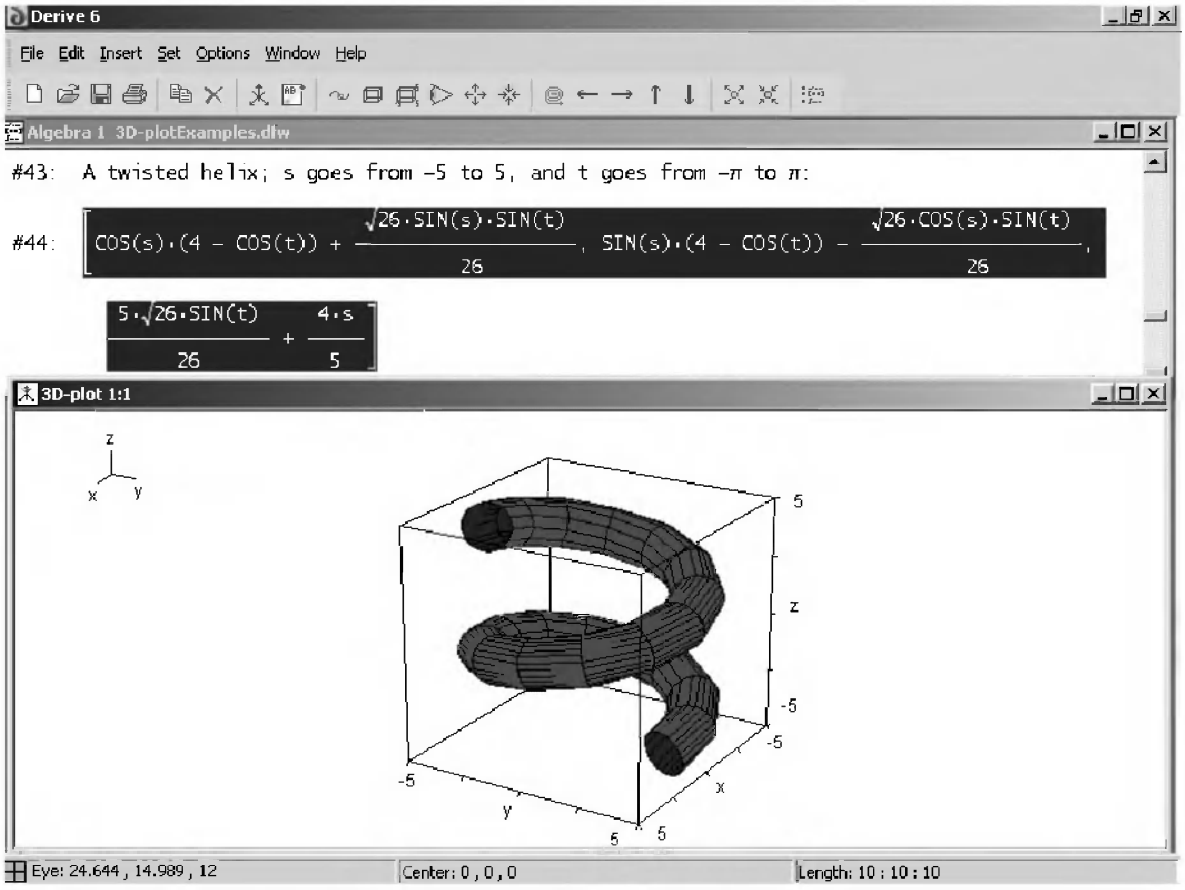


Рис. 13.63. Пример построения 3D-фигуры, заданной параметрически

Как видно из приведенного рисунка, построенная 3D-фигура представляет собой отрезок толстой спирали в пространстве. По наглядности этот график немного уступает подобным графикам, которые строят более мощные системы компьютерной математики, например Mathcad, Maple и Mathematica.

13.8.6. Построение пересекающихся в пространстве фигур

Еще одна интересная возможность графики Derive 5/6 – построение пересекающихся в пространстве поверхностей или трехмерных фигур. Для этого достаточно задать список выражений для таких фигур в виде функций двух переменных x и y, задающих координаты точек z(x,y) каждой из фигур. Рисунок 13.64 показывает пример такого построения для двух поверхностей (см. также рис. 1.16, на котором график размещен в окне алгебры).

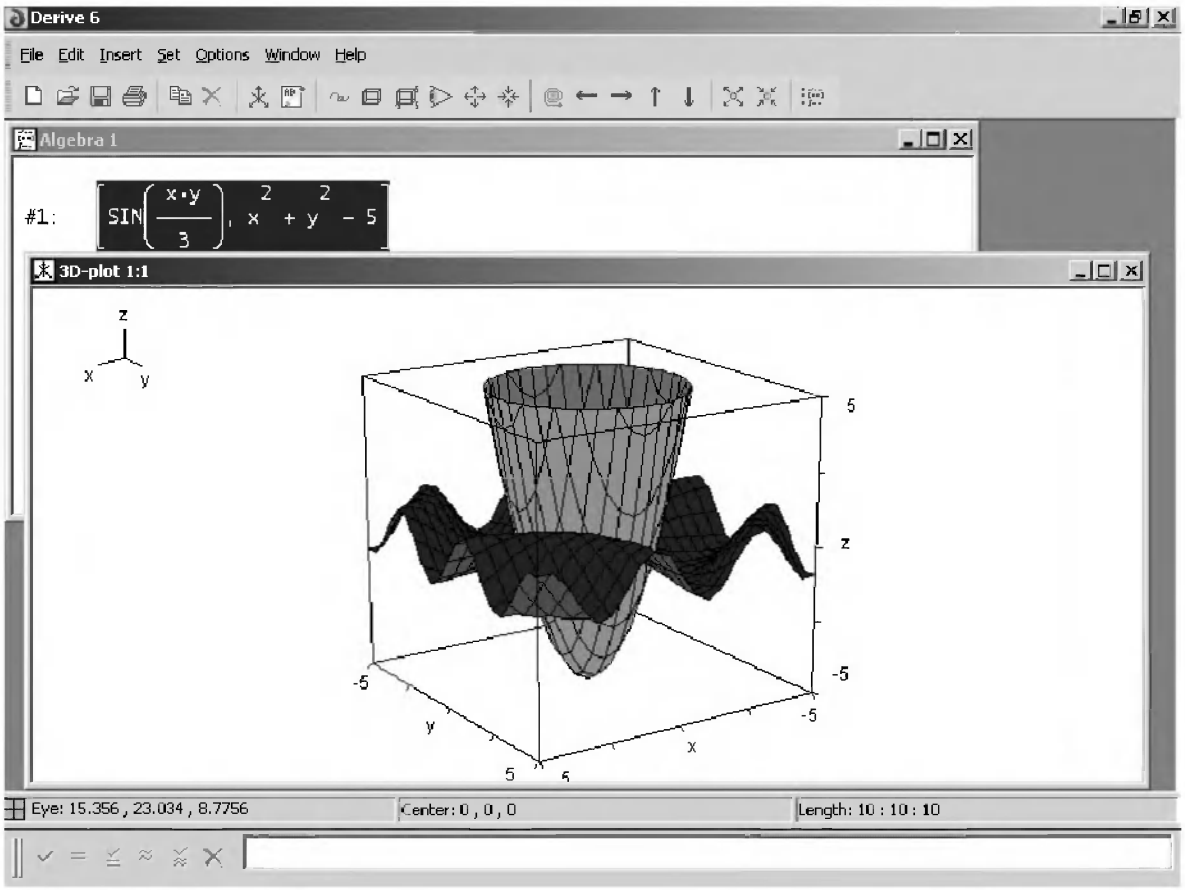


Рис. 13.64. Построение двух пересекающихся в пространстве поверхностей

В данном случае построены графики поверхности sin(x*y/4) и смещенной вниз пространственной параболы (x^2+y^2). Нетрудно заметить, что Derive 6 неплохо справляется с построением линий пересечения поверхностей в пространстве и эффективно использует алгоритм удаления невидимых частей поверхностей. Это делает графики (особенно при использовании функциональной окраски) весьма наглядными и полезными при проведении уроков по стереометрии.

Интересно, что эту возможность можно сочетать с последовательным построением трехмерных объектов на одном рисунке. Например, можно построить график поверхности sin(x*y), а затем три пространственные параболы со сдвигом по оси z, равным –5, –3 и без сдвига. Можно продолжить эксперименты с построени-

ем ряда фигур в пространстве. Рисунок 13.65 показывает построение 5 поверхностей в пространстве. Обратите внимание на то, что в ограниченную область пространства попадают только части некоторых фигур. Это придает графику не совсем обычный вид.

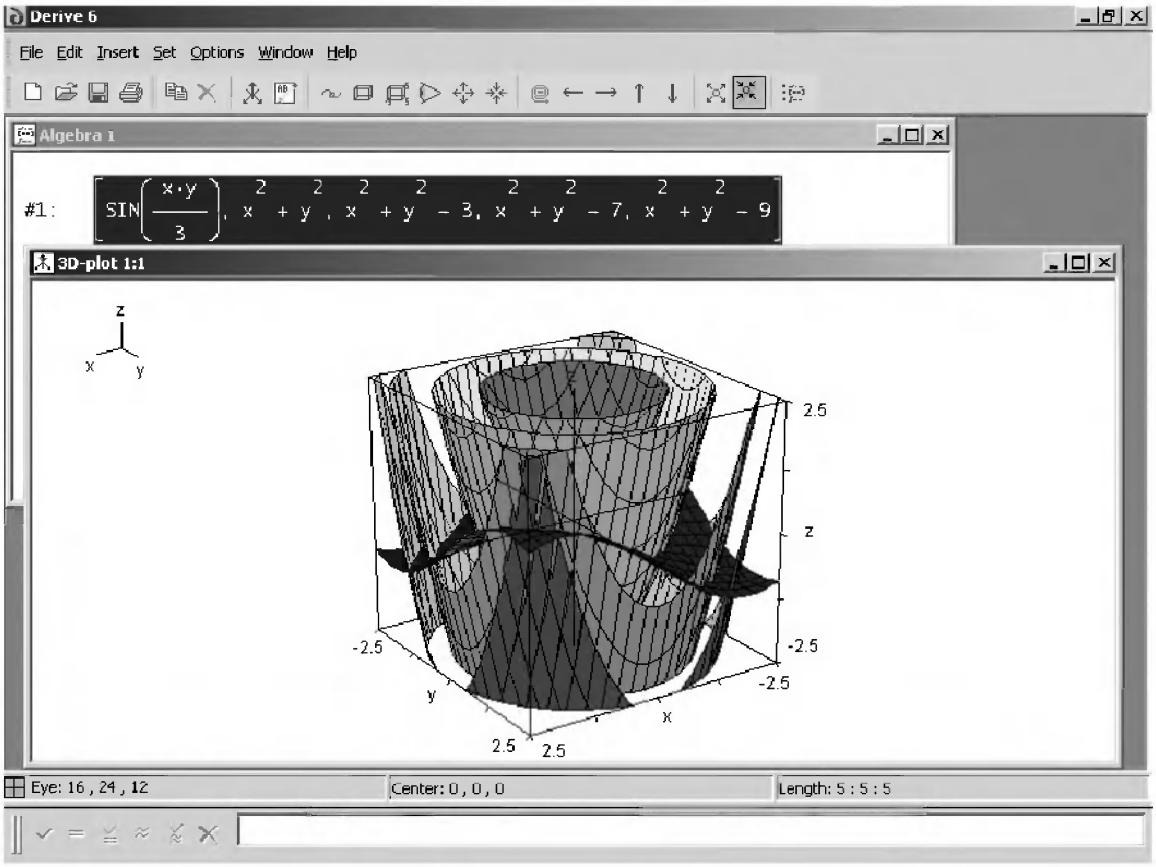


Рис. 13.65. Построение 5 поверхностей в пространстве

13.8.7. Двумерная графика с окраской по неравенствам

Еще один очень любопытный и полезный вид двумерной графики есть у системы Derive – это графики с окраской отдельных их сегментов, задаваемых с помощью неравенств. Пример такого графика приведен на рис. 13.66.

Как нетрудно заметить, в выражении для построения такого графика заданы синусоиды с амплитудами, равными 0.5 и 1. При этом ограничения записаны как положительной, так и отрицательной полуволнами синусоиды. Область между этими двумя синусоидами строится окрашенной.

Следующий рисунок показывает построение графика функции sin(x)/x с закраской областей, ограниченных графиком функции и осью абсцисс x, – рис. 13.67. На этом рисунке показана еще одна возможность Derive 6 – задание аннотации к графику с помощью специального окна **2D-plot Window Annotation**. Для выво-

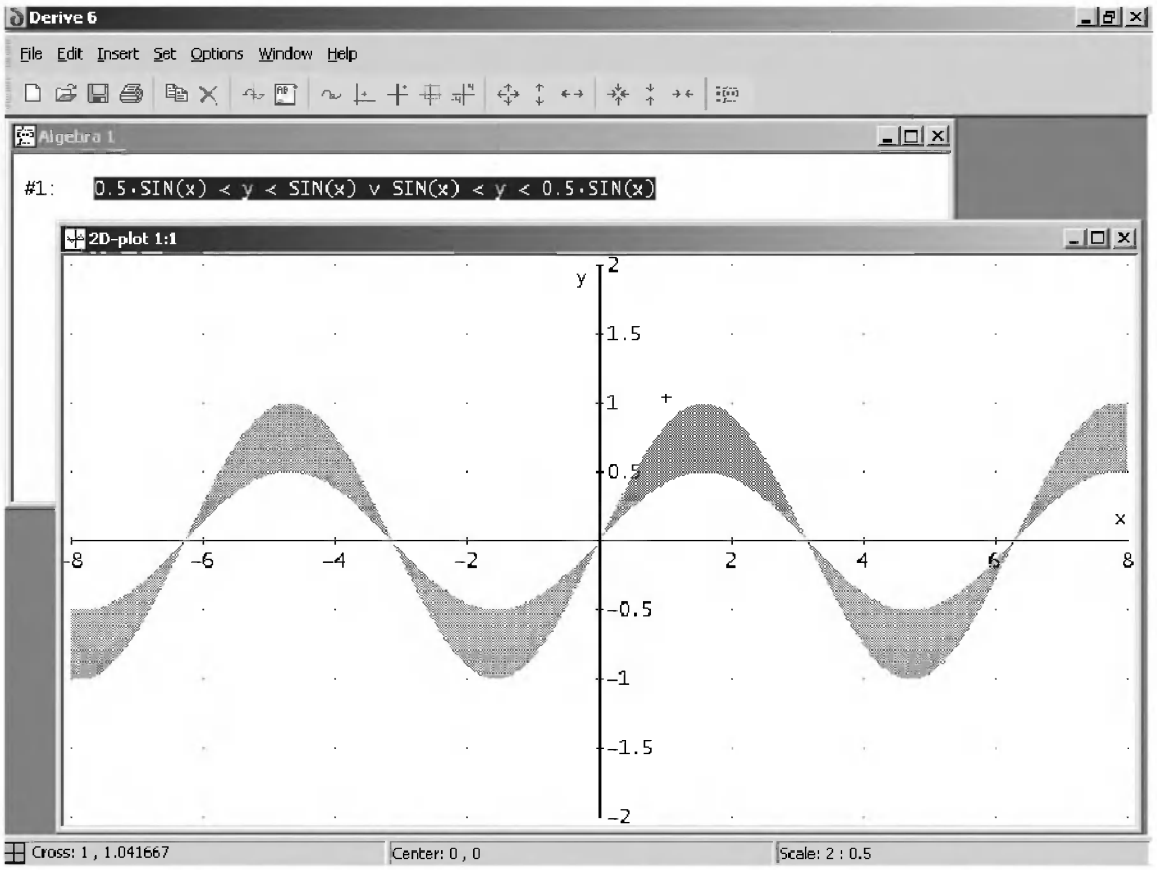


Рис. 13.66. Построение площадей, ограниченных синусоидами с амплитудой, равной 0.5 и 1

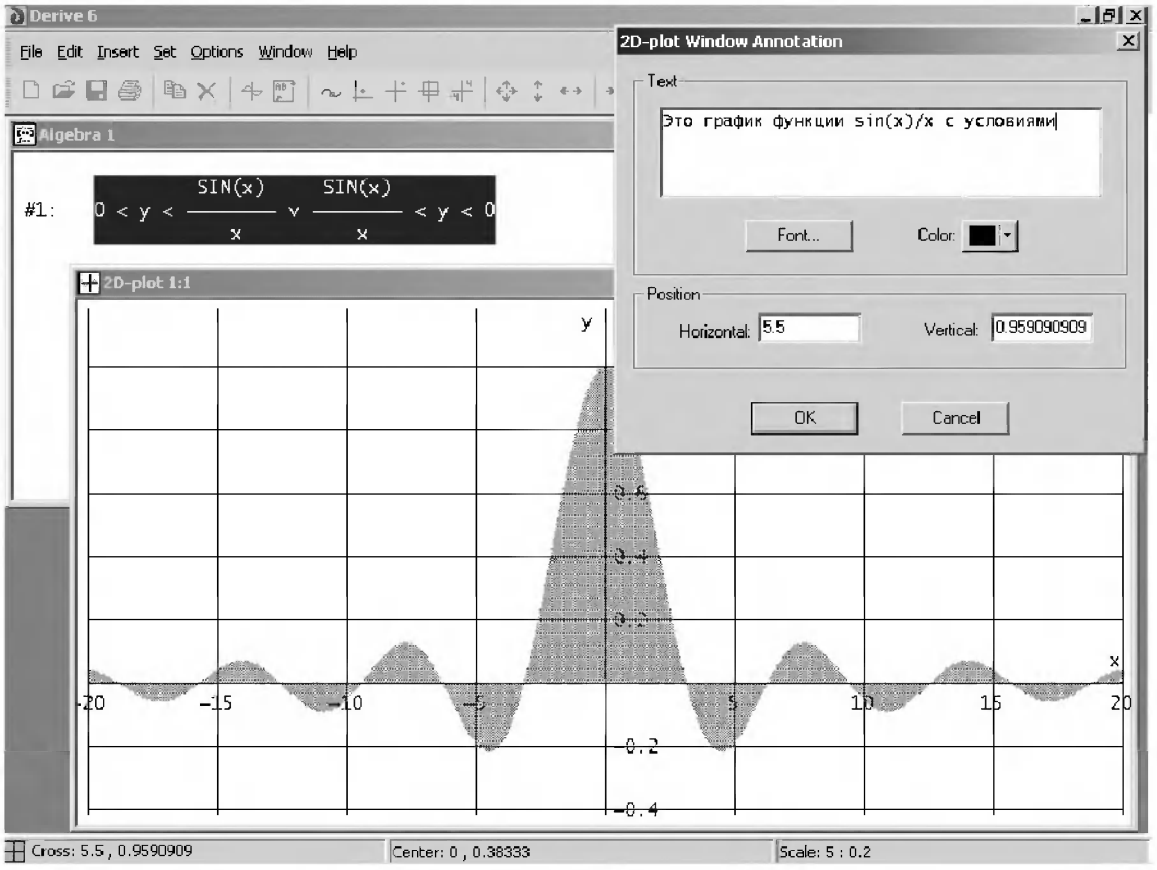


Рис. 13.67. График функции sin(x)/x с разной окраской положительных и отрицательных полуволн

да этого окна используется команда **Insert Annotation...** в контекстном меню правой клавиши мыши.

Следующий пример (рис. 13.68) показывает, что, помимо ограничений на значения y, можно ввести и ограничение на изменение x. В данном случае для значений sin(x)/x<y задано еще одно условие – x>0. Это означает, что будут закрашиваться только отрицательные полуволны зависимости sin(x)/x для x>0.

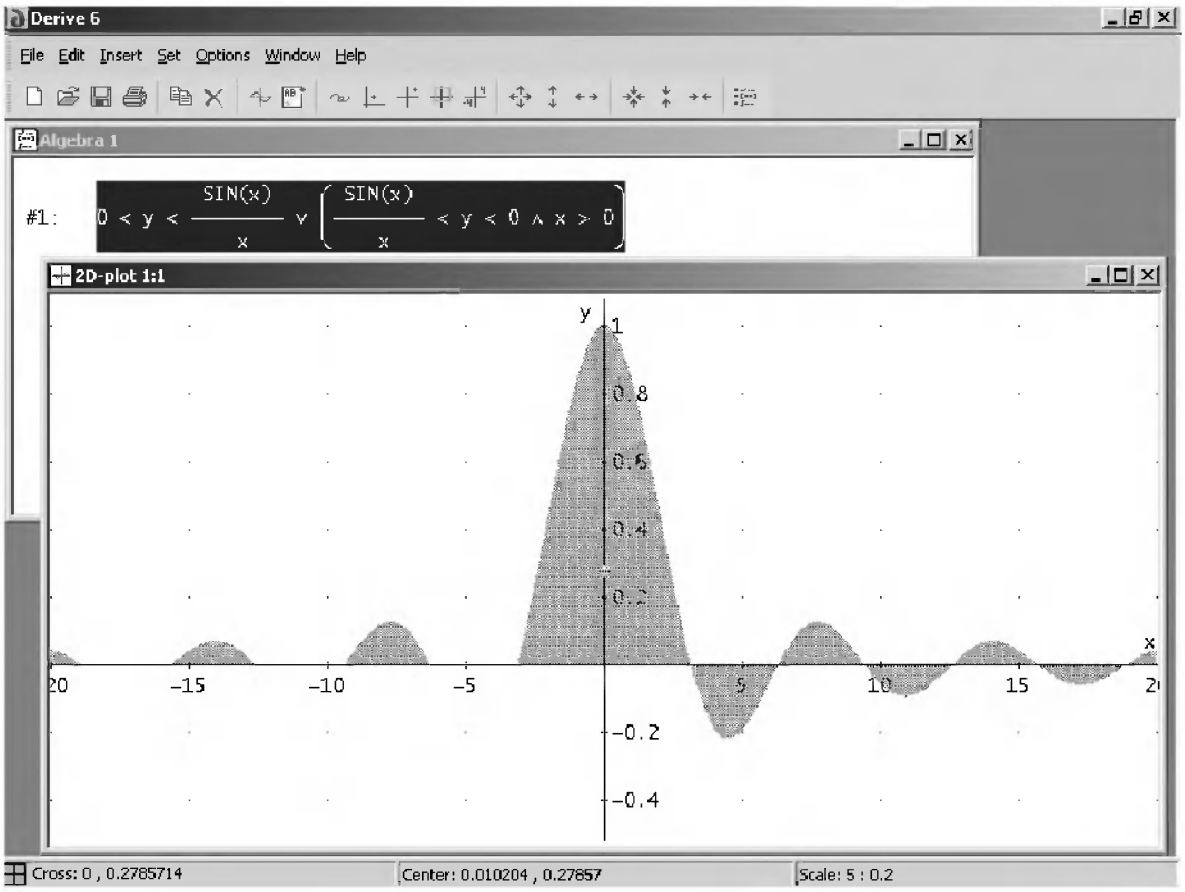


Рис. 13.68. График функции sin(x)/x при комбинированном задании условий окраски

Приведенные примеры позволят читателю опробовать свои силы в построении подобных графиков.

13.8.8. Представление вычислений в Derive в форме ноутбуков

В Derive 6 вычисления можно оформлять в форме ноутбуков, содержащих текстовые комментарии, математические формулы, результаты вычислений в численном или символьном виде и рисунки (см. пример на рис. 1.16). Составим ноутбук с графической иллюстрацией численного интегрирования. Его вид в окне Derive 6 представлен на рис. 13.69.

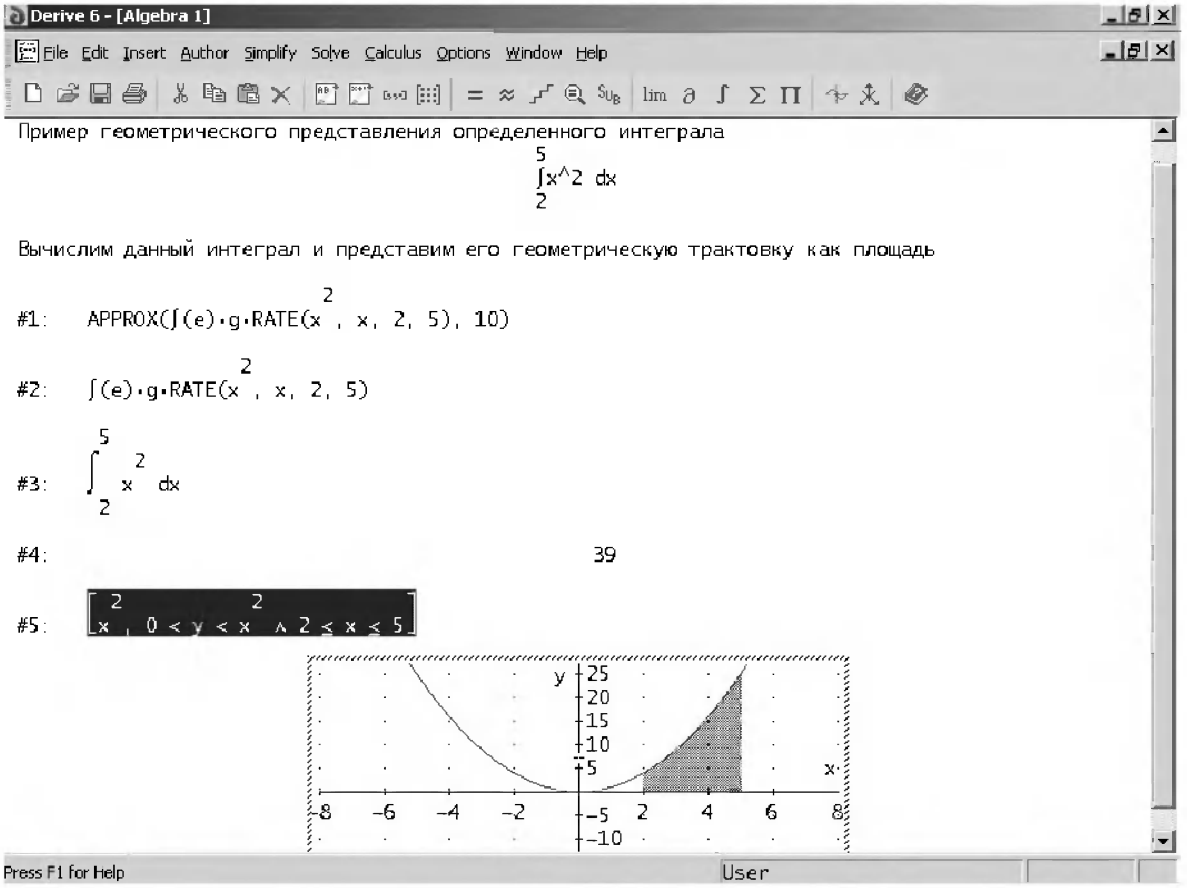


Рис. 13.69. Графическая иллюстрация численного интегрирования

Текстовые блоки в ноутбуки вставляются с помощью команды **Text Object...** в позиции **Insert** меню или с помощью соответствующей кнопки в панели инструментов. Текстовый комментарий может содержать символы кириллицы и выполняться в виде слов и фраз на русском языке. После подготовки рисунка его надо перенести из отдельного окна в окно документа. Для этого рисунок выделяется мышью и в меню **File** окна рисунка исполняется команда **Embed**.

13.8.9. Экспорт и печать файлов в Derive

В Derive 6 весьма полезной является команда **Export** в позиции **File** меню, позволяющая записывать содержимое графического окна в файлы различного формата (они перечислены в подменю **File-Export**). Важной также является группа команд, относящаяся к печати графика принтером. Она реализует типовые средства печати, присущие приложениям под Windows 98/NT/XP, но несколько упрощенные.

Команда **Print Preview** выводит окно просмотра графика перед печатью – рис. 13.70. Органы управления этого окно малочисленны и вполне очевидны. Прямо из окна (или командой **Print** из подменю **File**) можно задать распечатку графика. Точнее говоря, команда Print выводит окно того принтера, который используется компьютером для печати. Кнопка **Close** закрывает окно предварительного просмотра.

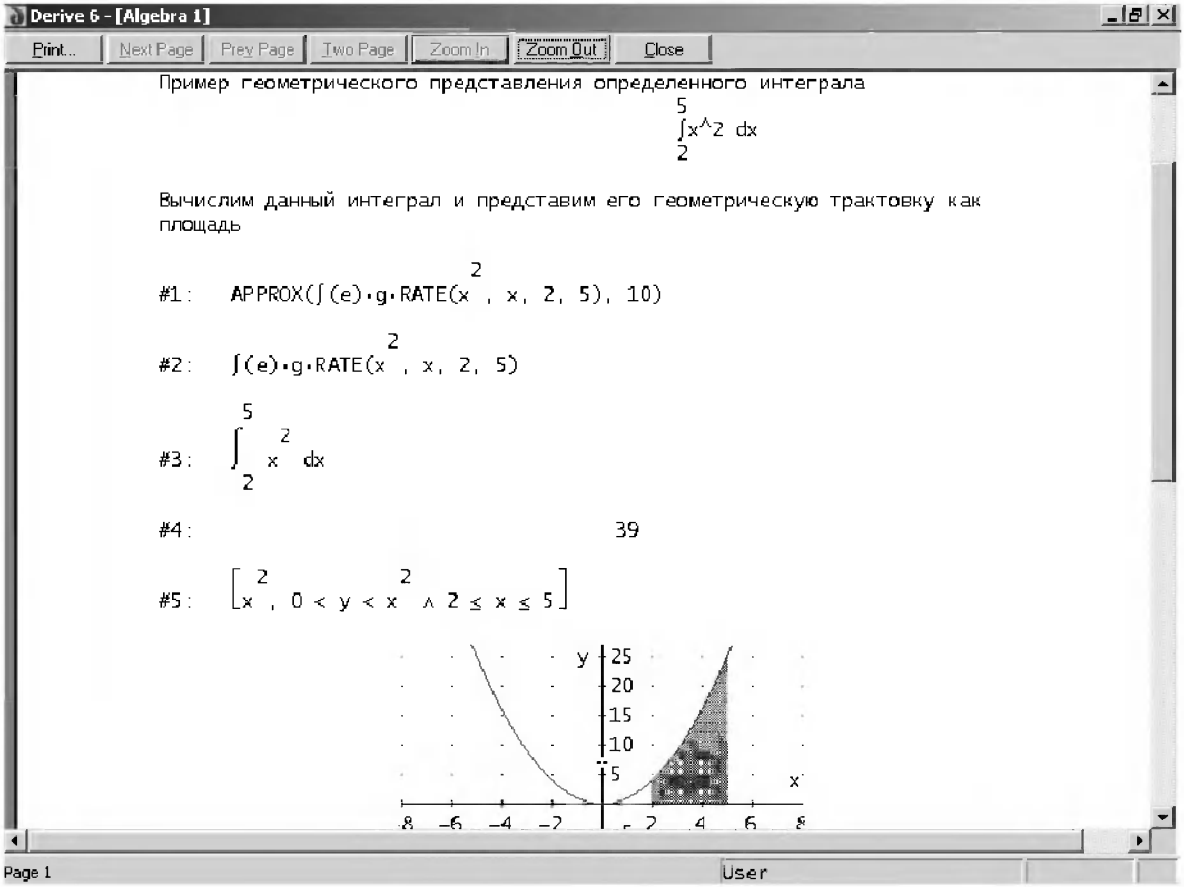


Рис. 13.70. Окно предварительного просмотра графика перед печатью

13.9. Графическая визуализация в системе MuPAD

Система MuPAD имеет довольно развитые графические возможности. В них реализован принцип, принятый в больших системах компьютерной математики, – построение графиков с помощью специальных графических операторов (команд) с множеством опций, позволяющих осуществлять форматирование графиков. Запись операторов напоминает программы на языках высокого уровня. Графики включаются в документы типа Notebook наряду со строками ввода и вывода.

13.9.1. Построение 2D-графиков функций plotfunc2d

Для оперативного построения графиков математических функций одной или двух переменных MuPAD имеет функцию:

```
plotfunc2d(f1,f2,...,fn,x=xs..xe[,y=ys..ye]);
```

где f1, f2,...,fn – функции в виде зависимостей f1(x), f2(y) и т. д., x и y – ранжированные переменные, заданные своими начальными (xs, ys,...) и конечными

(xe, ye,...) значениями, заданными вещественными числами или выражениями, возвращающими их. Никаких дополнительных опций эта команда не предусматривает и служит для оперативного построения графиков функций без их форматирования.

Применение этой команды для построения графиков трех функций одной переменной представлено на рис. 13.71. Нетрудно заметить, что эта команда легко справляется с построением графика функции $\sin(x)/x$, имеющего особенность в точке $x = 0$ в виде устранимой неопределенности $0/0 \to 1$.

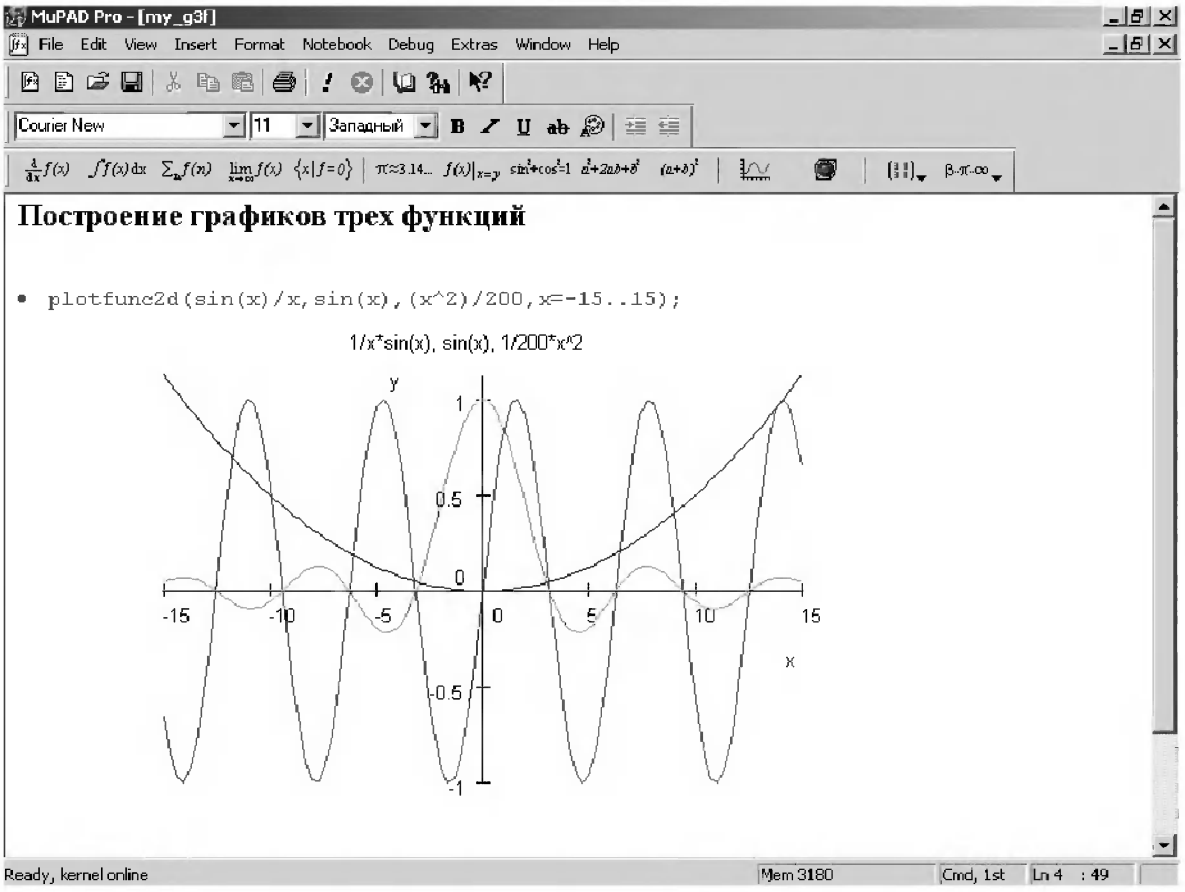


Рис. 13.71. Построение графиков трех функций системой MuPAD

Неплохо справляется команда **plotfunc2d** с построением графиков функций одной переменной с разрывами. На рис. 13.72 представлено построение графиков функций $\tan(x)$ и $1/(x-3) + 1/(x+3)$. Обе функции имеют разрывы со значениями, устремляющимися в $+\infty$ и $-\infty$. Важно отметить, что разрывы отображаются естественно, то есть как разрывы, а не в виде вертикальных отрезков линий.

Свойства графиков в MuPAD можно менять с помощью опций. Они будут рассмотрены немного позже. А пока представим график рис. 13.73, в котором одна из двух опций задает вывод координатной сетки, другая – желтый цвет основы (по умолчанию без опций он белый). Заодно на графике показано окно свойств графика с открытой вкладкой «Просмотр».

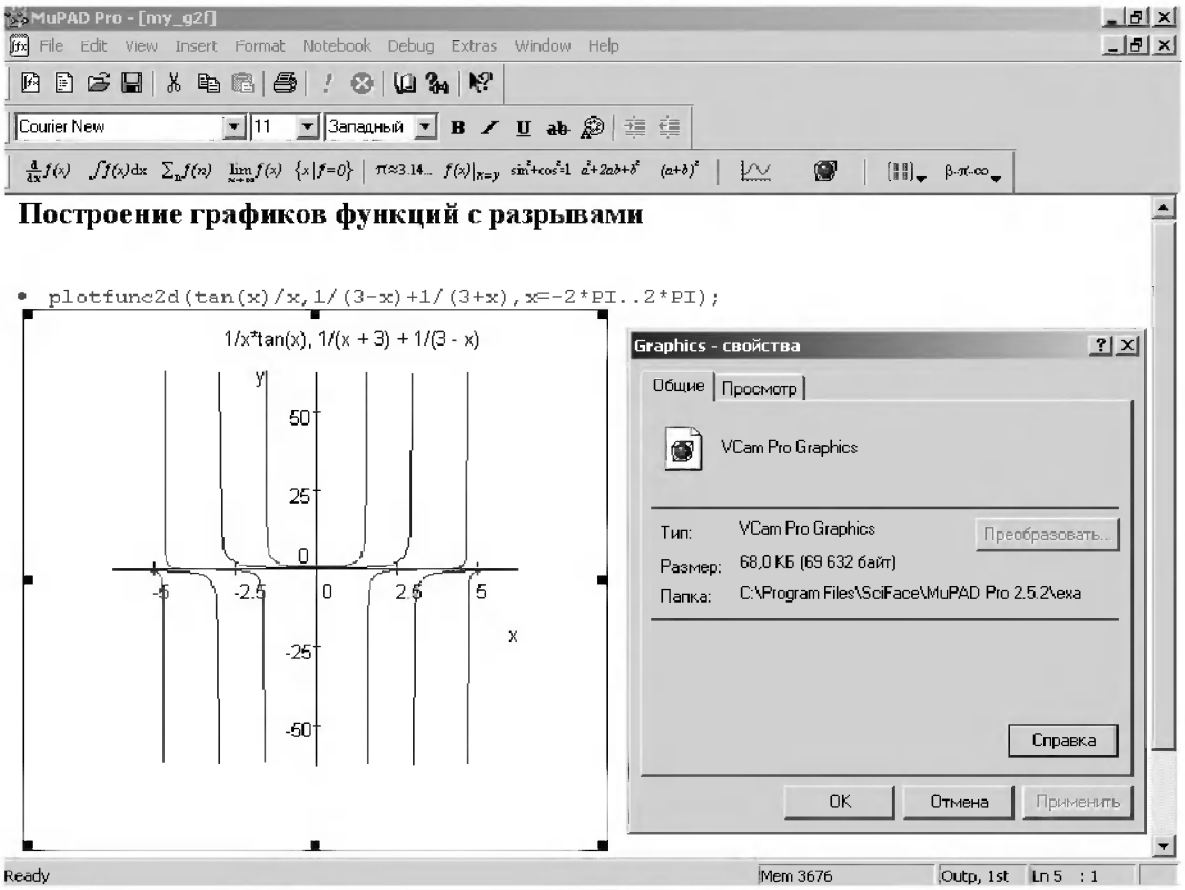


Рис. 13.72. Построение графиков двух функций с разрывами

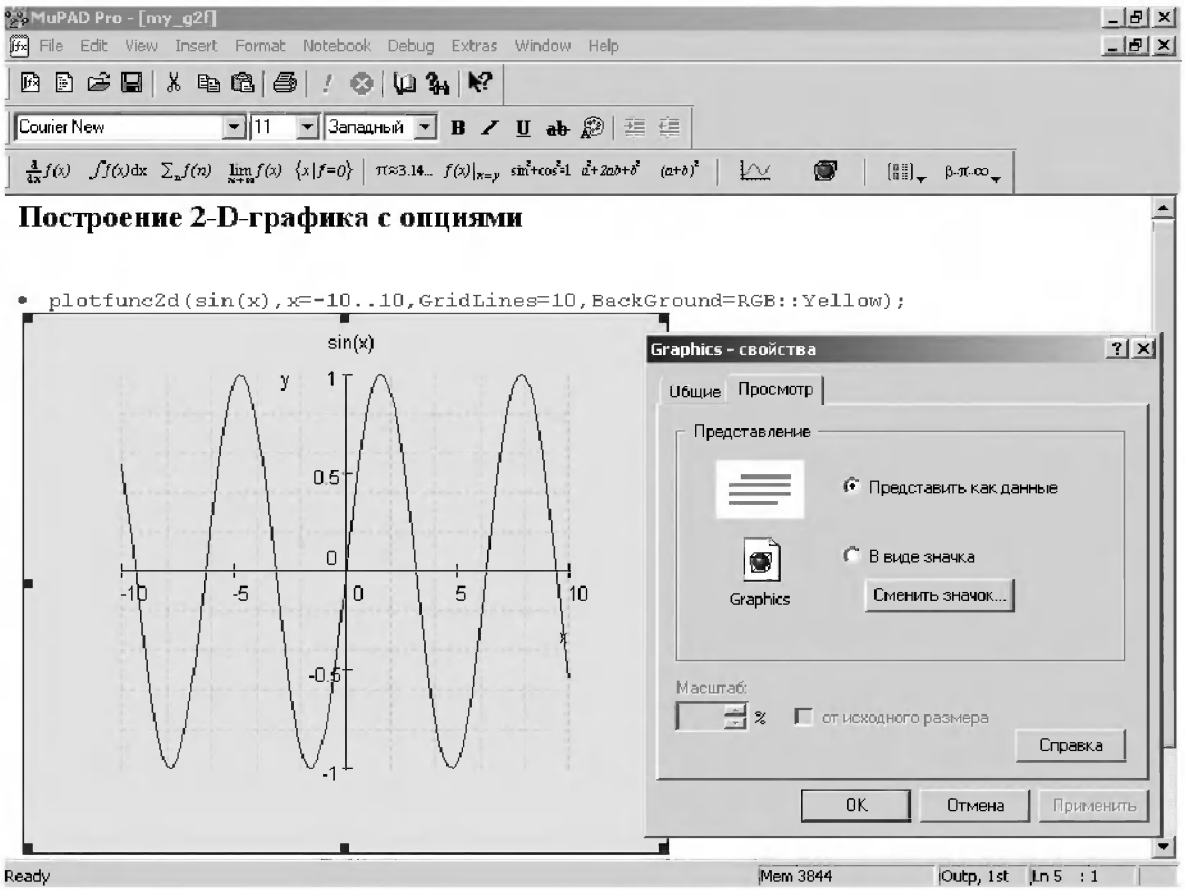


Рис. 13.73. Построение графика синусоиды с двумя опциями

13.9.2. Построение 3D-графиков функций plotfunc3d

Для построения 3D-графиков служит функция

```
Plotfunc3d(f1,f2,…[,opt])
```

Она может использоваться также для построения графиков поверхностей – функций двух переменных. Такое построение показано на рис. 13.74. Там же показано контекстное меню правой клавиши мыши для 3D-графика.

Данная функция способна строить графики ряда таких поверхностей, пересекающихся в пространстве. Это иллюстрирует рис. 13.75. Нетрудно заметить, что график строится с применением алгоритма удаления невидимых уровней и алгоритма функциональной окраски поверхности.

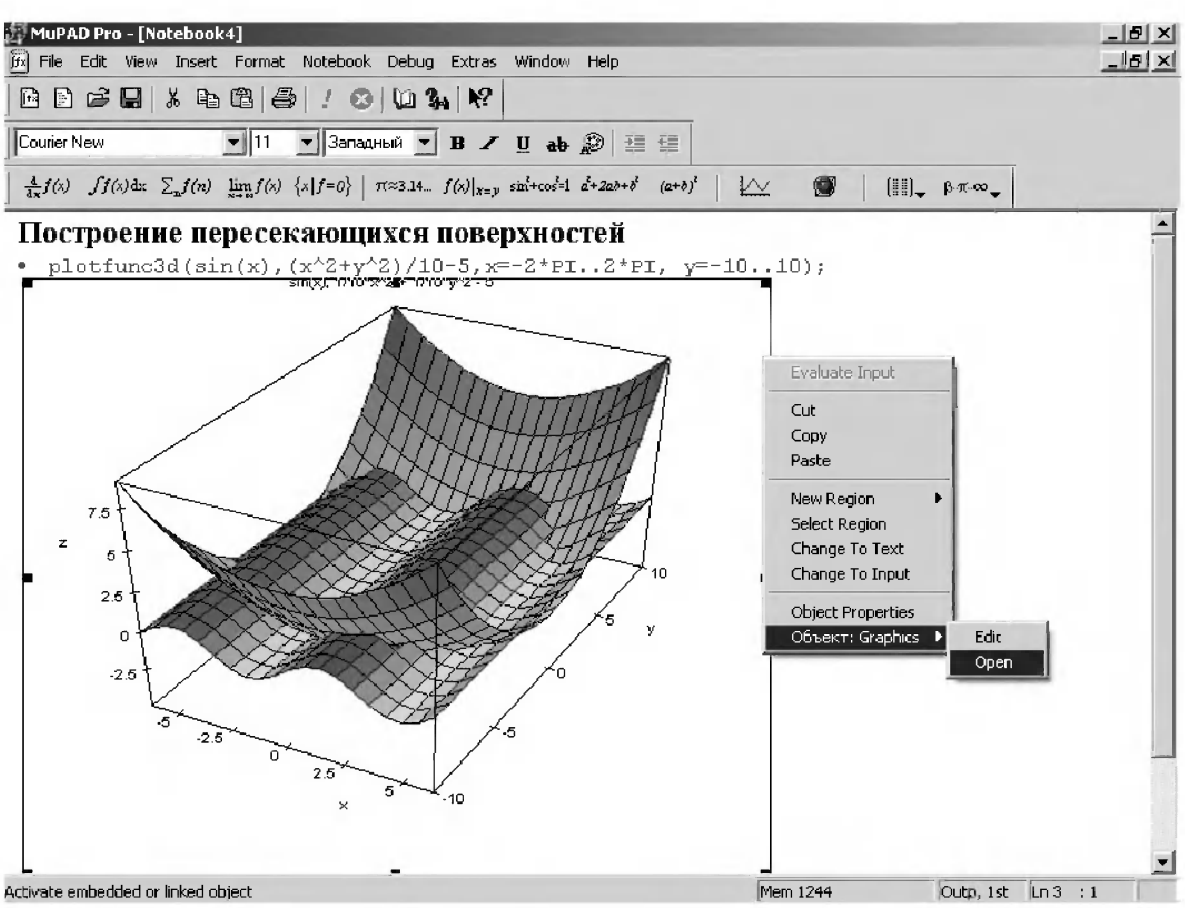


Рис. 13.75. Построение графиков двух поверхностей, пересекающихся в пространстве

Здесь график дан в выделенной области – для выделения достаточно, как обычно, один раз быстро нажать левую клавишу мыши при ее курсоре, расположенном в области графика. Выделенный график можно растягивать в разных направлениях, уцепившись курсором мыши за черные прямоугольники, расположенные на рамке выделения графика.

График можно также форматировать, выделив его на этот раз двойным коротким нажатием мыши, – см. пример на рис. 13.76. При выделенном графике его окно приобретает панель инструментов, расположенную под главным меню. С помощью этой панели можно задать повороты фигур в разном направлении, отображение графиков в перспективе (жирные двойные стрелки, указывающие вверх и вниз), а также задать вращение графиков мышью. Можно также редактировать титульную надпись и перемещать график в пределах его окна. Перед трансформацией графика он временно подменяется параллелепипедом.

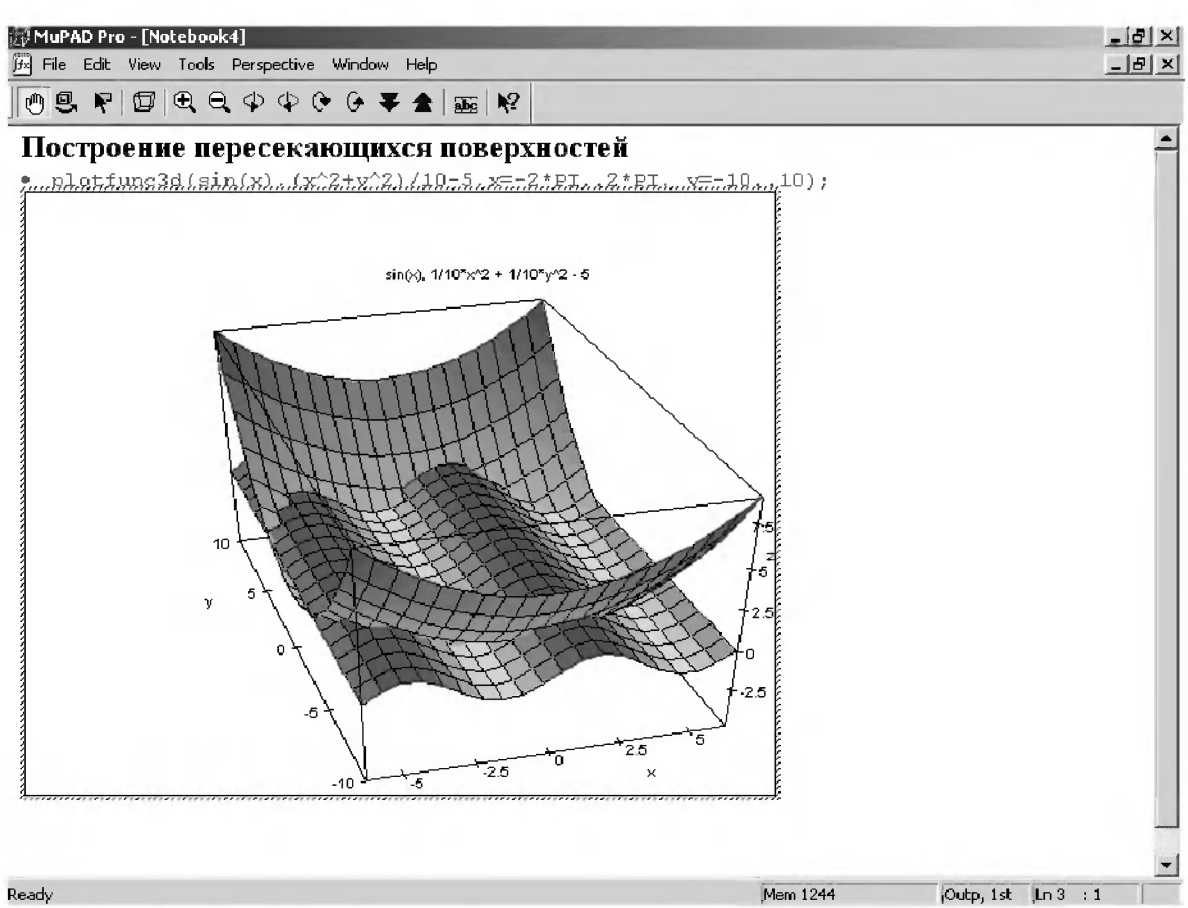


Рис. 13.76. Форматирование трехмерного графика

13.9.3. Многофункциональные команды plot2d и plot3d

Для построения графиков 2D- и 3D-функций служат многофункциональные команды:

```
plot2d(Опция_1, Опция_2,..., Объект_1, Объект_2),
plot3d(Опция_1, Опция_2,..., Объект_1, Объект_2).
```

Первая из них строит графики 2D-типа, а вторая – графики 3D-типа. Как нетрудно заметить, эти команды содержат опции и объекты. Последние, в свою очередь, могут иметь опции. Назначение опций указано ниже, при этом опции по умолчанию заданы жирным шрифтом.

Опции общего характера

Опция	Значение	Назначение
Arrow	TRUE/**FALSE**	Область графика
Axes	NON,ORIGEN, **CORNER**,BOX	Координатные оси
AxesOrigin	**Automatic**, [x0,y0]	Начало осей
AxesScaling	[**Lin**/Log,**Lin**/Log]	Масштаб (линейный/логарифмический)
BackGround	[R,G,B]	Цвет заднего фона
ForeGround	[R,G,B]	Цвет переднего фона
FontFamily	...,**Helvetica**	Фонты для надписей
FontSize	7,8,9,**10**,...,36	Размер фонтов
FontStyle	**Bold**,...	Стиль фонтов
Labeling	TRUE/**FALSE**	Обозначения осей (короткие)
Labes	[name,name]	Обозначения осей name
LineStyle	**SolidStyle**,DashedStyle	Стиль линий графиков
PointStyle	Squares,Circles, **FilledSquares**, FilledCircles	Стиль точек на графиках
PointWidth	1,2,**3**,...	Размер точки
Scaling	Constrained, **UnConstrained**	
Tiks	0, 1, 2,..., **10**,..., 20	Отметки на осях
Title	"String"	Титульная надпись
TitlePosition	**Above**, Bellow, [x,y]	Место титульной надписи

Опции объектов

Опция	Значение	Назначение
Color	[Flat],**[Flat]**,[R,G,B]], [Height]	Цвет
Grid	[2],[3],...,**[20]**,...	Сетка на графике
Smoothness	[0],[1],...,**[20]**	
Style	[Points], **[Lines]**, [LinesPoints], [Impulses]	Стиль графика
LineStyle	**SolidStyle**,DashedStyle	Стиль линий
PointStyle	Squares,Circles, **FilledSquares**, FilledCircles	Стиль точек
PointWidth	1, 2, **3**,...	Размер точек
Title	"String"	Титульная строка
TitlePosition	**Above**, Bellow, [x,y]	Позиция титульной строки

13.9.4. Построение двумерных графиков

На рис. 13.77 представлено построение графика функции $\sin(1/u)$ с помощью команды plot2d. Обратите внимание на то, что первой задана особая опция **Mode = Curve** (другие значения – **List** и **Surface**), которая указывает на построение графика линией – кривой. Функция задается списком [u, sin(1/u)], в котором первый элемент – независимая переменная u, а второй элемент – сама функция. Опция u = [-2*PI, 2*PI] задает диапазон изменения независимой переменной.

```
plot2d([Mode = Curve,
        [u, sin(1/u)], u = [-2*PI, 2*PI]]);
```

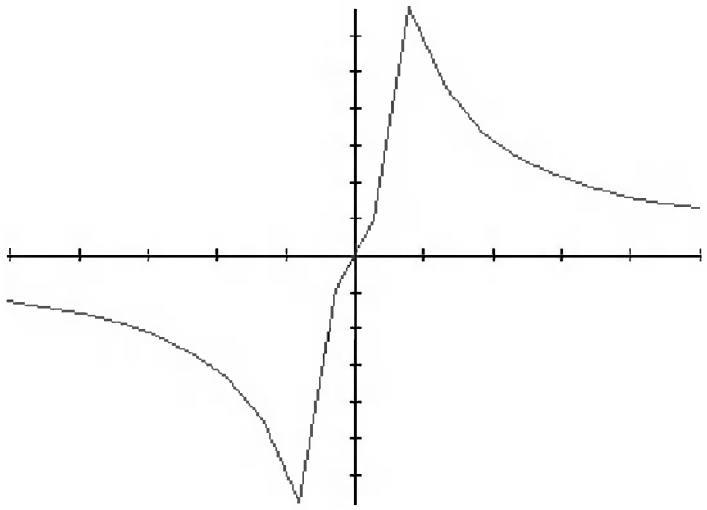


Рис. 13.77. Пример построения графика функции sin(1/u) с помощью команды plot2d MuPAD

13.9.5. Построение графиков функций, заданных параметрически

Команда **plot2d** позволяет строить графики параметрически заданных функций. На рис. 13.78 дан пример построения эллипса точками с помощью команды **plot2d** с предварительным заданием объекта Point_List – параметра функции.

```
alias(fsin = funcattr(sin, "float")):
alias(fcos = funcattr(cos, "float")):
point_list := [point(fsin(i*PI/80),
       fcos(i*PI/80)) $ i = -80..80]:
plot2d([Mode = List, point_list]);
```

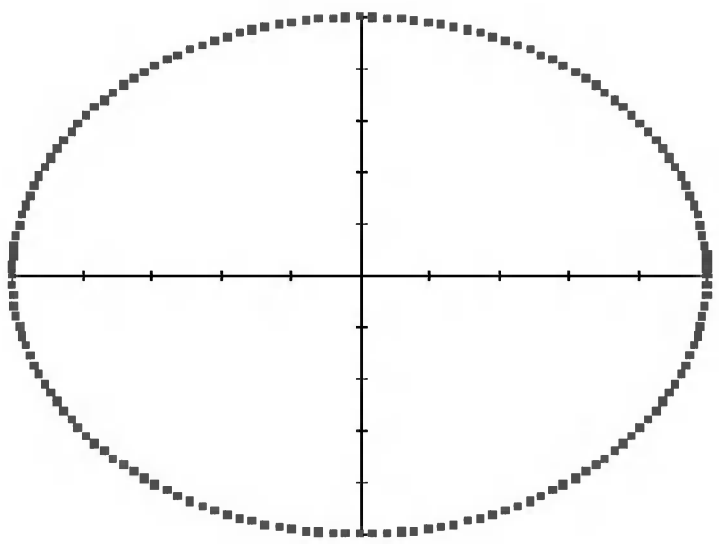


Рис. 13.78. Построение эллипса точками

13.9.6. Построение графика в полярной системе координат

Еще один пример на построение графика спирали в полярной системе координат дан на рис. 13.79. Обратите внимание на задание целого ряда опций для вывода осей в виде прямоугольника (ящика), нумерации делений на осях, вывода титульной надписи «SPIRAL», задания построения по 100 точкам и т. д.

```
plot2d(Axes = Box, Ticks = 8,Labeling = TRUE,Title="SPIRAL",
       [Mode = Curve,[u*cos(u), u*sin(u)],
            u = [0, 4*PI], Grid = [100]]);
```

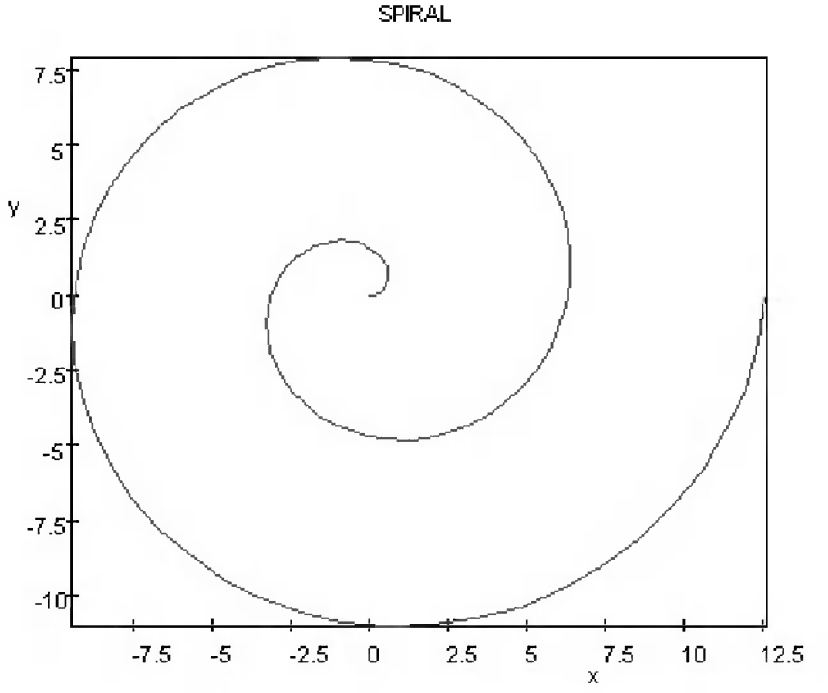


Рис. 13.79. Построение спирали с форматированием графика рядом опций

Еще один пример, показанный на рис. 13.80, иллюстрирует построение трех фигур с помощью команды **plot2d** с разным стилем линий. Здесь одна фигура построена сплошной линией, другая – квадратиками и третья – линией с квадратиками.

Этих примеров вполне достаточно, чтобы показать, что графика MuPAD по своим возможностям превосходит графику системы Derive и по ряду показателей приближается к графике более развитых систем компьютерной математики.

13.9.7. Построение 3D-графиков командой plot3d

Для построения поверхностей (в том числе нескольких на одном графике) и 3D-фигур используется команда **plot3d** с опциями и объектами, описанными выше. В связи с этим ограничимся приведением примера, который строит три трехмерные фигуры, пересекающиеся в пространстве:

```
plot2d(Axes = Box, Ticks = 0,
       [Mode = Curve,
            [sin(u), cos(u)], u = [0, 2*PI],
            Grid = [50], Style = [Points]],
       [Mode = Curve,
            [2*sin(u), cos(u)], u = [0, 2*PI],
            Grid = [50], Style = [Lines]],
       [Mode = Curve,
            [sin(u), 1.5*cos(u)], u = [0, 2*PI],
            Grid = [50], Style = [LinesPoints]]);
```

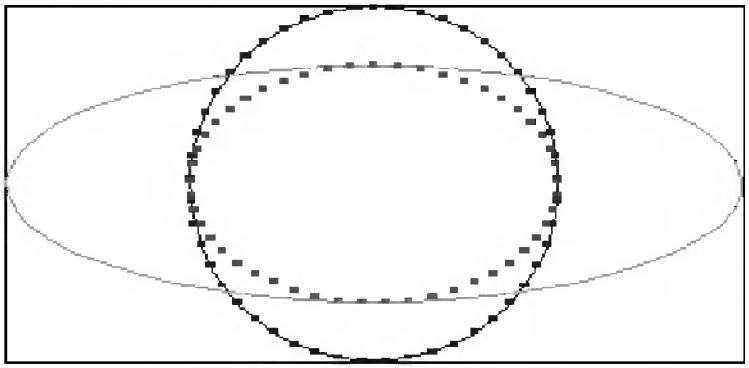


Рис. 13.80. Построение графиков трех функций, заданных параметрически

```
plot3d(Axes = None, Title = "Three different Surfaces",
    TitlePosition = Below, CameraPoint = [13.0, -24.0, 20.0],
    [Mode = Surface,
       [(4 + cos(v))*cos(u), (4 + cos(v))*sin(u), sin(v)],
       u = [0, 2*PI], v = [0, 2*PI], Grid = [30, 30],
       Style = [HiddenLine, Mesh]],
    [Mode = Surface,
       [2*sin(u)*cos(v), 2*sin(u)*sin(v), 2*cos(u)],
       u = [0, PI], v = [-PI, PI], Grid = [20, 20],
       Style = [ColorPatches, AndMesh]],
    [Mode = Surface,
       [u, v, -3.0], u = [-5.0, 5.0], v = [-5.0, 5.0],
       Grid = [20, 20], Style = [ColorPatches, AndMesh]]);
```

Нетрудно заметить, что по существу запись этой команды является типовой линейной программой. Каждая из трех функций, графики которых строятся, описывается своими математическими выражениями и опциями и представляет собой объекты. Все эти объекты размещены в списке параметров графической команды **plot3d**. На рис. 13.81 представлен вид трех поверхностей, соответствующих описанному выше примеру.

Реалистичность графиков можно заметно улучшить, обратившись к графической подсистеме Vcam.

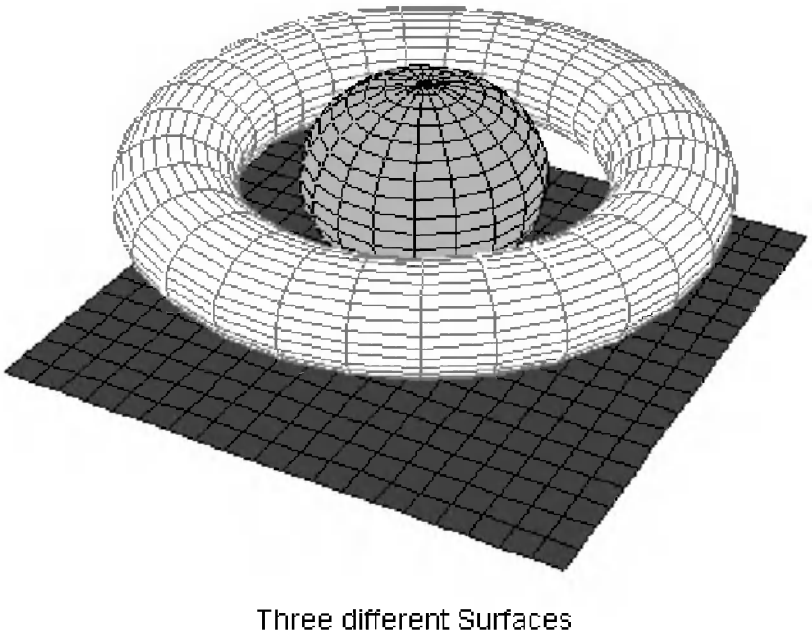


Рис. 13.81. Построение трех поверхностей с помощью функции plot3d

13.9.8. Работа со средствами Vcam

MuPAD имеет специальную *графическую подсистему Vcam*, расширяющую возможности графиков. К ней можно обратиться двойным щелчком левой клавиши мыши при установке ее курсора на графический объект. На рис. 13.82 представлено окно подсистемы Vcam, которое появляется, если команда графики допускает работу с данной подсистемой.

Окно графики системы Vcam имеет панель инструментов, которая уже описывалась. Она позволяет осуществлять трансформирование графиков почти в реальном масштабе времени.

13.9.9. Применение графической библиотеки

Помимо отмеченных выше команд, MuPAD имеет библиотеку с рядом дополнительных полезных команд графики. Команда **info(plot)** выводит список всех команд, включенных в библиотеку графики:

```
info(plot);
Library 'plot': graphical primitives and functions for two- and
three-dime\
nsional plots

- Interface:
plot::Curve2d,      plot::Curve3d,        plot::Ellipse2d,
plot::Function2d,   plot::Function3d,     plot::Group,
plot::HOrbital,     plot::Lsys,           plot::Point,
```

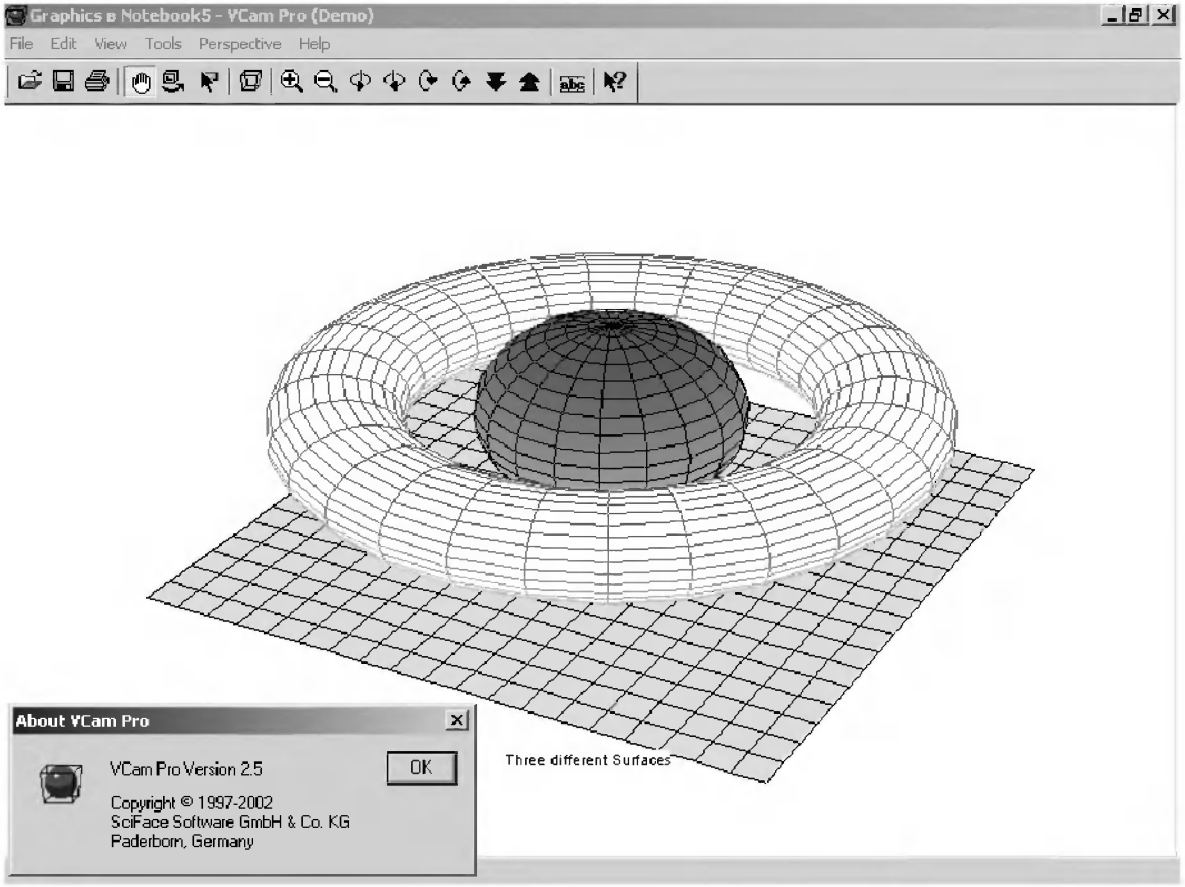


Рис. 13.82. Окно графической подсистемы Vcam математической системы MuPAD

```
plot::Pointlist,    plot::Polygon,        plot::Rectangle2d,
plot::Scene,        plot::Surface3d,      plot::Turtle,
plot::bars,         plot::boxplot,        plot::contour,
plot::copy,         plot::cylindrical,    plot::data,
plot::density,      plot::implicit,       plot::inequality,
plot::line,         plot::matrixplot,     plot::modify,
plot::ode,          plot::piechart2d,     plot::piechart3d,
plot::polar,        plot::spherical,      plot::surface,
plot::vector,       plot::vectorfield,    plot::xrotate,
plot::yrotate

- Exported:
plot, plot2d, plot3d, plotfunc2d, plotfunc3d
```

Заметим, что в старых версиях MuPAD эта библиотека называлась plotlib и содержала намного меньше функций. Как нетрудно заметить, библиотека plot содержит команды для построения контурных графиков, графиков поля и плотности, графиков данных, графиков в полярной, цилиндрической и сферической системах координат, графики полигонов и графики вращения. Поскольку на каждый вид графика в базе данных помощи есть несколько примеров, ограничимся рассмотрением пары примеров на применение библиотечных команд. На рис. 13.83 показано построение контурного импликативного графика.

```
>> plot(plot::implicit(
     (x^2 + y^2)^3 - (x^2 - y^2)^2, x = -1..1, y = -1..1
   ))
```

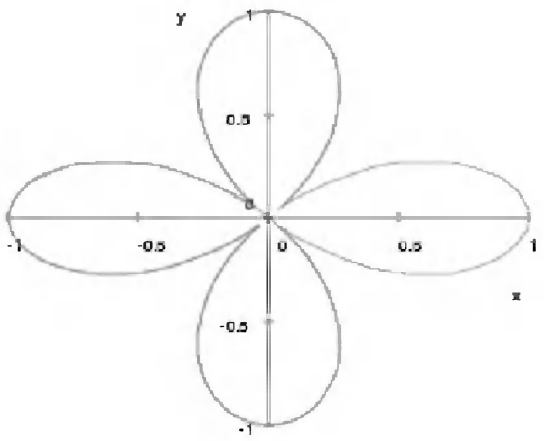


Рис. 13.83. Пример построения контурного импликативного графика поверхности в системе MuPAD

Приведенные примеры показывают, что система MuPAD обеспечивает построение практически всех важных типов графиков. В ней есть даже такой редкий вид графики, как TURTLE-графика (графика черепашки), заимствованная из языка программирования ЛОГО и представляющая интерес для школьного образования.

Глава 14

Программирование в системах компьютерной алгебры

Все СКА и СКМ имеют свой язык программирования. Как правило, по своим возможностям в решении математических задач языки программирования СКА намного превосходят обычные языки программирования высокого уровня, такие как Fortran, Algol, Pascal и их варианты, даже самых новейших версий.

14.1. Основные средства программирования СКМ

14.1.1. Понятие о входном языке системы и языке реализации

Системы компьютерной математики изначально создавались с целью исключить программирование при решении математических задач. Тем не менее собственный язык программирования вовсе не вредит СКМ и СКА, а лишь расширяет их функциональные возможности.

Язык программирования высокого уровня – это совокупность средств, позволяющих управлять компьютером с помощью достаточно простых и понятных пользователю инструкций (команд), а не с помощью машинных кодов. Следует отметить, что у всех СКМ и СКА можно выделить три языка:

- входной язык общения системы;
- собственный язык программирования системы;
- язык (или даже языки) реализации системы.

Все описанные в данной книге языки общения и языки программирования являются *интерпретирующими* и *математико-ориентированными*. Напомним, что интерпретирующие языки программирования просматривают текст программы и немедленно исполняют любые команды (операторы, функции и т. д.) по мере их опознания (интерпретации). Это дает следующие преимущества перед языками компилирующего типа:

- отсутствие этапов компиляции программ и линковки;
- возможность выполнения программ по шагам в интерактивном режиме;
- включение в состав языка программирования всех средств входного языка общения.

Существует неверная точка зрения, что интерпретирующие языки программирования проигрывают компилирующим в скорости выполнения вычислений, особенно в задачах большой размерности. Это было верно на заре развития этих языков. При этом компилирующие языки решали задачи только численными методами.

В наше время ситуация изменилась. Прежде всего надо отметить, что ядро СКМ содержит множество команд и функций, которые не только тщательно оптимизированы, используют новейшие алгоритмы вычислений, но и представлены в откомпилированном виде. Таким образом, в них все критические вычисления (например, матричные, быстрые преобразования Фурье и т. д.) выполняются

столь же быстро, как и в языках программирования компилирующего типа. Кроме того, надо отметить, что выигрыш от компиляции обычно заметно меньше, чем выигрыш от применения хороших алгоритмов вычислений, применяемых в современных СКМ. К примеру, при решении задач линейной алгебры в Maple используются самые совершенные алгоритмы группы NAG. В Mathematica используются свои алгоритмы матричных вычислений, которые, по мнению разработчиков системы, обеспечивают более высокие скорости матричных вычислений, чем у системы MATLAB.

По существу, все представленные выше главы описывали языки общения СКМ и СКА. Они обеспечивают решение огромного числа задач как без программирования, так и с ним. В этой главе описаны дополнительные средства, традиционно относящиеся к языкам программирования высокого уровня. Это управляющие структуры и модули программ, процедуры и функции, циклы, специальные средства ввода/вывода и т. д. Отдельные примеры их применения уже не раз приводились, но настал момент, когда эти средства нужно рассмотреть более детально и целенаправленно.

Язык реализации систем – это стандартный язык программирования высокого уровня, чаще всего C или C++. Явным исключением тут является система Deruve, реализованная на языке программирования MuLISP – одной из реализаций хорошо известного языка программирования для систем искусственного интеллекта LISP. Выбор языка C (или C++) обусловлен высокой эффективностью машинного кода, который создается компиляторами этого языка.

14.1.2. О языках программирования СКМ и СКА

Фактически СКМ и СКА имеют проблемно-ориентированный на математические расчеты *язык программирования сверхвысокого уровня*. По своим возможностям в этой области эти языки намного превосходят обычные универсальные языки программирования, такие как Фортран, Бейсик, Паскаль или Си (C/C++). Достаточно отметить, что такие сложные операции, как дифференцирование, интегрирование, решение сложных уравнений и т. д., языки СКМ и СКА реализуют в виде простых операторов или функций – как правило, тех же самых, которые определены во входном языке общения.

На языках программирования пишутся *программы*, содержащие набор инструкций и программных конструкций, обеспечивающих выполнение тех или иных операций и алгоритмов – в нашем случае математических. Программы обычно создаются в текстовых редакторах или в специальных *редакторах/отладчиках программ*.

Программы, все инструкции которых выполняются строго последовательно, называются *линейными программами*. Большинство же программ относятся к разветвляющимся программам, у которых возможны переходы от одной ветви с командами к другой в зависимости от условий работы. Для создания таких программ используются *управляющие структуры*.

14.1.3. Алфавит, ключевые слова, процедуры и функции языков программирования

Любой язык программирования имеет алфавит, лексемы, синтаксис и семантику. Конкретный их смысл описан ниже.

Алфавит языка – это совокупность символов, с помощью которых создаются те или иные конструкции языка. Во всех языках алфавит состоит из символов основной таблицы ASCII. В них входят большие и малые латинские буквы, арабские цифры от 0 до 9, арифметические операторы, скобки всех видов и прочие специальные символы (иногда двойные). Все они уже были рассмотрены.

Ключевые слова – это цепочки символов, обозначающие те или иные детали конструкций языков программирования. Ключевые слова, как правило, запрещено применять в качестве идентификаторов других конструкций языков, например имен функций пользователя, имен переменных, имен процедур и функций.

Во всех языках программирования встречаются *константы* и *функции*, достаточно подробно описанные в главе 2. Детали их синтаксиса (правил задания и применения) были многократно описаны выше, и это описание будет продолжено, как и описание правил синтаксического контроля языка системы Maple. При нарушении правил синтаксиса вычисления прерываются, появляется сообщение об ошибке.

Процедурой называют модуль программы, имеющий самостоятельное значение и выполняющий одну или несколько операций, обычно достаточно сложных и отличных от операций, выполняемых встроенными операторами и функциями.

Процедуры являются важнейшим элементом структурного программирования и служат средством расширения возможностей СКМ и СКА пользователем. *Процедуры-функции* возвращают некоторое значение в ответ на обращение к ним.

14.2. Язык программирования системы Maple

14.2.1. Задание функции пользователя

Хотя ядро Maple, библиотека и пакеты расширений содержат свыше трех тысяч функций, всегда может оказаться, что именно нужной пользователю (и порой довольно простой) функции все же нет. Тогда возникает необходимость в создании собственной функции, именуемой *функцией пользователя*. Для этого в Maple-языке используется следующая, не совсем обычная конструкция:

```
name:=(x,y,…)->expr
```

После этого вызов функции осуществляется в виде `name(x,y,…)`, где `(x,y,…)` – список формальных параметров функции пользователя с именем

name. Переменные, указанные в списке формальных параметров, являются локальными. При подстановке на их место фактических параметров они сохраняют их значения только в теле функции (expr). За пределами этой функции переменные с этими именами либо оказываются неопределенными, либо сохраняют ранее присвоенные им значения. Следующие примеры иллюстрируют сказанное:

```
> restart;
> x:=0;y:=0;
```

$$x := 0$$
$$y := 0$$

```
> m:=(x,y)->sqrt(x^2+y^2);
```

$$m := (x,y) \to \sqrt{x^2+y^2}$$

```
> m(3,4);
```

$$5$$

```
> m(3.,4);
```

$$5.000000000$$

```
> [x,y];
```

$$[0, 0]$$

Нетрудно заметить, что при вычислении функции m(x,y) переменные x и y имели значения 3 и 4, однако за пределами функции они сохраняют нулевые значения, заданные им перед введением определения функции пользователя. Использование хотя бы одного параметра функции в виде числа с плавающей точкой ведет к тому, что функция возвращает результат также в виде числа с плавающей точкой.

Еще один способ задания функции пользователя базируется на применении *функции-конструктора* unapply:

```
name:=unapply(expr,var1,var2,...)
```

Ниже даны примеры такого задания функции пользователя:

```
> restart;
> fm:=unapply(sqrt(x^2+y^2),x,y);
```

$$fm := (x,y) \to \sqrt{x^2+y^2}$$

```
> fm(4.,3.);
```

$$5.000000000$$

```
> fe:=unapply(x^2+y^2,x,y);
```

$$fe := (x, y) \to x^2 + y^2$$

```
> fe(sin(x),cos(x));
```

$$\sin(x)^2 + \cos(x)^2$$

```
> simplify(fe(sin(x),cos(x)));
```

$$1$$

Последний пример показывает возможность проведения символьных операций с функцией пользователя.

В ряде случаев весьма желательна визуализация результатов выполнения функций пользователя. Порой она может давать неожиданный результат. На рис. 14.1 представлены примеры задания двух функций пользователя от двух переменных и построение их графиков с помощью функции plot3d.

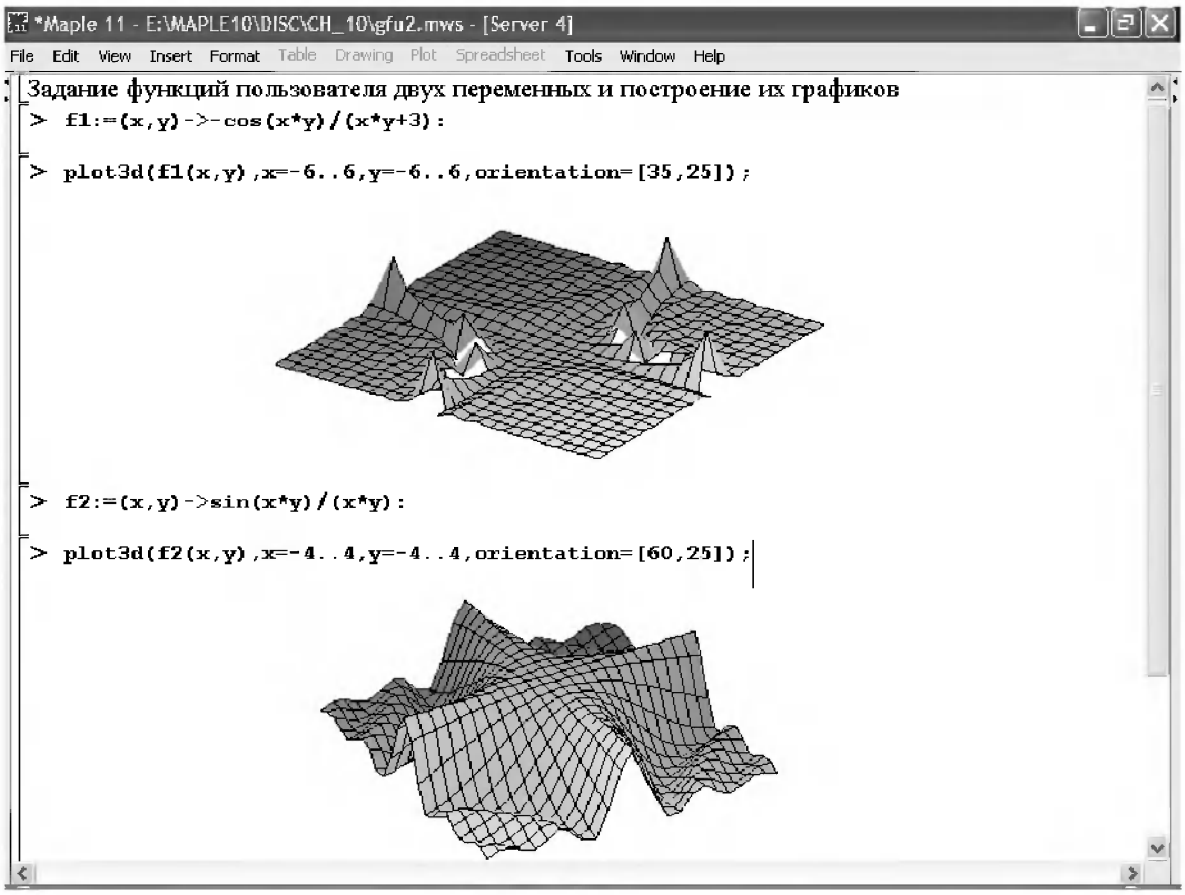


Рис. 14.1. Примеры задания функций пользователя двух переменных с построением их графиков

При задании функций пользователя рекомендуется просмотреть их графики в нужном диапазоне изменения аргументов. К сожалению, наглядными являются только графики функций одной и двух переменных.

14.2.2. Импликативные функции

Другой важный класс функций, которые нередко приходится задавать, – *импликативные функции*, в которых связь между переменными задана неявно в виде какого-либо выражения. Самый характерный пример такой функции – это выражение для задания окружности радиуса r: $x^2 + y^2 = r^2$.

Итак, импликативные функции записываются как уравнения. Соответственно, их можно решать с помощью функции `solve`. Следующие примеры иллюстрируют задание уравнения окружности в общем и частном (численном) виде:

```
> impf:=x^2+y^2=r^2;
```

$$impf := x^2 + y^2 = r^2$$

```
> subs(x=a,impf);
```

$$a^2 + y^2 = r^2$$

```
> solve(%);
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
> impf1:=x^2+y^2=25;
```

$$impf1 := x^2 + y^2 = 25$$

```
> subs(x=4,impf1);
```

$$16 + y^2 = 25$$

```
> solve(%);
```

$$3, -3$$

Для графической визуализации импликативных функций служит функция `implicitplot` пакета plots. На рис. 14.2 представлены задание двух импликативных функций и построение их графиков.

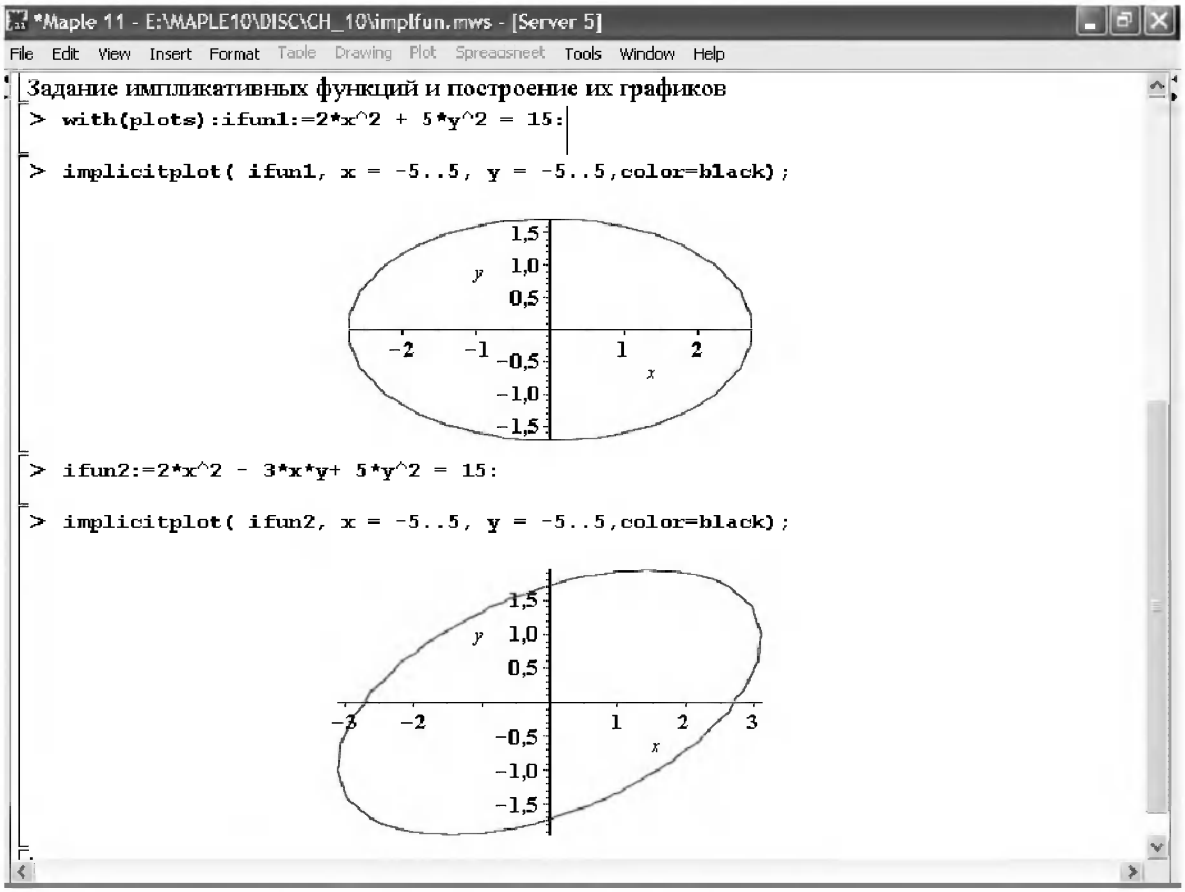


Рис. 14.2. Задание двух импликативных функций и построение их графиков

В данном случае задано построение двух эллипсов. Верхний – это окружность, сплюснутая по вертикали, а второй – наклонный эллипс.

14.2.3. Условные выражения

Простейшую конструкцию разветвляющихся программ в Maple-языке программирования задает оператор `if`, или оператор *условного выражения*:

```
if <Условие сравнения> then <Элементы>
|elif <Условие сравнения> then <Элементы>|
|else <Элементы>|
fi;
```

В вертикальных чертах | | указаны необязательные элементы данной конструкции. Следующие два вида условных выражений чаще всего используются на практике:

- `if <Условие> then <Элементы 1> fi` – если `Условие` выполняется, то исполняются `Элементы 1`, иначе ничего не выполняется;
- `if <Условие> then <Элементы 1> else <Элементы 2> fi` – если `Условие` выполняется, то исполняются `Элементы 1`, иначе исполняются `Элементы 2`.

В задании условий используются любые логические конструкции со знаками сравнения (<, <=, >, >=, =, <>) и логические операторы `and`, `or` и `not`, конструкции с которыми возвращают логические значения `true` и `false`. Рассмотрим следующий простой пример:

```
> x:=-5:
> if x<0 then print(`Negative`) fi;
```

Negative

```
> x:=1:
> if x<0 then print(`Negative`) fi;
```

В этом примере анализируется значение `x`. Если оно отрицательно, то с помощью функции вывода `print` на экран выводится сообщение «Negative». А вот если `x` неотрицательно, то не выводится никакого сообщения. В другом примере если `x` неотрицательно, то выводится сообщение «Positive»:

```
> x:=-5:
> if x<0 then print(`Negative`) else print(`Positive`) fi;
```

Negative

```
> x:=1:
> if x<0 then print(`Negative`) else print(`Positive`) fi;
```

Positive

Приведем еще один пример, показывающий особую форму задания конструкции `if-then-else-fi`:

```
> x:=-5:
> `if` (x<0, print(`Negative`),print(`Positive`));
```

Negative

```
> x:=1:
> `if` (x<0, print(`Negative`),print(`Positive`));
```

Positive

В этой конструкции вида

```
'if '(Условие, Выражение1, Выражение 2)
```

если `Условие` выполнятся, то будет исполнено `Выражение1`, в противном случае будет исполнено `Выражение2`. Ввиду компактности записи такая форма условного выражения нередко бывает предпочтительна, хотя она и менее наглядна. На рис. 14.3 представлено применение данной конструкции для моделирования трех типов сигналов.

К сожалению, функции на базе конструкции `if` не всегда корректно обрабатываются функциями символьной математики. Поэтому надо тщательно контролировать полученные в этом случае результаты.

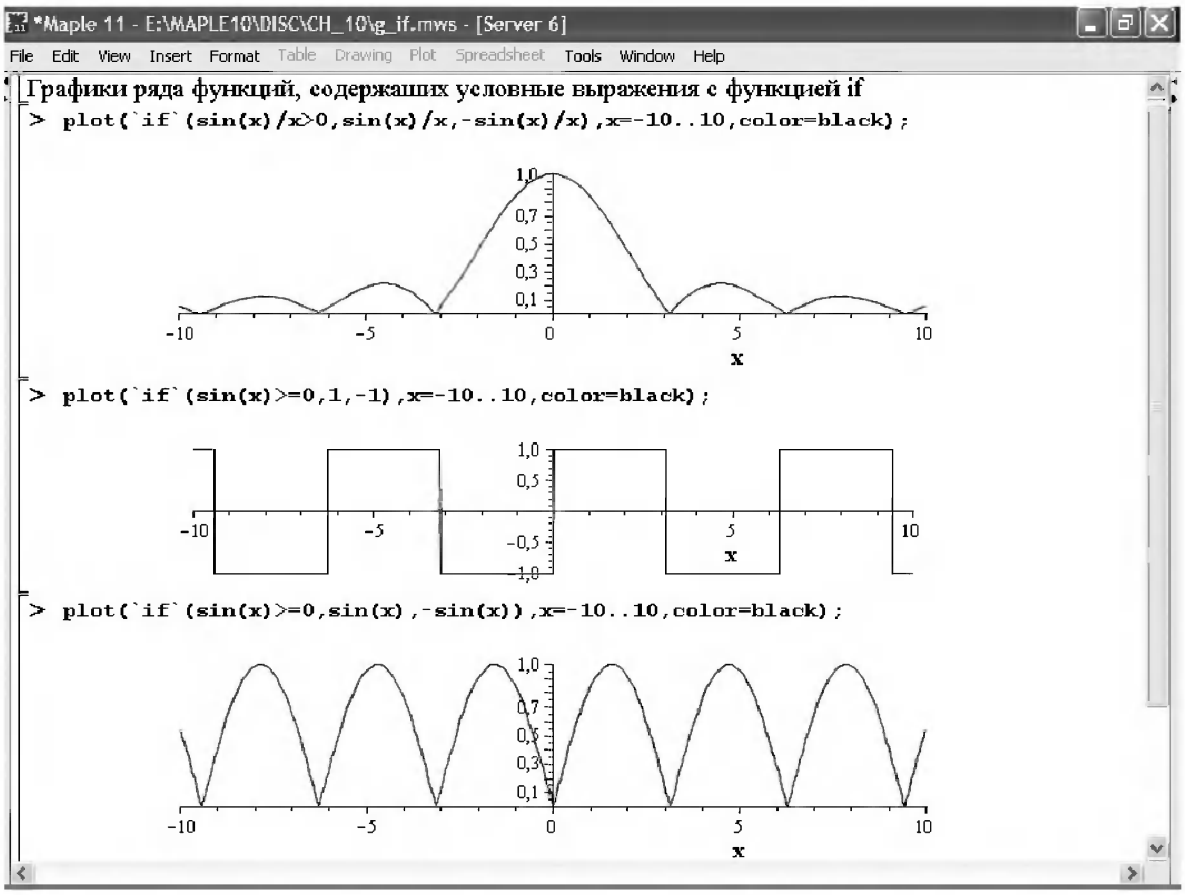


Рис. 14.3. Применение конструкции с функцией if для моделирования сигналов

14.2.4. Конструкции циклов в Maple

Зачастую необходимо циклическое повторение выполнения выражения заданное число раз или до тех пор, пока выполняется определенное условие. Maple имеет обобщенную конструкцию *цикла*, которая задается следующим образом:

```
|for <name>| |from <expr1>| |to <expr3>| |by <expr2>| |while <expr4>|
do <statement sequence> od;
```

Здесь `name` – имя *управляющей переменной* цикла, `expr1`, `expr2` и `expr3` – выражения, задающие начальное значение, конечное значение и шаг изменения переменной `name`, `expr4` – выражение, задающее условие, пока цикл (набор объектов между словами `do` и `od`) будет выполняться.

В ходе выполнения цикла управляющая переменная меняется от значения `expr1` до значения `expr2` с шагом, заданным `expr3`. Если блок `by <expr2>` отсутствует, то управляющая переменная будет меняться с шагом +1 при `expr1<expr2`. Это наглядно поясняет следующий пример:

```
> for i from 1 to 5 do print(i) od;
                                1
                                2
                                3
                                4
                                5
```

В нем выводятся значения переменной i в ходе выполнения цикла. Нетрудно заметить, что она и впрямь меняется от значения 1 до значения 5 с шагом +1. Следующий пример показывает, что границы изменения управляющей переменной можно задать арифметическими выражениями:

```
> for i from 7/(2+5) to 2+3 do print(i) od;
                                    1
                                    2
                                    3
                                    4
                                    5
```

А еще один пример показывает задание цикла, у которого переменная цикла меняется от значения 1 до 10 с шагом 2:

```
> for i from 1 to 10 by 2 do print(i) od;
                                    1
                                    3
                                    5
                                    7
                                    9
```

В этом случае выводятся нечетные числа от 1 до 9. Шаг может быть и отрицательным:

```
> for i from 9 to 1 by -2 do print(i) od;
                                    9
                                    7
                                    5
                                    3
                                    1
```

Следует отметить, что если expr1>expr2 задать заведомо невыполнимое условие, например expr1>expr2, и положительное значение шага, то цикл выполняться не будет. Цикл можно прервать с помощью дополнительного блока while <expr4>. Цикл с таким блоком выполняется до конца или до тех пор, пока условие expr4 истинно:

```
> for i from 1 to 10 by 2 while i<6 do print(i) od;
                                    1
                                    3
                                    5
```

Таким образом, конструкция цикла в Maple-языке программирования вобрала в себя основные конструкции циклов for и while. Есть еще одна, более специфическая конструкция цикла:

```
|for <name>| |in <expr1>| |while <expr2>| do <statement sequence> od;
```

Здесь expr1 задает список значений, которые будет принимать управляющая переменная name. Цикл будет выполняться, пока не будет исчерпан список и пока выполняется условие, заданное выражением expr2. Следующий пример иллюстрирует сказанное:

```
> for i in [1,2,5,-1,7,12] do print(i) od;
```

$$1$$
$$2$$
$$5$$
$$-1$$
$$7$$
$$12$$

```
> for i in [1,2,5,-1,7,12] while i>0 do print(i) od;
```

$$1$$
$$2$$
$$5$$

В цикле этого вида управляющая переменная может меняться произвольно.

Циклы могут быть *вложенными*. Это иллюстрирует следующий пример, создающий единичную матрицу на базе заданного массива `M`:

```
> M:=array(1..3,1..3);
```

$$M := \mathrm{array}(1 .. 3, 1 .. 3, [\ ])$$

```
> for i to 3 do for j to 3 do M[i,j]:=0;if i=j then M[i,j]:=1 fi;
od od;
> evalm(M);
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Этот пример имеет не более чем познавательное значение, поскольку для создания такой матрицы Maple имеет функции `identity`, с помощью которой функция `array` позволяет сразу создать единичную матрицу:

```
> array(1..3,1..3,identity);
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

В заключение отметим, что возможна упрощенная частная конструкция цикла типа `while`:

```
while expr do statseq od;
```

Здесь выражения `statseq` выполняются, пока выполняется логическое условие `expr`. Пример такого цикла:

```
> n:=1;
```

$$n := 1$$

```
> while n<16 do n:=2*n od;
```

$$n := 2$$
$$n := 4$$
$$n := 8$$
$$n := 16$$

В этом примере идет удвоение числа n с начальным значением n =1 до тех пор, пока значение n меньше 16.

Иногда бывает нужным пропустить определенное значение переменной цикла. Для этого используется оператор `next` (следующий). Приведенный ниже пример иллюстрирует применение оператора `next` в составе выражения `if-fi` для исключения вывода значения $i = -2$:

```
> for i in [1,2,3,-2,4] do if i=-2 then next else print(i) fi od;
                                    1
                                    2
                                    3
                                    4
```

Другой оператор – `break` – прерывает выполнение фрагмента программы (или цикла). Его действие поясняет слегка модифицированный предшествующий пример:

```
> for i in [1,2,3,-2,4] do if i=-2 then break else print(i) fi od;
                                    1
                                    2
                                    3
```

В данном случае при значении $i = -2$ произошло полное прекращение выполнения цикла. Поэтому следующее значение 4 переменной i присвоено не было и это значение на печать не попало.

Любой из операторов `quit`, `done` или `stop` обеспечивает также прерывание выполнения текущей программы (в частности, цикла), но при этом окно текущего документа закрывается.

14.2.5. Процедуры и процедуры-функции

Простейшая форма задания процедуры в Maple-языке программирования следующая:

```
name := proc(Параметры)
  Тело процедуры
end;
```

Параметры процедуры задаются перечислением имен переменных, например `proc(x)` или `proc(x,y,z)`. С помощью знака `::` после имени переменной можно определить ее тип, например в объявлении `prog(n::integer)` объявляется, что переменная `n` является целочисленной.

При вызове процедуры выражением вида

```
name(Фактические_параметры)
```

фактические параметры подставляются на место формальных. Несоответствие фактических параметров типу заданных переменных ведет к сообщению об ошибке и к отказу от выполнения процедуры.

В качестве примера ниже приведена процедура вычисления модуля комплексного числа `z` – в данном случае это единственный параметр процедуры:

```
> modc:=proc(z)
> evalf(sqrt(Re(z)^2+Im(z)^2))
> end;
```

$$modc := \mathbf{proc}(z)\text{evalf}(\text{sqrt}(\Re(z)\text{^}2+\Im(z)\text{^}2))\ \mathbf{end\ proc}$$

После ввода заголовка процедуры под строкой ввода появляется сообщение: «Warning, premature end of input». Оно указывает на то, что ввод листинга процедуры не закончен и должен быть продолжен до тех пор, пока не будет введено завершающее слово `end` листинга процедуры. Если после этого слова поставить символ точки с запятой, то листинг процедуры будет выведен на экран дисплея.

Теперь для вычисления модуля достаточно задать обращение к процедуре `modc(z)`, указав вместо `z` конкретное комплексное число:

```
> modc(3.+I*4.);
```

$$5.000000000$$

Нетрудно заметить, что при знаке `;` после завершающего слова `end` текст процедуры повторяется в строке вывода (в общем случае в несколько ином виде). Если это повторение не нужно, после слова `end` надо поставить знак двоеточия. Обратите также внимание на то, что для обозначения действительной и мнимой частей процедуры в ее тексте появились готические буквы.

Как отмечалось, процедуры, которые возвращают значение результата в ответ на обращение к ним, во многом тождественны функциям. Будем называть их процедурами-функциями. Обычно процедура возвращает значение последнего выражения в ее теле или выражения, намеченного к возврату специальным *оператором возврата* `RETURN`:

```
> modc:=proc(z)
> evalf(sqrt(Re(z)^2+Im(z)^2)):
> RETURN(%)
> end;
```

$$modc := \mathbf{proc}(z)\text{evalf}(\text{sqrt}(\Re(z)\text{^}2+\Im(z)\text{^}2)); \text{RETURN}(\%)\ \mathbf{end\ proc}$$

```
> modc(3.+I*4.);
```

$$5.000000000$$

Параметром оператора `RETURN` может быть любое выражение. В Maple не принято выделять процедуры-функции в какой-то отдельный класс. Действует правило: если не использован оператор `RETURN`, процедура возвращает значение последнего выражения в ее теле. Для устранения выдачи значений выражений внутри процедуры-функции после них просто надо установить знак двоеточия.

Выше мы рассмотрели основные частные формы задания процедур. Все они могут быть объединены в *общую форму задания процедуры*:

```
name:=proc(<argseq>)          # объявление процедуры
  local<nseq>;                # объявление локальных переменных
  global<nseq>;               # объявление глобальных переменных
  options<nseq>;              # объявление расширяющих ключей
  description<stringseq>;     # объявление комментария
  <stateq>                    # выражения — тело процедуры
end; (или end:)               # объявление конца процедуры
```

Эта форма охватывает все описанные выше частные формы и позволяет готовить самые сложные и надежно работающие процедуры. Применение в процедурах ключей будет описано чуть ниже.

14.2.6. Статус переменных

Переменные, которые указываются в списке параметров (например, z в нашем примере) внутри процедуры, являются *локальными*. Это означает, что изменение их значений происходит лишь в теле процедуры, то есть локально. За пределами тела процедуры эти переменные имеют то значение, которое у них было до использования процедуры. Это хорошо поясняет следующий пример:

```
> restart:z:=1;
```

$$z := 1$$

```
> modc:=proc(z)
> evalf(sqrt(Re(z)^2+Im(z)^2));
> end;
```

$$modc := \mathbf{proc}(z)\mathrm{evalf}(\mathrm{sqrt}(\Re(z)\,\hat{}\,2+\Im(z)\,\hat{}\,2))\ \mathbf{end\ proc}$$

```
> modc(3.+I*4.);
```

$$5.000000000$$

```
> z;
```

$$1$$

Нетрудно заметить, что внутри процедуры $z = 3 + I*4$, тогда как вне ее значение $z = 1$. Таким образом, имена переменных в списке параметров процедуры могут совпадать с именами глобальных переменных, используемых за пределами процедуры.

Переменные, которым впервые присваивается значение в процедуре, также относятся к локальным. Кроме того, переменные, применяемые для организации циклов, являются локальными. Все остальные переменные – глобальные.

Если в теле процедуры имеются операции присваивания для ранее определенных (глобальных) переменных, то изменение их значений в ходе выполнения процедуры создает так называемый побочный эффект. Он способен существенно изменить алгоритм решения сложных задач и, как правило, недопустим.

Поэтому Maple-язык программирования имеет встроенные средства для исключения побочных эффектов. Встречая такие операции присваивания, Maple-язык корректирует текст процедуры, вставляет в нее объявление переменных локальными с помощью ключевого слова `local` и выдает предупреждающую надпись о подобном применении:

```
> restart:m:=0;
```

$$m := 0$$

```
> modc:=proc(z)
> m:=evalf(sqrt(Re(z)^2+Im(z)^2)):RETURN(m)
> end;
Warning, `m` is implicitly declared local to procedure `modc`
```

$$modc := \mathbf{proc}(z)\mathbf{local}\ m; m := \mathrm{evalf}(\mathrm{sqrt}(\Re(z)\,\hat{}\,2+\Im(z)\,\hat{}\,2)); \mathrm{RETURN}(m)\ \mathbf{end\ proc}$$

```
> modc(3.+I*4.);
```

$$5.000000000$$

```
> m;
```

$$0$$

Обратите внимание на то, что в тело процедуры было автоматически вставлено определение `local m`, задающее локальный статус переменной `m`. Оператором `print` можно вывести текст процедуры:

```
> print(modc);
```

$$\mathbf{proc}(z)\mathbf{local}\ m; m := \mathrm{evalf}(\mathrm{sqrt}(\Re(z)\,\hat{}\,2 + \Im(z)\,\hat{}\,2)); \mathrm{RETURN}(m)\ \mathbf{end\ proc}$$

Говорят, что запретный плод сладок! Что бы ни говорили о нежелательности работы с глобальными переменными, бывает, что их применение желательно или даже необходимо. Чтобы сделать переменные внутри процедуры *глобальными*, достаточно объявить их с помощью ключевого слова `global`, после которого перечисляются идентификаторы переменных.

Следующий пример поясняет применение оператора `global` в процедуре:

```
> a:=1;b:=1;
```

$$a := 1$$
$$b := 1$$

```
> fg:=proc(x,y)
> global a,b;
> a:=x^2:b:=y^2:
> RETURN(sqrt(a+b));
> end;
```

$$fg := \mathbf{proc}(x,y)\mathbf{global}\ a,b; a := x\,\hat{}\,2; b := y\,\hat{}\,2; \mathrm{RETURN}(\mathrm{sqrt}(a+b))\ \mathbf{end\ proc}$$

```
> fg(3,4);
```

$$5$$

```
> [a,b];
```

$$[9, 16]$$

В данном примере переменным `a` и `b` вначале присвоены значения 1. Поскольку они в процедуре объявлены глобальными, то внутри процедуры они принимают новые значения x^2 и y^2. В результате при выходе из процедуры они имеют уже новые значения. Это и есть побочный эффект при исполнении данной процедуры.

Следует отметить, что нельзя делать глобальными переменные, указанные в списке параметров процедуры, поскольку они уже фактически объявлены локальными. Такая попытка приведет к появлению сообщения об ошибке следующего вида: «Error, argument and global `x` have the same name». При этом соответствующие переменные останутся локальными.

14.2.7. Вывод сообщений об ошибках

При профессиональной подготовке процедур пользователь должен предусмотреть их поведение при возможных ошибках. Например, если он готовит процедуру или функцию, вычисляющую квадратный корень из действительных чисел, то надо учесть, что такой корень нельзя извлекать из отрицательных чисел (будем,

исключительно в учебных целях, считать, что комплексные числа в данном примере недопустимы).

Для контроля за типом данных обычно используются различные *функции оценки и тестирования*. При выявлении ими ошибки, как правило, предусматривается вывод соответствующего сообщения. Для этого используется функция ERROR:

```
ERROR(expr_1, expr_2, …),
```

где expr_1, … – ряд выражений (возможно, пустой). Наиболее часто ERROR выводит просто строковое сообщение об ошибке, например ERROR(`strings`). Полное сообщение об ошибке имеет вид:

```
Error, (in name) string, …
```

Приведем пример процедуры, в которой предусмотрен вывод сообщения об ошибке при задании переменной $x < 0$:

```
> f := proc (x) if x<0 then error «invalid variable x: %1», x else
x^(1/2) end if end proc;
```

f := proc(**x**)
if *x* < 0 **then error** "invalid variable x: %1", *x* **else** *x*^(1/2) **end if**
end proc

```
> f(3.);
```

1.732050808

```
> f(-3.);
Error, (in f) invalid variable x: -3.
> lasterror;
```

"invalid variable x: %1", -3

```
> lastexception;
```

f, "invalid variable x: %1", -3

Эта процедура вычисляет квадратный корень из числа *x*. При $x < 0$ выводится заданное сообщение об ошибке. Еще раз обращаем внимание читателя на учебный характер данного примера, поскольку вычисление квадратного корня (в том числе из комплексных и отрицательных действительных чисел) реализовано встроенной функцией sqrt.

14.2.8. Ключи в процедурах

В объявление процедуры можно включить ключевые слова, вводимые словом

```
options opseq
```

Иногда их называют *расширяющими ключами*. Предусмотрены следующие ключи:

- arrow – определяет процедуру-оператор в нотации ->;
- builtin – определяет функцию как встроенную;
- call_external – задает обращение к внешним программным модулям;
- copyright – защищает процедуру от копирования;
- inline – определяет процедуру как подчиненную (возможно, не для всех процедур – см. справку);

- `load=memberName` – загружает нужный для определений процедуры модуль (см. также опцию unload и детали в справке);
- `operator` – объявляет процедуру – функциональный оператор;
- `system` – определяет процедуру как системную,
- `remember` – определяет таблицу памяти для процедуры;
- `trace` – задает трассировку процедуры;
- `unload=memberName` – выгружает нужный для определений процедуры модуль (см. опцию load).

Ключ `remember` обеспечивает занесение результатов обращений к процедуре в таблицу памяти, которая используется при исполнении процедуры. Функция `op` позволяет вывести таблицу:

```
> f:=proc(x) options remember; x^3 end:
> f(2);
```

$$8$$

```
> f(3);
```

$$27$$

```
> op(4,eval(f));
```

$$\text{table}([2 = 8, 3 = 27])$$

Ключ `remember` особенно полезен при реализации итерационных процедур. К примеру, в приведенной ниже процедуре (без использования ключа `remember`) время вычисления n-го числа Фибоначчи растет пропорционально квадрату n:

```
> f:=proc(n) if n<2 then n else f(n-1)+f(n-2) fi end;
```

$$f := \textbf{proc}(n)\ \textbf{if}\ n < 2\ \textbf{then}\ n\ \textbf{else}\ \mathrm{f}(n-1) + \mathrm{f}(n-2)\ \textbf{end if end proc}$$

```
> time(f(30));
```

$$4.891$$

$$27.400$$

```
> f(30);
```

$$832040$$

Вычисление `f(30)` по этой процедуре на ПК в системе Maple 8 с процессором Pentium II 350 МГц занимает 27,4 с. На ПК с процессором Pentium IV HT 2,6 ГГц в системе Maple 9.5 время вычисления составляет менее 5 с – см. контроль этого времени с помощью функции `time` (результат в секундах).

Стоит добавить в процедуру ключ `remember`, и время вычислений резко уменьшится:

```
> restart;
> fe:=proc(n) options remember; if n<2 then n else fe(n-1)+fe(n-2)
fi end:
> fe(30);
```

$$832040$$

```
> time(fe(30));
```

$$0.$$

При этом вычисление `fe(30)` происходит практически мгновенно, так как все промежуточные результаты в первом случае вычисляются заново, а во втором они

берутся из таблицы. Однако это справедливо лишь тогда, когда к процедуре было хотя бы однократное обращение. Обратите внимание на то, что данные процедуры являются *рекурсивными* – в их теле имеется обращение к самой себе.

Ключ `builtin` придает процедуре статус *встроенной*. Он должен всегда использоваться первым. С помощью функции `eval(name)` можно проверить, является ли функция с именем `name` встроенной:

```
> eval(type);
```

proc() option *builtin*; 274 **end proc**

```
> eval(print);
```

proc() option *builtin*; 235 **end proc**

Числа в теле процедур указывают системные номера функций. Следует отметить, что они различны в разных версиях системы Maple.

Этот ключ придает процедуре статус *системной*. У таких процедур таблица памяти может быть удалена. У обычных процедур таблица памяти не удаляется и входит в так называемый «мусорный ящик» (garbage collector).

Эта пара ключей задает процедуре статус оператора в «стрелочной» нотации (->). Это достаточно пояснить следующими примерами:

```
> o:=proc(x,y) option operator , arrow; x-sqrt(y) end;
```

$$o := (x, y) \to x - \sqrt{y}$$

```
> o(4,2);
```

$$4 - \sqrt{2}$$

```
> o(4,2.);
```

$$2.585786438$$

Ключ `trace` задает вывод отладочной информации:

```
> o:=proc(x,y) option trace, arrow; x-sqrt(y) end;
```

$o :=$ **proc**(x, y) **option** *trace*, *arrow*; $x - \mathrm{sqrt}(y)$ **end proc**

```
> o(4,2.);
{-> enter o, args = 4, 2.
```

$$2.585786438$$

```
<- exit o (now at top level) = 2.585786438}
```

$$2.585786438$$

Этот ключ защищает тело процедуры от просмотра. Это поясняют следующие два примера:

```
> o:=proc(x,y) x-sqrt(y) end;
```

$o :=$ **proc**(x, y) $x - \mathrm{sqrt}(y)$ **end proc**

```
> oo:=proc(x,y) option Copyright; x-sqrt(y) end;
```

$oo :=$ **proc**(x, y) ... **end proc**

```
> oo(4,2.);
```

$$2.585786438$$

Нетрудно заметить, что во втором примере тело процедуры уже не просматривается. Для отмены защиты от просмотра можно использовать оператор `interface(verboseproc=2)`.

14.2.9. Средства контроля и отладки процедур

Для *контроля* и *отладки* процедур, а также для их разбора с целью выяснения примененного алгоритма работы прежде всего надо уметь вывести их текст. Для этого служит функция

```
print(name);
```

где name – имя процедуры.

Однако перед тем как использовать эту функцию, надо исполнить команду

```
> interface(verboseproc=2,prettyprint=1,version);
    1, 3, Standard Worksheet Interface, Maple 11.0, Windows XP,
  February 17 2007

  Build ID 277223
```

Ее смысл будет пояснен ниже. Пока же отметим, что эта команда обеспечивает полный вывод текста процедур библиотеки. Встроенные в ядро процедуры, написанные не на Maple-языке, в полном тексте не представляются. Поясним это следующими примерами:

```
> print(evalf);
    proc() option builtin = evalf, remember;  end proc;

> print(erf);

proc(x::algebraic)
 local res;
 option
   `Copyright (c) 1994 by the University of Waterloo. All
    rights reserved.`;
   if nargs <> 1 then
     error "expecting 1 argument, got %1", nargs
   elif type(x, 'complex(float)') then
     return evalf('erf'(x))
   elif type(x, 'infinity') then
     if type(x, 'cx_infinity') then
       res := undefined + undefined*I
     elif type(x, 'undefined') then
       res := NumericTools:-ThrowUndefined(x)
     elif type(Re(x), 'infinity') then
       res := CopySign(1, Re(x))
     elif type(x, 'imaginary') then
       res := x
     else
       res := infinity + infinity*I
     end if;
   elif type(x, 'undefined') then
```

```
    res := NumericTools:-ThrowUndefined(x, 'preserve' = 'axes')
  elif `tools/sign`(x) = -1 then
    res := -erf(-x)
  else
    res := 'erf'(x)
  end if;
  erf(args) := res;
end proc;
```

Здесь вначале выполнен вывод сокращенного листинга встроенной в ядро процедуры `evalf`, а затем выведен полный листинг процедуры вычисления функции ошибок `erf`.

Но вернемся к функции `interface`. Эта функция служит для управления выводом и задается в виде

`interface( arg1, arg2, … )`,

где аргументы задаются в виде равенств вида `name=value` и слов-указателей:

ansi	**autoassign**	**echo**	**errorbreak**	**errorcursor**
imaginaryunit	**indentamount**	**labelling**	**labelwidth**	**latexwidth**
longdelim	**patchlevel**	**plotdevice**	**plotoptions**	**plotoutput**
postplot	**preplot**	**prettyprint**	**prompt**	**quiet**
rtablesize	**screenheight**	**screenwidth**	**showassumed**	**verboseproc**
version	**warnlevel**			

Рассмотрим только некоторые, наиболее важные возможности этой функции.

Указание `verboseproc=n` задает степень детальности вывода листинга процедур. При `n=0` текст не выводится, при `n=1` выводится текст только заданных пользователем процедур, а при `n=2` – всех процедур на Maple-языке. Пример этого был дан выше. Указание `prettyprint=0` или `1` управляет выводом стандартных сообщений. Указание `plotdevice=string` управляет выводом графики, например `plotdevice=gif` указывает на то, что запись графиков в виде файлов будет происходить в формате `.gif`.

Одним из основных средств отладки процедур является функция трассировки `trace(name)`. Детальность ее работы задается системной переменной `printlevel` (уровень вывода). При `printlevel:=n` (значение $n = 1$ по умолчанию) выводится результат только непосредственно исполняемой функции или оператора. Для вывода информации о выполнении k-го уровня вложенности надо использовать значение этой переменной от $5*k$ до $5*(k+1)$. Так, при n от 1 до 5 выводятся результаты трассировки первого уровня, при n от 6 до 10 – второго и т. д. Максимальное значение $n = 100$ обеспечивает трассировку по всем уровням вложенности процедуры `name`.

Следующий пример показывает осуществление трассировки для функции `int(x^n,x)`:

```
> printlevel:=5;
                          printlevel := 5
> trace(int);
{-> enter trace, args = int
```

```
<- exit trace (now at top level) = int}
                                    int
> int(x^n,x);
{-> enter int, args = x^n, x
                                     n + 1)
                                    x
                          answer := --------
                                      n + 1
                                   (n + 1)
                                  x
                                  --------
                                    n + 1
<- exit int (now at top level) = x^(n+1)/(n+1)}
                                   (n + 1)
                                  x
                                  --------
                                    n + 1
```

Действие функции трассировки отменяется командой `untrace`:

```
> untrace(int);
{-> enter untrace, args = int
<- exit untrace (now at top level) = int}
                                     int
> int(x^n,x);
{-> enter int, args = x^n, x
<- exit int (now at top level) = x^(n+1)/(n+1)}
                                   (n + 1)
                                  x
                                  --------
                                    n + 1
> printlevel:=1;
printlevel := 1

> int(x^n,x);
                                   (n + 1)
                                  x
                                  --------
                                    n + 1
```

При отладке алгоритмов выполнения вычислений надо тщательно следить за сообщениями об ошибках. Для этого в Maple предусмотрены функция `traceerr` и системная переменная `lasterr`, в которой сохраняется последнее сообщение об ошибке. При каждом обращении к `tracerr` переменная `lasterr` очищается:

```
> 2/0;
Error, numeric exeption:division by zero
> 2/4;
                                   1/2
> 2/.3;
                              6.666666667
```

```
> lasterror;
```

"division by zero"

```
> traperror(3/4);
```

3/4

```
> lasterror;
```

lasterror

```
> traperror(5/0);
```

Error, numeric exeption:division by zero

```
> lasterror;
```

"numeric exeption:division by zero"

Этот пример показывает, как может быть проведено отслеживание ошибок в ходе вычислений.

14.2.10. Работа с отладчиком программ

В большинстве случаев составители программ (процедур) редко прибегают к пошаговой их отладке. Средства общей диагностики уже в Maple 9.5 развиты настолько хорошо, что позволяют выявлять грубые ошибки в процедурах при их выполнении. Иногда, правда, для этого приходится неоднократно «прогонять» процедуру, пока она не начнет работать как задумано. Тем не менее для отладки процедур служит специальный интерактивный отладчик (debugger). Опишем, как его запустить и как с ним работать.

Допустим, мы составили некоторую процедуру `demo`, вычисляющую сумму квадратов чисел `(1^2+2^2+...+n^2)`:

```
> demo:=proc(n::integer) local y,i:
>      y:=0:
>      for i to n do y:=y+i^2 od
> end;
```

demo := **proc**(*n*::*unteger*) **local** *y*, *i*; *y* := 0; **for** *i* **to** *n* **do** *y* := *y* + *i*^2 **end do end proc**

```
> demo(3);
```

14

Чтобы включить отладчик в работу, надо исполнить команду `stopat`:

```
> stopat(demo);
```

[*demo*]

```
> demo(3);
demo:
      1*  y := 0;

DBG>
```

Признаком, указывающим на работу отладчика, является изменение приглашения к вводу со знака > на `DBG>` (как нетрудно догадаться, `DBG` означает *debugger*). Теперь, подавая команды `next` (следующий), `step` (шаг) и `stop` (остановка), можно проследить выполнение процедуры:

```
DBG> next
0
demo:
       2       for i to n do
                 ...
               end do

DBG> step
0
demo:
       3       y := y+i^2

DBG> step
1
demo:
       3       y := y+i^2

DBG> step
5
demo:
       3       y := y+i^2

DBG> step
14
```

В последнем случае процедура по шагам дошла до конца вычислений; на этом работа отладчика завершается сама собой.

Можно также вывести листинг процедуры с помощью команды `showstat`:

```
> showstat(demo);

demo := proc(n::integer)
local y, i;
       1*      y := 0;
       2       for i to n do
       3       y := y+i^2
               end do
end proc
```

Обратите внимание, что в этом листинге строки вычисляемых элементов пронумерованы. Это сделано для облегчения разбора работы процедуры.

В общем случае отладчик выключается при выполнении команд `stopat`, `stopwhen` или `stoperr`. Если используется команда `stopat`, то вывод на экран соответствует исполнению последней выполненной команды. Для отмены этой команды используется команда `unstopat`.

Команда `stopwhen` позволяет установить точку наблюдения за указанной в команде переменной. Отменить ее можно командой `unstopwhen`. Команда `stoperror` позволяет задать остановку при появлении определенной ошибки. Для отмены этой команды используется команда `unstoperror`.

Команда `cont` используется для продолжения работы до следующей точки прерывания, установленной указанными выше командами, или до конца процедуры. Для прерывания отладки можно использовать команду `quit`. После команды `stop` можно вычислить любое Maple-выражение.

В действительности команд отладчика намного больше, и их функции более развиты, чем это описано выше. Пользователи, заинтересованные в серьезной работе с отладчиком (скорее всего, их немного), могут просмотреть его подробное описание. Для этого в разделе справочной системы `Context` найдите раздел `Programming`, а в нем – раздел `Debugging`.

14.2.11. Операции ввода и вывода

Выше неоднократно рассматривалась работа с файлами документов и данных. Вводимые в текущий документ программные модули хранятся вместе с ним, так что при отказе от загрузки какого-либо документа все его программные блоки не могут использоваться в других документах. Кроме того, порой неудобно загружать объемный документ ради использования одного или нескольких модулей, например процедур. Поэтому в Maple введены средства, позволяющие записывать нужные модули (в том числе результаты вычислений) на диск и считывать их в случае необходимости.

Для записи на диск используется оператор `save`:

- `save filename` – запись всех определений текущего файла под именем filename;
- `save name_1, name_2, ..., name_k, filename` – запись избранных модулей с именами `name_1`, `name_2`, ..., `name_k` под именем filename.

Считывание имеющегося на диске файла filename осуществляется оператором `read`:

```
read <filename>
```

При считывании все имеющиеся в файле определения становятся доступными для рабочих документов Maple. При записи файлов отдельных определений используется специальный внутренний Maple-формат файлов. Для загрузки файлов типа `*.m` из стандартной библиотеки используется функция `readlib`. А для записи файлов в качестве библиотечных достаточно в имени `filename` оператора `save` указать расширение .m. Разумеется, можно считывать такие файлы и оператором `read`, указав в имени файла расширение .m:

```
> save my_proc,`my_lib.m`: # запись файла my_proc и
> # библиотечного файла my_lib.m;
> load `my_lib.m`: # считывание библиотечного файла
> # my_lib.m.
```

14.2.12. Создание своей библиотеки процедур

Если приведенные выше примеры составления процедур кажутся вам простыми, значит, вы неплохо знаете программирование и, скорее всего, уже имеете несколько полезных процедур, которые вы хотели бы сохранить – если не для потомков, то хотя бы для своей повседневной работы. Сделать это в Maple довольно просто.

Прежде всего надо определить имя своей библиотеки, например `mylib`, и создать для нее на диске каталог (папку) с заданным именем. Процедуры в Maple ассоциируются с таблицами. Поэтому вначале надо задать таблицу-пустышку под будущие процедуры:

```
> restart;
> mylib:=table();
```

$$mylib := \text{table}([])$$

Теперь надо ввести свои *библиотечные процедуры*. Они задаются с двойным именем – вначале указывается имя библиотеки, а затем в квадратных скобках имя процедуры. Для примера зададим три простые процедуры с именами `f1`, `f2` и `f3`:

```
> mylib[f1]:=proc(x::anything) sin(x)+cos(x) end:
> mylib[f2]:=proc(x::anything) sin(x)^2+cos(x)^2 end:
> mylib[f3]:=proc(x::anything) if x=0 then 1 else sin(x)/x fi end:
```

Рекомендуется тщательно проверить работу процедур, прежде чем записывать их на диск. Ограничимся, скажем, такими контрольными примерами:

```
> mylib[f1](x);
```

$$\sin(x) + \cos(x)$$

```
> mylib[f1](1.);
```

$$1.381773291$$

```
> mylib[f2](x);
```

$$\sin(x)^2 + \cos(x)^2$$

```
> simplify(mylib[f2](x));
```

$$1$$

```
> evalf(mylib[f3](x));
```

$$\frac{\sin(x)}{x}$$

```
> sin(0)/0;
Error, division by zero

> mylib[f3](0);
```

$$1$$

```
> evalf(mylib[f3](.5));
```

$$.9588510772$$

Можно построить графики введенных процедур-функций. Они представлены на рис. 14.4.

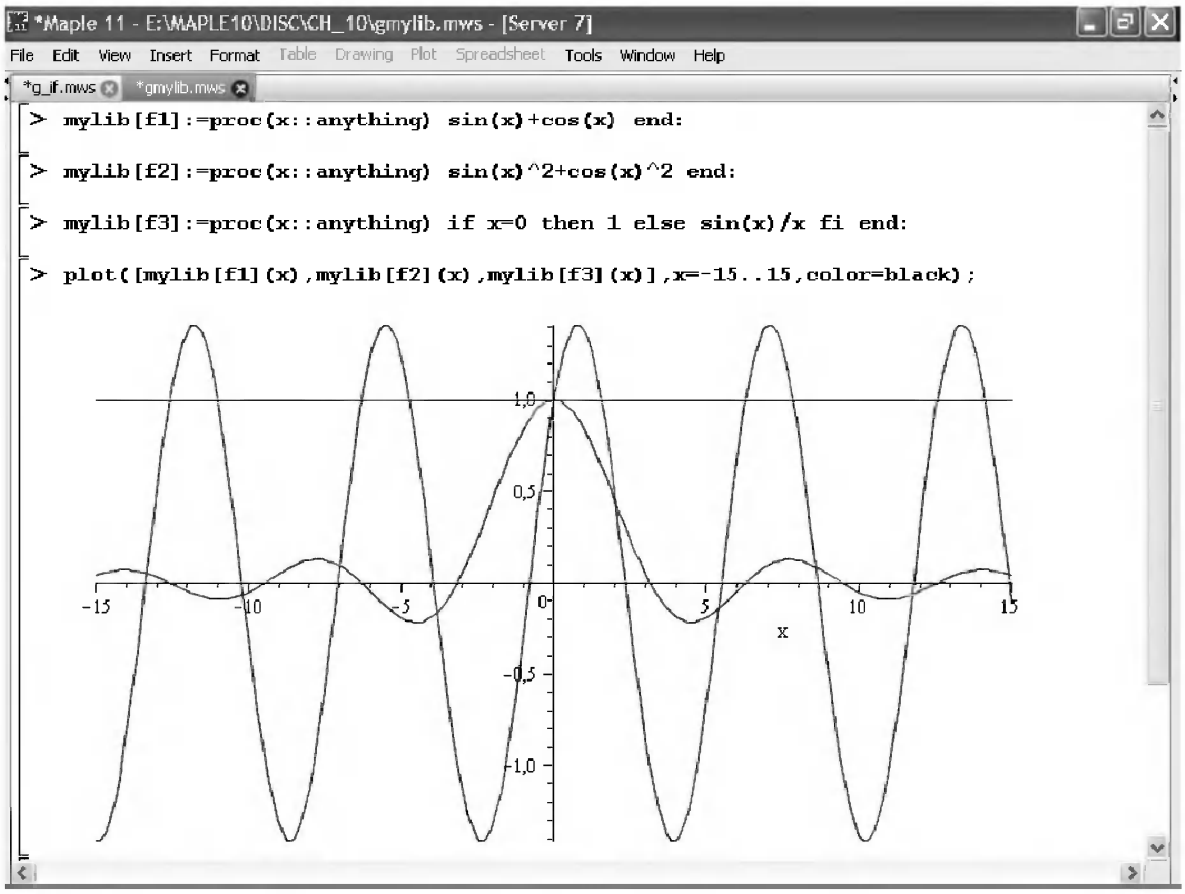


Рис. 14.4. Построение графиков процедур-функций f1, f2 и f3

С помощью функции `with` можно убедиться, что библиотека `mylib` действительно содержит только что введенные в нее процедуры. Их список должен появиться при обращении `with(mylib)`:

```
> with(mylib);
```

$$[f1, f2, f3]$$

Теперь надо записать эту библиотеку под своим именем на диск с помощью команды `save`:

```
> save(mylib,`c:/mylib.m`);
```

Обратите особое внимание на правильное задание полного имени файла. Обычно применяемый для указания пути знак \ в строках Maple-языка используется как знак продолжения строки. Поэтому надо использовать либо двойной знак \\, либо знак /. В нашем примере файл записан в корень диска C. Лучше поместить библиотечный файл в другую папку (например, в библиотеку, уже имеющуюся в составе системы), указав полный путь до нее.

После всего этого надо убедиться в том, что библиотечный файл записан. Затем можно сразу и считать его. Для этого вначале следует командой `restart` устранить ранее введенные определения процедур:

```
> restart;
```

С помощью команды `with` можно убедиться в том, что этих определений уже нет:

```
> with(mylib);
Error, (in pacman:-pexports) mylib is not a package
```

После этого командой `read` надо загрузить библиотечный файл:

```
> read(`c:/mylib.m`);
```

Имя файла надо указывать по правилам, определенным для команды `save`. Если все выполнено пунктуально, то команда `with` должна показать наличие в вашей библиотеке списка процедур `f1`, `f2` и `f3`:

```
> with(mylib);
[f1, f2, f3]
```

И наконец, можно вновь опробовать работу процедур, которые теперь введены из загруженной библиотеки:

```
> f1(x);
```

$$\sin(x) + \cos(x)$$

```
> simplify(f2(y));
```

$$1$$

```
> f3(0);
```

$$1$$

```
> f3(1.);
```

$$.8414709848$$

Описанный выше способ создания своей библиотеки вполне устроит большинство пользователей. Однако есть более сложный и более «продвинутый» способ ввода своей библиотеки в состав уже имеющейся. Для реализации этого Maple 8 имеет следующие операции записи в библиотеку процедур `s1`, `s2`, ... и считывания их из файлов file1, file2, ...:

```
savelib(s1, s2, …, sn, filename)
readlib(f, file1, file2, …)
```

С помощью специального оператора `makehelp` можно задать стандартное справочное описание новых процедур:

```
makehelp(n,f,b)
```

где `n` – название темы, `f` – имя текстового файла, содержащего текст справки (файл готовится как документ Maple), и `b` – имя библиотеки. Системная переменная `libname` хранит имя директории библиотечных файлов. Для регистрации созданной справки надо исполнить команду вида

```
libname:=libname, `/mylib`;
```

С деталями применения этих операторов можно ознакомиться в справочной системе.

К созданию своих библиотечных процедур следует относиться достаточно осторожно. Их применение лишает ваши Maple-программы совместимости со стандартной версией Maple. Если вы используете одну-две процедуры, проще поместить их в те документы, в которых они действительно нужны. Иначе вы будете вынуждены к каждой своей программе прикладывать еще и библиотеку процедур. Особенно рискованно изменять стандартную библиотеку Maple.

14.2.13. Программирование символьных операций

Найти достаточно простую и наглядную задачу, решение которой отсутствует в системе Maple, не очень просто. Поэтому для демонстрации решения задачи с применением аналитических методов воспользуемся примером, ставшим классическим, – реализуем итерационный метод Ньютона при решении нелинейного уравнения вида $f(x) = 0$.

Как известно, *метод Ньютона* сводится к итерационным вычислениям по следующей формуле:

$$x_{i+1} = x_i + f(x_i)/f'(x_i).$$

Реализующая его процедура выглядит довольно просто:

```
> restart:
 NI:=proc(expr,x)
 local iter;
 iter:=x-expr/diff(expr,x);
 unapply(iter,x)
 end;
```

NI := **proc**(*expr*, *x*) **local** *iter*; *iter* := *x* – *expr*/diff(*expr*, *x*); unapply(*iter*, *x*) **end proc**

Для получения итерационной формулы в аналитическом виде здесь используется функция `unapply`. Теперь, если задать решаемое уравнение, то можно получить искомое аналитическое выражение:

```
> expr:=sin(x)^2-0.5;
```

$$expr := \sin(x)^2 - .5$$

```
> F:=NI(expr,x);
```

$$F := x \to x - \frac{1}{2}\frac{\sin(x)^2 - .5}{\sin(x)\cos(x)}$$

Далее, задав начальное приближение для x в виде $x = x_0$, можно получить результаты вычислений для ряда итераций:

```
> x0:=0.2;
```

$$x0 := .2$$

```
> to 8 do x0:=F(x0);od;
```

$$x0 := 1.382611210$$
$$x0 := .117460944$$
$$x0 := 2.206529505$$

$$x0 := 2.360830634$$
$$x0 := 2.356194357$$
$$x0 := 2.356194490$$
$$x0 := 2.356194490$$
$$x0 := 2.356194490$$

Нетрудно заметить, что, испытав скачок в начале решения, значения x довольно быстро сходятся к конечному результату, дающему корень заданной функции. Последние три итерации дают одно и то же значение x. Заметим, что этот метод дает только одно решение, даже если корней несколько. Вычислить другие корни в таком случае можно, изменив начальное условие.

Можно попробовать с помощью полученной процедуры получить решение и для другой функции:

```
> expr:=ln(x^2)-0.5;
```

$$expr := \ln(x^2) - .5$$

```
> F:=NI(expr,x);
```

$$F := x \to x - \frac{1}{2}(\ln(x^2\_ - .5)x$$

```
>> x0:=0.2;
```

$$x0 := .2$$

```
> to 8 do x0:=F(x0);od;
```

$$x0 := .5718875825$$
$$x0 := 1.034437603$$
$$x0 := 1.258023119$$
$$x0 := 1.283760340$$
$$x0 := 1.284025389$$
$$x0 := 1.284025417$$
$$x0 := 1.284025416$$
$$x0 := 1.284025417$$

Здесь итерационная формула имеет (и вполне естественно) уже другой вид, но сходимость к корню также обеспечивается за несколько итераций.

Возможна и иная форма задания итерационной процедуры с применением оператора дифференцирования `D` и заданием исходной функции также в виде процедуры:

```
> MI:=proc(f::procedure)
> (x->x)-eval(f)/D(eval(f));
> end;
```

$$MI := \mathbf{proc}(f::procedure)(x \to x) - \mathrm{eval}(f)/\mathrm{D}(\mathrm{eval}(f))\ \mathbf{end\ proc}$$

```
> g:=x->x-cos(x);
```

$$g := x \to x - \mathrm{cox}(x)$$

```
> SI:=MI(g);
```

$$SI := (x \to x) - \frac{x \to x - \cos(x)}{x \to 1 + \sin(x)}$$

```
> x0:=0.1;
```

$$x0 := .1$$

```
> to 6 do x0:=SI(x0) od;
```

$$x0 := .9137633858$$
$$x0 := .7446642419$$
$$x0 := .7390919660$$
$$x0 := .7390851333$$
$$x0 := .7390851332$$
$$x0 := .7390851332$$

Вообще говоря, в программных процедурах можно использовать любые операторы и функции, присущие Maple-языку, в том числе и те, которые реализуют символьные вычисления. Это открывает широкий простор для разработки новых процедур и функций, обеспечивающих выполнение символьных операций.

Рассмотрим следующий пример на вычисление интеграла по известной формуле:

```
> Int(e^x*x^n,x)=int(e^x*x^n,x);
```

$$\int e^x x^n dx = -(-1)^{(-n)} \ln(e)^{(-1-n)} (x^n (-1)^n \ln(e)^n n\Gamma(n)(-x\ln(e))^{(-n)}$$
$$- x^n (-1)^n \ln(e)^n \mathbf{e}^{(x\ln(e))} - x^n (-1)^n \ln(e)^n n(-x\ln(e))^{(-n)} \Gamma(n,-x\ln(e)))$$

Ранние версии системы Maple не брали этот интеграл. Maple, начиная с версии Maple 7, вычисляет этот «крепкий орешек», но полученное выражение довольно сложно.

Из математики известно, что такой интеграл может быть представлен в следующем виде:

$$\int x^n e^x dx = n! e^x \sum_{i=0}^{n} (-1)^{n-i} \frac{x^i}{i!}.$$

Используя эту формулу, мы можем создать простую процедуру для численного и аналитического вычисления данного интеграла:

```
> IntExpMonomial:=proc(n::anything,x::name)
 local i;
 n!*exp(x)*sum(((-1)^(n-i)*x^i)/i!,i=0..n);
 end;
```

IntExpMonomail := **proc**(*n*::*anything*; *x*::*name*)
local *i*;
n!×exp(*x*)×sum((-1)^(*n* – *i*)×*x*^*i*/*i*!, *i* = 0 .. *n*)
end proc

Проверим ее в работе:

```
> IntExpMonomial(3,x);
```

$$6\mathbf{e}^x \left(-1 + x - \frac{1}{2}x^2 + \frac{1}{6}x^3\right)$$

```
> IntExpMonomial(5,x);
```

$$120\mathbf{e}^x \left(-1 + x - \frac{1}{2}x^2 + \frac{1}{6}x^3 - \frac{1}{24}x^4 + \frac{1}{120}x^5\right)$$

```
> IntExpMonomial(n,x);
```

$$\frac{n!(x^{(1+n)}(-1-n)\Gamma(1+n,-x)+(-1)^n(1+n)!(-x)^{(1+n)}+x^{(1+n)}\Gamma(2+n))}{(1+n)!(-x)^{(1+n)}}$$

Результат в аналитическом виде довольно прост для данного интеграла с конкретным значением n. Более того, мы получили несколько иной результат и для n в общем случае. Но точен ли он? Для ответа на этот вопрос продифференцируем полученное выражение:

```
> diff(%,x);
```

$$n!\left(\frac{x^{(1+n)}(1+n)(-1-n)\Gamma(1+n,-x)}{x}+x^{(1+n)}(-1-n)(-x)^n\mathbf{e}^x\right.$$
$$\left.+\frac{(-1)^n(1+n)!(-x)^{(1+n)}(1+n)}{x}+\frac{x^{(1+n)}(1+n)\Gamma(2+n)}{x}\right)/((1+n)!$$
$$(-x)^{(1+n)})-n!$$
$$(x^{(1+n)}(-1-n)\Gamma(1+n,-x)+(-1)^n(1+n)!(-x)^{(1+n)}+x^{(1+n)}\Gamma)2+n))$$
$$(1+n)/((1+n)!(-x)^{(1+n)}x)$$

Результат дифференцирования выглядит куда сложнее, чем вычисленный интеграл. Однако с помощью функции `simplify` он упрощается к подынтегральной функции:

```
> simplify(%);
```

$$\mathbf{e}^x x^n$$

Это говорит о том, что задача вычисления заданного интеграла в аналитической форме действительно решена.

14.2.14. Вложенные процедуры и интегрирование по частям

Теперь мы подошли к важному моменту, о котором читатель наверняка уже давно догадался, – в составляемых пользователем процедурах можно использовать ранее составленные им (или кем-то еще) другие процедуры! Таким образом, Maple-язык позволяет реализовать процедуры, вложенные друг в друга.

Для иллюстрации применения *вложенных процедур* рассмотрим операцию интегрирования по частям. Пусть нам надо вычислить интеграл

$$\int p(x)e^x dx,$$

где $p(x)$ – выражение, представляющее полином.

Вначале подготовим процедуру `IntExpMonomialR`, реализующую вычисление уже рассмотренного ранее интеграла, но рекурсивным способом:

```
> IntExpMonomialR:=proc(n::nonnegint,x::name)
  local i;if n=0 then RETURN(exp(x)) fi;
  x^n*exp(x)-n*IntExpMonomialR(n-1,x);
  end;
```

IntExpMonomialR := **proc**(*n*::*nonnegint*; *x*::*name*)
local *i*;
  if *n* = 0 **then** RETURN(exp(*x*)) **end if**;
  x^*n*×exp(*x*) – *n*×*IntExpMonomialR*(*n* – 1, *x*)
end proc

Проверим ее в работе:

```
> IntExpMonomialR(4,x);
```

$$x^4\mathbf{e}^x - 4x^3\mathbf{e}^x + 12x^2\mathbf{e}^x - 24x\mathbf{e}^x + 24\mathbf{e}^x$$

```
> collect(%,exp(x));
```

$$(x^4 - 4x^3 + 12x^2 - 24x + 24)\mathbf{e}^x$$

Теперь составим процедуру для вычисления по частям нашего интеграла:

```
> IntExpPolynomial:=proc(p::polynom,x::name)
  local i,result;
  ### WARNING: degree(0,x) now returns -infinity
  result:=add(coeff(p,x,i)*IntExpMonomialR(i,x),i=0..degree(p,x));
  collect(result,exp(x));
  end;
```

IntExpPolynomial := **proc**(*p*::*polynom*, *x*::*name*)
local *i*, *result*;
  result := add(coeff(*p*, *x*, *i*)×*IntExpMonomialR*(*i*, *x*), *i* = 0 .. degree(*p*, *x*));
  collect(*result*, exp(*x*))
end proc

В этой процедуре имеется обращение к ранее составленной процедуре `IntExpMonomialR`. Обратите внимание на то, что в процедуре введено предупреждение об определенных проблемах, связанных с использованием функции `degree` (сообщение начинается с символов ###). Тем не менее процедура работает, в чем убеждают, по крайней мере, следующие примеры:

```
> p:=(x^2+1)*(1-3*x);
```

$$p := (x^2 + 1)(1 - 3x)$$

```
> expand(p);
```

$$x^2 - 3x^3 + 1 - 3x$$

```
> int(p*exp(x),x);
```

$$10x^2\mathbf{e}^x - 23x\mathbf{e}^x + 24\mathbf{e}^x - 3x^3\mathbf{e}^x$$

```
> IntExpPolynomial(p,x);
```

$$(10x^2 - 23x + 24 - 3x^3)\mathbf{e}^x$$

14.2.15. Дополнительные возможности, модули и макросы Maple-языка

В большинстве случаев Maple-язык использует достаточно длинные идентификаторы для своих определений, например функций. Однако с помощью функции `alias` можно изменить любое определение на другое, если оно кажется пользователю более удобным. Функция `alias` записывается в виде

`alias(e1, e2, ..., eN)`,

где `e1`, `e2`, ..., `eN` – нуль или более равенств.

Эта функция возвращает список переназначений и осуществляет сами переназначения. Например, для замены имени функции `BesselJ` на более короткое имя `BJ` достаточно параметром функции `alias` записать `BJ=BesselJ`:

```
> alias(BJ=BesselJ);
```

$$BJ, Fx$$

```
> [BJ(0,1.),BesselJ(0,1.)];
```

$$[.7651976866, .7651976866]$$

Можно также переназначить функцию пользователя:

```
> alias(Fx=F(x));
```

$$BJ, Fx$$

```
> diff(F(x),x);
```

$$\frac{\partial}{\partial x} Fx$$

```
> int(F(x),x=a..b);
```

$$\int_a^b Fx\, dx$$

Для отмены переназначения, например `BJ`, используется та же функция `alias` с повтором переназначения:

```
> alias(BJ=BJ);
```

$$Fx$$

```
> BJ(0,1.);
```

$$\mathrm{BJ}(0, 1.)$$

Обратите внимание на то, что `BJ` исчезло из списка переназначений и функция `BJ(0,1.)` уже не вычисляется, поскольку ее больше нет.

Модули придают языку программирования Maple некоторые свойства языков объектно-ориентированного программирования. Они служат для реализации абстрактного типа данных на основе инкапсуляции – объединения данных и процедур их обработки. Модули задаются ключевым словом `module` с пустыми скобками () и завершаются словами `end module` или просто `end`:

```
name := module()
 export eseq;  local lseq;   global gseq;
 option optseq; description desc;
 Тело модуля
end module (или просто end)
```

Хотя структура модуля во многом напоминает структуру процедуры, включая объявление локальных и глобальных переменных, параметров и описаний, между ними есть существенная разница:

- модуль не имеет списка входных параметров;
- в модуле могут размещаться данные;
- модули могут использоваться для создания пакетов процедур, доступ к которым обеспечивается командой `with`;
- модули имеют свойства в виде локальных переменных и методы в виде процедур интерфейса модулей;
- реализация абстрактных типов данных с помощью модулей скрыта от пользователя;
- модули могут содержать оператор `export eseq`, объявляющий экспортируемые переменные модуля;
- для доступа к экспортируемым переменным модуля может использоваться специальный оператор «`:-`» (двоеточие и минус);
- модули и процедуры могут вкладываться друг в друга без ограничения уровня вложенности;
- модули могут иметь специальные конструкторы объектов.

Следующий пример демонстрирует создание модуля pt, в котором заданы две операции (сложения plus и умножения times) и показан доступ к ним:

```
> pt:= module()
    export plus, times;
    plus := (a,b) -> a + b;
    times := (a,b) -> a * b;
end module;
```

pt := **module() export** *plus*, *times*; **end module**

```
> pt:-plus(3,5);
```

8

```
> pt:-times(3,7);
```

21

Детальную информацию о модулях и конструкторах объектов можно найти в справках по ним. Некоторые пакеты уже в Maple 9.5 реализованы не в виде процедур, а в виде модулей (например, в виде модуля сделан пакет LinearAlgebra).

Макрос – это макрокоманда, короткая запись длинных определений. По сравнению с переназначениями макросы более гибки и могут использоваться для сокращения операций загрузки новых определений из библиотеки и пакетов. Макросы создаются с помощью функции `macro`:

`macro(e1, e2, …, en)`,

где `e1`, `e2`, ..., `en` – нуль или более равенств.

В следующем примере функция `numbperm` с помощью макроса заменена на `np`:

```
> numbperm([1,2,3,4]);
```

$$24$$

```
> macro(np=numbperm(V));
```

$$np$$

```
> V:=[1,2,3,4];
```

$$V := [1, 2, 3, 4]$$

```
> np(V);
```

$$24$$

Макросы могут быть использованы для конструирования выражений из их макроопределений. Maple имеет команду `system(string)`, с помощью которой можно исполнить любую команду MS-DOS, записанную в виде строки `string`. Например, для форматирования гибкого диска из среды Maple 9 можно использовать стандартную команду MS-DOS:

```
> system(`format a:`);
```

На экране появится окно MS-DOS с начальным диалогом форматирования диска `A`. Это окно показано на рис. 14.5.

При работе в операционной системе Windows эта возможность практически бесполезна, поскольку форматирование диска с большими удобствами можно вы-

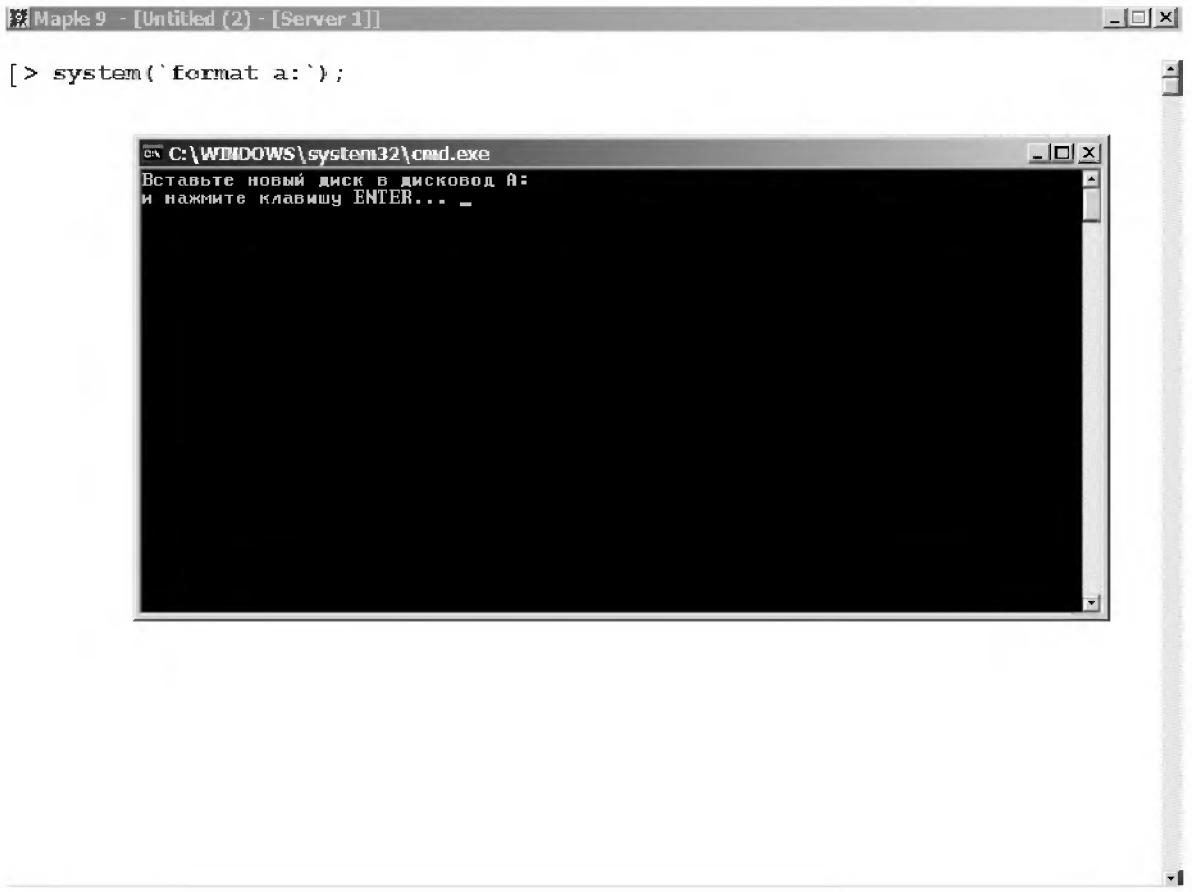


Рис. 14.5. Результат выполнения команды форматирования гибкого диска

полнить средствами Windows. В связи с этим в новых версиях Maple эта команда может не поддерживаться.

В заключение отметим, что Maple имеет средства для вызова внешних откомпилированных процедур, написанных на языке C.

14.3. Язык программирования систем Mathematica

14.3.1. Общая характеристика языка программирования Mathematica

Язык программирования системы Mathematica, так же как и Maple-язык, является типичным интерпретатором и не предназначен для создания исполняемых файлов. Впрочем, для отдельных выражений этот язык может осуществлять компиляцию с помощью функции **Compile**, что увеличивает скорость вычислений.

Внутреннее представление всех вычислений в языке программирования системы Mathematica базируется на применении полных форм выражений, представленных функциями. И вообще, функциям в системе Mathematica принадлежит решающая роль в организации вычислений любого вида. Таким образом, Mathematica фактически изначально реализует *функциональный* метод программирования – один из самых эффективных и надежных. А обилие логических операторов и функций позволяет полноценно реализовать и *логический* метод программирования. Множество операций преобразования выражений и функций позволяет осуществлять программирование на *основе правил преобразования*.

Возможно также создание полностью самостоятельных блоков – поименованных процедур и функций с локальными переменными, реализующее *структурное* и *модульное* программирование. Столь же естественно язык системы реализует ставшее модным в последнее время *объектно-ориентированное программирование*, основанное на следующих свойствах:

- *инкапсуляция* – объединение в одном объекте как данных, так и методов их обработки;
- *наследование* – означает, что каждый объект, производный от других объектов, наследует их свойства;
- *полиформизм* – свойство, позволяющее передать ряду объектов сообщение, которое будет обрабатываться каждым объектом в соответствии с его индивидуальными особенностями.

Приведенный ниже пример объектно-ориентированного программирования дает три определения, ассоциированные с объектом h:

```
h/:h[x_]+h[y_]:=hplus[x,y]
h/:p[h[x_],x]:=hp[x]
h/:f_[h[x_]]:=fh[f,x]
```

Возможно также *рекурсивное* программирование, при котором очередной шаг вычислений базируется на данных, полученных на предыдущих шагах. Наглядным примером этому может служить вычисление факториала рекурсивным методом.

Поскольку алфавит языка программирования системы и набор операторов и функций уже были рассмотрены ранее, в этой главе нам остается рассмотреть лишь специфические средства языка и его управляющие структуры.

14.3.2. Образцы и их применение

Образцы в системе Mathematica служат для задания выражений *различных классов* и придания переменным особых свойств, необходимых для создания специальных программных конструкций, таких как функции пользователя и процедуры. Признаком образца являются знаки подчеркивания «_» (от одного до трех). Они обычно выглядят слитно, так что надо внимательно следить за общей длиной символов образцов. Наиболее распространенное применение образцов – указание на локальный характер переменных при задании функций пользователя. Например, функция

```
fsc[x_,y_]:= x * Sin[y] + y * Cos[x] + z
```

в списке параметров содержит два образца x_ и y_. В правой части этого выражения переменные x и y, связанные с образцами x_ и y_, становятся локальными переменными, тогда как переменная z будет глобальной переменной. Обратите особое внимание на то, что символы образцов используются только в списках параметров – в правой части выражений они уже не используются.

Образцами можно задавать некоторые общие свойства функций. Например,

```
f[x_,x_] := p[x]
```

означает, что функция f двух идентичных аргументов становится тождественной функции p[x]. Следовательно:

```
f[a,a] + f[a,b]
```

даст выход в виде:

```
f[a,b] + p[a],
```

а

```
f[a^2 – 1, a^2 – 1]
```

даст выход

```
p[-1 + a^2]
```

Вообще говоря, вычисление таких функций, как факториал, в Mathematica можно запрограммировать более чем десятком способов. Вот несколько самых интересных реализаций этой функции:

```
f[n_]:=n!
f[n_]:=Gamma[n-1]
f[n_]:=n*f[n-1];f[0]=1;f[1]=1;
```

```
f[n_]:=Product[i,{i,n}]
f[n_]:=Module[{t=1}, Do[t=t*i,{i,n}];t]
f[n_]:=Module[{t=1},For[i=1,i<=n,i++,t*=i];t]
f[n_]:=Fold[Times, 1,Range[n]]
```

Все их можно проверить по примеру:

```
{f[0],f[1],f[5],f[10]}
 {1,1,120,3628800}
```

С образцом можно указывать его *тип данных*:

- **x_Integer** – образец целочисленный;
- **x_Real** – образец с действительным значением;
- **x_Complex** – образец с комплексным значением;
- **x_h** – образец с заголовком h (от слова head – голова).

Задание с помощью образцов типов данных делает программы более строгими и наглядными и позволяет избежать ошибок, связанных с несоответствием типов данных.

Следующие типы образцов используются в система Mathematica:

Обозначение	Назначение образца
_	Любое выражение
x_	Любое выражение, представленное именем x
x:pattern	Образец, представленный именем x
pattern ? test	Возвращает True, когда test применен к значению образца
_h	Любое выражение с заголовком h
x_h	Любое выражение с заголовком h, представленное именем h
__	Любая последовательность с одним и более выражениями
___	Любая последовательность с нулем или более выражений
x__ или x___	Последовательности выражений, представленные именем x
__h или h___	Последовательности выражений, каждое с заголовком h
x__h или x___h	Последовательности выражений с заголовком h, представленные именем x
x_:v	Выражение с определенным значением v
x_h:v	Выражение с заголовком h и определенным значением v
x_.	Выражение с глобально заданным определенным значением v
Optional[x_h]	Выражение с заголовком h и с глобально заданным значением
pattern..	Образец, повторяемый один или более раз
pattern...	Образец, повторяемый нуль или более раз

Еще раз отметим, что символ _ в образцах может иметь одинарную, двойную или тройную длину. Надо следить за правильностью его применения, поскольку эти применения различаются по смыслу. Образцы широко применяются при задании функций пользователя и в пакетах расширения системы.

14.3.3. Функциональное программирование в среде Mathematica

Суть функционального программирования заключается в использовании в ходе решения задач только функций пользователя. При этом возможно неоднократное вложение функций друг в друга и последовательное применение функций различного вида. В ряде случаев, особенно в процессе символьных преобразований, происходит взаимная рекурсия множества функций, сопровождаемая почти неограниченным углублением рекурсии и нарастанием сложности обрабатываемых системой выражений.

В Mathematica функции пользователя задаются именем и списком параметров. Переменные в списке параметров задаются как образцы, например:

```
powerxn[x_,n_]:=x^n
```

Вычисление по заданной функции пользователя возможно как в численном, так и в символьном виде, например:

```
powerxn[2,3]
 8
powerxn[a,b]
 a^b
```

Заметим, что для уничтожения определения заданной функции можно использовать команду – функцию

```
Clear[Name_function],
```

где Name_function – имя функции.

Можно также задать функцию пользователя, содержащую несколько выражений, заключив их в круглые скобки:

```
f[x_]:=(t=(1+x)^2;t=Expand[t])
```

Переменные списка параметров, после имени которых стоит знак «_», являются локальными в теле функции или процедуры с параметрами. На их место подставляется фактическое значение соответствующего параметра, например:

```
f[a+b]
 1 + 2a + a^2 + 2b + 2ab + b^2
t
 1 + 2a + a^2 + 2b + 2ab + b^2
```

Обратите внимание на то, что переменная t в функции f является глобальной. Это поясняет результат последней операции. Применение глобальных переменных в теле функции вполне возможно, но создает так называемый побочный эффект – в данном случае меняет значение глобальной переменной t. Для устране-

ния побочных эффектов надо использовать образцы и другие специальные способы задания функций, описанные ниже.

Параметрами функций могут быть списки, при условии допустимости их комбинации. Например, допустимо задать x списком, а n – переменной или числом:

```
powerxn[{1,2,3,4,5},z]
 {1, 2^z, 3^z, 4^z, 5^z}
powerxn[{1,2,3,4,5},2]
 {1,4,9,16,25}
```

После своего задания функции пользователя могут использоваться по тем же правилам, что и встроенные функции.

14.3.4. Чистые и анонимные функции в Mathematica

Иногда может потребоваться задание некоторой функции только в момент ее создания. Эта функция представляется лишь выражением без имени – отсюда и ее название: *чистая функция*. Для создания такого объекта служит встроенная функция **Function**, используемая в виде:

- **Function[body]** – создает чистую функцию с телом body;
- **Function[{x},body]** – создает чистую функцию параметра x с телом body;
- **Function[{x1,x2,...},body]** – создает чистую функцию ряда параметров x1, x2,... с телом body.

Для вычисления созданной таким образом функции после нее задается список параметров в квадратных скобках. Например, для взятой в качестве примера функции ее можно задать и использовать следующим образом:

```
Function[{x,n},x^n]
 Function[{x, n}, x^n]
%[2,3]
 8
```

Чистую функцию можно легко превратить в обычную функцию пользователя, что показывает следующий пример:

```
fun=Function[{x,n},x^n]
  Function[{x, n}, x^n]
fun[2,3]
  8
```

Предельно компактную форму задания функций имеют так называемые *анонимные функции*. Они не имеют ни названия, ни обычного определения и задаются только выражениями специального вида. В этом выражении вместо переменных используют обозначения # (для одной переменной), #1, #2,... для ряда переменных. Завершается тело функции символом &. Если надо вычислить функцию, то после ее записи в квадратных скобках указывается список фактических параметров.

Для нашего примера такая функция выглядит так:

```
#1^#2&[2,3]
 8
#1^#2&[y,z]
 y^z
```

С помощью анонимных функций нетрудно создать обычные функции пользователя:

```
f[x_,y_]=#1^#2&[x,y]
 x^y
f[2,3]
  8
```

14.3.5. Специальные средства работы с функциями

При функциональном программировании часто используется суперпозиция функций. Для ее реализации используются функции:

- **Nest[expr,x,n]** – применяет выражение (функцию) к заданному аргументу x n раз;
- **NestList[f,x,n]** – возвращает список приложений функции f к заданному аргументу x n+1 раз;
- **Fold[f,x,list]** – дает следующий элемент в FoldList[f,x,list];
- **FoldList[f,x,{a,b,...}]** – дает {x, f[x, a], f[f[x, a], b], ... };
- **ComposeList[{f_1,f_2,...},x]** – генерирует список в форме {x, a[x], a[a[x]], ...}.

Примеры, иллюстрирующие действие этих функций, представлены ниже:

```
Clear[f]
Nest[f,x,5]
 f[f[f[f[f[x]]]]]
Nest[Exp[x],x,5]
 e^x[e^x[e^x[e^x[e^x[x]]]]]
NestList[f,x,3]
 {x, 1 + 2x + x^2, 4 + 8x + 8x^2 + 4x^3 + x^4,
  25 + 80x + 144x^2 + 168x^3 + 138x^4 + 80x^5 + 32x^6 + 8x^7 + x^8}
Fold[f,x,{1,2,3}]
 x^6
FoldList[f,x,{1,2,3}]
 {x, x, x^2, x^6}
ComposeList[{Exp,Ln,Sin},x]
{x, e^x, Ln[e^x], Sin[Ln[e^x]]}
```

В функциональном программировании вместо циклов, описанных далее, может использоваться следующая функция:

- **FixedPoint[f,expr]** – вычисляет expr и прикладывает к нему f, пока результат не повторяется;
- **FixedPoint[f,expr,SameTest->comp]** – вычисляет expr и прикладывает к нему f, пока два последующих результата не дадут True в тесте.

Пример применения функции FixedPoint:

```
FixedPoint[Function[t, Print[t]; Floor[t/2]], 27]
 27
 13
 6
 3
 1
 0
 0
```

Последний результат – 0 – выводится в отдельной (нумерованной) ячейке вывода и означает завершение процесса итераций – деления t на 2.

Следующий пример показывает, как можно создать цепную дробь с помощью функции **Nest**:

```
Nest[ Function[t, 1/(1+t)], y, 3 ]
```

$$\frac{1}{1+\frac{1}{1+\frac{1}{1+y}}}$$

Еще одна функция такого рода – это **Catch**:

- **Catch[expr]** – вычисляет expr, пока не встретится Throw[value], затем возвращает value;
- **Catch[expr, form]** – вычисляет expr, пока не встретится Throw[value,tag], затем возвращает value;
- **Catch[expr, form, f]** – возвращает f[value, tag] вместо value.

Ниже представлены некоторые конструкции циклов с оператором **Catch**:

```
Catch[ Throw[ x, y ], y, fun ]
  fun[x,y]
Catch[ NestList[1/(# + 1)&, -3, 5] ]
```

$$\left\{-3, -\frac{1}{2}, 2, \frac{1}{3}, \frac{3}{4}, \frac{4}{7}\right\}$$

```
Catch[ NestList[1/(# + 1)&, -3., 5] ]
 {-3.,-0.5,2.,0.333333,0.75,0.571429}
```

14.3.6. Реализация рекурсивных и рекуррентных алгоритмов

Рассмотрим несколько простых примеров, выявляющих суть функционального программирования. Вначале пусть это будет пример, в котором задана функция scn[x,n], вычисляющая сумму синуса в степени n и косинуса в степени n:

```
scn[x_,n_]:=Sin[x]^n+Cos[x]^n
scn[1,2]
 Cos[1]^2 + Sin[1]^2
scn[x,2]
 Cos[x]^2 + Sin[x]^2
```

```
scn[x,n]
 Cos[x]^n + Sin[x]^n
```

В этом простейшем примере результат вычислений есть возвращаемое функцией scn символьное значение.

Важное место в решении многих математических задач занимают реализации рекурсивных и рекуррентных алгоритмов. Рассмотрим, как это делается с помощью описанных выше функций.

В следующих примерах показано вычисление квадратного корня из выражения $f(x)$ при начальном значении $x_0 = a$ по следующим формулам:

$$x_0 = a \qquad \text{и} \qquad x_n = x_{n-1} - f(x_{n-1})/f'(x_{n-1}).$$

Эту функцию можно записать следующим образом:

```
newtoniter[f_,x0_,n_]:=Nest[(#-f[#]/f'[#])&,N[x0],n]
```

Тогда вычисления корня из выражения e^x-2 с начальным приближением x0 = 0.5 при числе итераций n можно организовать с помощью функций **Nest** и **NestList**:

```
newtoniter[Function[{x},Exp[x]-2.0],0.5,5]
  0.693147
newtoniter[Function[{x},Exp[x]-2.0],0.5,#]&/@Range[5]
  {0.713061,0.693344,0.693147,0.693147,0.693147}
newtoniter1[f_,x0_,n_]:=NestList[(#-f[#]/f'[#])&,N[x0],n]
newtoniter1[Function[{x},Exp[x]-2.0],0.5,5]
  {0.5,0.713061,0.693344,0.693147,0.693147,0.693147}
```

В первом случае возвращается последний результат, а в других – все промежуточные. Функция **FixedPoint** позволяет осуществлять итерацию до тех пор, пока результат не перестанет изменяться (с машинной точностью). Это иллюстрирует следующий пример:

```
newtonfp[f_, init_, options___] := FixedPoint[#1 - f[#1]/f'[#1] &, init, options];
newtonfp[Function[{x},Exp[x]-2.0],0.5,5]
   0.693147
```

Более сложные примеры функционального программирования мы рассмотрим позже при описании создания пакетов расширения систем Mathematica.

14.3.7. Примеры программирования графических задач

Графические задачи составляют значительную часть задач, решаемых системой Mathematica. С точки зрения программирования эти задачи не имеют особой специфики. Большая часть из них сводится к заданию функции, описывающей график, и применению одной из многочисленных графических функций системы с соответствующими опциями и директивами.

На рис. 14.6 показано задание функции GrayCode (Большие коды) и ее графическое представление, полученное с помощью встроенной функции ListPlot.

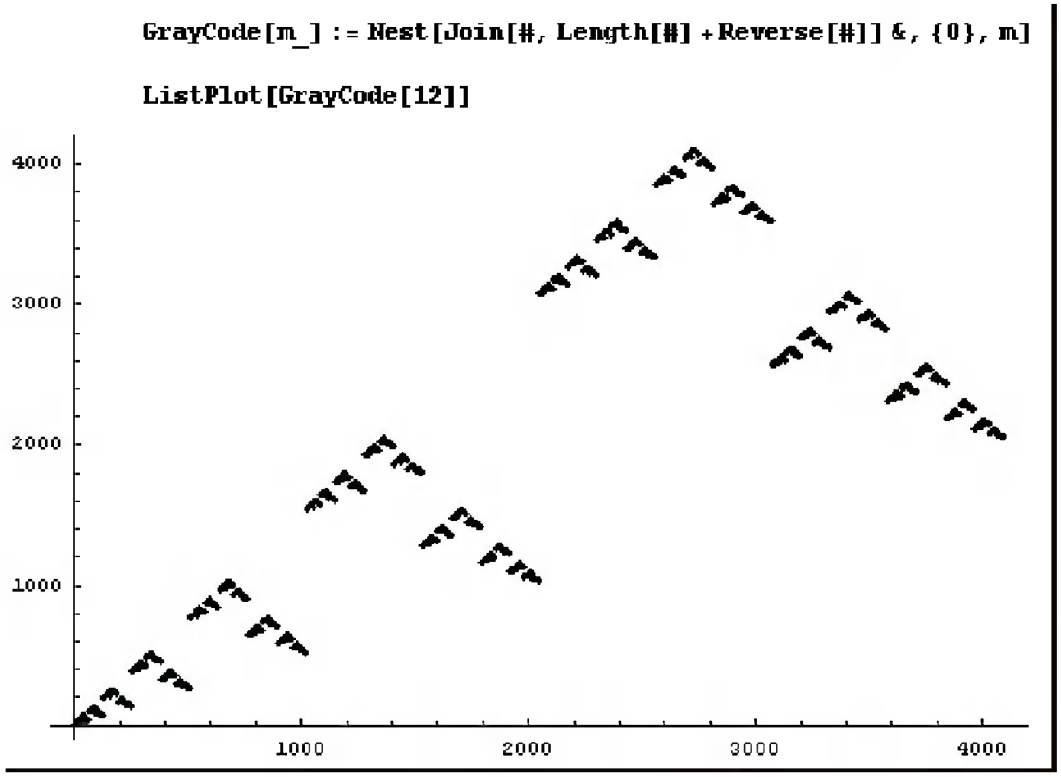


Рис. 14.6. Задание функции GrayCode и ее графическое представление на плоскости

В качестве следующего примера рассмотрим задачу на построение сложного графика Mandelbrot. Пример задания соответствующей функции MandelbrotFunction и применения графической функции DensityPlot для наглядного визуального представления функции MandelbrotFunction на комплексной плоскости представлен на рис. 14.7.

Еще более сложную и любопытную задачу демонстрирует рис. 14.8. Здесь задана функция JuliaFunction, которая представляет одну из моделей деления клеток. На этом же рисунке дано построение множества графиков, дающих прекрасное визуальное представление данной функции.

Разумеется, приведенные примеры далеко не исчерпывают всего многообразия графических возможностей языка программирования систем Mathematica.

14.3.8. Процедуры и блоки процедурного программирования

В основе процедурного программирования лежит понятие процедуры и типовых средств управления – циклов, условных и безусловных выражений и т. д. Процедурный подход – самый распространенный в программировании, и разработчики Mathematica были вынуждены обеспечить его полную поддержку. Хотя программирование систем Mathematica и в этом случае остается функциональным, поскольку элементы процедурного программирования заданы в виде функций.

Процедуры являются полностью самостоятельными программными модулями, задаются своим именем и отождествляются с выполнением некоторой после-

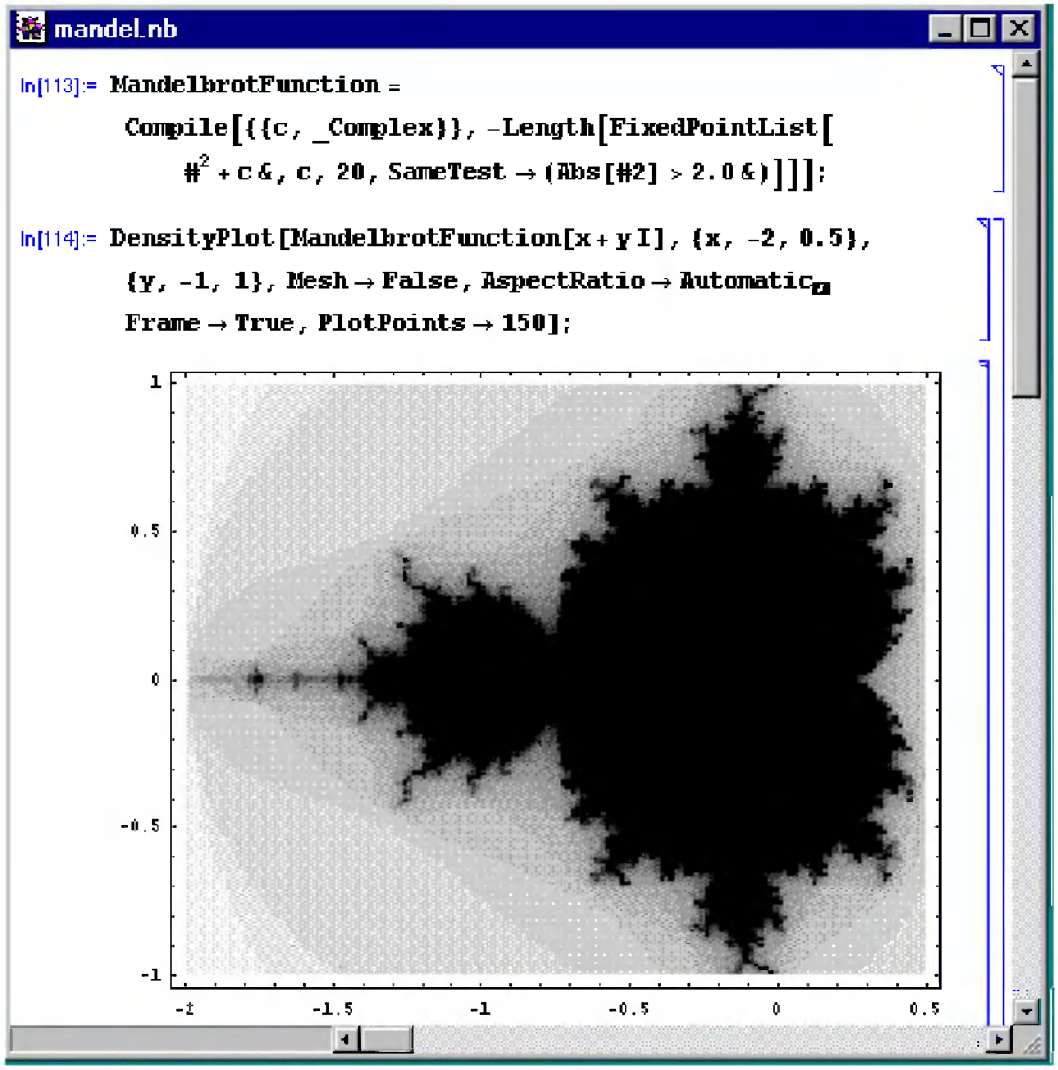


Рис. 14.7. Пример задания функции MandelbrotFunction и построения ее контурного графика

довательности операций. Они могут быть заданы в одной строке с использованием в качестве разделителя символа «;» (точка с запятой). Вот пример задания однострочной процедуры, отождествленной с именем r:

```
r=(1+x)^2;r=Expand[r];r-1
```

$$2x + x^2$$

Обратите внимание, что в теле процедуры символ r используется как вспомогательная переменная. Эта процедура возвращает символьное выражение **Expand[(1+x)^2] – 1**.

Для задания процедуры со списком локальных переменных {a, b,...} и телом proc может использоваться функция

```
Module[{a, b, ...},proc]
```

С применением этой функции мы столкнемся ниже.

Для создания полноценных процедур и функций, которые могут располагаться в любом числе строк, может использоваться базовая структура – блок:

- **Block[{x, y, ...}, procedure]** – задание процедуры с декларацией списка локальных переменных x, y, ...;

```
Show[
  GraphicsArray[
   Table[DensityPlot[JuliaFunction[x+I y, a+b I], {x, -1.1, 1.1}, {y, -1.1, 1.1},
     Mesh→False, Frame→False, Axes→False, PlotPoints→35, AspectRatio→1,
     DisplayFunction→Identity], {b, -1.0, 1.0, 0.333}, {a, -2.0, 1.0, 0.333}]]];
```

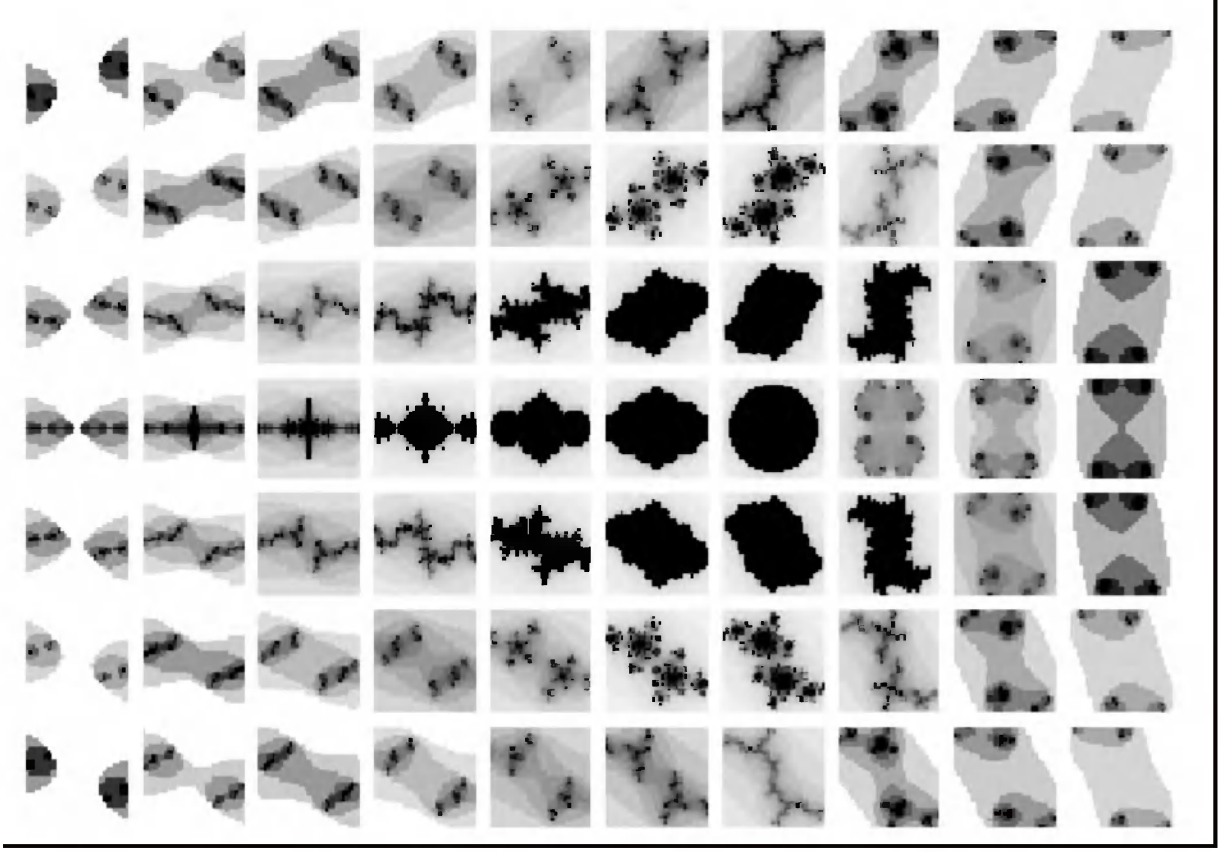


Рис. 14.8. Задание функции JuliaFunction и ее графическое представление

- **Block[{x = x0, y=y0, ...}, procedure]** – задание процедуры с декларацией списка переменных x, y, ... с заданными начальными значениями.

Пример использования базовой структуры:

```
g[x_] := Block[{u}, u = (1+x)^2; u=Expand[u] ]
g[a+b]
 1 + 2a + a^2 + 2b + 2ab + b^2
u
 u
u=123456;g[2]
 9
u
  123456
```

Обратите внимание, что последние действия показывают на то, что переменная u, введенная в тело базовой структуры, является действительно локальной переменной и присвоение ей символьного выражения (1+x)^2 в теле блока игнорируется вне блока. Если переменная u до применения в функции была не определена, то она так и остается неопределенной. А если она имела до этого некоторое значение (например, 123456 в нашем случае), то и по выходе из процедуры она будет иметь это значение.

14.3.9. Организация циклов

Циклы с заданным числом повторений в Mathematica организуются с помощью функции Do:

- **Do[expr, {imax}]** – выполняет imax раз вычисление expr.
- **Do[expr, {i, imax}]** – вычисляет expr с переменной i, последовательно принимающей значения от 1 до imax (с шагом 1).
- **Do[expr, {i, imin, imax}]** – вычисляет expr с переменной i, последовательно принимающей значения от imin до imax с шагом 1.
- **Do[expr, {i, imin, imax, di}]** – вычисляет expr с переменной i, последовательно принимающей значения от 1 до imax с шагом di.
- **Do[expr, {i, imin, imax}, {j, jmin, jmax}, ...]** – вычисляет expr, организуя ряд вложенных циклов с управляющими переменными j, i и т. д.

Примеры организации цикла **Do** и его исполнения представлены ниже:

```
Do[Print["hello"],{5}]
 hello
 hello
 hello
 hello
 hello
Do[Print[i],{i,3}]
 1
 2
 3
Do[Print[i],{i,5,8}]
 5
 6
 7
 8
Do[Print[i],{i,0,1,0.25}]
 0
 0.25
 0.5
 0.75
 1.
```

Нетрудно убедиться в том, что переменная i в теле цикла – итератор является локальной и по выходе из цикла ее значение остается тем же, что и до входа:

```
i=2
 2
Do[Print[i],i,1,5]
 1
 2
 3
 4
 5
i
 2
```

Вся программа с циклом является содержанием одной ячейки, и ее листинг охвачен квадратной скобкой. Для иллюстрации вывода здесь использована команда **Print** в теле цикла. Нетрудно заметить, что управляющая переменная цикла может иметь как целочисленные, так и вещественные значения. Возможность организации цикла в цикле иллюстрируется следующим примером:

```
Do[Do[Print[i,"   ",j,"   ",i+j],{j,1,3}],{i,1,3}];
 1      1      2
 1      2      3
 1      3      4
 2      1      3
 2      2      4
 2      3      5
 3      1      4
 3      2      5
 3      3      6
```

Здесь используются два цикла с управляющими переменными i и j. Командой **Print** выводятся значения переменных i и j, а также их суммы i+j.

Следующий пример показывает применение цикла Do для задания функции, вычисляющей n-ое число Фибоначчи:

```
fibonacci[(n_Integer)?Positive] :=
 Module[{fn1 = 1, fn2 = 0},
  Do[{fn1, fn2} = {fn1 + fn2, fn1}, {n - 1}]; fn1]
fibonacci[10]
 55
fibonacci[100]
 354224848179261915075
fibonacci[-10]
 fibonacci[-10]
```

Обратите внимание на применение в этом примере функции **Module**. Она создает программный модуль с локальными переменными (в нашем случае fn1 и fn2), в котором организовано рекуррентное вычисление чисел Фибоначчи.

Наконец, последний пример показывает применение цикла Do для создания цепной дроби:

```
x = y; Do[x = 1/(1 + k x), k, 2, 8, 2]; x
```

$$\cfrac{1}{1+\cfrac{8}{1+\cfrac{6}{1+\cfrac{4}{1+2y}}}}$$

Другой вид цикла For реализуется функцией:

```
For[start, test, incr, body]
```

В ней внутренняя управляющая переменная вначале приобретает значение start, затем циклически меняется от этого значения до значения body с шагом изменения incr – и так до тех пор, пока условие test не перестанет давать логическое значение True. Когда это случится, то есть test даст False, цикл заканчивается.

Следующий пример показывает задание простой программы с циклом **For** и результат ее выполнения:

```
Print["i   x"]
For [x=0;i=0,i<4,i++
[x+=5*i,Print[i,"   ",x]]]
 i     x
 1     5
 2     15
 3     30
 4     50
 Return[x]
 Return[50]
```

Программа, приведенная выше, позволяет наблюдать за изменением значений управляющей переменной цикла i и переменной x, получающей за каждый цикл приращение, равное 5*i. В конце документа показан пример на использование функции возврата значений **Return[x]**. В цикле For не предусмотрено задание локальных переменных, так что надо следить за назначением переменных – при использовании глобальных переменных неизбежны побочные эффекты.

Итак, функция **For** позволяет создавать циклы, которые завершаются при выполнении (эволюции) какого-либо условия. Такие циклы можно организовать и с помощью функции **While**:

- **While[test, expr]** – выполняет expr до тех пор, пока test дает логическое значение True.

Ниже дан практический пример организации и использования цикла **While**.

```
i:=1;x:=1;Print["i   x"];
While[i<5,i+=1;x+=2*i;Print[i,"   ",N[x]]]
 i     x
 2     5.
 3     11.
 4     19.
 5     29.
Return[x]
 Return[50]
```

Циклы типа **While**, в принципе, могут заменить другие, рассмотренные выше типы циклов. Однако это усложняет запись и понимание программ. Аппарат локальных переменных в этом типе циклов не используется.

В указанных типах циклов и в иных управляющих структурах можно использовать следующие директивы-функции:

- **Abort[ ]** – вызывает прекращение вычислений с сообщением $Aborted;
- **Break[ ]** – выполняет выход из тела цикла или уровня вложенности программы, содержащего данный оператор (циклы тира Do, For и While или тело оператора – переключателя Shith). Оператор возвращает Null – значение (без генерации секции выхода).
- **Continue[ ]** – задает переход на следующий шаг текущего цикла Do, For или While.

- **Interrupt[ ]** – прерывает вычисления с возможностью возобновления их.
- **Return[]** – прерывает выполнение с возвратом Null.
- **Return[expr]** – прерывает выполнение с выводом значения выражения expr.
- **Throw[value]** – задает прекращение выполнения цикла Catch, если в ходе эволюции expr встречается значение value (см. примеры выше).

На рис. 14.9. представлено применение директив **Abort[]** и **Interrupt[]** в середине набора команд. Нетрудно заметить, что директива **Abort[]** просто прерывает выполнение цепочки команд и выводит сообщение **$Aborted**. А вот директива **Interrupt[]** выводит диалоговое окно, с помощью которого можно либо прервать вычисления, либо продолжить их.

Если продолжить вычисления (нажав кнопку **Continue Evaluation**), то вывод выражений командами **Print** будет продолжен, что видно из рис. 14.10.

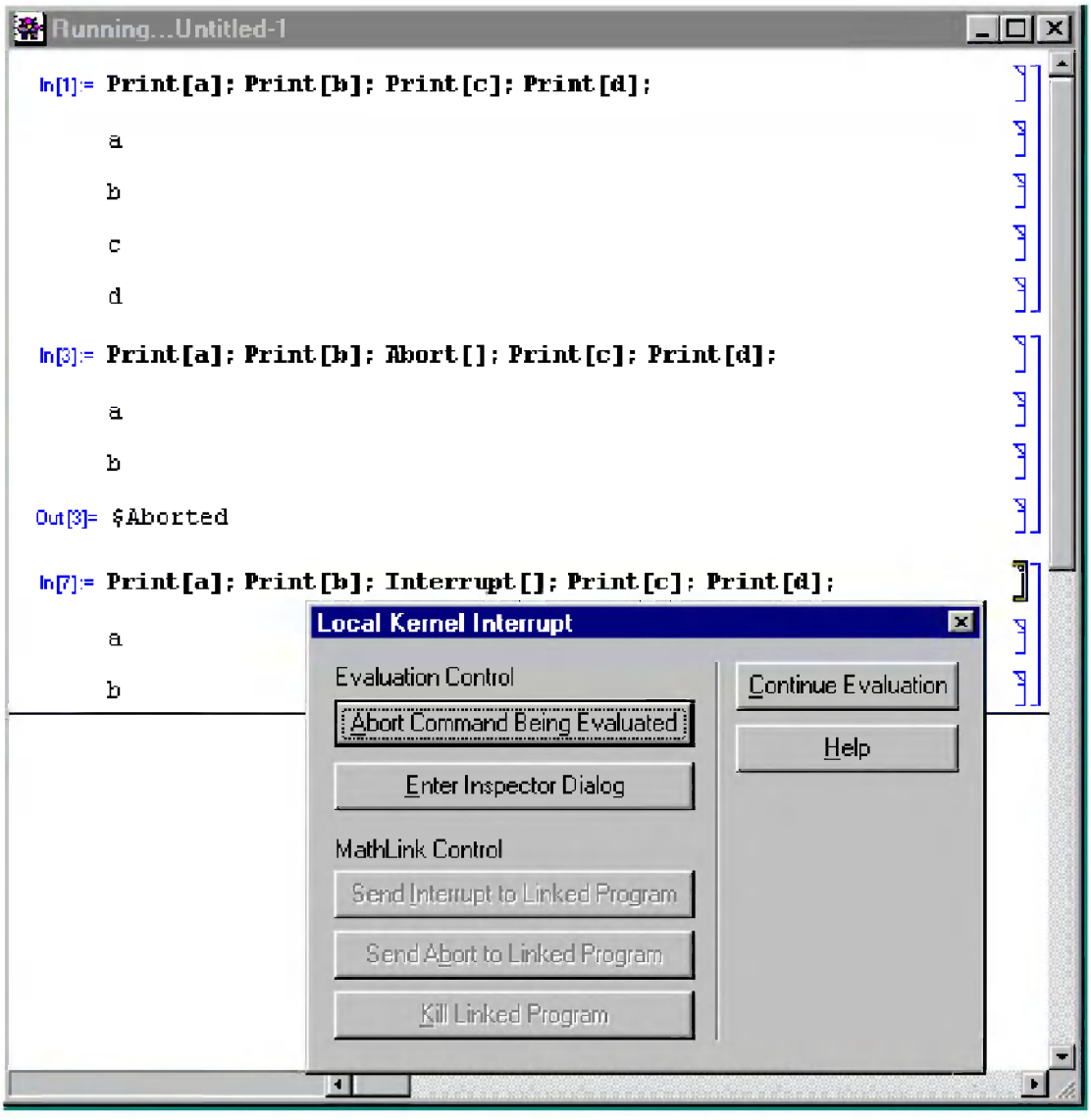


Рис. 14.9. Действие директив Abort[] и Interrupt[]

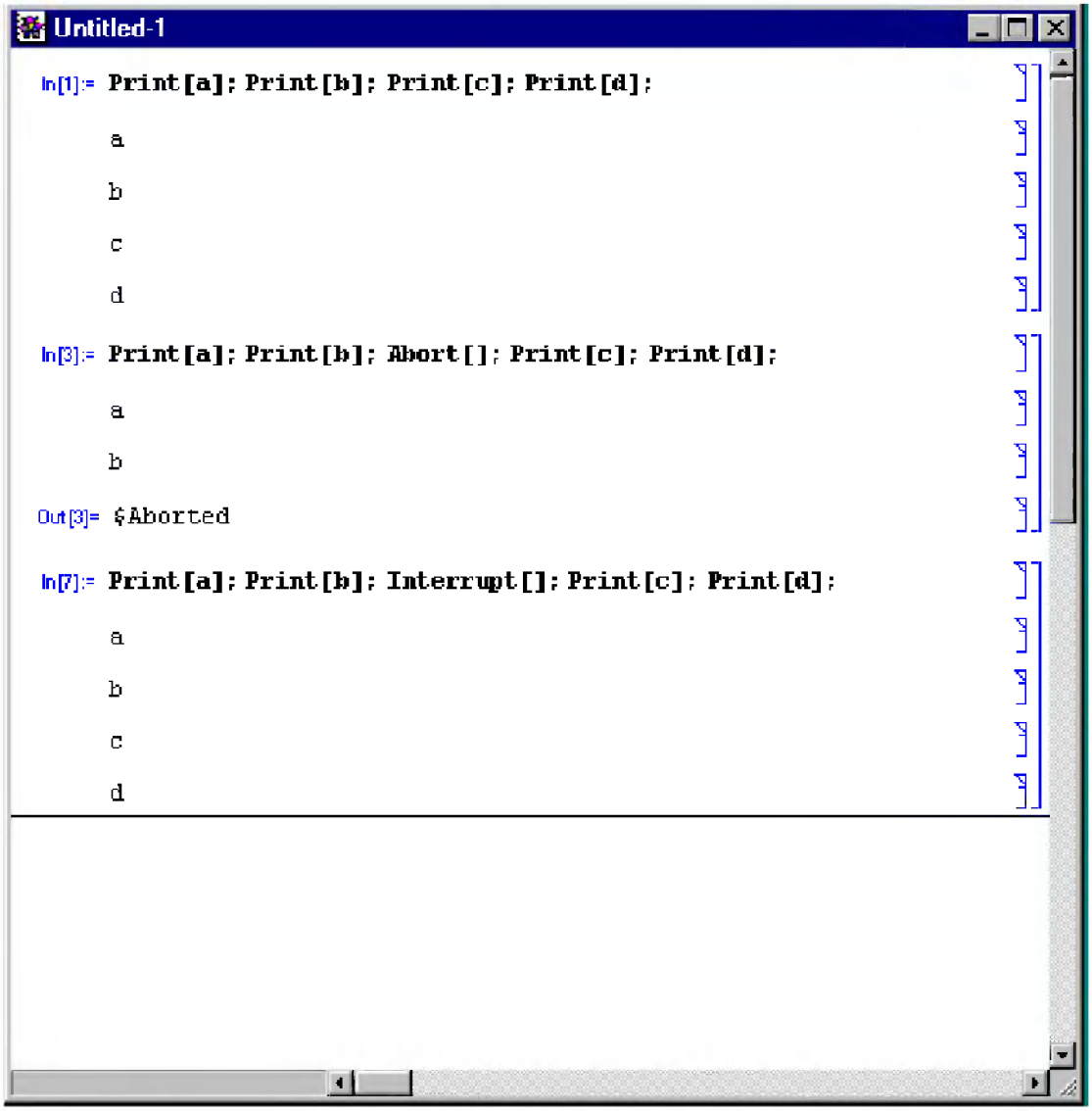


Рис. 14.10. Продолжение вычислений после команды ***Interrupt[ ]***

14.3.10. Условные выражения и безусловные переходы

Для подготовки полноценных программ, помимо средств организации циклов, необходимы и средства для создания разветвляющихся программ произвольной структуры. Обычно они реализуются с помощью условных выражений, позволяющих в зависимости от выполнения или невыполнения некоторого условия (condition) выполнять те или иные фрагменты программ.

Как у большинства языков программирования, условные выражения задаются с помощью оператора или функции **If**. Система Mathematica имеет функцию **If**, формы которой представлены ниже:

- **If[condition, t, f]** – возвращает t, если результатом вычисления condition будет True, и даст f, если результат False.
- **If[condition, t, f, u]** – даст u, если в результате вычисления condition не будет получено ни True, ни False.

Следующий пример поясняет задание программной процедуры с циклом **Do**, выход из которой задается с помощью функции **If** и директивы прерывания **Aborted[ ]**:

```
x:=1;Print["i   x"];
Do[{If [i==5,Abort[],None],
i+=1;x+=2*i;Print[i,"   ",N[x]]},
{i,1,100}]
 i     x
 2     5.
 3     11.
 4     19.
 5     29.
 $Aborted
Return[x]
 Return[29]
```

Тот же пример, но с применением директивы выхода из цикла **Break[]** в функции **If** показан ниже:

```
x:=1;Print["i   x"];
Do[{If [i==5,Break[],None],
i+=1;x+=2*i;Print[i,"   ",N[x]]},
{i,1,100}]
 i     x
 2     5.
 3     11.
 4     19.
 5     29.
Return[x]
 Return[29]
```

В данном случае никаких специальных сообщений о выходе из цикла не дается.

Функция **If** обеспечивает ветвление максимум по двум ветвям программы. Для ветвления по многим направлениям можно использовать древовидные структуры программ с множеством функций **If**. Однако это усложняет программы.

14.3.11. Функции-переключатели

Для организации ветвления по многим направлениям в современных языках программирования используются операторы-переключатели. В системе Mathematica множественное ветвление организовано с помощью функций Which и Switch:

- **Which[test1, value1, test2, value2, ...]** – вычисляет в порядке следования каждый из **testi**, сразу возвращая именно ту величину из **valuei**, которая относится к первому **testi**, давшему True.
- **Switch[expr, form1, value1, form2, value2, ...]** – вычисляет **expr**, затем сравнивает его последовательно с каждым **formi**, вычисляя и возвращая то **valuei**, которое соответствует первому совпадению.

Приведем примеры работы функции **Which**:

```
Which[1==2,1,2==2,2,3==3,3]
 2
Which[1==2,x,2==2,y,3==3,z]
 y
```

Следующие примеры иллюстрируют работу функции **Switch**:

```
Switch[2,1,a,2,b,3,c]
 b
Switch[3,1,a,2,b,3,c]
 c
Switch[8,1,a,2,b,3,c]
 Switch[8,
 1,a,
 2,b,
 3,c]
```

Обратите внимание на последний пример – при неверном задании первого параметра (выбора) просто повторяется запись функции.

Следующий пример показывает возможность выбора с применением вещественных значений параметра выбора и меток:

```
Switch[8.,1.5,a,2.5,b,8.,c]
 c
Switch[1.5,1.5,a,2.5,b,8.,c]
 a
Switch[8,1.5,a,2.5,b,8.,c]
 Switch[8.2,
1.5,a,
2.5,b,
8.,c]
```

Обратите опять-таки внимание на последний пример – параметр выбора здесь выбран как целочисленное число 8, тогда как метка выбора – вещественное число 8. Выбор при этом не происходит, поскольку целочисленное значение 8 не является тождественным вещественной восьмерке.

14.3.12. Безусловные переходы

В целом условные выражения в языке программирования системы Mathematica позволяют реализовать любой вид ветвления в программах. Однако иногда бывает полезно без лишних раздумий указать в программе явный переход к какой-либо ее части. Для этого используется оператор безусловного перехода **Goto[tag]**, который дает переход к тому месту программы, которое отмечено меткой **Label[tag]**. Возможны также формы **Goto[expr]** и **Label[expr]**, где expr – вычисляемое выражение.

Применение оператора **Goto** иллюстрирует следующий пример:

```
(q = 2; Label[start]; Print[q]; q += 2;
                        If[q < 7, Goto[start]])
```

```
2
4
6
```

Здесь с помощью оператора **Goto[start]** организован цикл с возвратом на метку Label[start], действующий до тех пор, пока значение q меньше 7. При этом q меняется от начального значения 2 с шагом 2, причем добавление 2 к текущему значению q осуществляется укороченным оператором сложения q+=2.

Интересной особенностью языка программирования Mathematica является возможность создания переходов по значению вычисляемого выражения. Например, Goto[2+3] дает переход к метке Label[5] или даже Label[1+4], что видно из следующего примера:

```
Goto[2+3];
Print["aaaaa"];
Label[1+4];
Print["bbbbb"]
  bbbbb
```

Переходы, задаваемые выражениями, и метки, меняющие свой идентификатор, редко встречаются в обычных языках программирования, хотя они обеспечивают новые обширные и довольно необычные возможности по созданию программ с различными ветвлениями.

Для языка программирования системы Mathematica, ориентированного на безупречное и строгое структурное программирование, введение оператора **Goto** может оцениваться как отступничество от основополагающих идей структурного программирования. Поэтому на применение этого оператора наложено табу в методах структурного программирования. Тем не менее этот оператор есть, а применять его или нет – дело пользователя.

14.3.13. Механизм контекстов

Мы уже не раз обращали внимание на то, что при создании документов нередки конфликты между переменными, назначаемыми пользователем, и переменными, входящими в программы ядра, между функциями пользователя и встроенными функциями, между их заголовками и т. д. Ситуация усложняется при использовании пакетов расширения, поскольку в них широко используются переменные и различные функции, причем нередко обозначенные так же, как и встроенные функции.

Особенно коварны побочные эффекты в конструкциях, содержащих вспомогательные переменные, например в итерационных циклах, функциях вычисления суммы и произведения и др. Они содержат переменные – итераторы i, j, k и т. д. Обычно избежать конфликтов можно с помощью механизма локализации итераторов. Вернемся к уже обсуждавшимся примерам. Возьмем пример с вычислением суммы:

```
i=2
  2
```

```
Sum[i,i,1,4]
 10
i
 2
```

Ясно, что сумма вычисляется с применением цикла с заданным числом повторений. В его конце итератор i получает значение 4. Но глобальная переменная с тем же именем имеет значение i=2, которое она получила до вычисления суммы с помощью функции Sum. В данном случае это достигнуто за счет того, что в теле функции переменная-итератор является локальной.

Нетрудно убедиться, что проблемы со статусом переменных возможны и в казалось бы изученных функциях суммирования и перемножения. На это явно указывает следующий пример:

```
func[x_]:=Sum[x^i,{i,4}]
{func[y],func[i]}
  {y + y^2 + y^3 + y^4, 288}
i
  1
```

Результат вычисления `func[y]` вполне понятен, тогда как вычисление func[i] носит явно обескураживающий характер. Причина его в том, что вместо символьного значения i в данном случае оказались использованы численные значения итератора i. А в этом случае функция **Sum** просто вычисляет численные значения. Говорят, что она работает по контексту!

А теперь возьмем пример с циклом **For**:

```
For [i=1,i<=4,i++,Print[i]]
 1
 2
 3
 4
i
 5
```

На этот раз переменная i изменила свое значение в конце цикла с 2 на 5. Это говорит о том, что пользователю-программисту надо очень внимательно относиться к статусу переменных во всех итерационных, да и других программах.

Разумеется, Mathematica содержит средства для устранения подобного смешения ролей переменных. Одно из них – применение конструкции **Module**:

```
i=2
 2
Module[{i},For[i=1,i<=4,i++,Print[i]]]
 1
 2
 3
 4
i
 2
```

На этот раз захвата итератором глобальной переменной i удалось избежать. Однако этот пример носит не более чем частный характер. Вообще говоря, если переменная-итератор задается в теле функции, то она будет локальной, а если она задается за пределами функций – то глобальной.

Для разрешения подобных противоречий в системе Mathematica введен особый механизм *контекстов*. Напомним, что под контекстом подразумевается некоторое разъяснение характера связанных с контекстом объектов. Другими словами, это означает, что с каждым объектом системы Mathematica (например, с переменными или функциями) связан некоторый контекст. Чисто внешне контекст задается в виде **Имя_контекста`** (обратный апостроф в конце имени и есть признак контекста).

Итак, контекст фактически является некоторым признаком объекта. Каждый объект системы Mathematica имеет свой контекст, который записывается перед именем объекта (знак ` при этом является разделителем). Обычно он не виден, но существует. Одни и те же по имени объекты могут иметь разные контексты и действовать по-разному – то есть по контексту. Пользователям ПК полезно усвоить такую аналогию: контексты – это как бы разные папки со своими именами, куда могут помещаться одноименные файлы – объекты.

С другой стороны, один и тот же контекст может принадлежать разным объектам. Например, все системные переменные и встроенные функции имеют контекст System`, то есть они относятся к системным объектам, а все символы, вводимые в начале работы с системой, имеют контекст Global` (глобальные).

14.3.14. Работа с контекстами

В системе Mathematica есть средства для визуализации контекстов. Прежде всего это функция **Context**:

```
Context[Tan]
 System`
Context[E]
 System`
Context/@{Cos,Pi,Abort}
 {System`,System`,System`}
```

Текущее значение контекста определяет системная переменная **$Context** или функция **Context[]**:

```
{$Context,Context[]}
 {Global`,Global`}
```

В начале сессии по умолчанию действует контекст Global`, что означает глобальный статус вводимых символов:

```
Context/@{q,p,w}
 {Global`,Global`,Global`}
```

Однако контекст может быть заменен на любой нужный пользователю просто указанием его перед соответствующим символом или словом:

```
{ new`q,new`w,Global` p}
 {  new`q,  new`w,  p}
Context/@{new`q,new`w,Global`p}
 {new`,new`,Global`}
```

Обратите внимание на то, что символы new`q и new`w имеют свой новый контекст new` и отображаются вместе с ним (но контекст указан перед символом). А вот символ Global` p отображается лишь кратким именем, а полностью не отображается. Причина этого – в том, что текущий контекст есть Global`, а контекст new отсутствует на так называемой контекстной дорожке – проще говоря, в списке контекстов. Что касается символов q, r и z, то сами по себе (без новой контекстной приставки) они по-прежнему имеют контекст Global`:

```
Context/@{q,p,w}
 {Global`,Global`,Global`}
```

Для вывода контекстной дорожки используется переменная **$ContextPath**:

```
$ContextPath
 {Global`,System`}
```

С помощью функции **Prepend** можно добавить в контекстную дорожку новый контекст, например new`:

```
$ContextPath=Prepend[$ContextPath,"new`"]
 {new`,Global`,System`}
```

Теперь функция **Context** возвращает только контексты символов new`q, new`w и Global`r:

```
Context/@{new`q,new`w,Global`p}
 {new`,new`,Global`}
```

С помощью функции **Begin** можно изменить текущий контекст на заданный, например Global` на new`:

```
Begin["new`"]
  new`
q=5;
{ q,Context[q] }
  {5,new`}
```

Теперь легко разобраться в том, как интерпретируются символы с разными контекстами. Любой символ, вводимый без контекстной приставки, то есть своим коротким именем, интерпретируется и выводится с этим именем, если его контекст является текущим. Если символ вводится полным именем, то проверяется, есть ли его контекст на контекстной дорожке. Если он есть, то к символу добавляется самый левый контекст из имеющихся на контекстной дорожке. Таким образом, по мере ввода новых контекстов, имена которых совпадают со старыми, происходит вытеснение новыми контекстами старых. Другими словами, это позволяет обновить уже имеющиеся определения, сохранив их на случай отмены старых контекстов.

Этот принципиально важный механизм модификации объектов играет решающую роль в создании пакетов расширений. В них часто уже имеющиеся функции

(со старыми контекстами) заменяются новыми одноименными с ними, но имеющими иные контексты.

Для получения списка всех определений с заданным контекстом можно использовать функции **Name["Context`S"]**, где S – символ или строка, определяющие имена определений. Например, для получения всех определений с контекстом System` можно использовать функцию **Name["System`*]**. Поскольку этот список довольно большой, ограничимся примером вывода всех определений с контекстом System`, начинающихся с буквы U:

```
Names[«System`U*»]
{UnAlias,Underflow,Underoverscript,UnderoverscriptBox,
UnderoverscriptBoxOptions,Underscript,UnderscriptBox,
UnderscriptBoxOptions,UndocumentedTestFEParserPacket,
UndocumentedTestGetSelectionPacket,Unequal,Unevaluated,Uninstall,
Union,Unique,UnitStep,Unprotect,UnsameQ,Unset,Up,Update,UpperCaseQ,
UpSet,UpSetDelayed,UpValues,URL,Using}
```

Функция **Name[]** без параметра выводит **полный список** всех определений как ядра, так и всех определений в пакетах расширений с указанием их контекстов. Таким образом, данная функция дает самую полную информацию об определениях (функциях, константах и т. д.), которые имеет текущая версия системы Mathematica.

14.3.15. Подготовка пакетов расширений системы Mathematica

Мощным средством расширения возможностей системы Mathematica является подготовка пакетов ее расширений. Пакеты расширений позволяют создавать новые процедуры и функции и хранить их на диске в виде файлов с расширением .m. После считывания такого пакета с диска все входящие в него определения функций становятся доступными для использования в соответствии с правилами, принятыми для встроенных функций. Текст пакета расширения не выводится после его вызова, чтобы не загромождать документ вспомогательными описаниями.

Структура пакета расширений в минимальном виде выглядит следующим образом:

```
(*Вводный комментарий*)
BeginPackage["Имя_пакета`"]
Mean::usage = "Имя функции[Параметры] Текстовый комментарий"
.....................................................
Begin["`Private`"]
Unprotected[Список_имен]
Определения новых функций
End[ ]
Установка атрибутов защиты
EndPackage[ ]
(* Завершающий комментарий *)
```

Особая структура пакетов расширений связана с реализацией описанной выше идеологии контекстов. Пакет открывается необязательным текстовым комментарием, который обрамляется двойными символами (* и *). Он может быть как однострочным, так и многострочным. Обычно вводный комментарий включает в себя имя пакета, наименование фирмы и автора – создателей пакета, историю развития, дату создания и т. д. Если вы программируете для себя, можете на первых порах опустить все эти комментарии. Но не забудьте их ввести после отладки пакета, как того требуют культура и дисциплина программирования.

Затем пакет открывается словом **BeginPackage**. Это слово дается с квадратными скобками, в которых указывается контекст (см. выше) пакета. Обратите внимание на то, что после имени пакета должен стоять апостроф или цепочка символов, обрамленная апострофами. Имя пакета не должно совпадать ни с одним из известных, то есть должно быть уникальным.

Эта команда изменяет контекстную дорожку, и она принимает вид {Имя_пакета`,System`}. Таким образом, на первом месте контекстной дорожки оказывается имя пакета, а на втором – контекст System`. Теперь любой вводимый и невстроенный символ приобретает контекстную приставку с именем данного пакета.

Обратите внимание на то, что контекст System` сохранился на новой контекстной дорожке, но вторым. Это значит, что если вы вводите слова и символы, встроенные в систему, то они будут замещены новыми определениями. К примеру, если вы решили вычислять функцию Sin[x] по новому и ценному для вас алгоритму, то ему будет отдаваться предпочтение при каждом использовании этой функции, до тех пор, пока вы работаете с данным пакетом расширения. Однако, как только вы перестаете работать с пакетом, восстановится роль встроенной функции Sin[x].

Следующий блок пакета – сообщения о назначении функций. Эти сообщения выводятся, если после загрузки пакета задать вопросительный знак с последующим именем функции. Данные сообщения не обязательны, но они обеспечивают единство диалога с системой и, безусловно, нужны при профессиональной подготовке пакета. Обычно в этих сообщениях кратко указываются синтаксические правила использования функций и назначение их параметров, даваемых в квадратных скобках.

Затем следует главная часть пакета – определения новых функций. Она открывается определением **Begin["`Private`"]**. Оно, не меняя контекстной дорожки, устанавливает новый текущий контекст Имя_пакета`Private`. Он присваивается всем ранее не встречавшимся символам. Имя Private принято в пакетах расширения системы Mathematica, хотя в принципе может быть любым другим именем. После него следуют сами определения, в которых могут использоваться любые средства, включенные в ядро системы.

В некоторых случаях имена функций могут повторять ранее определенные в ядре системы. Это полезно, если пользователь считает, что введенное им определение уже известной функции более точно или более универсально, чем использованное в системе. В таких случаях надо позаботиться о снятии с них защиты перед новым применением с помощью функции **Unprotected**. Именно эта часть и определяет существо пакета и его ценность.

Завершается эта часть определением **End[ ]**. При этом восстанавливается контекст, который был до определения Begin["`Private`"], то есть контекст с именем пакета. После этого идет необязательная часть с указанием атрибутов защиты. Пакет завершается определением **EndPackage[ ]**, которое восстанавливает бывший до загрузки пакета контекст (например, Global`), а контекст Имя_пакета` помещает в начало прежней контекстной дорожки.

Контексты в системах Mathematica 3 и 4 идентичны – иначе и быть не может, поскольку всякая старшая версия системы должна обеспечивать совместимость с предшествующей версией. Впрочем, в Mathematica 4 включены два новых контекста – Developer` и Experimental`.

Необязательный заключительный комментарий чаще всего дает список тестовых примеров. Он особенно желателен, если пакет содержит определения не вполне очевидных функций. Не забывайте, что этот комментарий не выводится и не исполняется – он нужен лишь на этапе знакомства с пакетом. Разумеется, такое знакомство необходимо при каждой серьезной попытке применения того или иного пакета расширения или использования системы.

В принципе, текстовые комментарии могут вводиться на русском языке. Однако при этом возникают определенные трудности. Такие пакеты не читаются вьюверами (программами просмотра файлов), ориентированными на просмотр текстов в стандарте ASCII. Да и при выводе их на экран дисплея при работе с оболочкой системы Mathematica также наблюдаются несоответствия между шрифтами, установленными при вводе комментариев и при их выводе. Поэтому лучше использовать комментарии на английском языке, тем более что комментарии ко всем встроенным функциям и к поставляемым расширениям системы даны, естественно, на английском языке.

Для создания пакетов расширений в общем случае используются следующие средства системы:

- **Begin["context`"]** – устанавливает текущий контекст.
- **BeginPackage["context`"]** – делает context единственным активным контекстом. Возможна также форма: **BeginPackage["context`", {"need1`", "need2`", ...}]**.
- **CallProcess[\"command\", f, {x1, x2, ...}]** – вызывает функцию f во внешнем процессе с аргументами xi (CallProcess заменяется операциями MathLink).
- **Do[]** – возвращает Null или аргумент для первого Return, если происходила эволюция.
- **End[ ]** – возвращает текущий контекст и переходит к предыдущему.
- **EndAdd[ ]** – возвращает настоящий контекст и переходит к предыдущему, предварительно добавляя настоящий контекст к $ContextPath.
- **EndPackage[ ]** – восстанавливает $Context и $ContextPath в их значениях до предшествующего BeginPackage и дополняет текущий контекст к списку $ContextPath.
- **EndProcess[\"command\"]** – завершает внешний процесс, в котором функции, возможно, вызывались из системы Mathematica (EndProcess заменен командами MathLink).

- **Exit[ ]** – завершает сеанс работы Mathematica.
- **Goto[tag]** – просматривает текущее составное выражение в поиске Label[tag] и передает управление в эту точку.
- **Interrupt[ ]** – производит прерывание в теле расширения.
- **Label[tag]** – представляет точку в составном выражении, в которую передается управление директивой Goto.
- **Quit[ ]** – завершает сеанс работы системы Mathematica.
- **StartProcess[\"command\"]** – запускает внешний процесс, в котором функции могут вызываться из системы Mathematica. StartProcess должен поддерживаться операциями MathLink.

Приведем пример простого фрагмента программы, дающего определение новой функции **ExpandBoth** с помощью некоторых из представленных средств:

```
(* :Title: ExpandBoth *)
(* :Context: ProgrammingInMathematica`ExpandBoth` *)
(* :Author: Roman E. Maeder *)
ExpandBoth::usage = "ExpandBoth[e] expands all numerators and
denominators in e."
Begin["`Private`"]
  ExpandBoth[x_Plus] := ExpandBoth /@ x
  ExpandBoth[x_] := Expand[ Numerator[x] ] / Expand[
  Denominator[x] ]
End[]
Null
```

Этот пример настолько прост, что читателю будет нетрудно разобраться с его сутью – расширением выражения по числителю и знаменателю. Ниже представлен сеанс работы с этим пакетом, файл которого expboth.m размещен в каталоге mypack, включенном в общий каталог пакетов расширений:

```
<<mypack/expboth.m
?ExpandBoth
  ExpandBoth[e] expands all numerators and denominators in e.
ExpandBoth[124/12]
  31
  --
  3
ExpandBoth[1234/12]
  617
  ---
   6
```

Для создания текстовых комментариев различного назначения (как выводимых, так и не выводимых в ходе исполнения пакета) в языке программирования системы Mathematica используются следующие средства:

- **(* Comment *)** – задание невыводимого текстового комментария в любом месте пакета, как однострочного, так и многострочного.
- **Message[symbol::tag]** – вывод сообщения symbol::tag, если только не выполнено отключение.
- **Message[symbol::tag, e1, e2, ...]** – выводит сообщение, вставляя значения ei по мере необходимости.

- **$MessageList** – глобальная переменная, возвращающая список имен сообщений, вырабатываемых во время вычисления текущей входной строки. Имя каждого сообщения заключено в HoldForm[ ]. $MessageList сохраняется в MessageList[n] и переустанавливается в { } после того, как произведена n-ая выходная строка.
- **MessageList[n]** – глобальный объект, который является списком имен (сообщений), которые вырабатываются в процессе обработки n-ой входной линии.
- **MessageName** – применяется в виде: **symbol::tag** или
- **MessageName[symbol, \"tag\"]** – имя для сообщения.
- **$MessagePrePrint** – глобальная переменная, чье значение, если установлено, применяется к выражениям перед тем, как они помещаются в тексте сообщений.
- **$Messages** – возвращает список файлов и каналов, в которые направляется вывод сообщений.
- **Messages[symbol]** – возвращает все сообщения, присвоенные данному символу symbol.

Следует отметить, что широкое применение комментариев обычно является признаком культуры программирования. Это особенно важно для математических систем, реализующих вычисления по сложным и подчас малопонятным для неспециалистов алгоритмам.

14.3.16. Защита идентификаторов от модификации

Как отмечалось, система Mathematica позволяет вводить константы, переменные и функции со своими именами – идентификаторами. Между функциями можно задавать различные отношения, в том числе и те, которые не соответствуют правилам, заданным в ядре системы.

Идентификаторы должны быть уникальными, то есть не совпадать с именами встроенных функций, директив, опций, переменных и констант. Однако как быть, если нужно задать новое отношение для уже имеющихся встроенных функций или изменить их определения?

Для решения таких вопросов в систему введена защита идентификаторов от модификации, которая при необходимости может сниматься. Все встроенные в ядро поименованные объекты языка программирования системы являются защищенными по умолчанию. Они имеют признак – атрибут Protected (защищенный).

Для управления средствами защиты от модификации используются следующие директивы:

- **Protect[s1, s2, ...]** – устанавливает атрибут защиты от модификации Protected для перечисленных символов si.
- **Protect[\"form1\", \"form2\", ...]** – устанавливает атрибут защиты от модификации для всех символов, имена которых сопоставимы с любым из указанных строковых шаблонов formi.

- **Unprotect[s1, s2, ...]** – удаляет атрибут защиты от модификации Protected для символов si, что делает возможной их модификацию.
- **Unprotect[\"form1\", \"form2\", ...]** – снимает защиту всех символов, имена которых текстуально (по буквам) сопоставимы с любым из указанных formi.

Примеры на применение средств модификации функций уже приводились.

Следующие атрибуты и директивы также используются при управлении модификацией:

- **NProtectedAll** – атрибут, устанавливающий, что ни один из аргументов функции не будет модифицирован при применении N[ ].
- **NProtectedFirst** – атрибут, указывающий, что первый аргумент функции не будет модифицирован применением N[ ].
- **NProtectedRest** – атрибут, устанавливающий, что все аргументы после первого аргумента функции не будут модифицированы применением N[ ].

Мы уже рассматривали модификацию функций, в частности снятие и назначение атрибутов защиты. Отметим лишь, что из последующих примеров будет ясно, что эти операции широко применяются в пакетах расширений.

14.3.17. Практика подготовки пакетов расширений и применений

Наиболее сложным моментом работы с системой Mathematica является разработка пакетов расширения профессионального качества. Именно такие пакеты позволяют приспособить всю мощь системы к решению тех задач, которые полезны конкретному пользователю.

В файловой системе Mathematica 4/5 можно найти множество уже подготовленных пакетов расширения. Поэтому ограничимся примером пакета, который содержит определение функции AlgExpQ[expr], позволяющей выяснить, является ли выражение expr алгебраическим:

```
(* :Title: AlgExp *)
(* :Context: ProgrammingInMathematica`AlgExp` *)
BeginPackage["ProgrammingInMathematica`AlgExp`"]
AlgExpQ::usage = "AlgExpQ[expr] returns true if expr is an
algebraic expression."
Begin["`Private`"]
  SetAttributes[AlgExpQ, Listable]
  AlgExpQ[ _Integer ]  = True
  AlgExpQ[ _Rational ] = True
  AlgExpQ[ c_Complex ] := AlgExpQ[Re[c]] && AlgExpQ[Im[c]]
  AlgExpQ[ _Symbol ]   = True
  AlgExpQ[ a_ + b_ ] := AlgExpQ[a] && AlgExpQ[b]
  AlgExpQ[ a_ * b_ ] := AlgExpQ[a] && AlgExpQ[b]
  AlgExpQ[ a_ ^ b_Integer ]  := AlgExpQ[a]
  AlgExpQ[ a_ ^ b_Rational ] := AlgExpQ[a]
  AlgExpQ[_] = False
End[]
EndPackage[]
```

Если выражение является алгебраическим, то функция AlgExpQ возвращает логическое значение True, иначе она возвращает значение False:

```
<<mypack\algexp.m
?AlgExpQ
 AlgExpQ[expr] returns true if expr is an algebraic expression.
AlgExpQ[a*x^2+b*x+c]
 True
AlgExpQ[Sqrt[x]]
 True
AlgExpQ["x^2+1"]
 False
AlgExpQ[1]
 True
AlgExpQ[1.0]
 False
```

Пакеты применений – это группы документов, предназначенные для решения определенного класса математических или научно-технических проблем и задач. В отличие от пакетов расширения, в документах пакетов применений обычно дается подробно комментируемое описание всех основных алгоритмов решения задач. При этом комментарий обычно выводится на экран дисплея.

14.3.18. Трассировка программ

В практике подготовки и отладки программ важное значение имеет наличие специальных средств отладки программ по шагам – средств **трассировки**. Mathematica имеет ряд функций для осуществления трассировки своих программных конструкций.

Функция **Trace[expr]** позволяет выполнить трассировку выражения expr. Возьмем простой пример – вычисление выражений $2*(3+4)^2/5$:

```
Trace[2 (3 + 4)^2 / 5]
```

$$\left\{\left\{\{3+4,\ 7\},\ 7^2,\ 49\right\},\ \left\{\frac{1}{5},\ \frac{1}{5}\right\},\ \frac{49}{5},\ \frac{49}{5}\right\},\ \frac{2\,49}{5},\ \frac{98}{5}\Bigg\}$$

Результат трассировки представлен вложенными списками, имеющими два элемента – вычисляемое выражение и результат вычислений. В частности, для приведенного примера отчетливо видно, что вначале вычисляется выражение в круглых скобках (3+4) и получается результат 7, который затем возводится в квадрат – получается число 49. Далее вызывается явно не записанная единица для деления на 5, потом 49 умножается на 1/5, и, наконец, 49/5 умножаются на 2, и получается конечный результат. Отсюда ясно, что даже равноценные операции умножения и деления Mathematica разделяет по приоритету – деление выполняется перед умножением!

Символьные операции также могут трассироваться – см. пример ниже:

```
Trace[a*a/(b*b)]
```

$$\left\{\left\{\left\{\{b\,b,\ b^2\},\ \frac{1}{b^2},\ \frac{1}{b^2}\right\},\ \frac{a}{b^2}\right\},\ \frac{a\,a}{b^2},\ \frac{a\,a}{b^2},\ \frac{a^2}{b^2}\right\}$$

Можно выполнить и трассировку рекуррентных вычислений. Функция **TracePrint[expr]** дает распечатку последовательности действий по вычислению выражения expr:

```
TracePrint[a*b/c]
 ab
 ─
 c
Times
a
b
 1
 ─
 c
Power
c
- 1
 ab
 ─
 c
```

Помимо указанных примеров выполнения трассировки и отладки, возможны и иные их варианты с помощью ряда функций. Они представлены ниже:

- **Off[s]** – отключает сообщения трассировки, связанные с символом s.
- **Off[m1, m2, ...]** – отключает несколько сообщений.
- **Off[ ]** – отключает все сообщения трассировки.
- **On[s]** – включает трассировку для символа s.
- **On[m1, m2, ...]** – включает ряд сообщений.
- **On[ ]** – включает трассировку для всех символов.

Применяются также функции **TraceDialog**, **TraceLevel**, **TracePrint** и **TraceScan** с различными опциями.

14.3.19. Динамическое изменение переменных в Mathematica 6 и модель Dymamic

В новейшую Mathematica 6 введена новая концепция динамического изменения значений переменных. Рассмотрим ее суть.

Обычно глобальные переменные в ноутбуке доступны в любом его месте. Например, мы можем задать где-то значение переменной x, равное 1, и проверить ее значение:

```
x=1
 1
x
 1                                                    (1)
```

Затем мы можем изменить это значение на 0.5 и также проверить его:

```
x=0.5
 0.5                                                  (2)
```

Принципиально важно, что в строке вывода (1) остается прежнее значение переменной x, поскольку никакой динамической связи между новым и старым значениями переменных нет.

Совсем иное дело, если мы объявим динамическую связь с помощью функции **Dynamic**:

```
Dynamic[x]
 0.5                                                    (3)
```

В строке вывода (3) появится значение переменной x, равное 0.5, то есть последнее присвоенное переменной значение. Но теперь изменим значение переменной x на иное, например 123:

```
x=123
 123                                                    (4)
```

Вы увидите, что выход в строках выхода, расположенных выше, тут же изменится с 0.5 на 123. Если значение переменной где-то задавалось слайдером, то изменение ее значения тут же приведет к изменению положения движка слайдера. Тем самым реализуется динамическая связь между конечными и предыдущими значениями переменной, включенной в список пареметров функции **Dynamic**.

Описанная возможность применима к любому выражению, то есть основная форма функции имеет вид **Dynamic[expr]**. В форме **Dynamic[expr,None]** устанавливается запрет на динамическую связь выражения. Например, команда

```
{Slider[Dynamic[x,None]],Dynamic[x]}
```

выводит слайдер для изменения x, но перемещение его движка невозможно.

В следующих формах записи

Dynamic[expr,f] **Dynamic[expr,{f,fend}]** **Dynamic[expr,{fstart,f,fend}]**

динамическая связь устанавливается на множественные значения переменных в соответствии со значениями переменной f.

Большинству пользователей описанная возможность может показаться экзотикой. Но это революционное изменение техники программирования, и надо полагать, что должно пройти время, прежде чем это изменение будет оценено должным образом и всерьез применено на практике.

14.3.20. Динамический модуль DynamicModule в Mathematica 6

Часто желательно локализовать динамическую связь пределами одного модуля. Для этого служит функция создания модуля с динамической связью:

DynamicModule[{*x*,*y*,...},*expr*] **DynamicModule[{$x=x_0$,$y=y_0$,...},*expr*]**

Работу этого модуля хорошо поясняет рис. 14.11. Из него видно, что заданное изначально значение переменной x=123 сохраняется и после изменения переменной x в модуле. Речь, стало быть, идет о разных переменных с одним именем – переменная x в модуле локальная, тогда как за пределами модуля (до него и пос-

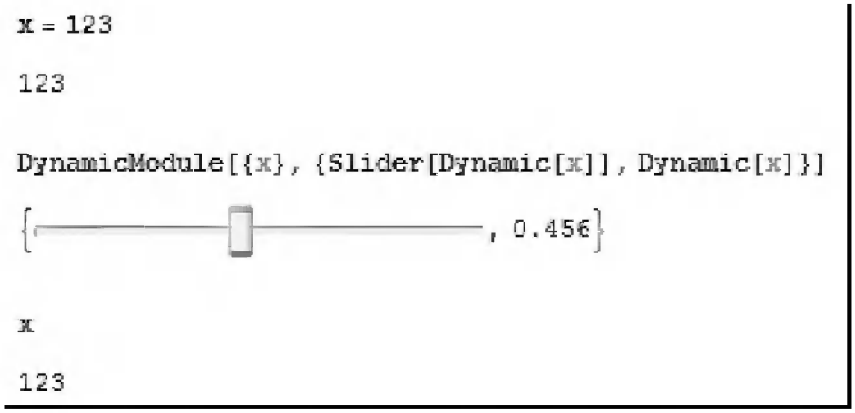


Рис. 14.11. Иллюстрация локализации переменной с динамическим изменением ее значения в модуле DynamicModule

ле) она глобальная. Примеры применения модуля DynamicModule уже приводились неоднократно.

14.3.21. Сброс интерактивных изменений в Mathematica 6

Функция **Deploy[*expr*]** сбрасывает введенные изменения выражения. Например, конструкция

{Graphics[{Disk[],Inset[Slider2D[]]}],Deploy[Graphics[{Disk[],Inset[Slider2D[]]}]]}

выводит два двумерных слайдера, движки которых можно устанавливать в любое положение. Однако стоит исполнить эту команду, как установки станут нулевыми и движки двумерных слайдеров установятся в центральное положение.

14.3.22. Модуль манипуляций Manipulate в Mathematica 6

Динамическая интерактивность – новое качество ноутбуков системы Mathematica 6. Оно заключается в применении довольно простых средств, позволяющих превращать ноутбуки (документы) системы в диалоговые (интерактивные) окна с элементами, дающими возможность динамически управлять параметрами исходных данных для вычислений (в том числе символьных) и построения графиков.

Модуль манипуляций Manipulate позволяет создавать различные интерактивные средства по заданному выражению expr и заданным изменением его параметров:

Manipulate[*expr*, {*u*, u_{min}, u_{max}}]
Manipulate[*expr*, {*u*, u_{min}, u_{max}, *du*}]
Manipulate[*expr*, {{*u*, u_{init}}, u_{min}, u_{max}, ...}]
Manipulate[*expr*, {{*u*, u_{init}, u_{lbl}}, ...}]
Manipulate[*expr*, {*u*, {u_1, u_2, ...}}]
Manipulate[*expr*, {*u*, ...}, {*v*, ...}, ...]
Manipulate[*expr*, "c_u"->{*u*, ...}, "c_v"->{*v*, ...}, ...]

Полезно обратить внимание на то, что хотя модуль Manipulate выводит GUI-средства, например слайдеры, вид средств явно не указывается и зависит от того, какая форма записи функции взята. Приведенные ниже примеры дают прекрасное представление об этом.

На рис. 14.12 показан пример создания модуля, создающего фигурку человечка, состоящую из диска и ряда отрезков прямых. Содержание модуля совершенно очевидно. Отрезки прямых – ручки и отрезки прямых – ножки могут попарно поворачиваться с помощью верхнего и нижнего слайдеров, которые автоматически создаются модулем Manipulate с помощью динамического объекта Slider.

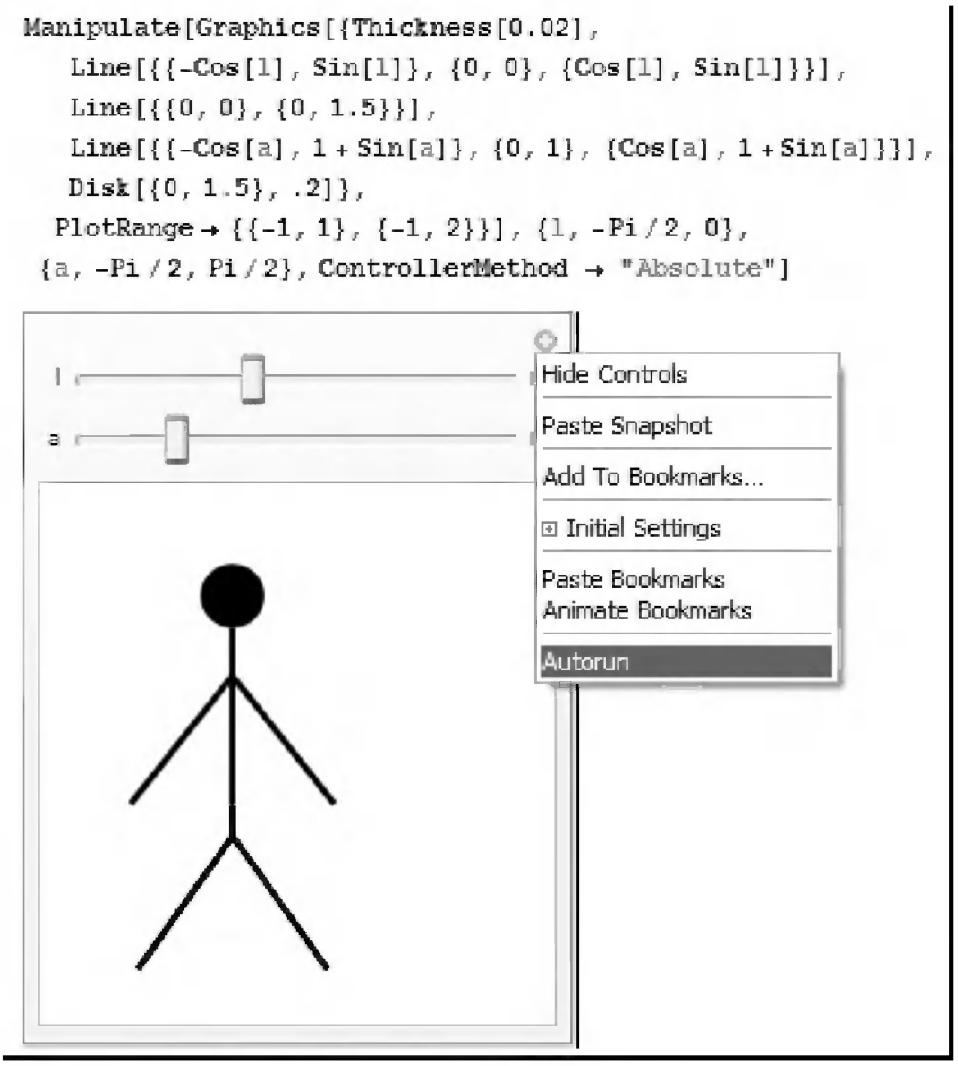


Рис. 14.12. Модуль, строящий фигуру человечка

Щелкнув мышью по кнопочке в виде серого кружка со знаком «+», можно вывести контекстное меню управления созданным в строке вывода графическим объектом. Последняя позиция **Autorun** меню запускает слайдеры на поочередное перемещение движка взад-вперед, то есть создает анимацию объекта – человечек поочередно начинает синхронно вращать туда-сюда ручками и ножками.

Для иллюстрации применения функции-блока Manipulate приведем пример ноутбука (блока) для визуализации фигур Лиссажу. Его листинг представлен ниже:

```
Manipulate[
ParametricPlot[{a1 Sin[n1 (x+p1)],a2 Cos[n2 (x+p2)]},{x,0,20 Pi},
PlotRange->1,PerformanceGoal->"Quality"],
   {{n1,1,"Frequency"},0.5,5,0.1},
   {{a1,1,"Amplitude"},0,1},
   {{p1,0,"Phase"},0,2 Pi},
 Dynamic[ParametricPlot[{a1 Sin[n1 (x+p1)],x},{x,0,2  Pi},
    ImageSize->100,AspectRatio->1,PlotRange->{{-1,1},{0,2 Pi}}]],
 Delimiter,
   {{n2,5/4,"Frequency"},0.5,5, 0.1},
   {{a2,1,"Amplitude"},0,1},
   {{p2,0,"Phase"},0,2 Pi},
 Dynamic[Plot[a2 Sin[n2 (x+p2)],{x,0,2 Pi}, ImageSize->100,
    AspectRatio->1,PlotRange->1]], ControlPlacement->Left]
```

Структура этого ноутбука-блока предельно проста. Внутри функции Manipulate задано построение в параметрической форме фигуры Лиссажу, которая получается из вертикальных и горизонтальных синусоидальных функций с заданными слайдерами частотой от 0,5 до 5 с шагом 0,1 (для удобства останова фигур), амплитудой от 0 до 1 и фазой от 0 до 2Pi. В модуле задано построение синусоидальных функций по осям и построение фигуры Лиссажу. Пример окна с GUI-интерфейсом, которое создает этот модуль, показан на рис. 14.13. Опция **ControlPlacement->Left** выводит блок слайдеров в левую часть окна в ячейке вывода.

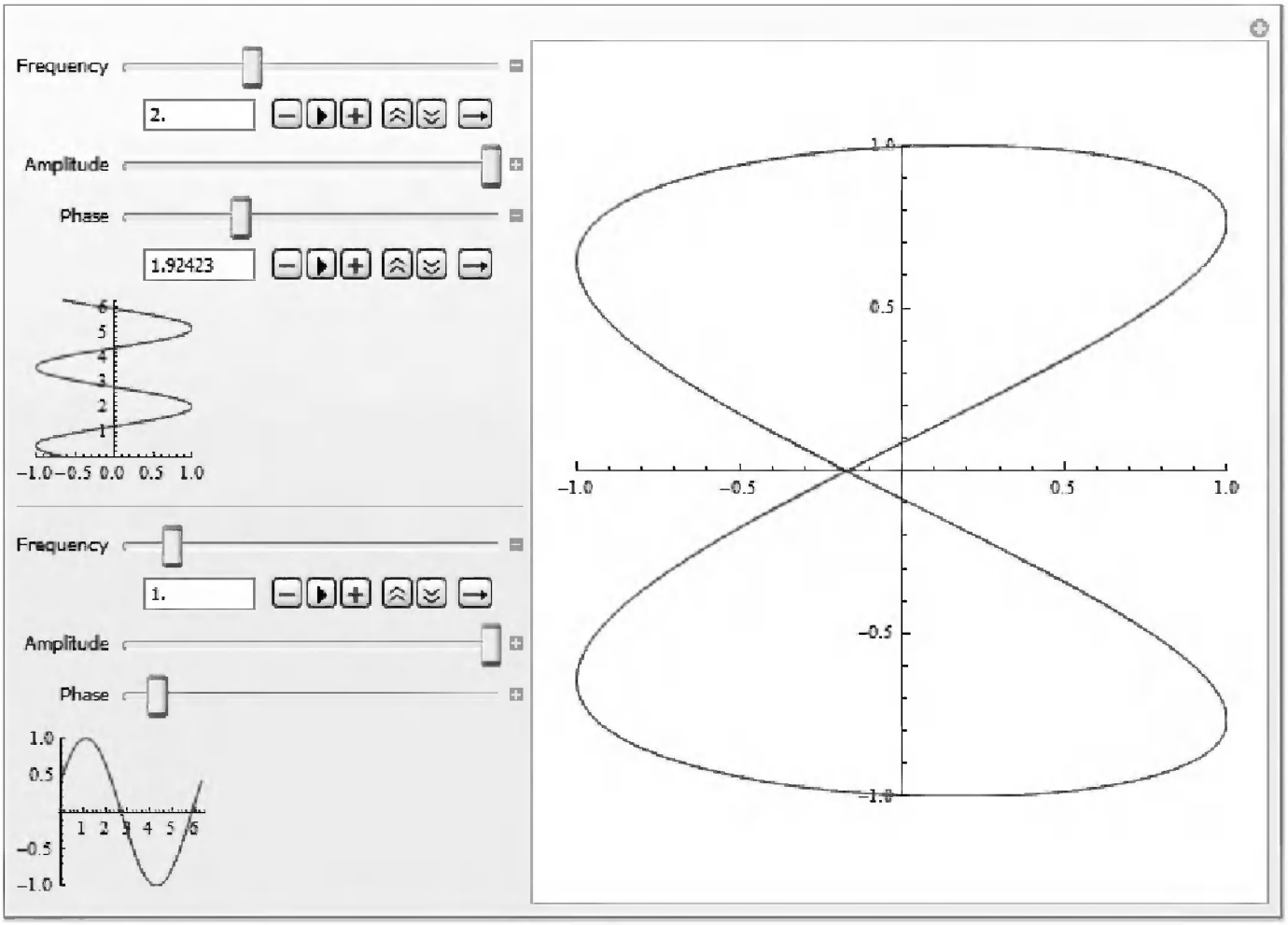


Рис. 14.13. Окно с GUI-интерфейсом для построения фигур Лиссажу

14.4. Функциональное программирование в Derive 5/6

14.4.1. Создание функций в системе Derive

Derive также реализует функциональное программирование, хотя его средства представлены в минимальном объеме. В придачу к встроенным функциям Derive позволяет создавать функции пользователя следующего вида:

Имя_функции(Список переменных) := Выражение

Список переменных (параметров функции) является набором имен переменных, разделенных запятыми. При этом переменные, входящие в список, локальны.

Функция в Derive ведет себя как самостоятельный модуль и связывается с другими функциями через свои входные параметры, представленные списком переменных, и через возвращаемое функцией значение. Размеры выражения в Derive не ограничены, возможны многострочные выражения, хотя надо прямо сказать, что выглядят они очень необычно и не очень понятно. Вот так, например, выглядит программа Derive, вычисляющая функцию Бесселя `BESSEL_J(n,z)` n-го порядка:

```
BESSEL_J_LIST(n,z):=BESSEL_J2(BESSEL_J1(n,z))
BESSEL_J(n,z):=IF(z=0,IF(n=0,1,0),IF(MOD(n)=0,IF(n<0,(-
1)^n,1)*(BESSEL_J_LIST(~ABS(n)+FLOOR(PrecisionDigits/2+
4*ABS(z))+4,z)) SUB (ABS(n)+1),(z/2)^n/(SQRT(pi~) *GAMMA(n+1/
2))*INT(COS(z*COS(t_))*SIN(t_)^(2*n),t_,0,pi)))
```

Программа Derive является просто набором взаимосвязанных функций. Так, в приведенном выше примере первая функция носит вспомогательный характер, а вторая определяет функцию для вычисления специальной функции Бесселя. Никаких интерактивных средств ввода/вывода в программе не предусмотрено – применяются обычные средства командного режима. Практически нет средств отладки программ. Поэтому гарантией их правильности является только представление сложных программ в виде простых функций, каждая из которых тщательно выверяется на контрольных примерах.

14.4.2. Функция выбора по условию IF

Для осуществления выбора по условию y в Derive применяется функция

IF (У, u, v).

Если условие У дает TRUE, то эта функция возвращает значение выражения u, а если условие дает FALSE – то возвращается значение выражения v.

14.4.3. Применение функций SUM и PRODUCT вместо циклов

Циклов как таковых в Derive изначально не было. Детальный анализ областей применения циклов показывает, что нередко они нужны не сами по себе, а для реализации определенных классов задач, например для суммирования и перемножения членов ряда, создания последовательностей данных и т. д. В Derive эти возможности уже реализованы в функциях SUM и PRODUCT для рядов и VECTOR для создания последовательностей данных.

Ниже представлен пример задания и применения специальной функции EI(m,x) с применением функции **SUM** (вид программы дан как на экране дисплея):

```
1:   "Вычисление  функции EI(m,x)"
2:   ce := 0.577215
                                          k
                                    m    x
3:   EI (m, x) := ce + LN (x) +    Σ  ------
                                   k=1  k k!
4:   EI (10, 0.5)
5:   0.454219
```

Derive не содержит явных средств для вычислений сумм или произведений сходящихся рядов при неограниченном числе их членов. Поэтому проще всего заменять такие ряды рядами с конечным и достаточно большим числом членов. Так, в приведенном примере при m=10 точными являются все цифры результата для Ei(0.5).

14.4.4. Функции ITERATES и ITERATE для создания итераций

Итак, некоторая замена циклам в Derive есть. Остается необходимость в создании итерационных алгоритмов. Такие алгоритмы в Derive реализует функция **ITERATES**:

- **ITERATES (u,x,x0,n)** – выполнение *n* итераций при начальном значении *x=x0* с возвратом результатов каждой итерации;
- **ITERATE (u,x,x0,n)** – аналогична ITERATES, но возвращает результат последней итерации.

Функция **ITERATES** позволяет организовать итерационные циклы вычисления функции $u(x)=x$. Если указан параметр *n* (число итераций), то функция **ITERATES** проводит именно *n* итераций. Вначале она подставляет в $(u(x)-x)$ начальное значение *x=x0* и выполняет первую итерацию, используя простой метод

итераций. Затем полученное значение *x=x1* вновь подставляется в $(u(x)-x)$, выполняется вторая итерация и т. д. Таким образом получается вектор значений *x* (*x0*, *x1*,...).

14.4.5. Примеры функционального программирования в Derive

Функции **ITERATES** и **ITERATE** можно использовать для реализации различных методов решения нелинейных уравнений. Так, следующий пример показывает задание функции, решающей систему нелинейных уравнений:

```
FIXED_POINT(g,x,x0,n) := ITERATES(LIM(g,x,xk),xk,x0,n)
```

Здесь g – вектор правых частей системы нелинейных уравнений вида f(x)=x, x – вектор переменных, x0 – вектор начальных значений переменных и n – число переменных.

Однако во многих случаях желательно, чтобы созданная пользователем функция могла иметь в качестве параметра не просто вычисляемое тут же арифметическое выражение, а произвольно заданную функцию. Derive позволяет передавать символическое значение функций, введенных в качестве параметров в лист параметров, если эти функции используются в составе встроенных функций Derive. Эта возможность становится ясной из следующего простого примера:

```
1:   "Функция пользователя с параметром — функцией"
     F (y, x, a) := LOG (lim  y)
2:                          x->a
         2
3:   F (x , x, 2)
4:   1.38629
5:   "Проверка"
           2
6:   LOG (2 )
7:   1.38629
         3
8:   F (x , x, 2)
9:   2.07944
10:  "Проверка"
           3
11:  LOG (2 )
12:  2.07944
```

В этом отнюдь не тривиальном примере задана некоторая функция **F(y,x,a)**. Однако здесь *y* и *x* – необычные параметры. Они позволяют передать в правую часть выражения #2 некоторую произвольную функцию $y(x)$, указанную обобщенным именем *y* и имеющую аргумент *x*. А вот *a* – обычный параметр. Данная функция вычисляет натуральный логарифм для любого значения $y(x)$ при $x=a$.

Для этого несколько необычно использована функция вычисления предела – в данном случае она просто вычисляет значение функции $y(x)$ в точке a. В этом примере параметры y и x передают в тело функции **F(y,x a)** вначале функцию x^2, а затем функцию x^3.

Итак, с помощью функции **IF**, а также функций **STEP** и **CHI** можно осуществлять условный выбор нужных функций. Создание Derive на языке muLISP позволяет в полной мере реализовать возможности **рекурсии**, путем задания рекурсивных функций, которые могут обращаться к самим себе. Наглядным примером может служить рекурсивная функция задания факториала:

```
FACT(n) := IF(n=0, 1, n FACT (n-1))
```

Она соответствует рекурсивному определению факториала:

n! = 1 если n=0
n! = n (n-1)! если n>0

и содержит внутри себя обращение к самой себе.

При задании рекурсивных функций следует не забывать об условиях их завершения. Обычно они создаются с помощью условных функций **IF**. К примеру, в функции FACT (n–1) значение n при каждой рекурсии уменьшается на 1, пока не станет равным нулю – в итоге работа функции прекращается.

Возможна также *взаимная рекурсия*. При ней одна функция обращается к другой, та – к третьей и т. д., причем последующие функции могут обращаться к первой функции. Глубокая рекурсия и взаимная рекурсия могут потребовать больших затрат памяти. Тем не менее рекурсия – основополагающий прием программирования. Следует отметить, что сам по себе язык muLISP базируется на множестве функций, реализующих взаимную рекурсию.

А вот так в Derive выглядит программа решения системы дифференциальных уравнений методом Рунге-Кутта четвертого порядка:

```
RK_AUX3(p,v,u_,c1,c2,c3):=(c1+LIM(p,v,u_+c3)+2*(c2+c3))/6
RK_AUX2(p,v,u_,c1,c2):=RK_AUX3(p,v,u_,c1,c2,LIM(p,v,u_+c2/2))
RK_AUX1(p,v,u_,c1):=RK_AUX2(p,v,u_,c1,LIM(p,v,u_+c1/2))
RK_AUX0(p,v,v0,n):=ITERATES(u_+RK_AUX1(p,v,u_,LIM(p,v,u_)),u_,v0,
n)RK(r,v,v0,h,n):=RK_AUX0(h*APPEND([1],r),v,v0,n)
```

Это определение дает прекрасный пример техники функционального программирования на языке системы Derive и свидетельствует о компактности программ. В этой функции **r** – вектор левых частей системы обыкновенных дифференциальных уравнений первого порядка вида:

$y' = f1(...x,y,z)$ ... $z' = fn(...x,y,z)$.

Вектор **v** должен содержать перечисление всех переменных x, y, z, а вектор **v0** – их начальные значения. Шаг решения по переменной x задается значением h, а число точек решения – значением n. Решение представляется в виде матрицы, столбцы которой дают значения переменных ...x, y, z. Примеры использования этой функции были приведены выше.

14.4.6. Некоторые замечания по функциональному программированию в Derive

Ограничимся этими примерами функционального программирования на входном языке Derive как достаточно типовыми. Следует отметить, что прилагаемая с Derive библиотека функций, описанных в следующей главе, является множеством прекрасно подобранных примеров на практическое программирование в среде Derive. Они наглядно показывают, что с помощью весьма ограниченных средств задания управляющих структур Derive способен решать довольно сложные математические задачи – от вычисления численными методами различных специальных функций до решения систем дифференциальных уравнений современными численными и даже аналитическими методами.

Завершая этот раздел по функциональному программированию Derive, стоит отметить одно важное обстоятельство – соглашение о назначении идентификаторов внутри функций, заменяющих управляющие структуры, например **SUM**, **PRODUCT**, **VECTOR**, **ITERATES** и **ITERATE** и др. Такие идентификаторы нежелательно назначать обычным образом, то есть в виде букв a, b, ... z. Дело в том, что изменение их значений внутри этих функций является для пользователя неожиданным и нежелательным эффектом, способным резко нарушить казалось бы верный алгоритм работы программы.

Для избежания таких эффектов в системе Derive принято обозначать внутренние и служебные переменные со знаком «_» после букв, например a_, b_,...,z_. Разумеется, надо взять за правило не вводить этот знак в идентификаторы обычных глобальных переменных. Тогда вы будете застрахованы от неожиданных эффектов непредусмотренного изменения значений переменных при исполнении той или иной функции.

14.4.7. Средства процедурного программирования

В новые версии Derive 5/6 введена конструкция процедуры, которая начинается со слова **PROG** и может включать в себя цикл типа **LOOP**. Кроме того, процедура может оканчиваться функцией **REST(var)** возврата значения переменной var. Приведем пример заданной таким образом функции:

```
Rev(x,y) := PROG(y:=[], LOOP(IF(x=[], RETURN y),
y:=ADJOIN(FIRST(x),y), x:=REST(x)))
```

Она может быть записана в виде многострочного выражения с короткими строками:

```
Rev(x, y) :=
  Prog
    y := []
```

```
Loop
  If x = []
     RETURN y
   y := ADJOIN(FIRST(x), y)
   x := REST(x)
```

Среди примеров применения Derive и определения дополнительных функций системы можно найти множество примеров, записанных в виде таких конструкций.

14.5. Программирование в системе MuPAD

В отличие от системы Derive, система начального уровня MuPAD имеет все традиционные (можно даже сказать, классические) средства операторного и процедурного программирования. Операторы интерактивного ввода/вывода ее были уже описаны. В набор проблемно-ориентированных программных средств системы входят ее встроенные и библиотечные функции, операторы и команды. Они хорошо дополняются стандартными средствами программирования – условным оператором, циклами различного вида и процедурами.

14.5.1. Условный оператор if-then-else

Условный оператор в MuPAD имеет вид:

if Условие then Выражение_1 else Выражение2 end_if

Если условие выполняется, то выполняется Выражение_1, иначе выполняется Выражение_2. Каждое выражение может быть в единственном числе или представлять ряд выражений.

Примеры применения условного оператора:

- ```
  x:="two":
  if x="two" then print("x=2")
     else print("x<>2") end_if;
  ```

  ```
  "x=2"
  ```

- ```
  x:="one":
  if x="two" then print("x=2")
     else print("x<>2") end_if;
  ```

  ```
  "x<>2"
  ```

14.5.2. Оператор – переключатель case-end_case

С помощью оператора if, применяемого неоднократно, можно осуществить и множественное ветвление в программе. Однако проще это сделать с помощью **оператора-переключателя**:

```
• i:=3:
  case i
     of 1 do print("One");break;
     of 2 do print("Two");break;
     of 3 do print("Three");break;
     of 4 do print("Four");break;
  end_case;
```

```
"Three"
```

Мы ограничимся этим простым и наглядным примером, хотя данный оператор имеет и иные формы записи. Вместе с ним и с описанными ниже операторами цикла могут использоваться операторы **break** (выход из программного объекта) и **next** (выполнение следующего программного объекта).

14.5.3. Циклы вида for-end_for

Для создания **циклов с заданным числом повторений** используются типовые конструкции цикла типа:

for i from is to ie [step di] do
Тело цикла
end_for

for i from is downto ie [step di] do
тело цикла
end_for

В первой управляющая переменная i возрастает от значения i=is до i=ie (is<ie) с шагом 1. Во второй конструкции i уменьшается от значения is до значения ie (is>ie). Следующие примеры поясняют работу этих конструкций и заодно оператора **ptint**:

```
• for i from 1 to 4 do
      q:=i^3;
      print("Число ", i,"в кубе равно ",q)
  end_for;
```

```
"Число ", 1, "в кубе равно ", 1
"Число ", 2, "в кубе равно ", 8
"Число ", 3, "в кубе равно ", 27
"Число ", 4, "в кубе равно ", 64
```

```
• for i from 5 downto 2 do
      print(i,i^2,i^3);
  end_for;
```

```
5, 25, 125
4, 16, 64
 3, 9, 27
 2, 4, 8
```

Выражение **step di** применяется, если шаг изменения управляющей переменной должен отличаться от 1 или –1. Приведенный ниже пример иллюстрирует это:

```
• for x  from 0 to 1 step 0.25 do
      print(x)
```

```
end_for;
                                    0
                                  0.25
                                  0.5
                                  0.75
                                  1.0
```

Еще один вид цикла, в котором значения управляющей переменной выбираются из списка, иллюстрирует следующий пример:

-
```
for i in [1,3,5,9] do print(i,i^2)end_for;
                                 1, 1
                                 3, 9
                                5, 25
                                9, 81
```

Такую возможность, кстати, имеют далеко не все обычные языки программирования.

14.5.4. Циклы типа repeat-until-end_repeat и while-end_while

Циклы с условием типа **until** и **while** также есть в MuPAD:

repeat **while Условие do**
Тело цикла **Тело цикла**
until Условие end_repeat **end_while**

Эти циклы отличаются местом условия сравнения: в первом случае оно расположено в конце цикла, а во втором – в его начале. Ниже даны примеры на их применение:

-
```
x:=1:
repeat
   x:=x+2;print(x)
until x>9 end_repeat;
                                    3
                                    5
                                    7
                                    9
                                    11
```

-
```
x:=1:
while x<9 do
   x:=x+2;print(x)
end_while:
                                    3
                                    5
                                    7
                                    9
```

Обычно эти циклы могут давать отличные результаты, поскольку в одном случае условие проверяется в начале цикла, а в другом – в конце.

14.5.5. Процедуры и процедуры-функции в системе MuPAD

Важным средством расширения языков программирования, помимо функций пользователя, являются *процедуры* и *процедуры-функции*. Они являются самостоятельными модулями языка и обеспечивают обмен данными через свои входные и выходные параметры. Программа в системе MuPAD – это некоторая процедура. Внутри нее могут быть другие процедуры и функции, а также описанные выше управляющие конструкции. Наличие процедур обусловливает возможность процедурного и модульного программирования.

Процедура в MuPAD может иметь одну из двух конструкций:

```
Имя:=proc(x,y,z,....)          proc(x,y,z,...)
  [local a,b,c...;]              name Имя
  [options;]                     [local a, b, c, ...;]
  begin                          [options;]
    Тело процедуры                begin
  end_proc                          Тело процедуры
                                 end_proc
```

В первом варианте процедуры обычно задаются пользователем в его программе, во втором – применяются в библиотеках процедур. Имена процедур задаются по правилам задания идентификаторов. Главное, что они должны быть уникальными. В библиотечных процедурах имена имеют вид **Имя_пакета::Имя_процедуры**.

Помимо локального *статуса переменных*, указанных в списке параметров процедур, в них можно задать такой статус и для любых других переменных, объявив их с помощью слова **local**. Возможно также задание опций – options. После этого процедура открывается словом **begin**. Тело процедуры может содержать любые выражения, в том числе математические. Завершается процедура словом **end_proc**. В любом месте процедуры можно ввести комментарий с помощью символов #, например # Это комментарий #.

Процедура может быть процедурой-функцией, то есть в ответ на обращение к ней **Имя(x, y, z, ...)** она может возвращать некоторое значение. Принято, что это значение последнего выражения в теле процедуры. С помощью слова **return** могут возвращаться и значения выражений в любом месте процедуры, если это предусмотрено циклами с условиями их прекращения или условными операторами.

Для иллюстрации статуса переменных в процедурах рассмотрим следующий пример:

- ```
  a:=0:b:=0:c:=0:
  modz:=proc(z) #Вычисление модуля комплексного числа#
  begin
  a:=Re(z):b:=Im(z):
  c:=sqrt(a^2+b^2)
  end_proc:
  ```
  ```

```
• modz(3+4*I);
                                    5
• a;b;c;z;
                                    3
                                    4
                                    5
                                    z
```

Здесь задана процедура **modz(z)**, вычисляющая модуль комплексного числа. В ней использованы три глобальные переменные с начальными нулевыми значениями. После исполнения процедуры значения этих переменных изменились. Это принято называть *побочным эффектом* процедуры. Однако значение переменной z осталось прежним – неопределенным, что связано с особым статусом данной переменной – она является входящей в список параметров процедуры.

Как уже говорилось, появление побочных эффектов – серьезный недостаток. В сложных программах он может существенно усложнить их отладку. Поэтому желательно проектировать процедуры так, чтобы эти эффекты не возникали или были полностью подконтрольны.

Теперь модернизируем процедуру, введя в нее объявление переменных a, b и c локальными:

```
• a:=0:b:=0:c:=0:
  modz:=proc(z) #Вычисление модуля комплексного числа#
  local a,b,c;
  begin
  a:=Re(z):b:=Im(z):
  c:=sqrt(a^2+b^2)
  end_proc:
• modz(3+4*I);
                                    5
• a;b;c;z;
                                    0
                                    0
                                    0
                                    z
```

Видно, что здесь как переменная z, так и переменные a, b и c имеют статус локальных переменных и на выходе процедуры сохраняются их исходные значения – нулевые для переменных a, b и c и статус необъявленной переменной для z.

В MuPAD можно задавать процедуры в процедурах, например:

```
f:=proc(...)
  local g;
  begin
    g:=proc(...) begin ... end_proc; # Подпроцедура#
  .....#Тело основной процедуры с вызовами g(...)#
  end_proc
```

Процедуры могут быть рекурсивными, то есть внутри процедуры можно задавать обращение к ней же самой.

Любую процедуру можно вывести для просмотра, указав ее имя. Ниже дан пример этого для библиотечной процедуры **numeric::fsolve**:

```
• numeric::fsolve;

proc(p, x, a, b)
  name numeric::fsolve;
begin
  if type(subs(p, x = Dom::Interval(a..b), Simplified)) <>
Dom::Int\
erval then
    error("cannot evaluate to interval")
  end_if;
  p:=[numeric::fsolve2(args())];
  a:=1;
  while a < nops(p) do
    for b from a + 1 to nops(p) do
      if op(p[b — 1], 2) <> op(p[b], 1) then
        break
      end_if
    end_for;
    p[a]:=op(p[a], 1)..op(p[b — 1], 2);
    for b from a + 1 to b — 1 do
      p[a + 1]:=NIL
    end_for;
    a:=a + 1
  end_while;
  op(p)
end_proc
```

Даже эта небольшая процедура демонстрирует наличие в ней условного оператора **if** и циклов типа **while** и **for**. Обратите внимание на запись процедуры с выделением ее конструкций (структур) отступами. Такая запись является внешним признаком структурного программирования и способствует лучшему пониманию программ.

14.6. Средства программирования системы Mathcad

В Mathcad уже давно были введены традиционные средства программирования. Они задаются в виде оригинальных программных боков, выделенных жирными вертикальными прямыми. Так что программный блок стал еще одним типом блоков, используемых в системах Mathcad наряду с текстовыми, формульными и графическими блоками.

14.6.1. Задание операторов пользователя

Начнем обсуждение программных средств Mathcad с функций пользователя, которые являются первым признаком программирования. Однако Mathcad поддерживает еще одну интересную возможность – задание новых *операторов* пользователя. Такой оператор задается практически так же, как функция пользователя, но вместо имени выбирается какой-либо подходящий знак. Например, можно задать оператор деления в виде:

$$\div(A,B) := \frac{A}{B}.$$

После этого новым оператором можно пользоваться:

- `÷(6,2) = 3` – пример применения новой *функции* деления;
- `6÷2 = 3` – пример применения нового *оператора* деления.

При кажущейся простоте такого задания здесь есть проблемы. Встроенные в систему операторы нельзя переопределять, поэтому набор доступных знаков для обозначения новых операторов ограничен. К примеру, нельзя задать новый оператор деления знаком / (он уже использован), но можно взять знак ÷, поскольку этот символ как знак деления системой не используется.

Вторая проблема связана с вводом символа нового оператора. Скорее всего, напрямую ввести его будет нельзя. Придется воспользоваться типовыми приемами ввода новых символов в документы Windows, например используя приложение, выдающее таблицу символов и предоставляющее возможность экспорта символа из этой таблицы в документ другого приложения (в нашем случае – в документ Mathcad).

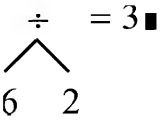
Можно также воспользоваться подходящим знаком из набора **Extra Math Symbols** (Дополнительные математические символы), имеющегося в составе быстрых «шпаргалок» (QuickSheets), доступ к которым предоставляет Центр ресурсов. Для переноса знака в документ можно скопировать его в буфер обмена командой **Copy** (Копировать), а затем вставить в документ командой **Paste** (Вставить). Знак можно дополнить. Например, если вы хотите создать оператор с именем °C, то возьмите за основу знак ° и добавьте C. Для получения оператора °F придется проделать все заново, начав с ввода набора **Extra Math Symbols**.

Для работы с операторами пользователя служит панель математических знаков `Evaluation`. В ней для этого имеются четыре кнопки:

- `fx` – использование унарного оператора пользователя;
- `xf` – использование инфиксного оператора пользователя;
- `xfy` – использование бинарного оператора пользователя;
- `xfy` – использование бинарного оператора пользователя в форме «дерева».

Итак, после того как оператор задан, его можно использовать и как функцию, и как оператор. При использовании нового оператора надо вывести его шаблон

с помощью панели математических знаков **Evaluation**. В нашем примере следует щелкнуть на кнопке `xfy` этой панели, которая выводит особый шаблон с тремя местами ввода. Введите операнды, например 6 и 2, в крайние места ввода, а символ оператора – в среднее. Поставив после этой конструкции знак равенства, увидите результат – число 3.



Разумеется, можно задавать и другие операторы, в том числе для работы с одним операндом. Так, например, можно следующим образом определить операторы пересчета температуры по шкале Цельсия в температуру по шкале Фаренгейта:

$$°C(x) := \frac{9}{5} \left| \; x + 32 \quad °F := 1. \right.$$

Используя затем кнопку `xf` на панели символов отношения, можно выполнять операцию пересчета в виде:

37°C = 98.6·°F.

14.6.2. Задание и применение функций пользователя

Функции пользователя в Mathcad задаются предельно просто и естественно:

Имя_функции(Список_параметров) := Тело_функции

Имя функции составляется по правилам задания имен переменных. Список параметров содержит перечень переменных, значения которых под именем переменных передаются в тело функции. Примеры задания функций пользователя:

$\sin3(x) := \sin(x)^3$

$\text{dist}(x,y) := \sqrt{x^2 + y^2}$

Переменные в списке параметров функции пользователя являются локальными переменными. На их место могут быть записаны константы или выражения, которые передаются в тело функции. Но за пределами функции пользователя эти переменные могут иметь иные значения или быть неопределенными.

14.6.3. Задание программных модулей

Возможность задания *программных блоков* (*модулей*) появилась в версии Mathcad PLUS 6.0 и в расширенном варианте поддерживается в последующих версиях, включая Mathcad 12/13/14. Средства программирования сосредоточены в палитре программных элементов, показанной на рис. 14.14 на фоне фрагмента документа Mathcad.

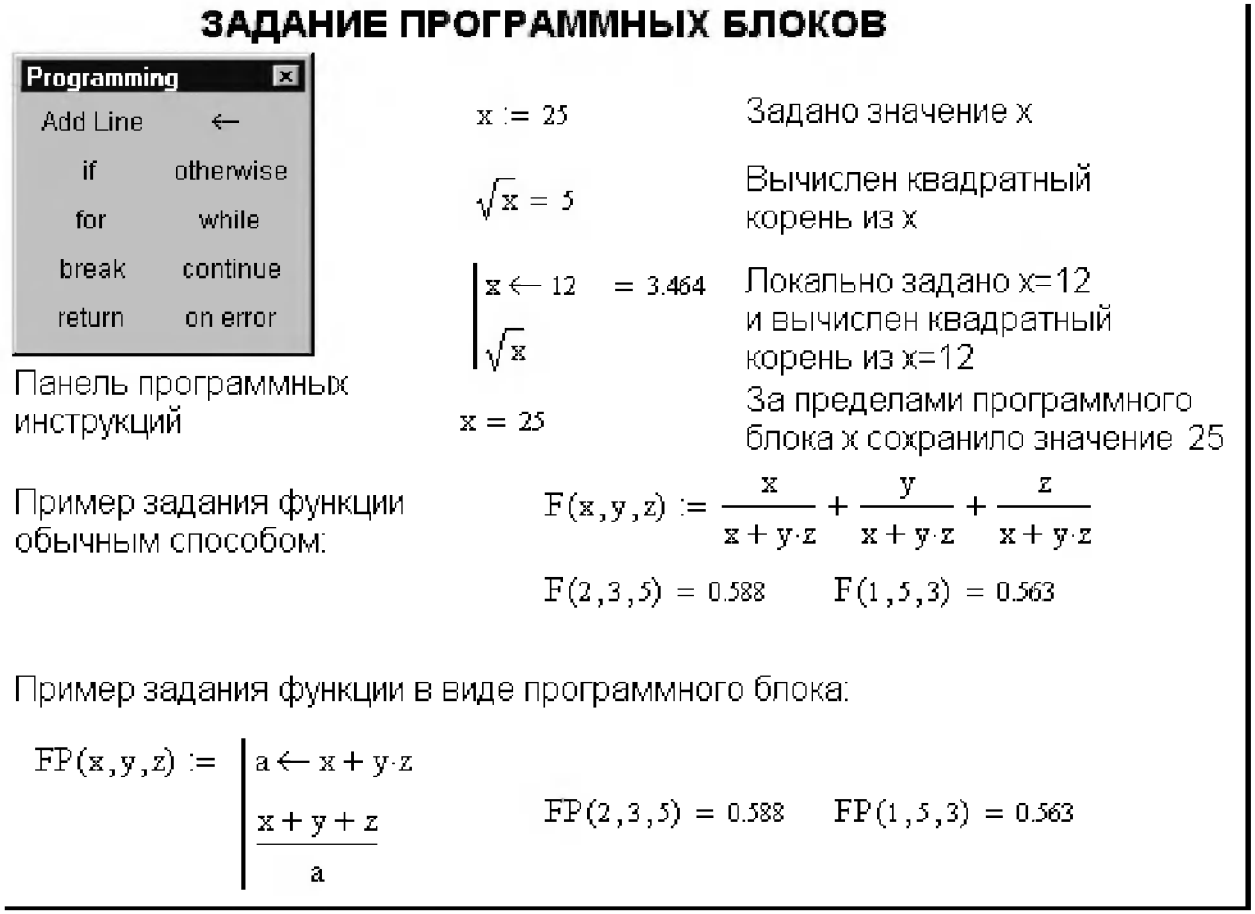


Рис. 14.14. Задание программных модулей

Как видно из рисунка, программный блок в системе Mathcad превратился в самостоятельный модуль, выделяемый в тексте документа жирной вертикальной чертой. Модуль может вести себя как безымянная функция без параметров, но возвращающая результат (см. первый пример – вычисление квадратного корня из числа 12). Программный модуль может выполнять и роль тела функции пользователя с именем и параметрами (см. пример в нижней части рисунка).

Нетрудно заметить (см. палитру программных элементов на рис. 14.14), что набор инструкций для создания программных модулей весьма ограничен и содержит следующие элементы:

- `Add Line` – создает и при необходимости удлиняет жирную вертикальную линию, справа от которой в местах ввода производится запись программного блока;
- ← – символ локального (в теле модуля) присваивания;
- `if` – условная инструкция;
- `otherwise` – инструкция иного выбора (обычно применяется с `if`);
- `for` –инструкция задания цикла с фиксированным числом повторений;
- `while` – инструкция задания цикла, действующего до тех пор, пока выполняется некоторое условие;
- `break` – инструкция прерывания;
- `continue` – инструкция продолжения;
- `return` – инструкция возврата;
- `on error` – инструкция обработки ошибок.

14.6.4. Инструкция добавления линий в модуль Add Line

Инструкция `Add Line` выполняет функции расширения программного блока. Расширение фиксируется удлинением вертикальной черты программных блоков или их древовидным расширением. Благодаря этому, в принципе, можно создавать сколь угодно большие программы.

14.6.5. Оператор внутреннего присваивания

Оператор ← выполняет функции внутреннего (локального) присваивания. Например, выражение `x ← 123` присваивает переменной `x` значение 123. Локальный характер присваивания означает, что такое значение переменной `x` хранится только в теле программного модуля. За пределами тела программы значение переменной `x` может быть неопределенным либо равным значению, которое задается вне программного блока операторами локального (`:=`) или глобального (≡) присваивания.

Не стоит путать оператор внутреннего присваивания ← с оператором символьного вывода →, у которого стрелка направлена в другую сторону. Эти операторы решают совершенно разные задачи.

14.6.6. Условная инструкция if

Инструкция `if` позволяет строить условные выражения. Она задается в виде:

```
Выражение if Условие
```

Если `Условие` выполняется, то возвращается значение `Выражения`. Совместно с этой инструкцией часто используются инструкции прерывания `break` и иного выбора `otherwise`.

14.6.7. Инструкция организации цикла for

Инструкция `for` служит для организации циклов с заданным числом повторений. Она записывается в виде:

```
for Var ∈ Nmin .. Nmax
```

Эта запись означает, что выражение, помещенное в расположенное ниже место ввода, будет выполняться для значений переменной `Var`, меняющихся от `Nmin` до `Nmax` с шагом +1. Переменную счетчика `Var` можно использовать в исполняемом выражении.

14.6.8. Инструкция организации цикла while

Инструкция `while` служит для организации циклов, действующих до тех пор, пока выполняется некоторое условие. Она записывается в виде:

```
while Условие
```

Выполняемое выражение записывается в расположенное ниже место ввода.

14.6.9. Инструкция otherwise

Инструкция иного выбора `otherwise` обычно используется совместно с инструкцией `if`. Это поясняет следующая программная конструкция:

$$f(x):=\begin{vmatrix} 1 \ if \ x>0 \\ -1 \ otherwise \end{vmatrix}$$

То есть функция `f(x)` возвращает 1, если `x > 0`, и –1 во всех остальных случаях.

14.6.10. Инструкция прерывания break

Инструкция `break` вызывает прерывание выполнения программы. Чаще всего эта инструкция используется совместно с условной инструкцией `if` и инструкциями циклов `while` и `for`, обеспечивая переход в конец тела цикла.

14.6.11. Инструкция continue

Инструкция `continue` используется для продолжения работы после прерывания программы. Она также чаще всего используется совместно с инструкциями циклов `while` и `for`, обеспечивая возвращение в точку прерывания и продолжение вычислений.

14.6.12. Инструкция return

Особая инструкция `return` прерывает выполнение программы и возвращает значение операнда, стоящего следом за ней. Например, в приведенном ниже случае будет возвращаться значение 0 при `x < 0`:

```
return 0 if x<0
```

14.6.13. Инструкция on error и функция error

Инструкция `on error` позволяет создавать процедуры обработки ошибок. Эта инструкция задается в виде:

```
Выражение_1 on error Выражение_2
```

Если при выполнении `Выражения_1` возникает ошибка, то выполняется `Выражение_2`. Для обработки ошибок полезна также функция `error(S)`, которая, будучи помещенной в программный модуль, при возникновении ошибки выводит всплывающую подсказку с сообщением, хранящимся в символьной переменной `S`.

14.6.14. Простейшие примеры создания программных модулей

Несмотря на скромность набора программных средств, имеющихся в Mathcad, они дают системе именно те возможности, которые ранее попросту отсутствова-

ли: задание специальных функций, задание различных видов циклов (в том числе вложенных), упрощение алгоритмов ряда вычислений и реализацию различных итерационных и рекурсивных процедур. Рекомендуется внимательно изучить фрагмент документа Mathcad, показанный на рис. 14.15 и иллюстрирующий часть этих возможностей.

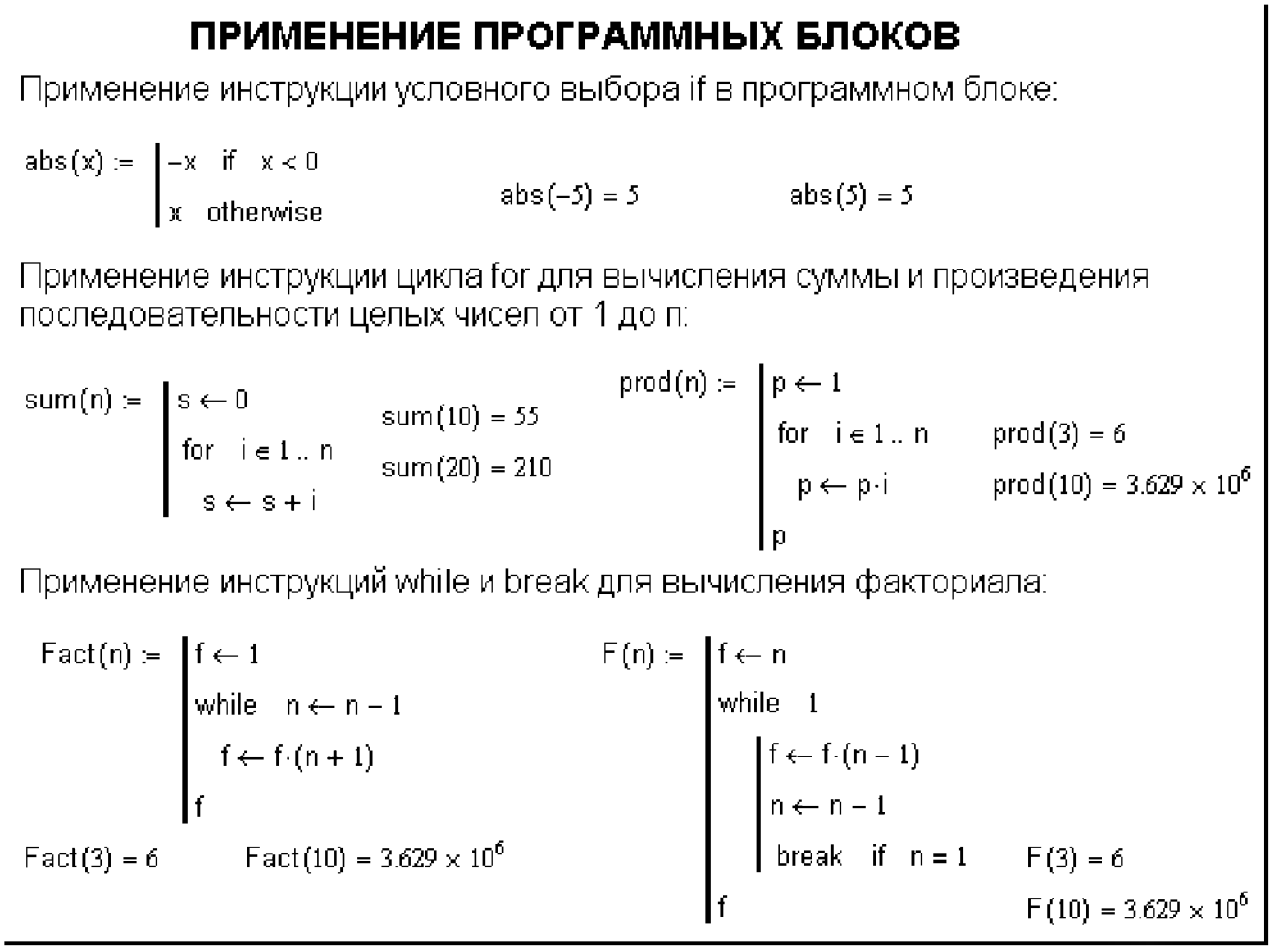


Рис. 14.15. Примеры задания программных блоков

Обратите особое внимание на пример вычисления факториала. Здесь один программный модуль задается внутри другого. Вообще говоря, для нескольких подмодулей, которые должны выполняться в составе циклов, служит команда **Add Line** (добавить линию), добавляющая в модуль дополнительную вертикальную черту для подмодуля.

Программный модуль, в сущности, является функцией, но описанной с применением упомянутых программных средств. Она возвращает значение, определяемое последней инструкцией (если не предусмотрено иное с помощью инструкции `return`). Это значит, что после такого модуля, выделенного как целый блок, можно поставить знак равенства для вывода результата его работы. В блоке могут содержаться любые операторы и функции входного языка системы. Для передачи в блок значений переменных можно использовать переменные документа, которые ведут себя в блоке как глобальные.

Обычно модулю присваивается имя со списком переменных, после которого идет оператор присваивания :=. Переменные в списке являются локальными, и им можно присваивать значения при вызове функции, заданной модулем. Локальный характер таких переменных позволяет использовать для их идентификаторов те же имена, что и у глобальных переменных документа. Однако лучше этого не делать и назначать разные имена для локальных переменных программных модулей и переменных документа.

14.6.15. Обработка ошибок в программных модулях

На рис. 14.16 показан фрагмент документа Mathcad с примерами применения инструкций `on error` и `return`, а также примером действия функции `error`, задающей вывод всплывающей подсказки при указании мышью на выражение, содержащее ошибку.

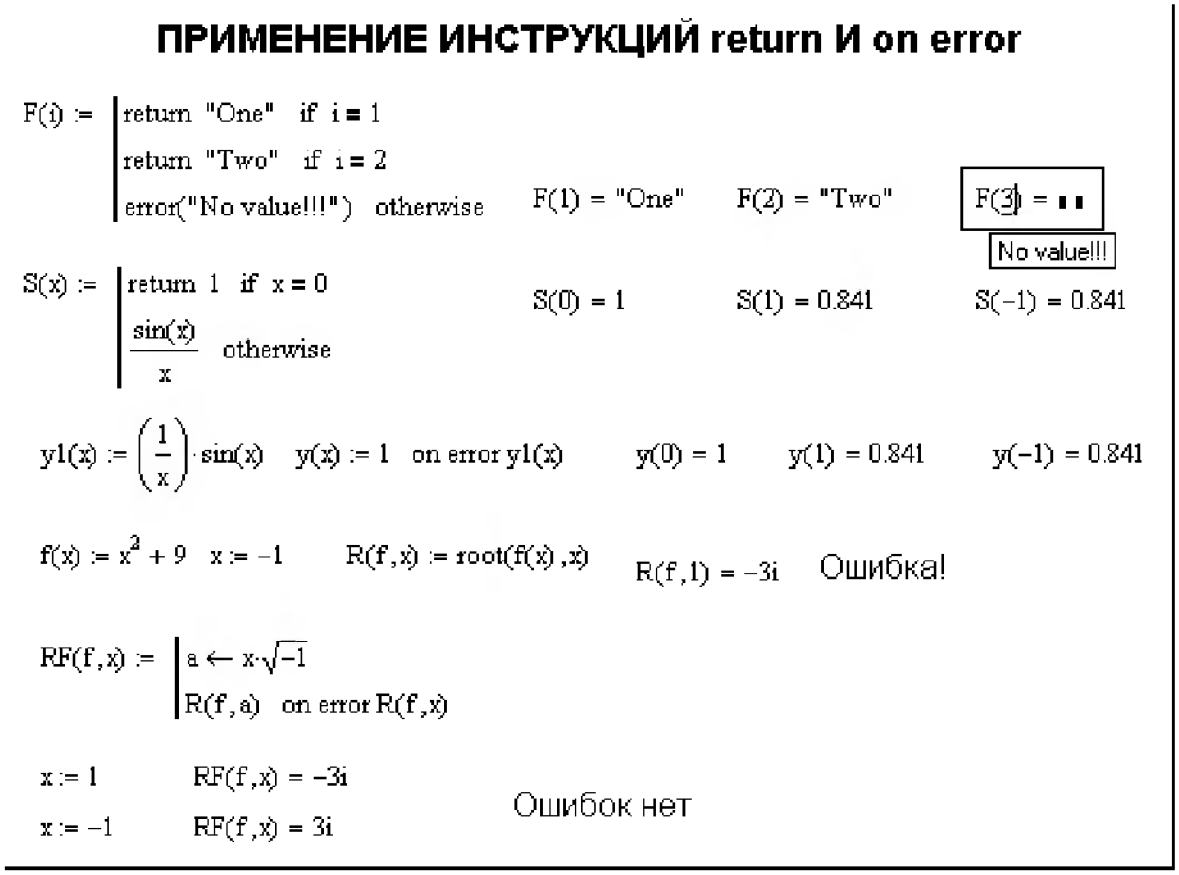


Рис. 14.16. Применение инструкций on error и return

В этих примерах надо обратить внимание на два момента. Первый – возможность применения инструкции `on error` вне программного блока для задания функции `y(x)=sin(x)/x` с доопределением ее при `x=0: y(0) = 1`. Второй – применение этой инструкции для решения уравнения, имеющего комплексные корни. Когда при таком решении задается начальное значение `x` (действительное

число), то функция `root(f,x)` вызывает ошибку. Она исправляется с помощью инструкции `on error` при задании функции `RF(f,x)`, имеющей процедуру обработки ошибок.

Таким образом, инструкция `on error` фактически играет роль процедуры обработки ошибок. Она позволяет создавать программные модули, защищенные от грубых ошибок.

14.6.16. Модуль построения точек в пространстве

На рис. 14.17 показан фрагмент документа Mathcad с примером применения программного модуля для изображения в трехмерном пространстве точек с координатами `x`, `y` и `z = f(x,y)`, где `f(x,y)` – некоторая функция. Точки изображены на фоне параллелепипеда, внутри которого они находятся.

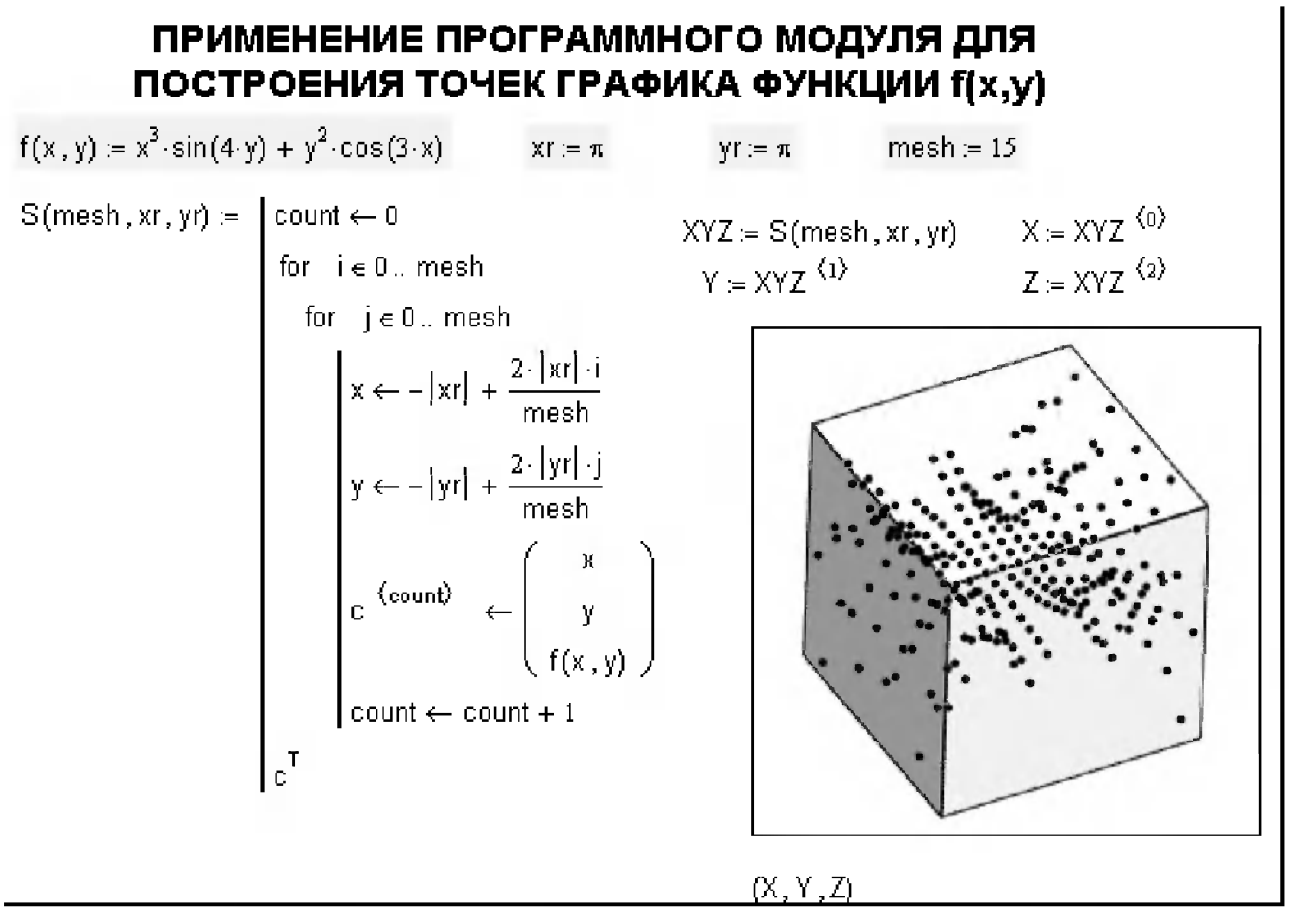
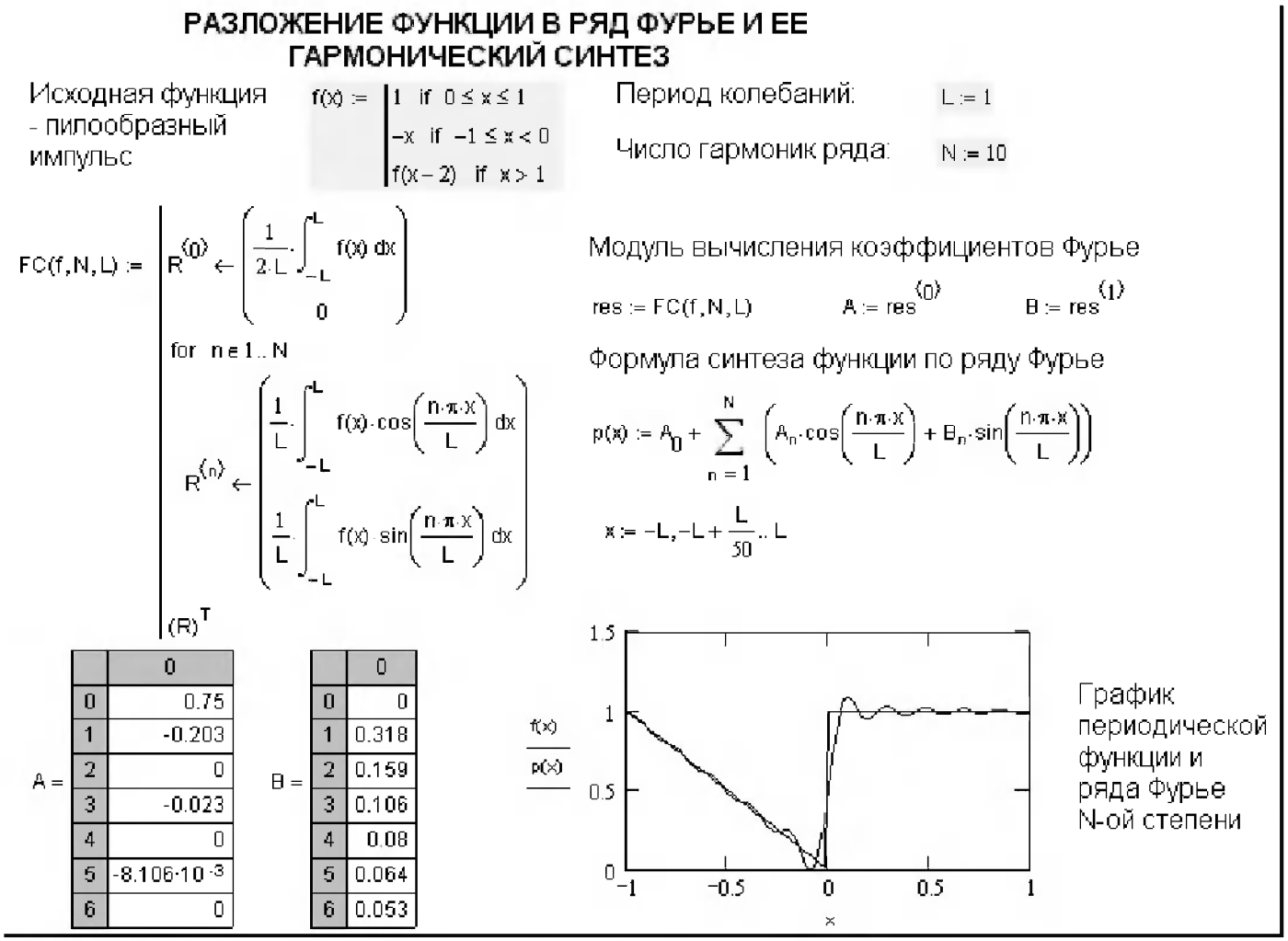


Рис. 14.17. Применение программного модуля для построения в пространстве точек с функционально связанными координатами

Этот пример иллюстрирует возможность применения внутри программного модуля графических функций – в данном случае функции `mesh`.

14.6.17. Модуль Фурье-анализа

Теперь рассмотрим достаточно сложный, но поучительный пример применения программного блока. На рис. 14.18 показана верхняя часть документа Mathcad, в котором с помощью программного модуля вычисляются коэффициенты Фурье для заданной функции (в рассматриваемом примере – пилообразного импульса).



	0
0	0.75
1	-0.203
2	0
3	-0.023
4	0
5	$-8.106\cdot10^{-3}$
6	0

	0
0	0
1	0.318
2	0.159
3	0.106
4	0.08
5	0.064
6	0.053

Рис. 14.18. Разложение функции в ряд Фурье

Здесь вычисление коэффициентов ряда Фурье выполнено по их классическим интегральным представлениям. Это хорошо иллюстрирует возможность применения в программных блоках весьма мощных средств входного языка системы (например, блоков с интегралами). Завершается документ реализацией гармонического синтеза сигнала. На рисунке показаны векторы коэффициентов Фурье A и B, формула гармонического синтеза исходной функции и графики исходной функции с ее представлением рядом Фурье с ограниченным числом гармоник N.

Вы получите прекрасные результаты, когда число точек спектрального анализа и синтеза порядка десятка или несколько более. Однако чем больше точек использовать, тем медленнее выполняется программа. Причина этого в том, что она использует довольно медленный алгоритм численного интегрирования. Встроен-

ные в Mathcad функции быстрого преобразования Фурье (БПФ) при большом числе точек (сотни, тысячи) оказываются куда более быстрыми. Их и применяют при решении серьезных задач спектрального анализа и синтеза.

14.6.18. Рекурсивная генерация простых чисел

Простыми называют числа, которые делятся только сами на себя и не могут быть разложены на множители. Все они – нечетные числа, кроме единственного четного числа 2. На рис. 14.19 представлен программный модуль, который реализует рекурсивный алгоритм поиска простых чисел из множества из n натуральных чисел.

Интересно отметить, что зависимость значений простых чисел от значений натуральных чисел близка к линейной и представлена на рис. 14.19.

Рис. 14.19. Реализация гармонического синтеза

14.6.19. Программа моделирования аттрактора Лоренца

В физике хорошо известны колебательные системы – аттракторы, у которых наблюдается неустойчивый и порой весьма причудливый характер колебаний. Обсуждение таких систем явно выходит за рамки данной книги. Поэтому ограничимся практическим примером. На рис. 14.20 показан начальный фрагмент документа Mathcad с программной реализацией такого аттрактора на основе решения уравнений колебаний конечно-разностным методом.

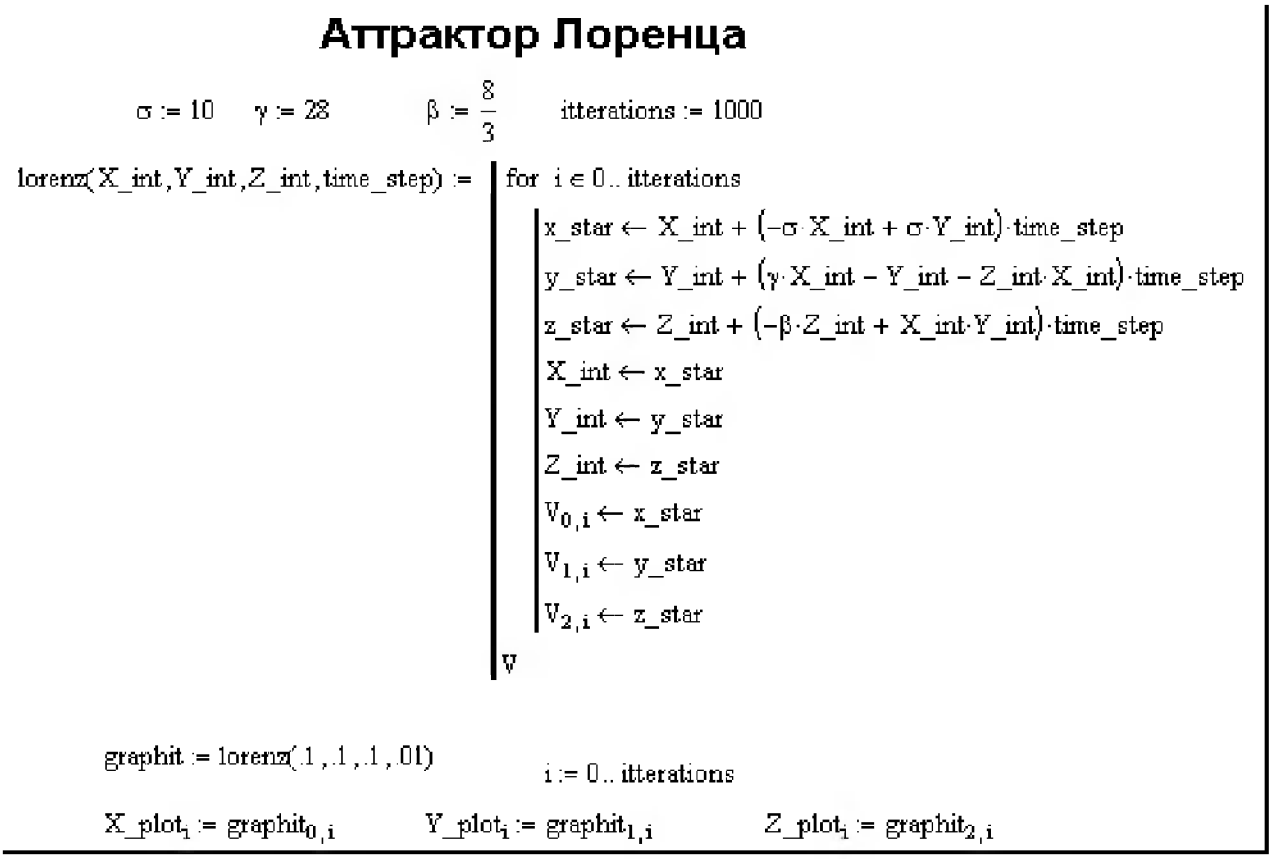


Рис. 14.20. Программа моделирования аттрактора Лоренца (начало)

Следующий фрагмент документа Mathcad с фазовым портретом колебаний в пространстве представлен на рис. 14.21. Сам фазовый портрет задан программным модулем, показанным на рис. 13.20. Нетрудно заметить, что характер колебаний и впрямь довольно причудливый. Вид колебаний можно менять в широких пределах, изменяя исходные параметры (см. рис. 14.21 сверху).

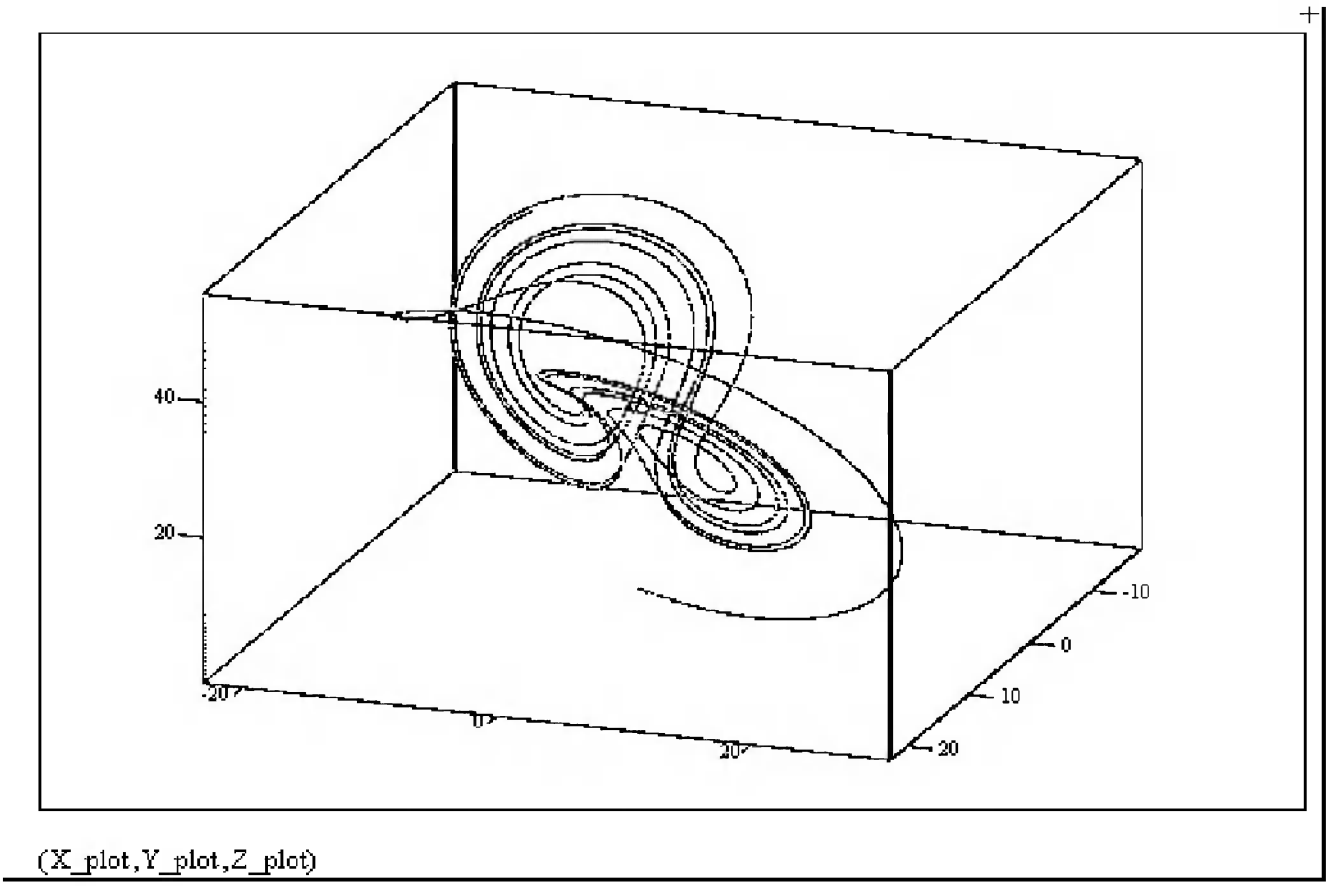


Рис. 14.21. Программа моделирования аттрактора Лоренца (окончание)

14.6.20. Построение фрактала «кукуруза»

К числу занимательных разделов математики относятся фракталы – поверхности, малые части которых как бы повторяют общую структуру поверхности. Фрактальная геометрия – особая область математической графики. Не слишком вторгаясь в нее, рассмотрим один из документов Mathcad, в котором задано построение фрактальной поверхности, напоминающей ячейки кукурузы (рис. 14.22).

Этот пример приведен не столько для иллюстрации методов построения фракталов, сколько в качестве еще одного довода в поддержку программирования в системе Mathcad. Кроме того, это хороший пример организации рекуррентных вычислений и применения программного оператора цикла.

Фрагмент документа Mathcad с фракталом, определяемым этим программным модулем, представлен на рис. 14.23. Он и впрямь напоминает ячеистую структуру кукурузного початка.

14.6.21. Замечания по программированию в Mathcad

Много интересных и поучительных примеров задания и применения программных модулей можно найти в «быстрых шпаргалках» (QuickSheets) центра ресурсов системы. Нельзя не отметить, что характер задания программных модулей

ПОСТРОЕНИЕ ФРАКТАЛА - "КУКУРУЗА"

$x_{n+1} = x_n - h \cdot f(y_n)$ $y_{n+1} = y_n - h \cdot f(x_n)$ $\frac{dx}{dt} = -f(y)$ $\frac{dy}{dt} = -f(x)$ $f(z) := \sin(z + \tan(3 \cdot z))$

```
popcorn(N,h,a) := | t ← -1
                  | d ← 2·a/N
                  | for j ∈ 0..N
                  |   for k ∈ 0..N
                  |     t ← t + 1
                  |     x_t ← d·j - a
                  |     y_t ← d·k - a
                  |     for n ∈ 0..N
                  |       xx ← x_t - h·f(y_t)
                  |       yy ← y_t - h·f(x_t)
                  |       t ← t + 1
                  |       x_t ← xx
                  |       y_t ← yy
                  |       z_t ← x_t + y_t·i
                  | z
```

N := 50

h := 0.05

a := 6

z := popcorn(N,h,a)

Рис. 14.22. Программа построения фрактала «кукуруза»

в Mathcad весьма удачен: модули прекрасно вписываются в документы, выглядят просто и естественно, чего нельзя сказать о программах на обычных языках программирования.

В принципе, в Mathcad есть возможность включения в систему функций пользователя, написанных на языке C или C++. Однако ситуация с этой возможностью очень напоминает нашу крылатую фразу «за что боролись, на то и напоролись» – ведь смысл разработки систем класса Mathcad и заключается в том, чтобы избавить пользователя от программирования на сложных языках высокого уровня. В связи с внедрением в систему Mathcad основных программных конструкций надобность в программировании на языке C++ практически отпала. Точнее, она стала прерогативой «фанатов» системного программирования.

Наконец, следует отметить еще одну важную возможность – применение откомпилированных дополнительных библиотек, расширяющих возможности ядра системы, а также специальных исполняемых модулей, которые обновляют версии системы. Их можно получить у разработчика через Интернет или электронную почту. Эти модули обычно сохраняются в основной папке Mathcad и запускаются как самостоятельные программы. После однократного исполнения они модифицируют текущую версию Mathcad, превращая ее в очередную, более мощную.

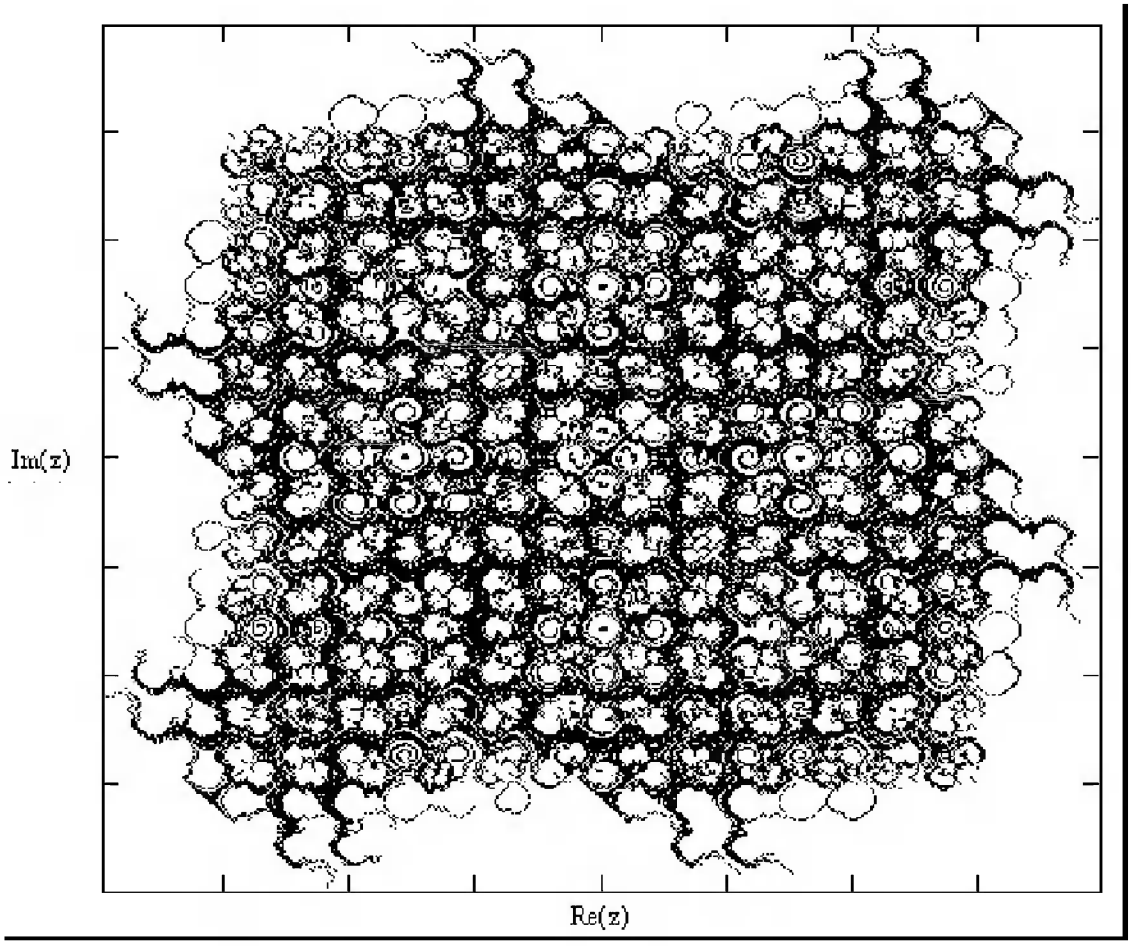


Рис. 14.23. Фрактал «кукуруза», построенный программным модулем с рис. 14.22

Глава 15

Аналитическое и спектральное моделирование

В этой главе приводится полное решение некоторых вполне конкретных учебных и научно-технических задач из области физики, квантовой механики и электрорадиотехники [73–75, 144–146, 158–162]. Эти задачи хорошо иллюстрируют технику решения научно-технических задач с применением прежде всего аналитических вычислений в среде Maple 11. Однако они с равным успехом могут решаться в версиях Maple 9/9.5/10. Приведены примеры решения задач и в других СКМ.

15.1. Исследование и моделирование линейных систем

15.1.1. Демпфированная система второго порядка

Резонансные *LCR*-контуры в электрорадиотехнике, механические маятники и даже молекулы и атомы различных веществ – все это примеры систем второго порядка. Они могут быть линейными и нелинейными, сильно или слабо демпфированными и находящимися в режиме свободных колебаний или под внешним воздействием.

Замечательно то, что огромное число таких систем описывается системой из двух линейных дифференциальных уравнений или одним линейным дифференциальным уравнением второго порядка. Рассмотрим типичную *сильно демпфированную систему* – вне зависимости от ее конкретной реализации (см. рис. 15.1). Из характеристического полинома данного дифференциального уравнения видно, что его корни действительные, что является признаком апериодичности анализируемой системы, и отрицательные, что указывает на затухание собственных колебаний системы.

Дифференциальное уравнение DE представленного вида имеет два параметра – параметр p, определяющий степень демпфирования системы, и параметр q, определяющий резонансную частоту системы. В данном примере в качестве внешнего воздействия используется синусоидальное воздействие (сигнал в радиотехнических системах). Для решения дифференциального уравнения надо задать его начальные условия. Все это и сделано на рис. 15.1. На рис. 15.2 представлены два решения для данного примера – при отсутствии и при наличии воздействия. Обратите внимание на то, что решение при отсутствии воздействия представлено только экспоненциальными членами с отрицательными показателями степени. Это говорит об апериодическом поведении системы и затухании в ней энергии.

График исходного воздействия и реакций системы также представлен на рис. 15.2. Нетрудно заметить, что при p = 3 система выброса ведет себя как типичная апериодическая система – возникшее отклонение уменьшается без колебаний. Однако при наличии воздействия его колебательная компонента появляется в реакции системы.

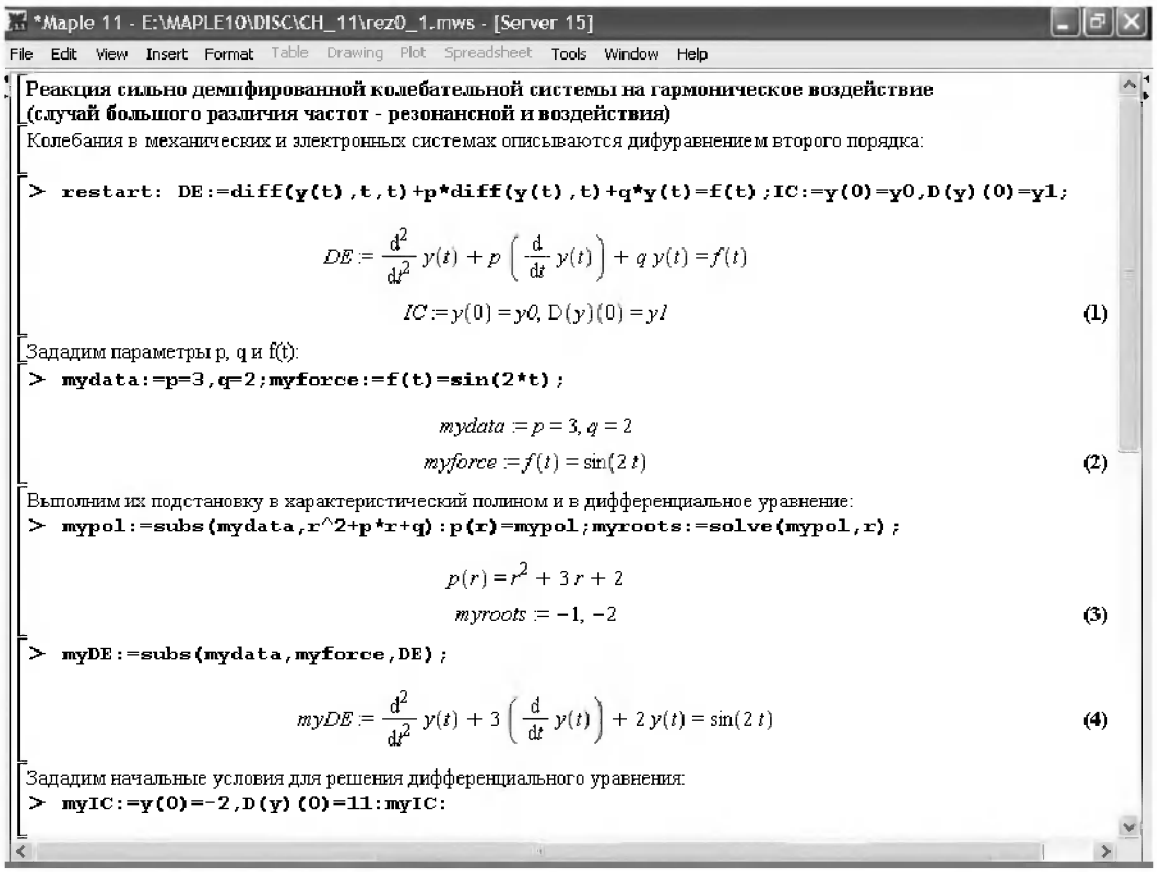


Рис. 15.1. Задание дифференциального уравнения второго порядка для сильно демпфированной системы второго порядка

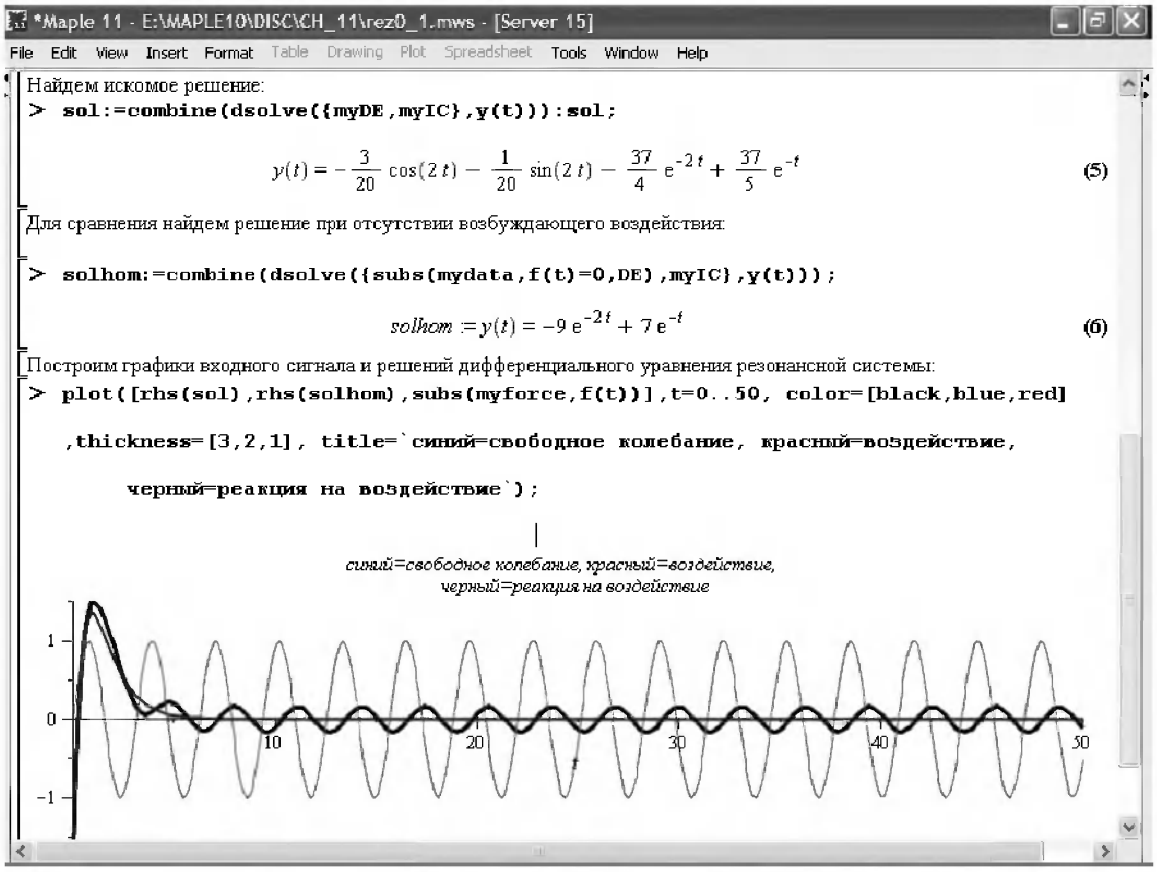


Рис. 15.2. Решение задачи моделирования системы второго порядка при синусоидальном воздействии

15.1.2. Система с малым демпфированием под внешним синусоидальным воздействием

Теперь слегка модернизируем представленный выше документ и зададим параметры p и q, соответствующие слабо демпфированной колебательной системе, – рис. 15.3. Сейчас характеристический полином имеет комплексные корни, что (для знающих теорию колебаний) указывает на колебательный характер поведения системы. Такие системы являются резонансными.

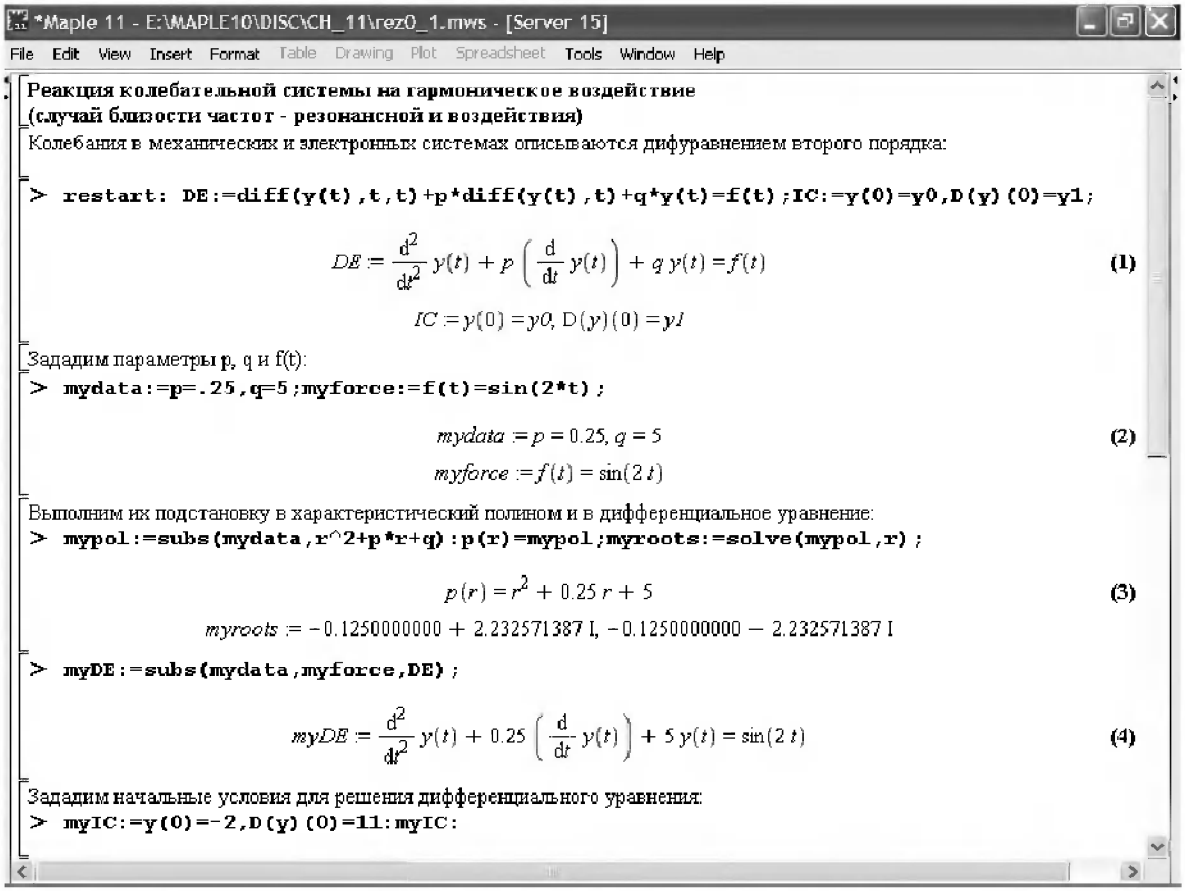


Рис. 15.3. Начало документа с примером моделирования резонансной системы с малым демпфированием при синусоидальном воздействии

Решение составленной системы представлено на рис. 15.4. Хотя мы рассматриваем достаточно простую систему, ее решение выглядит уже не слишком простым, хотя назвать его сложным было бы неверно. Решение как при отсутствии внешнего воздействия, так и при его наличии имеет члены с синусами и косинусами, что явно указывает на наличие периодических компонент решения. Характер решения нагляден из представленных графиков.

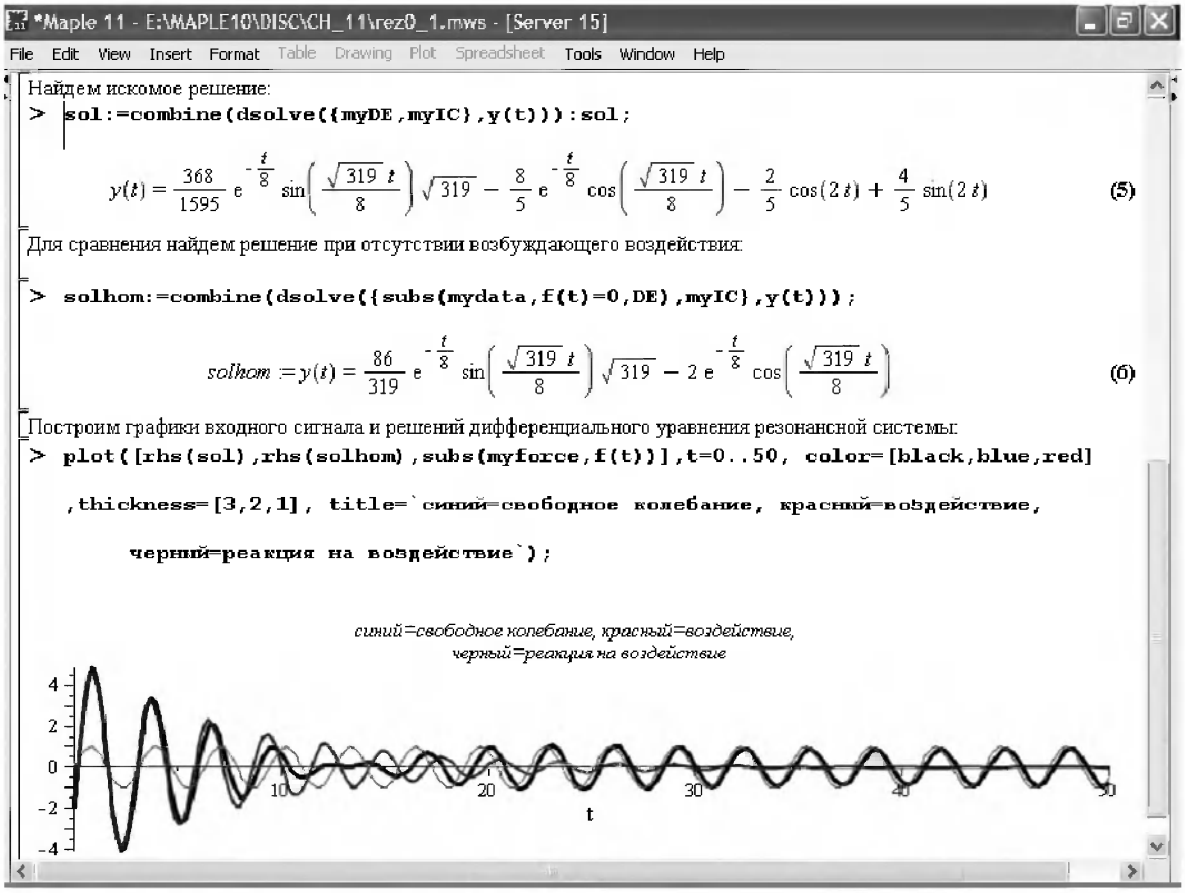


Рис. 15.4. Конец документа с примером моделирования резонансной системы с малым демпфированием при синусоидальном воздействии

15.1.3. Слабо демпфированная система под воздействием треугольной формы

Другим наглядным примером может служить анализ поведения системы при воздействии на нее треугольных колебаний. Как было уже показано, такие колебания легко получить, задав их функцией arcsin(sin(x)). Это и показано на рис. 15.5.

Конец документа рис. 15.5 представлен на рис. 15.6. Здесь прежде всего стоит обратить внимание на аналитическое решение задачи. Назвать его простым язык уже не поворачивается, хотя решение занимает на экране всего две строки (нередки случаи, когда решение имеет десятки-сотни строк). Любопытно, что в решение входит даже определенный интеграл. А это указывает уже на то, что время вычислений может значительно возрасти из-за вычисления интеграла численными методами.

Решение системы при отсутствии внешнего воздействия здесь не приводится, поскольку оно абсолютно идентично представленному на рис. 15.4. А вот график общего решения весьма показателен. Так, видно, что благодаря резонансу форма выходных колебаний остается синусоидальной. Но гораздо сильнее, чем на рис. 15.4, видны биения с разностной частотой. Впрочем, что уже заметно и на рис. 15.6, они затухают, так что в стационарном режиме сигнал на выходе представляет собой синусоидальную функцию с частотой внешнего воздействия.

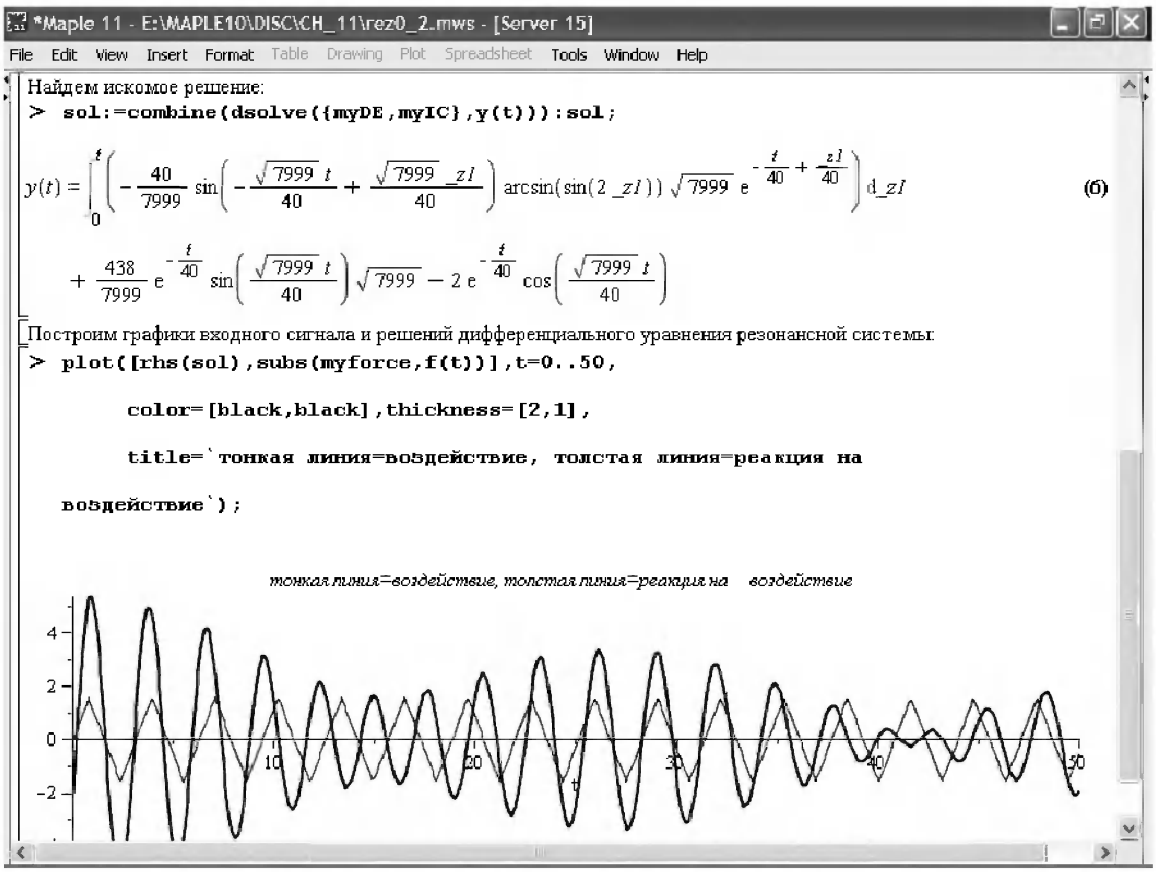


Рис. 15.5. Начало документа с примером моделирования резонансной системы с малым демпфированием при треугольном воздействии

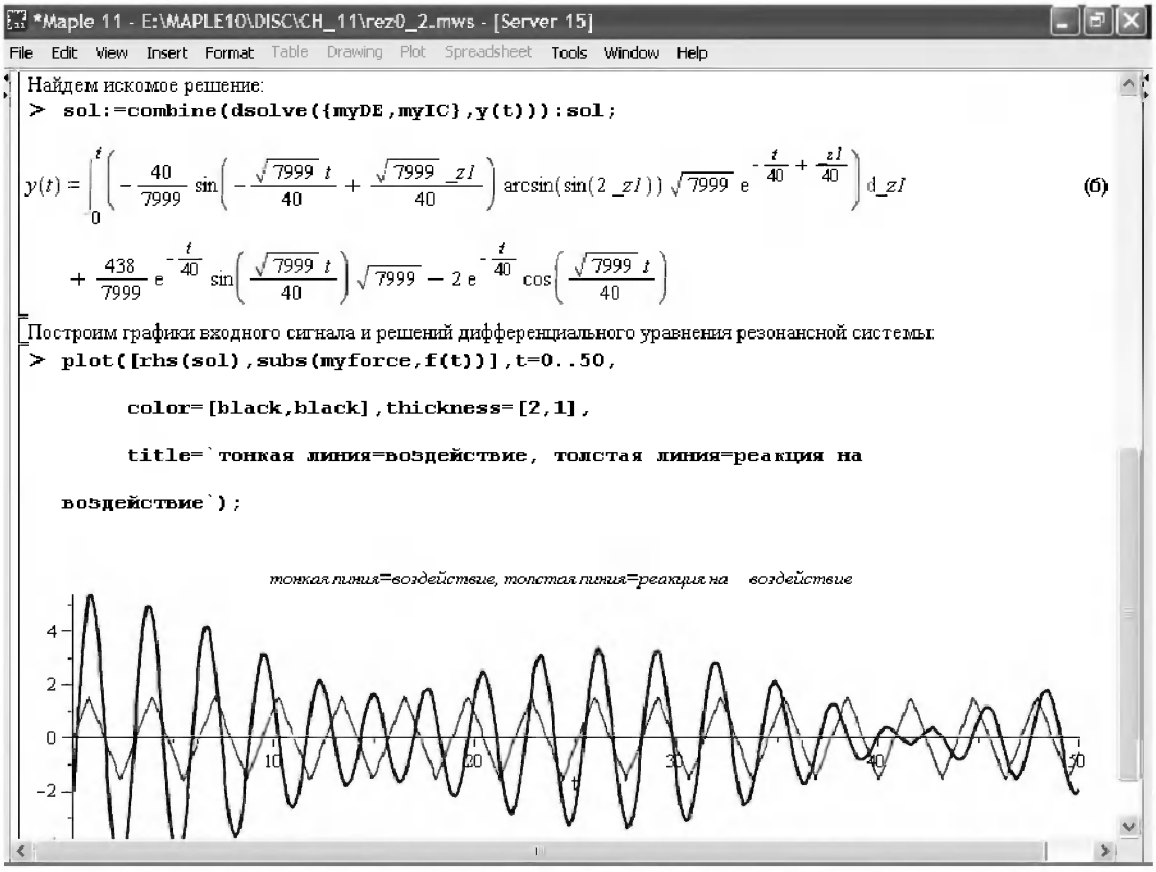


Рис. 15.6. Конец документа с примером моделирования резонансной системы с малым демпфированием при треугольном воздействии

15.1.4. Слабо демпфированная система при произвольном воздействии

При произвольном воздействии ожидать возможности аналитического решения, скорее всего, уже не приходится. В качестве примера рассмотрим решение задачи на поведение *резонансной системы* при воздействии на нее прямоугольных импульсов. Чтобы упростить записи выражений, будем считать импульсы нормированными, то есть пусть их амплитуда будет равна π, длительность тоже равна π, и период 2π. Такие импульсы можно задать, используя соотношение signum(sin(x)) и выполнив указанное выше нормирование. Это и показано на рис. 15.7.

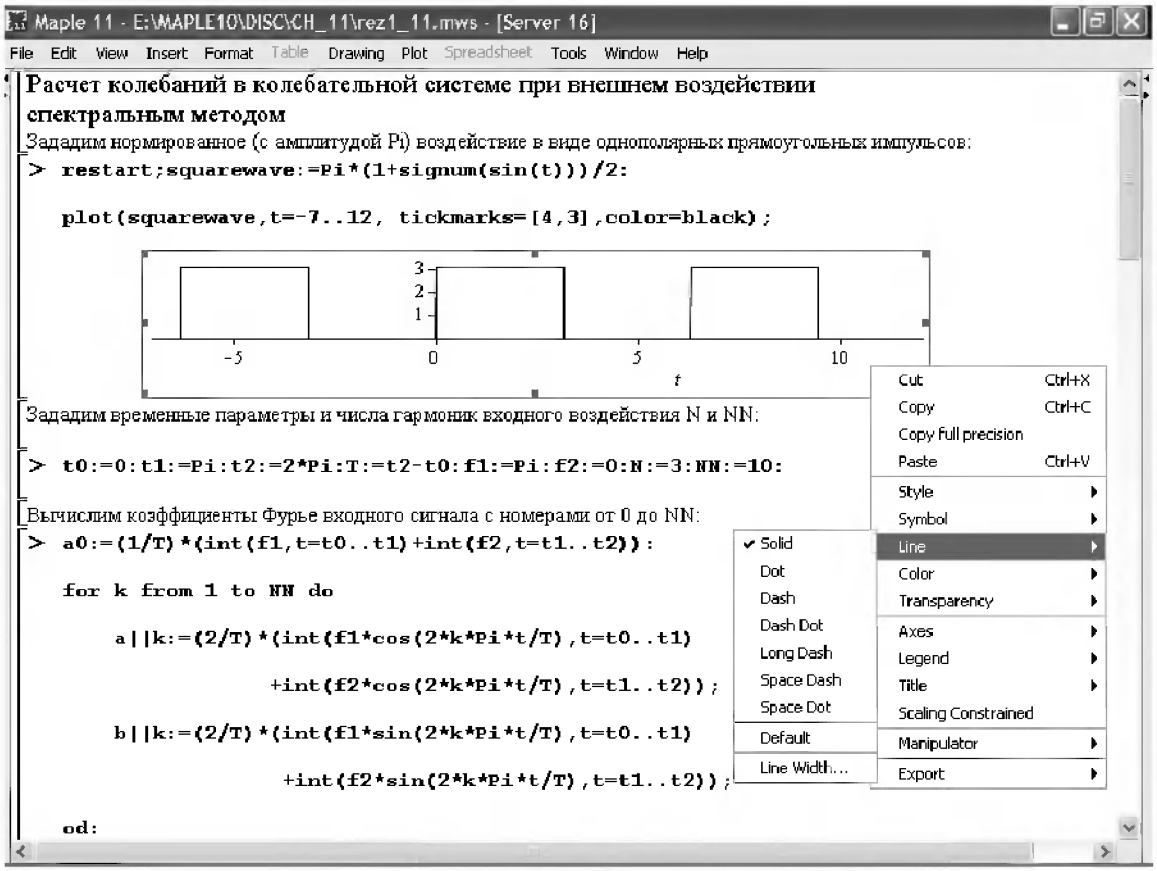


Рис. 15.7. Представление входного сигнала рядом Фурье

К сожалению, сформированный таким образом сигнал нельзя считать строго аналитическим, поскольку функция signum не относится ни к элементарным, ни к специальным математическим функциям. А потому желающие могут это легко проверить: такой сигнал не может стоять в правой части дифференциального уравнения, поскольку оно в этом случае аналитически не решается и просто повторяется в строке вывода.

Однако подобный сигнал, как и множество других сигналов, может быть представлен своим разложением в ряд Фурье или просто синтезирован рядом гармоник, что и показано на рис. 15.7. В нем задано построение сигнала с числом гармоник $N = 3$ и $NN = 10$ и заданы коэффициенты a_k и b_k ряда Фурье. Заметим, что,

поставив после оператора od точку с запятой вместо двоеточия, можно вывести значения этих коэффициентов.

Рисунок 15.8 показывает Фурье-синтез приближенного входного сигнала для 3 и 10 гармоник, а также построение сигнала вместе с идеальным (исходным) сигналом. Нетрудно заметить, что из-за эффекта Гиббса полученный сигнал (особенно при трех гармониках) довольно сильно отличается от идеального. Однако не стоит забывать, что резонансная система эффективно гасит все колебания, за исключением того, которое имеет частоту, близкую к резонансной частоте.

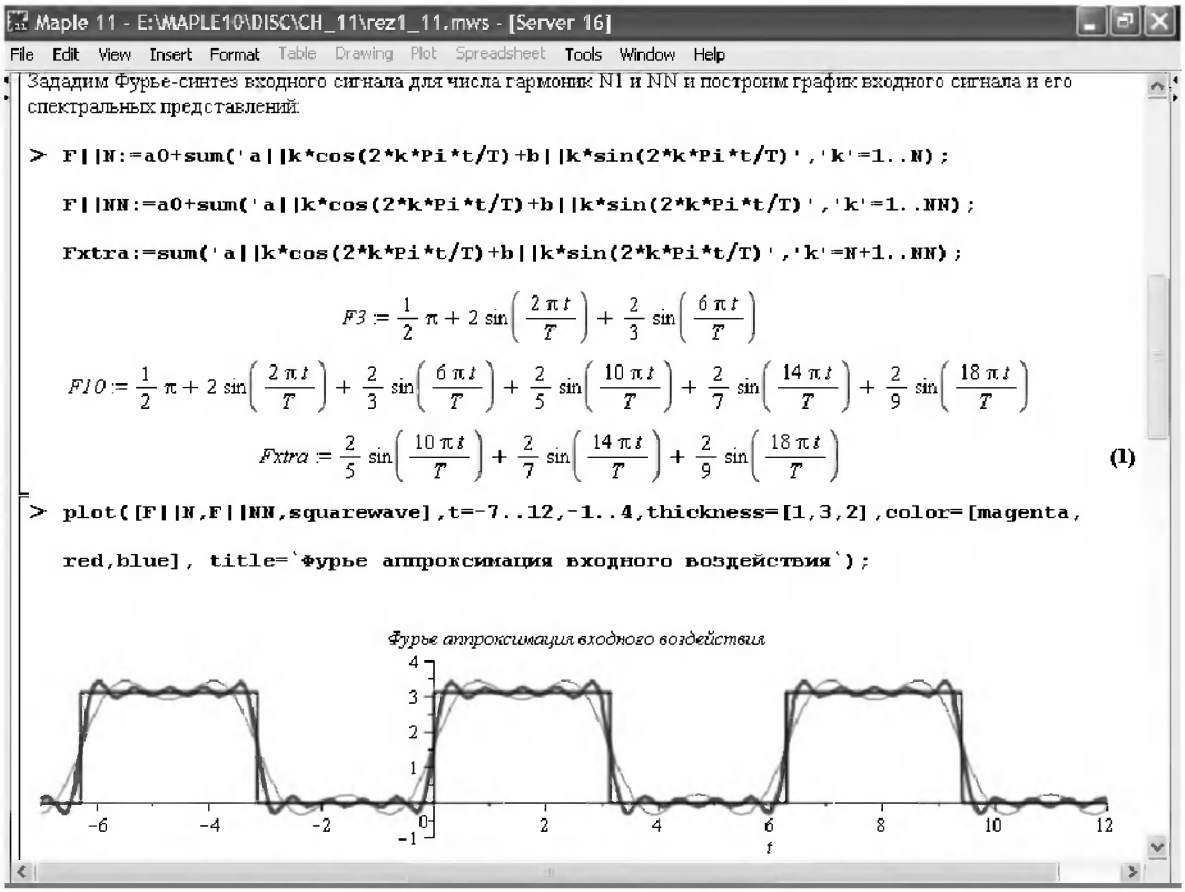


Рис. 15.8. Синтез приближенного сигнала и его сравнение с идеальным сигналом

Хотя временные зависимости сигнала, показанные на рис. 15.8, могут показаться сложными, на самом деле это всего лишь суммы синусоидальных колебаний кратной частоты. К тому же представлены только нечетные гармоники. Но главное – этот сигнал уже имеет аналитическое представление, и его можно использовать в правой части дифференциального уравнения. Это и показано на рис. 15.9, где заданы дифференциальные уравнения для рассмотренных выше случаев.

Теперь остается решить представленные дифференциальные уравнения и получить графики решений, представленные на рис. 15.10. В данном случае частоты сигнала и собственных колебаний системы заметно различаются, и выходной сигнал системы представляет собой в основном выделенную вторую гармонику воздействия.

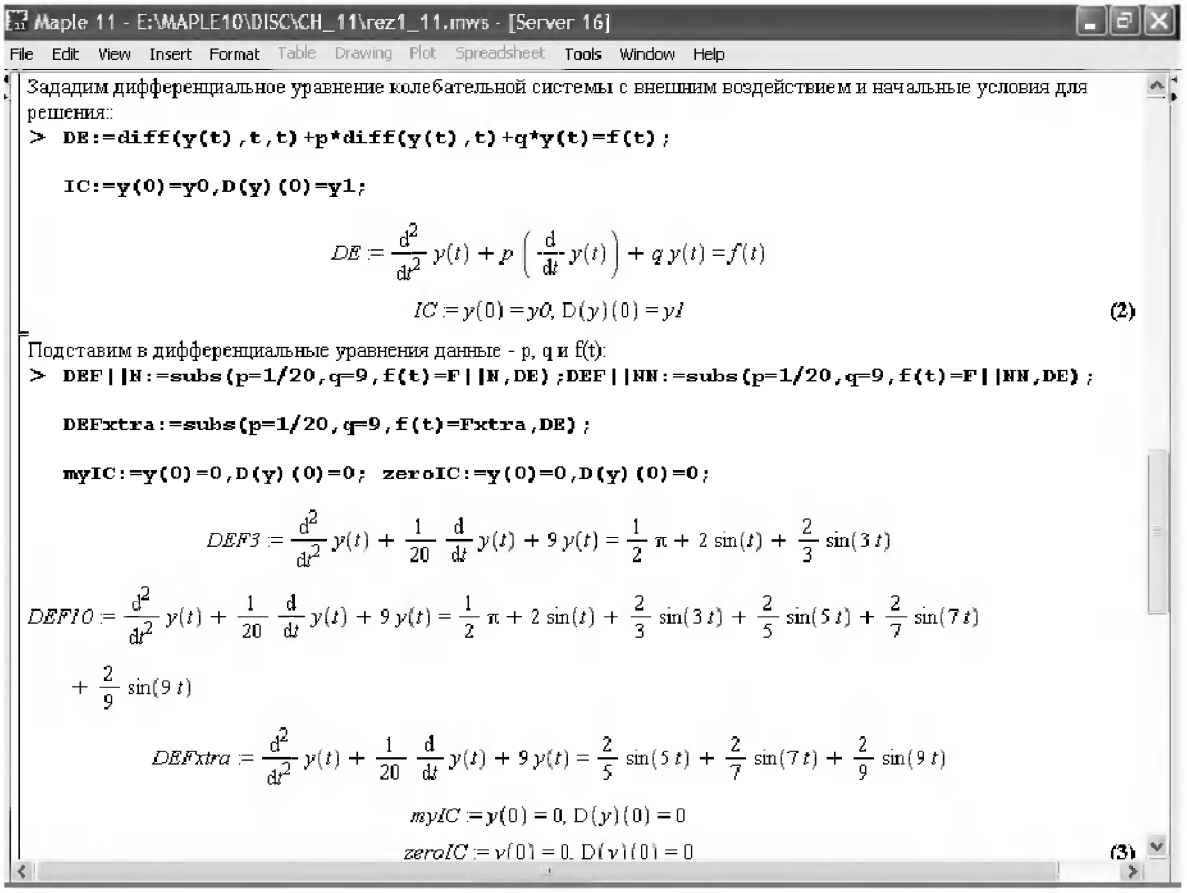


Рис. 15.9. Составление дифференциального уравнения второго порядка и задание начальных условий

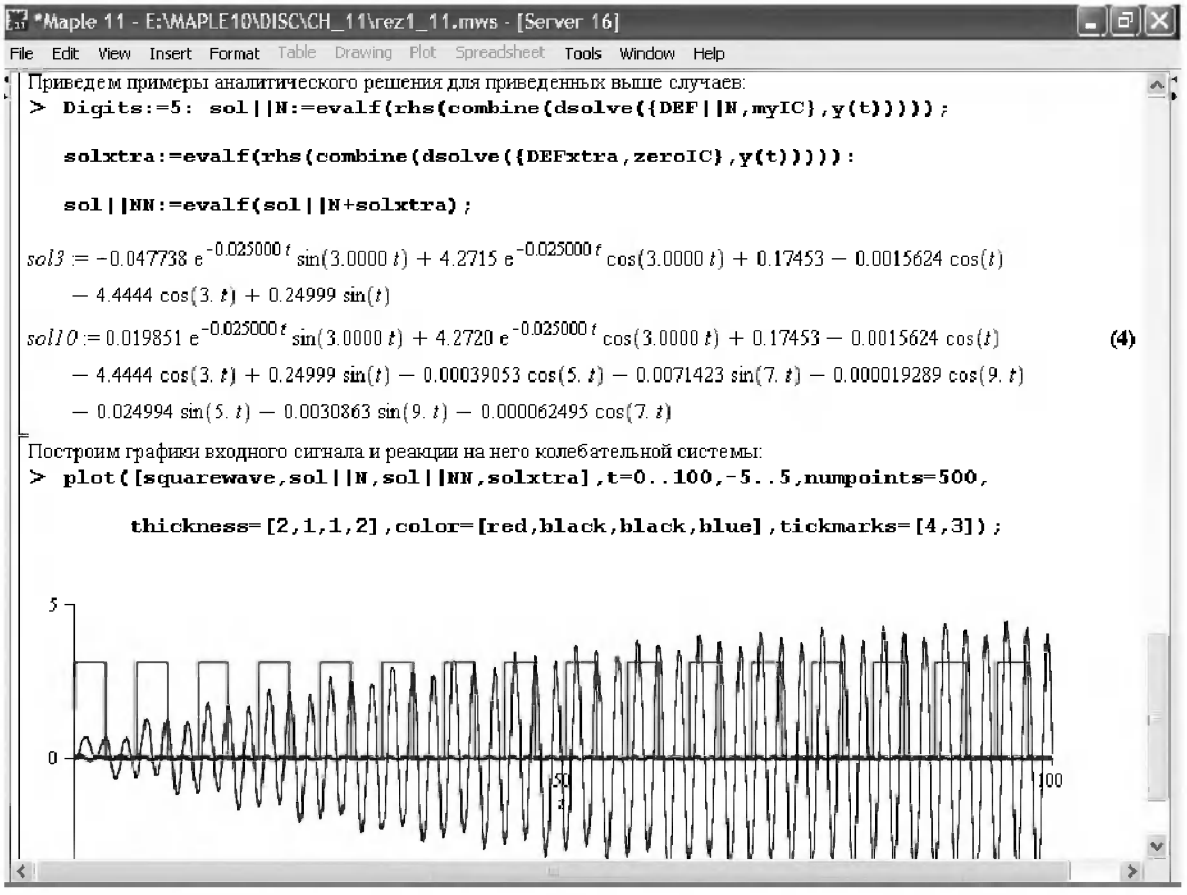


Рис. 15.10. Решение дифференциальных уравнений и его визуализация

Разумеется, представленный вариант анализа носит частный характер, поскольку синтезируется вполне конкретный вид сигнала – прямоугольные импульсы с заданными выше параметрами. Однако если использовать разложение в ряд Фурье произвольного воздействия, то подобным способом можно решить задачу получения реакции любой линейной системы на заданное воздействие.

15.1.5. Улучшенное моделирование свободных колебаний

Вернемся к задаче моделирования системы второго порядка и попытаемся найти решения в более удобном виде, обычно приводимом в учебниках. Для этого достаточно воспользоваться пакетом расширения DEtools. Рисунок 15.11 показывает начало документа с составленным дифференциальным уравнением и его решением. Нетрудно заметить, что теперь решение представлено в классическом виде, который обычно приводится в учебниках по теории колебаний.

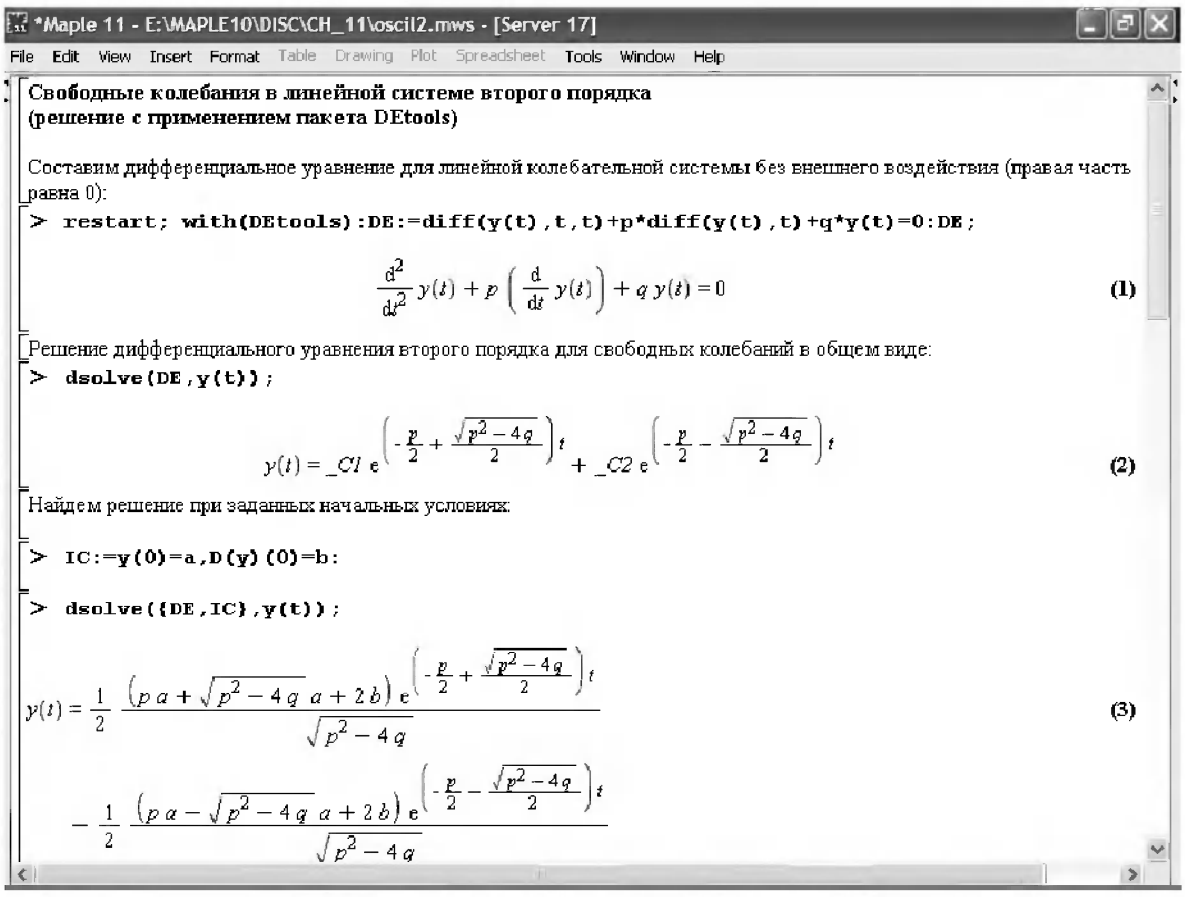


Рис. 15.11. Решение дифференциального уравнения свободных колебаний с применением пакета DEtools

На рис. 15.12 показана вторая часть документа с решением для конкретных данных и построением графика временной зависимости свободных колебаний. Нетрудно заметить, что свободные колебания системы имеют вид затухающих

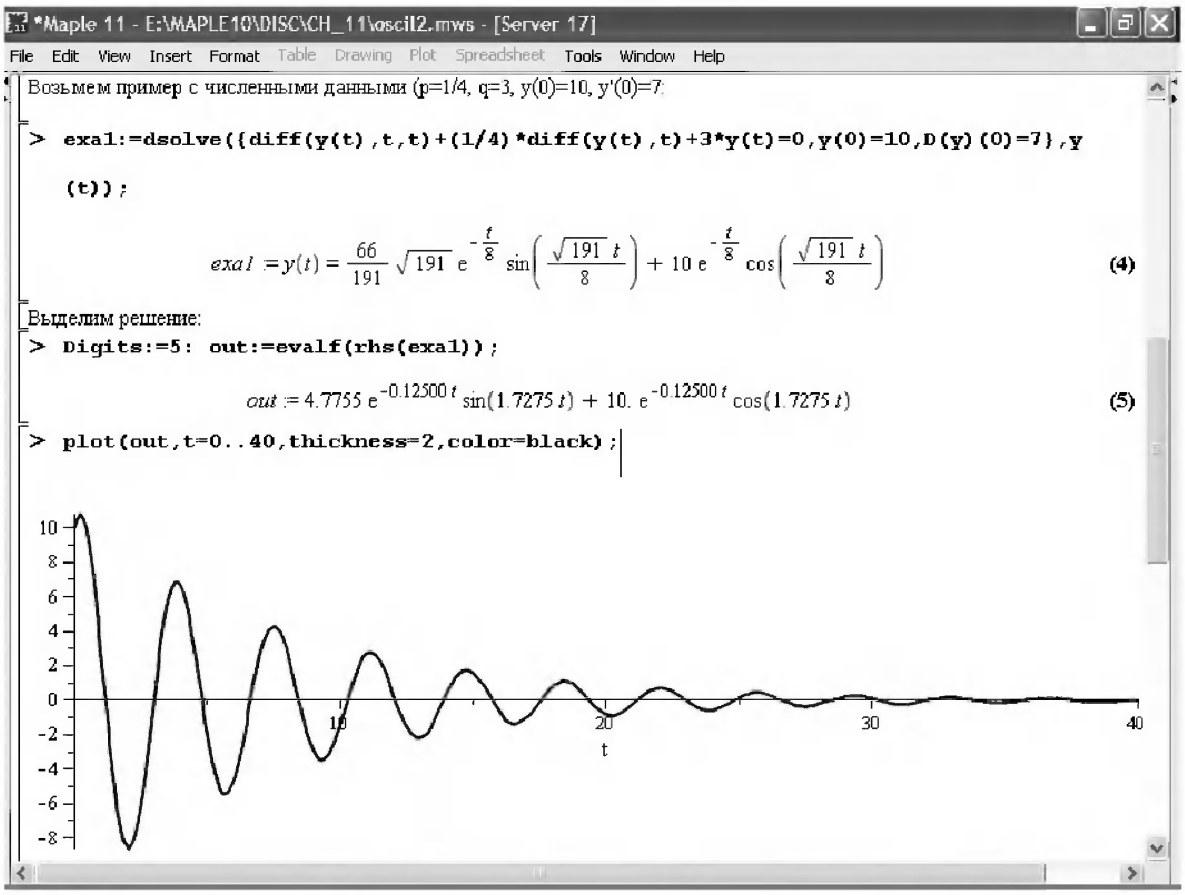


Рис. 15.12. Пример вычисления временной зависимости свободных колебаний и построения их графика

синусоидальных колебаний. Вы можете проверить, что при $p < 0$ колебания будут нарастать по экспоненциальному закону, что характерно для генераторных систем.

Нередко о характере колебаний удобно судить по фазовому портрету колебаний. Он задается графиком в параметрической форме, при которой по одной оси откладывается зависимость $y(t)$, а по другой – ее производная. Это показано на рис. 15.13. Фазовый портрет в данном случае представляет собой сворачивающуюся спираль.

15.1.6. Улучшенное моделирование колебаний при синусоидальном воздействии

По аналогии с последним примером можно рассмотреть поведение системы второго порядка при синусоидальном воздействии. На рис. 15.14 представлено начало документа, в котором задано исходное дифференциальное уравнение и получено его общее и частное аналитические решения.

На рис. 15.15 представлены временные диаграммы реакции системы и синусоидального воздействия. Кроме того, построен фазовый портрет колебаний. Он заметно отличается от спирали и хорошо иллюстрирует сложность колебаний в начале их развития.

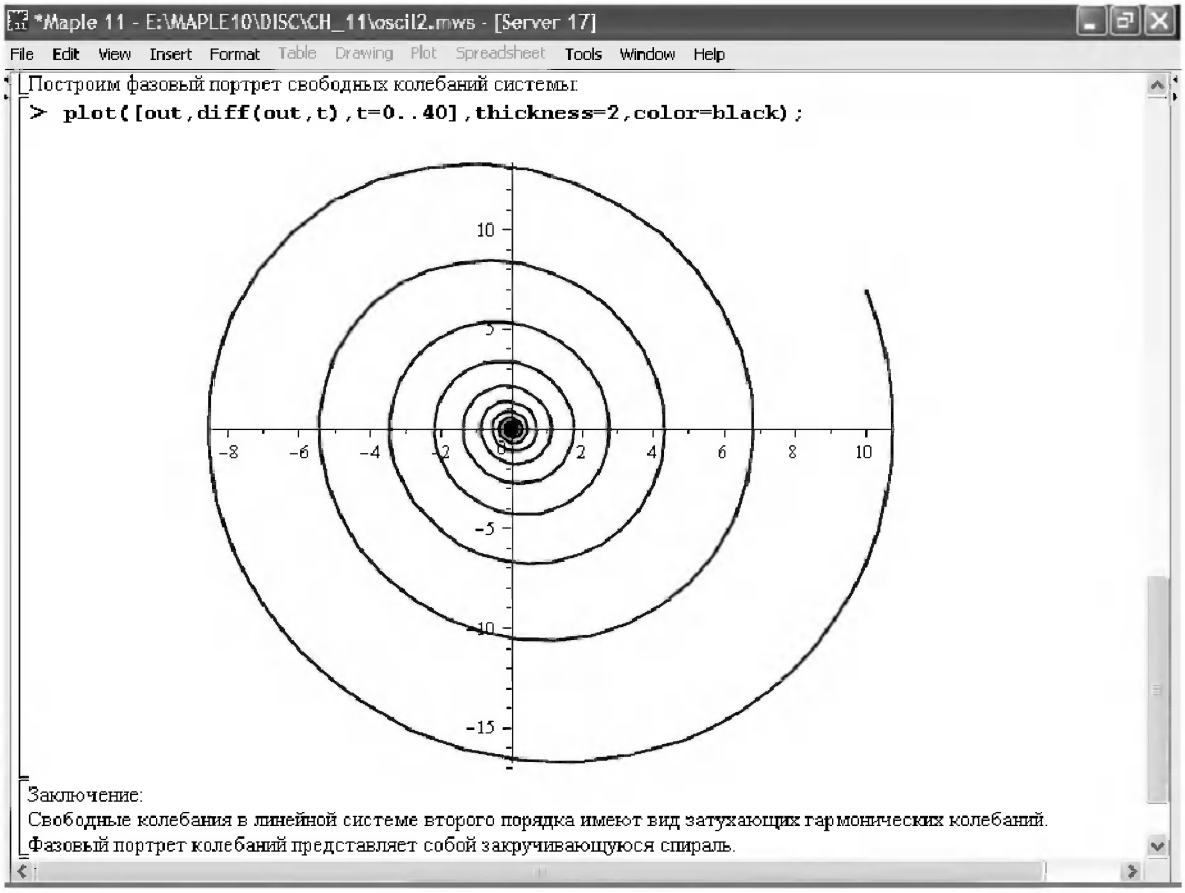


Рис. 15.13. Фазовый портрет затухающих свободных колебаний

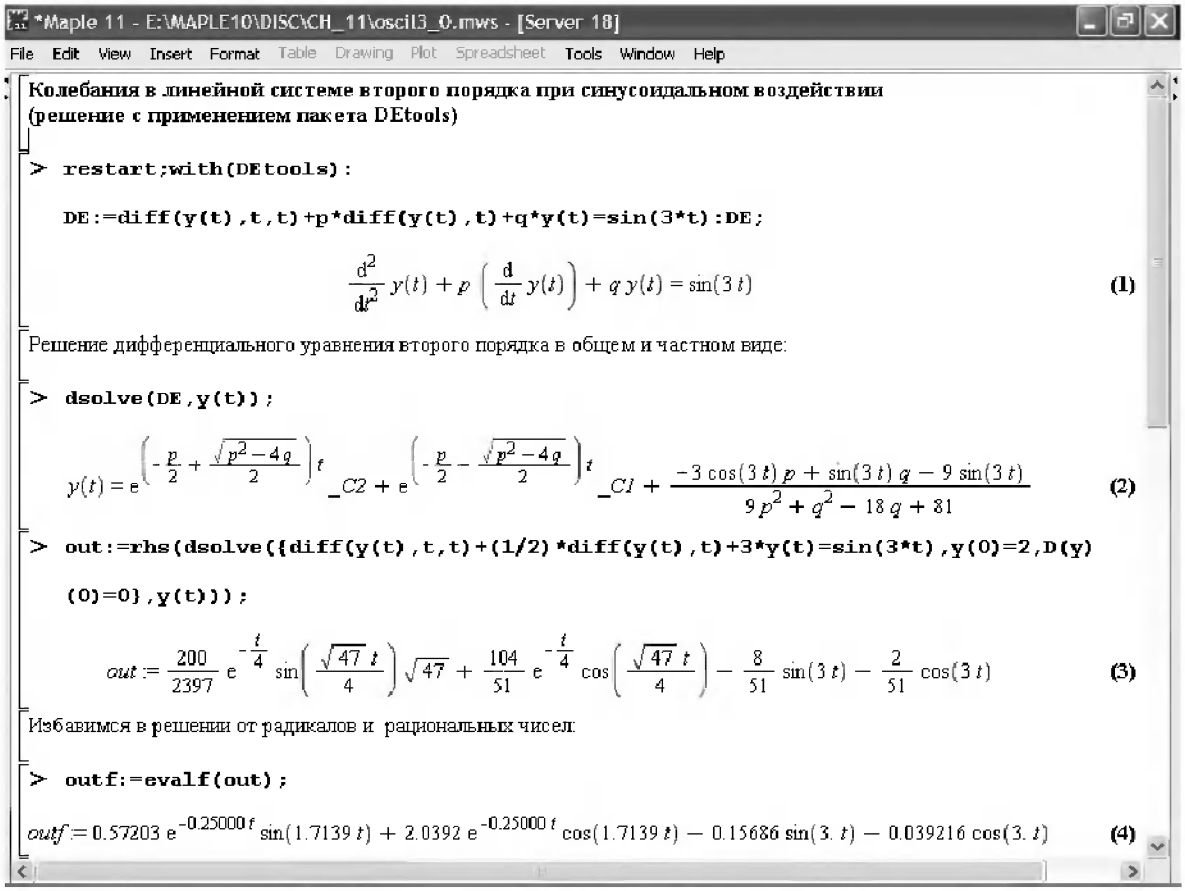


Рис. 15.14. Пример аналитического решения задачи на поведение системы второго порядка при синусоидальном воздействии

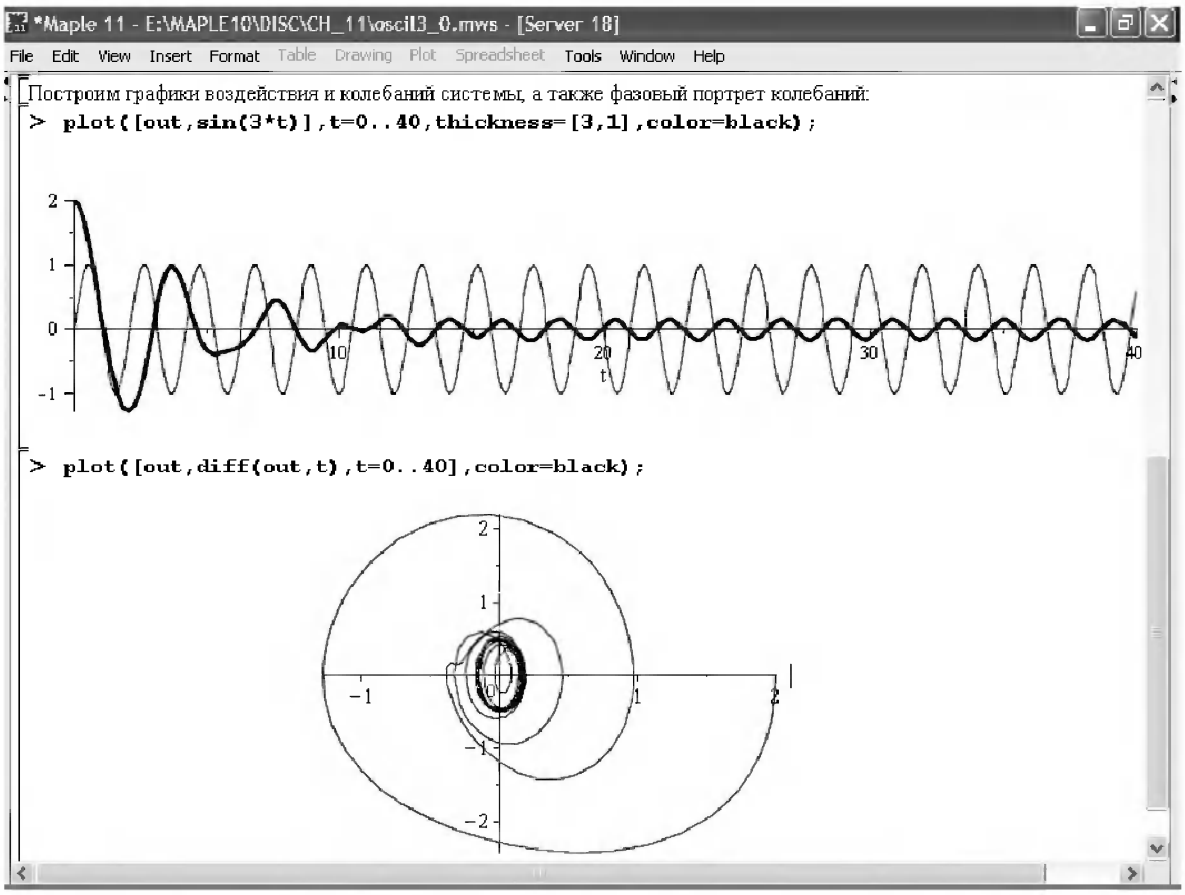


Рис. 15.15. Результаты моделирования цепи второго порядка при синусоидальном воздействии

К сожалению, применение пакета расширения DEtools усложняет фунцию dsolve решения дифференциальных уравнений. В результате время моделирования даже простых систем удлиняется до минут, а более сложные системы могут потребовать куда более длительного времени моделирования. В этом случае может оказаться целесообразным отказаться от получения аналитических зависимостей для результатов моделирования и перейти к численному моделированию.

15.1.7. Улучшенное моделирование колебаний при пилообразном воздействии

Рассмотрим методику улучшенного моделирования еще на одном примере – вычислении реакции системы при пилообразном воздействии. На рис. 15.16 показано задание такого воздействия с помощью функции `floor`. Для упрощения расчетных выражений амплитуда и период воздействия взяты равными π. Поскольку в данном случае аналитическое решение получить невозможно (функция `floor` не позволяет этого), то заменим воздействие рядом Фурье. Его коэффициенты также представлены на рис. 15.16.

На рис. 15.17 представлены графики воздействия в идеальном случае и при его представлении рядом Фурье с пятью гармониками. Показано также аналитическое решение для временной зависимости $y(t)$ при таком воздействии.

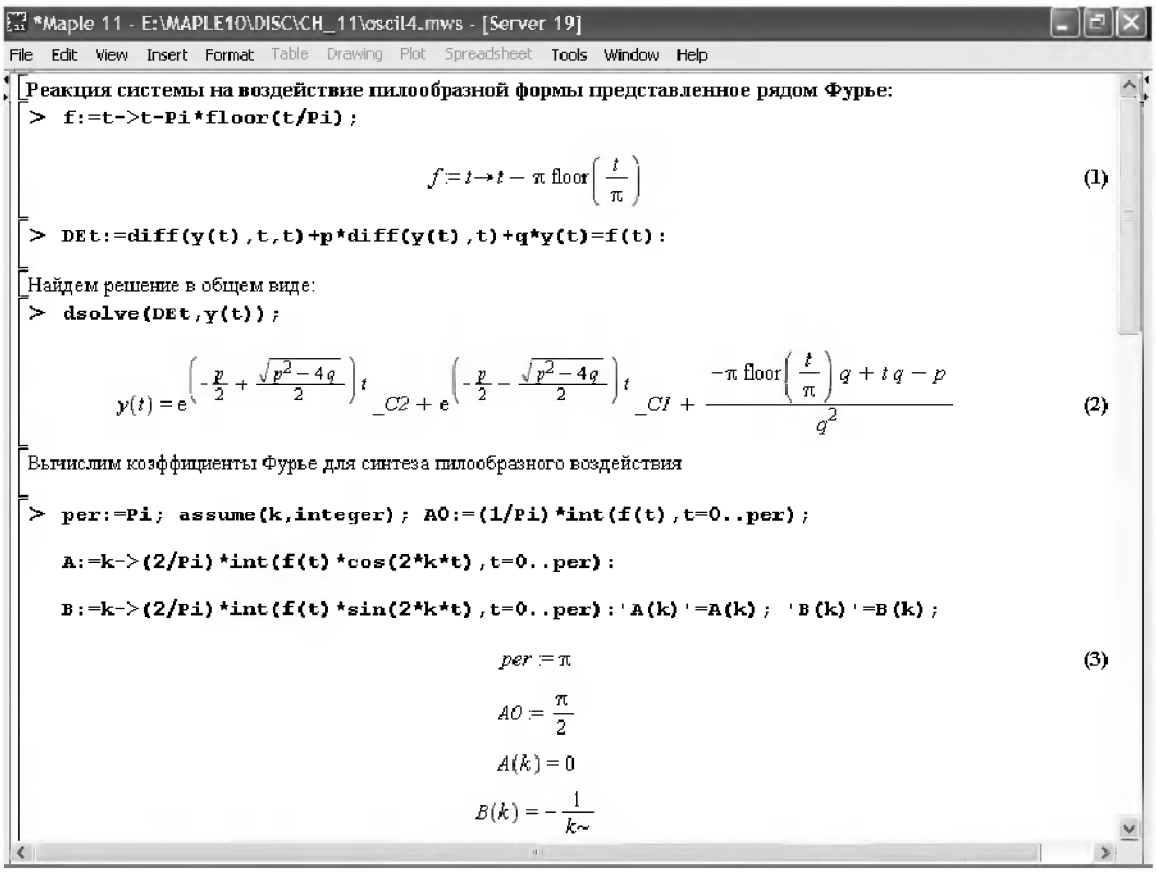


Рис. 15.16. Начало моделирования системы с пилообразным воздействием, представленным рядом Фурье

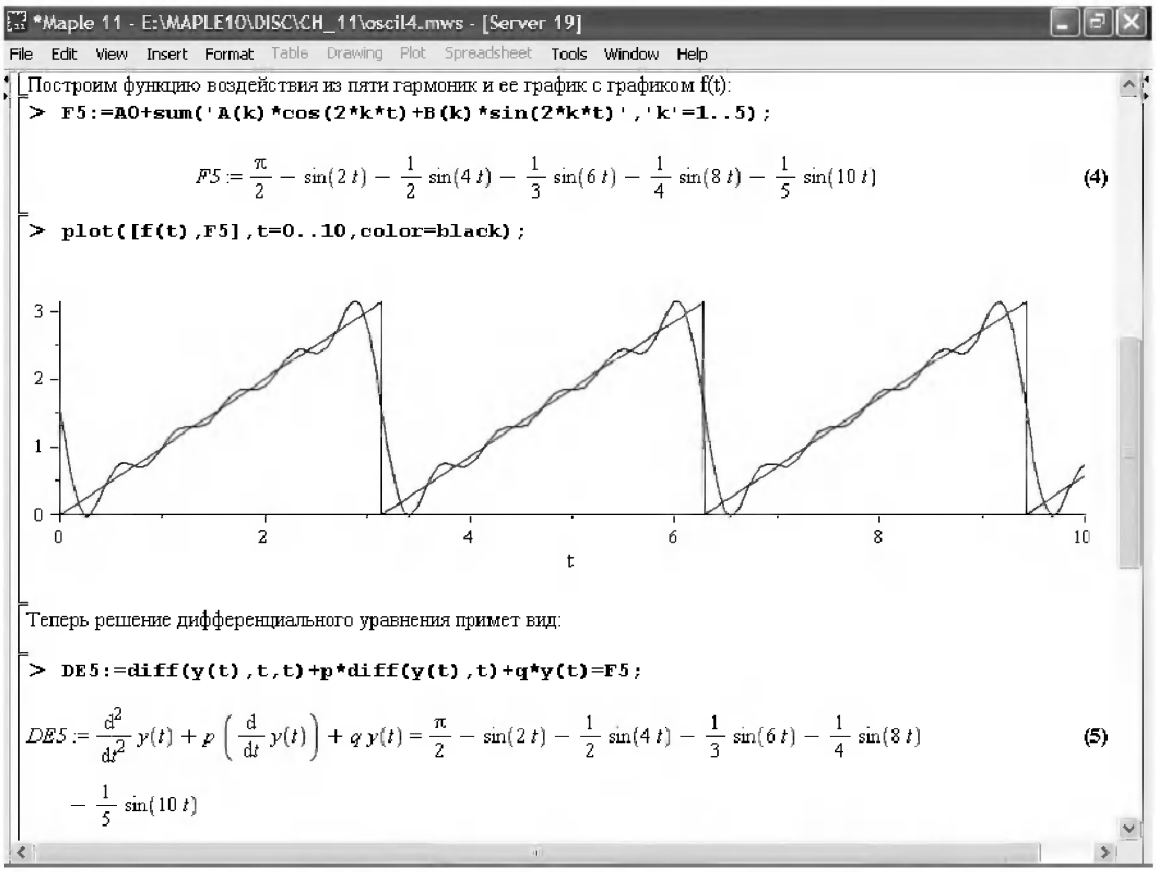


Рис. 15.17. Воздействие и временная зависимость реакции системы при пилообразной форме воздействия

Наконец, на рис. 15.18 показаны график реакции системы на пилообразное воздействие и фазовый портрет колебаний в ней. Нетрудно заметить, что форма воздействия достаточно слабо влияет на форму временной зависимости реакции системы на заданное воздействие. Это следствие резонансных свойств системы.

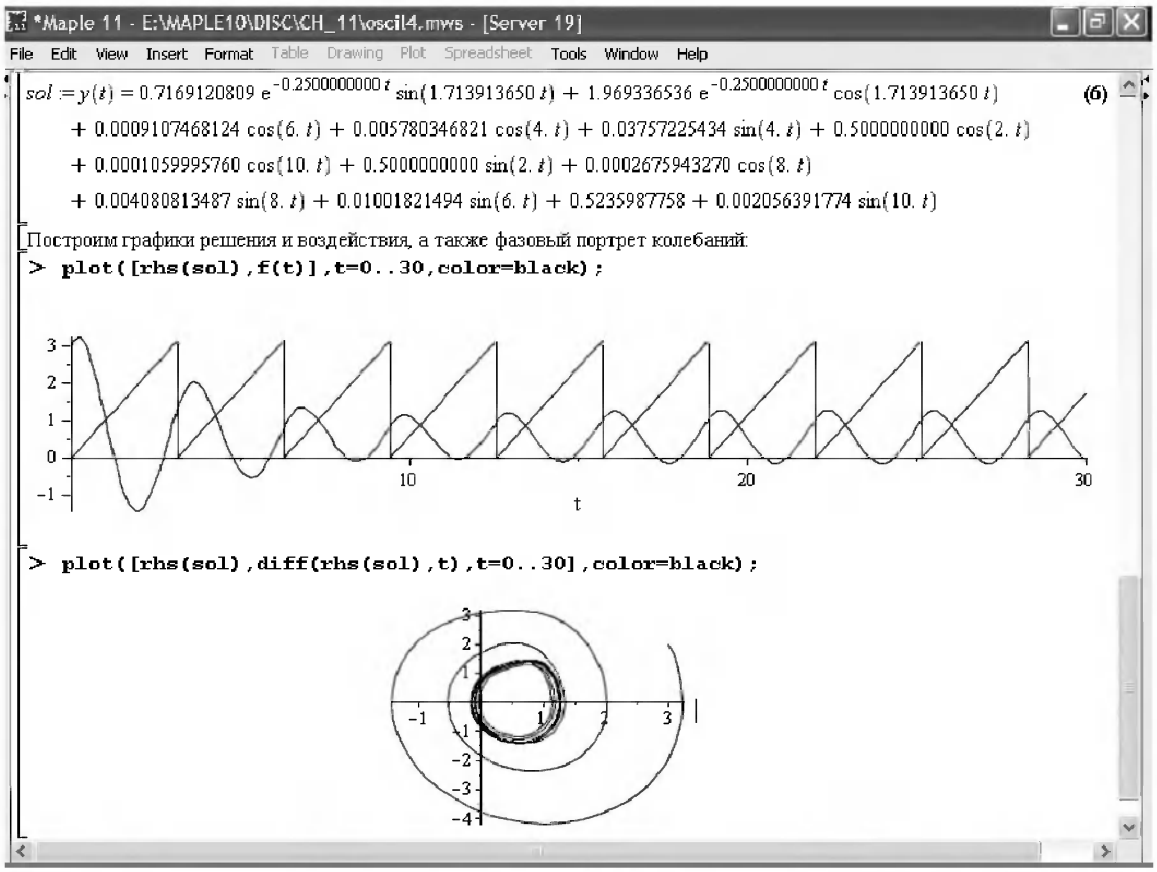


Рис. 15.18. Реакция системы на пилообразное воздействие и фазовый портрет колебаний при таком воздействии

Нелинейные системы второго порядка, к сожалению, не имеют общих аналитических решений, и для моделирования таких систем следует использовать численные методы решения дифференциальных уравнений.

15.1.8. Анализ и моделирование линейных систем операторным методом

Произвольные линейные системы могут анализироваться и моделироваться хорошо известным (особенно в электротехнике и радиотехнике) операторным методом. При этом методе система и ее воздействие представляются операторными выражениями, то есть в виде функций параметра – оператора Лапласа s (в литературе встречается и обозначение p). Не вникая в детали этого общеизвестного метода, рассмотрим конкретный пример. Он, для сравнения с предшествующими примерами, дан для системы второго порядка, хотя в данном случае никаких ограничений на порядок системы нет.

Для начала зададим инициализацию применяемых пакетов расширения

```
> restart:with(plots): readlib(spline): with(inttrans):
Warning, the name changecoords has been redefined
```

Далее зададим операторные выражения для коэффициента передачи системы G и входного сигнала R (в виде единичного перепада) и вычислим с упрощением их произведение:

```
> G := K/(M*s^2+C*s+1); R := 1/s;
```

$$G := \frac{K}{Ms^2 + Cs + 1}$$

$$R := \frac{1}{s}$$

```
> X := simplify(R*G);
```

$$X := \frac{K}{s(Ms^2 + Cs + 1)}$$

Теперь, используя обратное преобразование Лапласа, найдем временную зависимость реакции системы в аналитическом (что наиболее ценно) виде:

```
> h := simplify(invlaplace(X,s,t));
```

$$h := -K\left(-\sqrt{C^2-4M} + \mathbf{e}^{\left(-\frac{tC}{2M}\right)}\cosh\left(\frac{t\sqrt{C^2-4M}}{2M}\right)\sqrt{C^2-4M}\right.$$

$$\left. + \mathbf{e}^{\left(-\frac{tC}{2M}\right)}C\sinh\left(\frac{t\sqrt{C^2-4M}}{2M}\right)\right)/\sqrt{C^2-4M}$$

Теперь мы можем построить график этой зависимости для конкретных значений M, C и K:

```
> h1 := subs(M=1,C=0.75,K=1,xt);
```

$$h1 := 0.5393598900I(-1.854049622I + 1.854049622I\mathbf{e}^{(-0.3750000000t)}\cosh(0.9270248110It) + 0.75\mathbf{e}^{(-0.3750000000t)}\sinh(0.9270248110It)$$

```
> linresp := plot(h1, t=0..20, axes=boxed, color=black):
display(linresp);
```

Вид этой зависимости представлен на рис. 15.19. Он соответствует реакции системы второго порядка для случая затухающих колебаний.

А теперь зададимся целью наглядно проиллюстрировать изменение временной зависимости реакции системы при изменении параметра C от 0 до 2 при M=1 и K=1. Для этого выполним следующие вполне очевидные команды:

```
> x := subs(M=1, K=1, h);
```

$$x := -\left(-\sqrt{C^2-4} + \mathbf{e}^{\left(-\frac{tC}{2}\right)}\cosh\left(\frac{t\sqrt{C^2-4}}{2}\right)\sqrt{C^2-4}\right.$$

$$\left. + \mathbf{e}^{\left(-\frac{tC}{2}\right)}C\sinh\left(\frac{t\sqrt{C^2-4}}{2}\right)\right)/\sqrt{C^2-4}$$

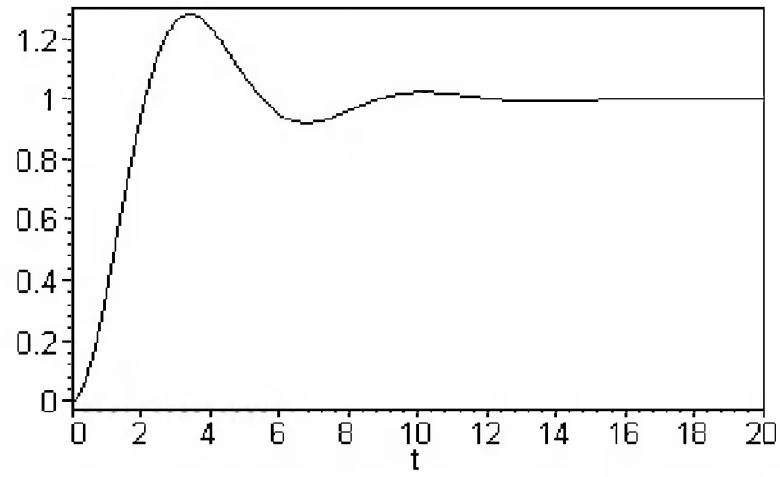


Рис. 15.19. Одна из временных зависимостей реакции системы второго порядка

```
> plot3d(x, C=0..2, t=0..20, axes=boxed);
```

Соответствующий график показан на рис. 15.20. Он прекрасно иллюстрирует переход от апериодического режима при C=2 к колебательному при C=0 при изменении времени от 0 до 20.

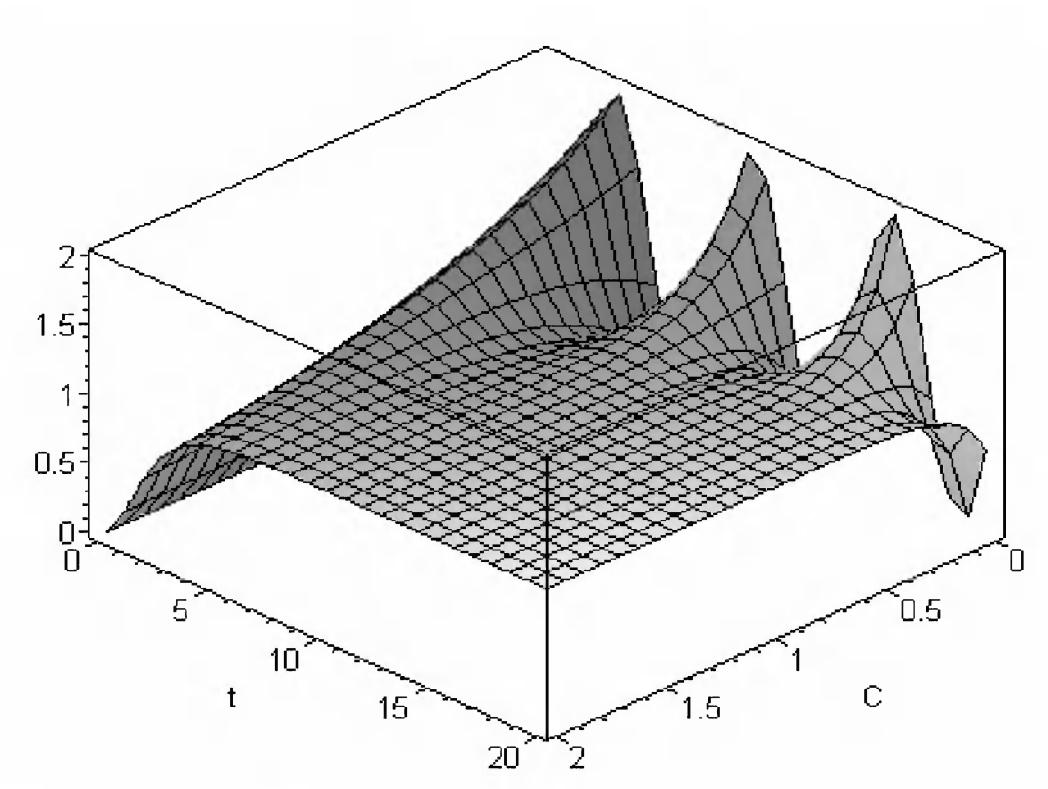


Рис. 15.20. Динамика развития колебаний в системе при изменении параметра C

Аналогичным образом можно построить трехмерный образ временной зависимости реакции системы для M=1, C=0.25 при изменении параметра K от 0 до 3. Для этого надо выполнить команды:

```
> x1 := subs(M=1, C=0.25, xt);
```

$$x1 := 0.5039526307IK(-1.984313483I + 1.984313483Ie^{(-0.1250000000t)}\cosh(0.9921567415It) + 0.25e^{(-0.1250000000t)}\sinh(0.9921567415It)$$

```
> plot3d(x1, K=0..3, t=0..20, axes=boxed);
```

Диаграмма временных зависимостей представлена на рис. 15.21.

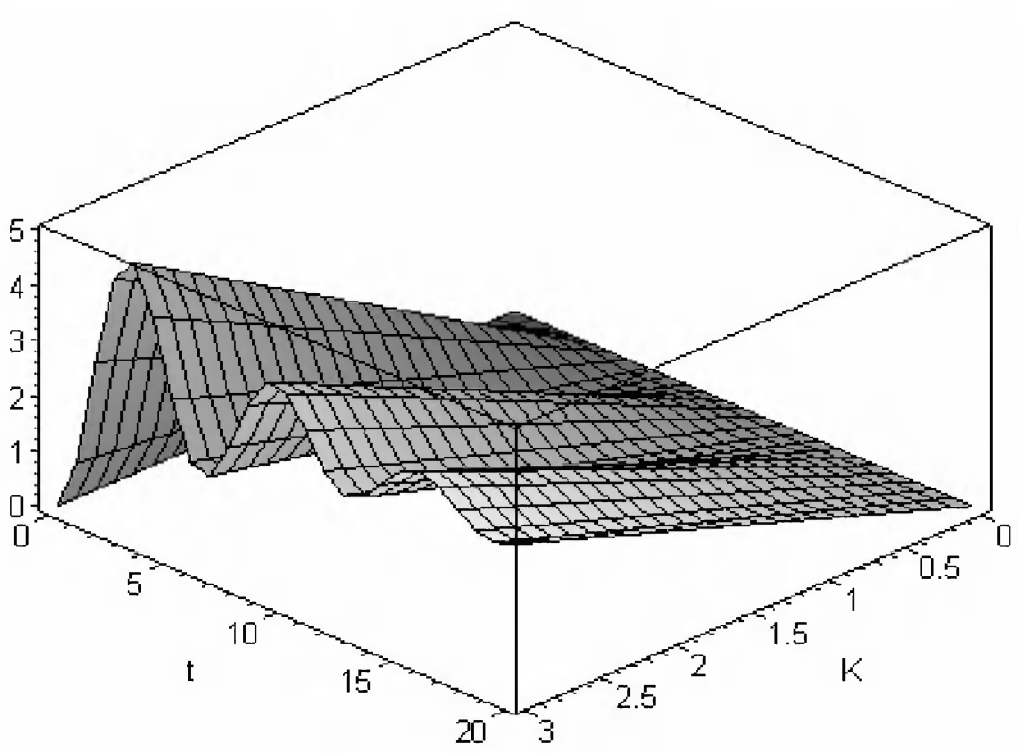


Рис. 15.21. Динамика развития колебаний в системе при изменении параметра K

Представленные на рис. 15.20 и 15.21 диаграммы дают весьма наглядное представление о динамике поведения рассмотренной системы. Но еще важнее то, что просто изменением операторной записи G и R по описанной методике можно анализировать и наглядно представлять работу множества линейных систем.

15.2. Моделирование динамических физических задач

15.2.1. Расчет траектории камня с учетом сопротивления воздуха

Вы хотите метнуть камень в огород вашего вредного соседа? Разумеется, во время его отсутствия. Давайте промоделируем эту ситуацию, предположив два актуальных случая: дело происходит на Земле в условиях, когда наша планета лишилась

воздуха и когда, слава богу, он все же есть. В первом случае сопротивления воздуха нет, а в другом сопротивление воздуха есть – и его надо учитывать. Иначе камень упадет в ваш огород, а не в огород соседа!

Учет сопротивления воздуха не просто усложняет нашу задачу – он делает ее нелинейной. В связи с этим мы применим численные методы решения дифференциальных уравнений. Кроме того, учитывая громоздкость документов, описывающих приведенные ниже задачи, перейдем к их записи прямо в тексте книги.

Итак, пусть подвернувшиеся под руку камни с массой 500 и 100 г брошены под углом 45° к горизонту со скоростью Vo=20 м/с. Найдем их баллистические траектории, если сила сопротивления воздуха Fтр=A*V, где A=0,1 Н*с/м. Сравним их с траекториями, получающимися без учета сопротивления воздуха.

Начнем с подключения пакета plots, нужного для визуализации данной задачи:

```
> restart; with(plots):
Warning, the name changecoords has been redefined
```

Составим параметрические уравнения для проекций скорости на оси координат:

```
> Vox:=Vo*cos(alpha);Voy:=Vo*sin(alpha);
```

$$Vox := Vo\cos(\alpha)$$
$$Voy := Vo\sin(\alpha)$$

Мы рассматриваем два случая: камень массой 500 г и камень массой 100 г. Поскольку для каждого случая мы предусматриваем расчет в двух вариантах (с учетом сопротивления воздуха и без такого учета), то мы должны составить 4 системы дифференциальных уравнений (ДУ). Каждая система состоит из двух ДУ второго порядка, и вид этих систем известен из курса физики. Ниже представлено задание этих систем ДУ (для первой системы дан вывод ее вида):

```
> sys1:=massa[1]*diff(x(t),t$2)=-
A[1]*diff(x(t),t),massa[1]*diff(y(t),t$2)=
-A[1]*(diff(y(t),t))-massa[1]*g;
```

$$sys1 := massa_1\left(\frac{d^2}{dt^2}\mathrm{x}(t)\right) = -A_1\left(\frac{d}{dt}\mathrm{x}(t)\right),$$
$$massa_1\left(\frac{d^2}{dt^2}\mathrm{y}(t)\right) = -A_1\left(\frac{d}{dt}\mathrm{y}(t)\right) - massa_1 g$$

```
> sys2:=massa[1]*diff(x(t),t$2)=-
A[2]*diff(x(t),t),massa[1]*diff(y(t),t$2)=-A[2]*(diff(y(t),t))-
massa[1]*g;
```

$$sys2 := massa_1\left(\frac{d^2}{dt^2}\mathrm{x}(t)\right) = -A_2\left(\frac{d}{dt}\mathrm{x}(t)\right),$$
$$massa_1\left(\frac{d^2}{dt^2}\mathrm{y}(t)\right) = -A_2\left(\frac{d}{dt}\mathrm{y}(t)\right) - massa_1 g$$

```
> sys3:=massa[2]*diff(x(t),t$2)=-
A[1]*diff(x(t),t),massa[2]*diff(y(t),t$2)=-A[1]*(diff(y(t),t))-
massa[2]*g;
```

$$sys3 := massa_2\left(\frac{d^2}{dt^2}\mathrm{x}(t)\right) = -A_1\left(\frac{d}{dt}\mathrm{x}(t)\right),$$

$$massa_2\left(\frac{d^2}{dt^2}\mathrm{y}(t)\right) = -A_1\left(\frac{d}{dt}\mathrm{y}(t)\right) - massa_2 g$$

```
> sys4:=massa[2]*diff(x(t),t$2)=-
A[2]*diff(x(t),t),massa[2]*diff(y(t),t$2)=-A[2]*(diff(y(t),t))-
massa[2]*g;
```

$$sys4 := massa_2\left(\frac{d^2}{dt^2}\mathrm{x}(t)\right) = -A_2\left(\frac{d}{dt}\mathrm{x}(t)\right),$$

$$massa_2\left(\frac{d^2}{dt^2}\mathrm{y}(t)\right) = -A_2\left(\frac{d}{dt}\mathrm{y}(t)\right) - massa_2 g$$

Зададим исходные числовые безразмерные данные для расчета:

```
> Vo:=20;massa:=[0.5,0.1];A:=[0.1,0];alpha:=Pi/4;g:=9.8;
```

$$Vo := 20$$

$$massa := [.5, .1]$$

$$A := [.1, 0]$$

$$\alpha := \frac{1}{4}\pi$$

$$g := 9.8$$

Выполним решение заданных систем ДУ:

```
> p1:=dsolve({sys1,x(0)=0,D(x)(0)=Vox,y(0)=0,D(y)(0)=Voy},
{y(t),x(t)},type=numeric ,output=listprocedure):
> p2:=dsolve({sys2,x(0)=0,D(x)(0)=Vox,y(0)=0,D(y)(0)=Voy},
{y(t),x(t)},type=numeric ,output=listprocedure):
> p3:=dsolve({sys3,x(0)=0,D(x)(0)=Vox,y(0)=0,D(y)(0)=Voy},
{y(t),x(t)},type=numeric ,output=listprocedure):
> p4:=dsolve({sys4,x(0)=0,D(x)(0)=Vox,y(0)=0,D(y)(0)=Voy},
{y(t),x(t)}, type=numeric ,output=listprocedure):
```

Громоздкий вывод решений блокируется, но заинтересованный читатель может его просмотреть, заменив двоеточие в конце подготовленных объектов на точку с запятой. Создадим графические объекты – результаты решения систем ДУ:

```
> a1:=odeplot(p1,[x(t),y(t)],0..3,color=green, view=[0..50,0..15],
thickness=2):
> a2:=odeplot(p2,[x(t),y(t)],0..3,color=red, view=[0..50,0..15],
thickness=2):
> a3:=odeplot(p3,[x(t),y(t)],0..3,color=blue, view=[0..50,0..15],
thickness=2:
> a4:=odeplot(p4,[x(t),y(t)],0..3,color=black, view=[0..50,0..15],
thickness=2):
```

Построим графики траекторий для первого случая:

```
> t:=textplot([[25,8,`A=0.1`],[35,9,`A=0`]],color=blue,
font=[TIMES,ROMAN,12]):
> t1:=textplot([[17,3,`A=0.1`],[35,9,`A=0`]],color=blue,
font=[TIMES,ROMAN,12]):
> display({a1,a2,t},title=`Траектория полета тела массой 500 г`,
labels=[x,y],labelfont=[TIMES,ROMAN,14]);
```

Графики траекторий полета камня с массой 500 г представлены на рис. 15.22.

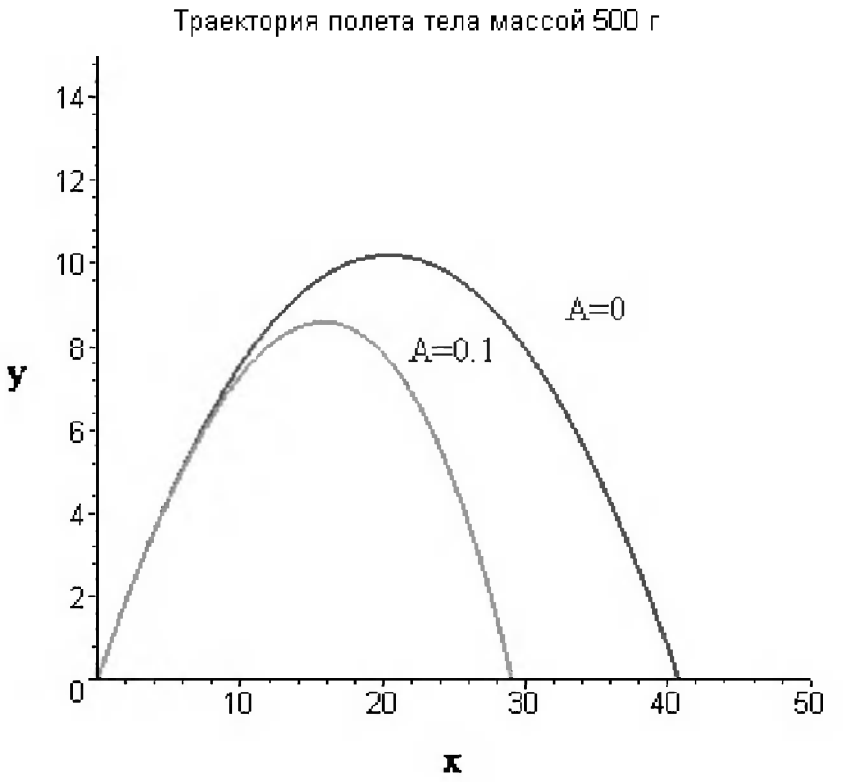


Рис. 15.22. Баллистические траектории камня с массой 500 г

Теперь построим графики траекторий для второго случая:

```
> display({a3,a4,t1},title=`Траектория полета тела массой 100 г`,
labels=[x,y],labelfont=[TIMES,ROMAN,14]);
```

Они представлены на рис. 15.23.

Из проведенных расчетов и графиков видно, что при учете силы сопротивления воздуха дальность и высота полета сильно уменьшаются по сравнению с полетом в вакууме, и эта разница зависит от массы тела, поэтому при небольшой массе тела сопротивлением воздуха пренебрегать нельзя.

15.2.2. Движение частицы в магнитном поле

От реального мира перейдем к микромиру. Пусть микрочастица массой $9*10^{-31}$ кг и зарядом $+1{,}6*10^{-19}$ Кл влетает в магнитное поле с индукцией B=0,1 Тл под углом a=80°. Рассчитаем траекторию движения частицы при начальной скорости $Vo=1*10^{7}$ м/с. Начнем с рестарта:

```
> restart;
```

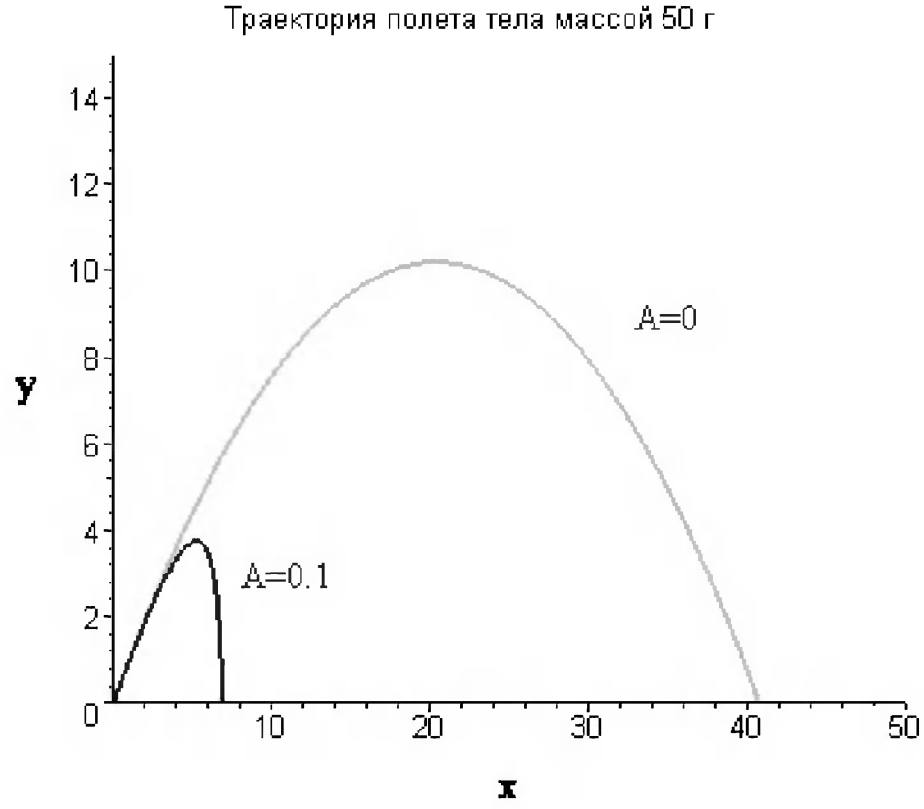


Рис. 15.23. Баллистические траектории камня при массе 100 г

Сила Лоренца, действующая на движущуюся частицу: F=q*(E+[v,B]). Проекции векторного произведения [v,B] на оси x, y, z:

```
[v,B]x=vy*Bz-vz*By      [v,B]y=vz*Bx-vx*Bz      [v,B]z=vx*By-vy*Bz
```

В соответствии с этим известные из курса физики дифференциальные уравнения, описывающие траекторию полета частицы по осям x,y,z, имеют вид:

```
> sys:=diff(x(t),t$2)=q*(Ex+(diff(y(t),t)*Bz-diff(z(t),t)*By))/
massa,diff(y(t),t$2)=q*(Ey+(diff(z(t),t)*Bx — diff(x(t),t)*Bz))/
massa,diff(z(t),t$2)= q*(Ez+(diff(x(t),t)* By-diff(y(t),t)*Bx))/
massa;
```

$$sys := \frac{\partial^2}{\partial t^2}\mathrm{x}(t) = \frac{q\left(Ex + \left(\frac{\partial}{\partial t}\mathrm{y}(t)\right)Bz - \left(\frac{\partial}{\partial t}\mathrm{z}(t)\right)By\right)}{massa},$$

$$\frac{\partial^2}{\partial t^2}\mathrm{y}(t) = \frac{q\left(Ey + \left(\frac{\partial}{\partial t}\mathrm{z}(t)\right)Bx - \left(\frac{\partial}{\partial t}\mathrm{x}(t)\right)Bz\right)}{massa},$$

$$\frac{\partial^2}{\partial t^2}\mathrm{z}(t) = \frac{q\left(Ez + \left(\frac{\partial}{\partial t}\mathrm{x}(t)\right)By - \left(\frac{\partial}{\partial t}\mathrm{y}(t)\right)Bx\right)}{massa}$$

Зададим исходные числовые данные (опустив размерности):

```
> q:=-1.6e-19:massa:=9.1e-31:V:=1e7:alpha:=80*Pi/180:
> Vx:=V*cos(alpha):Vy:=V*sin(alpha):Ex:=0:Ey:=0:Ez:=0:
Bx:=0.1:By:=0: Bz:=0:
```

Построим траекторию движения частиц в пространстве:

```
> with(DEtools):DEplot3d({sys},{x(t),y(t),z(t)},t=0..2e-
9,[[x(0)=0,D(x)(0)=Vx,y(0)=0,D(y)(0)=Vy,z(0)=0,D(z)(0)=0]],stepsize=1e-
11,orientation=[24,117]);
```

Полученная траектория представлена на рис. 15.24. Она имеет вид спирали в пространстве. При этом скорость движения частицы вдоль оси x неизменна, а вдоль осей y и z имеет характерную колебательную компоненту. Случай явно куда менее тривиальный, чем полет камня, описанный выше.

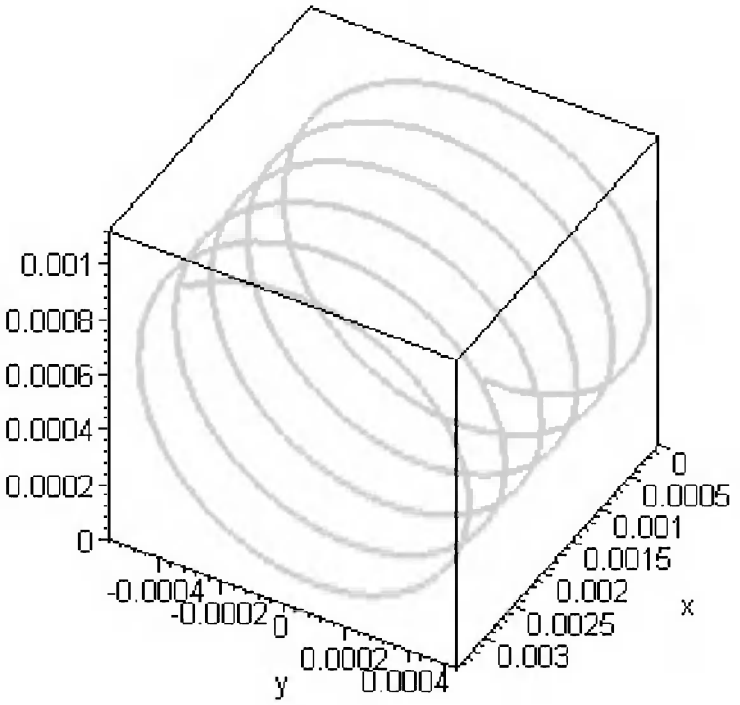


Рис. 15.24. Траектория движения частицы в магнитном поле

Мы можем найти аналитическое представление для траектории частицы в виде параметрически заданной (с параметром времени t) системы из трех уравнений:

```
>xyz:=dsolve({sys,x(0)=0,D(x)(0)=Vx,y(0)=0,D(y)(0)=Vy,z(0)=0,D(z)(0)=0},
{x(t),y(t),z(t)}, method=laplace);
```

$$xyz := \{z(t) = 175824175800000000\ \sin\left(\frac{4\pi}{9}\right)\left(\frac{1}{309141407957493056400}\right.$$

$$\left. - \frac{1}{309141407957493056400}\cos(17582417580t)\right),$$

$$y(t) = \frac{500000}{879120879}\sin\left(\frac{4\pi}{9}\right)\sin(17582417580t),$$

$$x(t) = 10000000\ \cos\left(\frac{4\pi}{9}\right)t\}$$

Моделирование движения заряженной частицы в пространстве с магнитным полем показывает, что для принятых для моделирования параметров решаемой задачи движение частицы происходит по спиралеобразной траектории. Получен как график траектории движения частицы, так и аналитические уравнения, описывающие это движение.

15.2.3. Разделение изотопов

Рассмотрим еще одну классическую задачу ядерной физики – *разделение изотопов* (атомов с одинаковым зарядом ядра, но разной массой). Для этого используют различные способы. В частности, это может быть масс-спектроскопический метод. Из точки A вылетают однозарядные ионы ($q=e=1.6*10^{-19}$ Кл) разной массы (от 20 до 23 а.е.м.) и под разными углами в пределах от 80 до 100° к оси x в плоскости xy (рис. 15.25). Вдоль оси z приложено магнитное поле $B=10^{-2}$ Тл. Рассчитать траектории полета частиц. Будем надеяться, что это подскажет способ разделения изотопов.

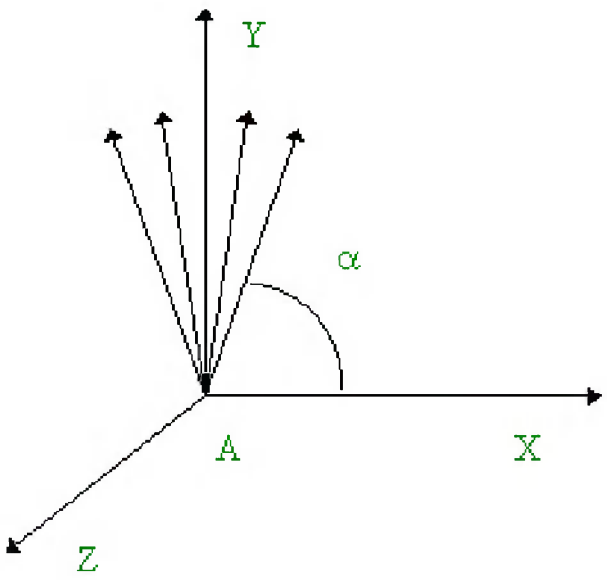


Рис. 15.25. Иллюстрация к методу разделения изотопов

Приступим к решению данной задачи. Сила Лоренца, действующая на движущуюся частицу: F=q*(E+[v,B]). Проекции векторного произведения [v,B] на оси x, y, z заданы выражениями:

```
[v,B]x=vy*Bz-vz*By  [v,B]y=vz*Bx-vx*Bz  [v,B]z=vx*By-vy*Bz
```

В соответствии с этим дифференциальные уравнения, описывающие траекторию полета частицы по осям x,y,z, имеют вид:

```
> restart;
> sys:=diff(x(t),t$2)=q*(Ex+(diff(y(t),t)*Bz-diff(z(t),t)*By))/
massa,diff(y(t),t$2)=q*(Ey+(diff(z(t),t)*Bx-diff(x(t),t)*Bz))/
massa,diff(z(t),t$2)=q*(Ez+(diff(x(t),t) *By-diff(y(t),t)*Bx))/
massa;
```

$$sys := \frac{\partial^2}{\partial t^2}\mathrm{x}(t) = \frac{q\left(Ex + \left(\frac{\partial}{\partial t}\mathrm{y}(t)\right)Bz - \left(\frac{\partial}{\partial t}\mathrm{z}(t)\right)By\right)}{massa},$$

$$\frac{\partial^2}{\partial t^2}\mathrm{y}(t) = \frac{q\left(Ey + \left(\frac{\partial}{\partial t}\mathrm{z}(t)\right)Bx - \left(\frac{\partial}{\partial t}\mathrm{x}(t)\right)Bz\right)}{massa},$$

$$\frac{\partial^2}{\partial t^2}\mathrm{z}(t) = \frac{q\left(Ez + \left(\frac{\partial}{\partial t}\mathrm{x}(t)\right)By - \left(\frac{\partial}{\partial t}\mathrm{y}(t)\right)Bx\right)}{massa}$$

Зададим исходные числовые данные для расчета:

```
> q:=1.6e-19:V:=1e4:
> Vx:=V*cos(alpha):Vy:=V*sin(alpha):Ex:=0:Ey:=0:Ez:=0:
  Bx:=0:By:=0:Bz:=1e-2:
```

Выполним решение составленной выше системы дифференциальных уравнений:

```
> xyz:=dsolve({sys,x(0)=0,D(x)(0)=Vx,y(0)=0,D(y)(0)=Vy,
  z(0)=0,D(z)(0)=0},{x(t),y(t),z(t)},method=laplace):
> XX:=(massa,alpha)->.6250000000e25*massa*(sin(alpha)-
  1.*sin(alpha)*cos(.1600000000e-
  20*t/massa)+cos(alpha)*sin(.1600000000e-20*t/massa));
> YY:=(massa,alpha)->.6250000000e25*massa*(-
  1.*cos(alpha)+cos(alpha)*cos(.1600000000e-
  20*t/massa)+sin(alpha)*sin(.1600000000e-20*t/massa));
```

$$XX := (massa, \alpha) \to 0.6250000000\ 10^{25} massa\left(\sin(\alpha)\right.$$

$$-1.\sin(\alpha)\cos\left(0.1600000000\ 10^{-20}\frac{t}{massa}\right)$$

$$\left.+\cos(\alpha)\sin\left(0.1600000000\ 10^{-20}\frac{t}{massa}\right)\right)$$

$$YY := (massa, \alpha) \to 0.6250000000\ 10^{25} massa\left(-1.\cos(\alpha)\right.$$

$$+\cos(\alpha)\cos\left(0.1600000000\ 10^{-20}\frac{t}{massa}\right)$$

$$\left.+\sin(\alpha)\sin\left(0.1600000000\ 10^{-20}\frac{t}{massa}\right)\right)$$

Построим графики решения:

```
> aem:=1.67e-27: ur:=3.14/180:
> plot([[XX(20*aem,80*ur),YY(20*aem,80*ur),
t=0..10e-5],[XX(20*aem,90*ur),YY(20*aem,90*ur),
t=0..10e-5],[XX(28*aem,80*ur),YY(28*aem,80*ur),
t=0..10e-5],[XX(28*aem,90*ur),YY(28*aem,90*ur),
t=0..10e-5],[XX(24*aem,80*ur),YY(24*aem,80*ur),
t=0..10e-5],[XX(24*aem,90*ur),YY(24*aem,90*ur),
t=0..10e-5]],view=[0..0.65,0..0.65],
color=[red,red,blue,blue,black,black],labels=[x,y]);
```

Эти графики показаны на рис. 15.26.

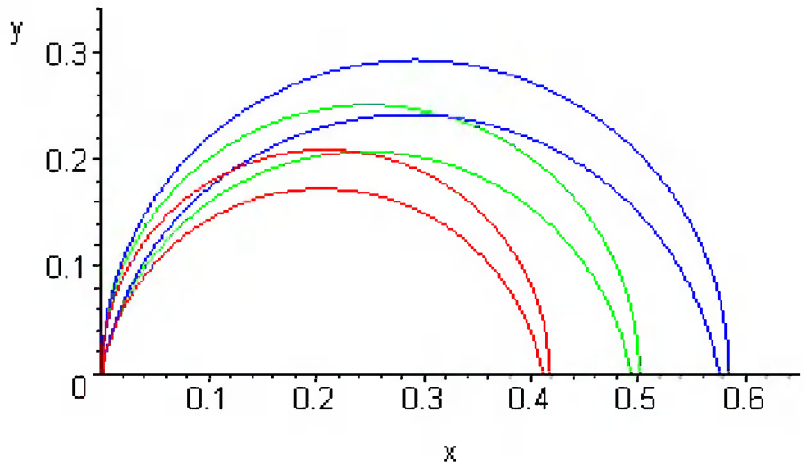


Рис. 15.26. Траектории движения частиц

Полученные графики (рис. 15.26) наглядно показывают на одну из возможностей разделения изотопов. Как говорится, осталось подставить «стаканчик» в нужное место для ловли нужных изотопов. Разумеется, это только изложение идеи одного из методов разделения изотопов. Увы, на практике приходится использовать сложнейшие и дорогие физические установки для решения этой актуальной задачи.

15.2.4. Моделирование рассеивания альфа-частиц

Одним из фундаментальных доказательств существования ядра у атомов стал опыт с бомбардировкой тонкой фольги из металла альфа-частицами с высокой энергией. Если бы «массивных» ядер не существовало, то альфа-частицы должны были бы спокойно пролетать тонкую фольгу, практически не отклоняясь. Однако, как физики и ожидали, некоторая часть частиц испытывала сильное отклонение и даже поворачивала назад. Очевидно, что имели место отскоки (упругие столкновения) с малыми, но массивными ядрами металла фольги.

В нашем распоряжении, увы (а может быть, и к счастью), нет ускорителя альфа-частиц. Так что мы, не опасаясь облучения и очередной Чернобыльской катас-

трофы, сможем смоделировать это интереснейшее физическое явление с помощью математической системы Maple.

Итак, пусть в нашем теоретическом опыте альфа-частицы с энергией 4 МэВ рассеиваются тонкой золотой фольгой. Рассчитать траекторию частицы, приближающейся к ядру атома Au. Прицельное расстояние p равно $2*10^{-15}$ м.

Приступим к решению задачи и зададим вначале систему дифференциальных уравнений для траектории альфа-частицы:

```
> restart;
> sys:=diff(x(t),t$2)=q1*q2*x(t)/(4*Pi*E0*massa*
(x(t)^2+y(t)^2)^(3/2)),diff(y(t),t$2)=q1*q2*y(t)/
(4*Pi*E0*massa*(x(t)^2+y(t)^2)^ (3/2));
```

$$sys := \frac{\partial^2}{\partial t^2}\mathrm{x}(t) = \frac{1}{4}\frac{q1q2\mathrm{x}(t)}{\pi\, E0\, massa(\mathrm{x}(t)^2+\mathrm{y}(t)^2)^{(3/2)}},$$

$$\frac{\partial^2}{\partial t^2}\mathrm{y}(t) = \frac{1}{4}\frac{q1q2\mathrm{y}(t)}{\pi\, E0\, massa(\mathrm{x}(t)^2+\mathrm{y}(t)^2)^{(3/2)}}$$

Введем исходные числовые данные для вычислений:

```
> q1:=2*1.6e-19:q2:=79*1.6e-19:massa:=4*1.67e-27:E0:=8.85e-12:
a:=4e-13:p:=5e-15:T:=4e6*1.6e-19:V0x:=sqrt(2*T/massa):
```

Создадим графическую структуру решения нашей системы дифференциальных уравнений для нескольких расчетных отклонений линии движения альфа-частицы от центра ядра атома, находящегося на ее пути:

```
> with(DEtools):ss:=DEplot({sys},{y(t),x(t)},t=0..7e-20,
[[x(0)=-a,D(x)(0)=V0x,y(0)=p,D(y)(0)=0],
[x(0)=-a,D(x)(0)=V0x,y(0)=p*4,D(y)(0)=0],
[x(0)=-a,D(x)(0)=V0x,y(0)=p*8,D(y)(0)=0],
[x(0)=-a,D(x)(0)=V0x,y(0)=p*12,D(y)(0)=0],
[x(0)=-a,D(x)(0)=V0x,y(0)=p*16,D(y)(0)=0],
[x(0)=-a,D(x)(0)=V0x,y(0)=p*20,D(y)(0)=0],
[x(0)=-a,D(x)(0)=V0x,y(0)=p*24,D(y)(0)=0],
[x(0)=-a,D(x)(0)=V0x,y(0)=p*28,D(y)(0)=0]],
x(t)=-a..a,scene=[x(t),y(t)],stepsize=1e-21,linecolor=black):
> with(plottools):yy:=circle([0,0],2E-14,color=red,thickness=2):
Warning, the name translate has been redefined
```

Построим центр ядра (кружок со знаком +) и траектории альфа-частиц:

```
> ss2:=PLOT(TEXT([0,-0.3e-14],`+`), FONT(HELVETICA, OBLIQUE,14)):
```

Осталось построить график траекторий движения альфа-частиц вблизи центра атома:

```
> with(plots):
Warning, the name changecoords has been redefined
> display([ss,yy,ss2],title=`Рассеивание a-частиц`,axes=framed);
```

График траекторий движения альфа-частиц вблизи ядра представлен на рис. 15.27. Этот график настолько нагляден, что не требует пояснения.

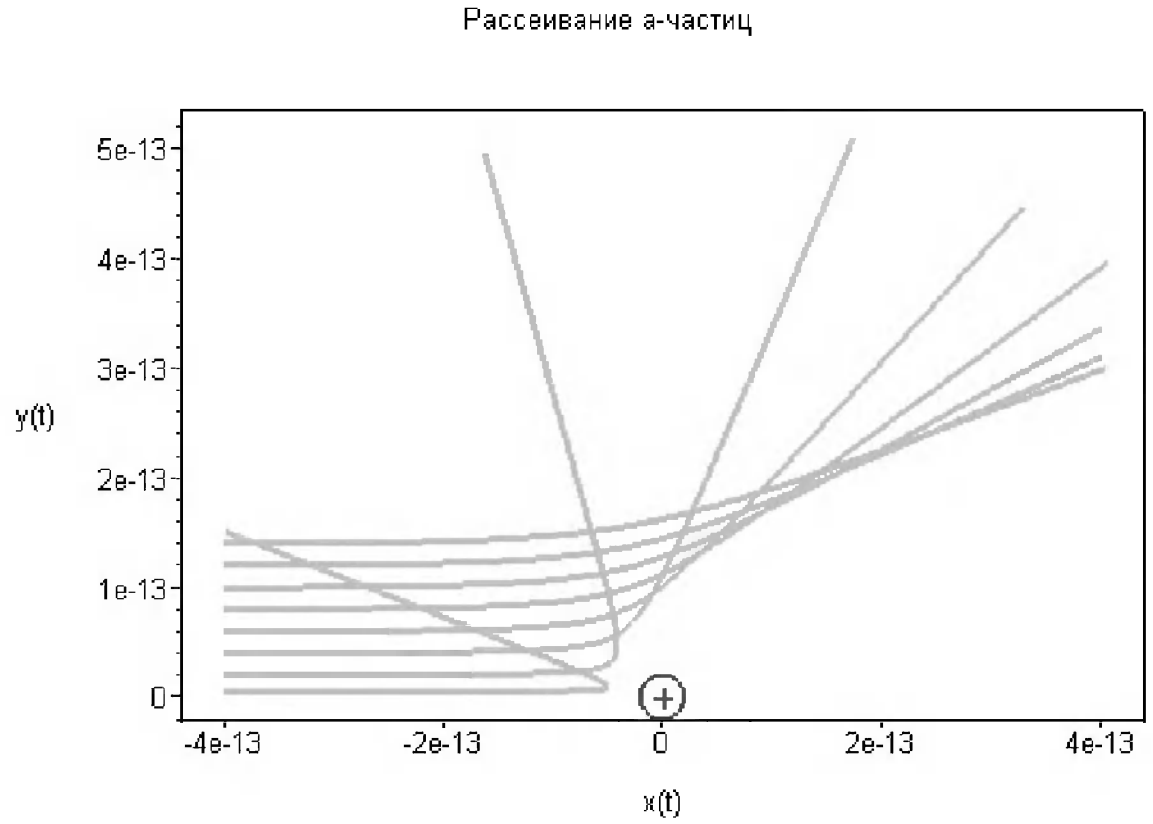


Рис. 15.27. Траектории движения альфа-частиц вблизи ядра атома

Моделирование движения альфа-частиц вблизи малого и «массивного» ядра атома дает наглядное представление о математической и физической сути данного опыта. Надо лишь помнить, что нельзя нацеливать быстро летящие альфа-частицы прямо в центр ядра. Более сложные, чем приведенные, расчеты показывают, что при этом альфа-частица настолько близко подходит к ядру, что надо учитывать новые факторы, возникающие при близком взаимодействии. Они могут привести к тому, что частица будет поглощена ядром. Но это уже тема нового разговора, выходящего за рамки данной книги.

15.3. Моделирование и расчет электронных схем

15.3.1. Нужно ли применять Maple для моделирования и расчета электронных схем?

Нужно ли применять системы компьютерной математики для анализа, расчета и моделирования *электронных схем*? Ответ на этот вопрос не так прост, как кажется с первого взгляда. С одной стороны, к услугам пользователя компьютера сейчас имеется ряд программ схемотехнического моделирования, например Micro-CAP,

Electronics Workbench, PSpice, Design Labs и др., автоматически составляющих и решающих большие системы уравнений состояния электронных схем и моделирующих работу бесчисленного множества электронных схем без кропотливого «ручного» составления уравнений.

Но, с другой – анализ схем в таких программах настолько автоматизирован, что начисто теряется его физическая и математическая сущность. И это явно плохо, когда объектом исследования и моделирования являются новые нетрадиционные схемы на новых или малоизвестных приборах или когда знание физических и математических основ работы таких схем принципиально необходимо.

13.3.2. Применение интеграла Дюамеля для расчета переходных процессов

Вернемся к линейным системам и рассмотрим еще один полезный метод расчета электрических цепей – с помощью *интеграла Дюамеля* [161, 162]. При нем можно рассчитать временную зависимость выходного напряжения u2(t) цепи по известному входному сигналу u1(t) и переходной характеристике цепи a(t). Возьмем в качестве первого классического примера дифференцирующую RC-цепь и вычислим ее реакцию на экспоненциально нарастающий перепад напряжения. Соответствующие расчеты приведены на рис. 15.28.

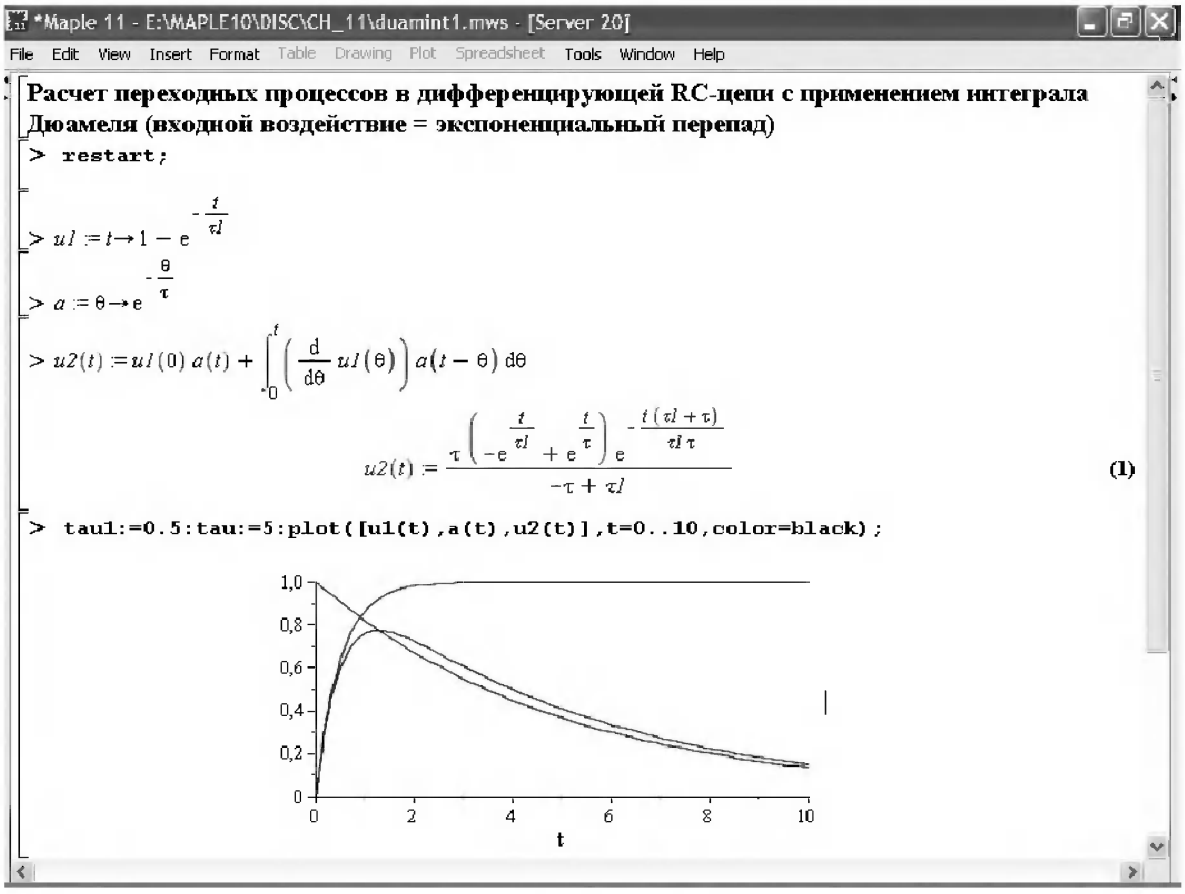


Рис. 15.28. Расчет реакции дифференцирующей цепи на экспоненциальный перепад напряжения

Рисунок 15.28 представляет начало документа, в котором выполнен указанный выше расчет. Представлены заданные зависимости u1(t) и a(t), аналитическое выражение для интеграла Дюамеля (одна из 4 форм) и аналитическое выражение для искомой зависимости u2(t). Пока последнее выражение довольно простое. В конце этого фрагмента документа построены графики зависимостей u1(t), a(t) и u2(t).

Окончание документа, представленное на рис. 15.29, демонстрирует расчет на основе интеграла Дюамеля реакции дифференцирующей RC-цепи на экспоненциально затухающий синусоидальный сигнал u1(t).

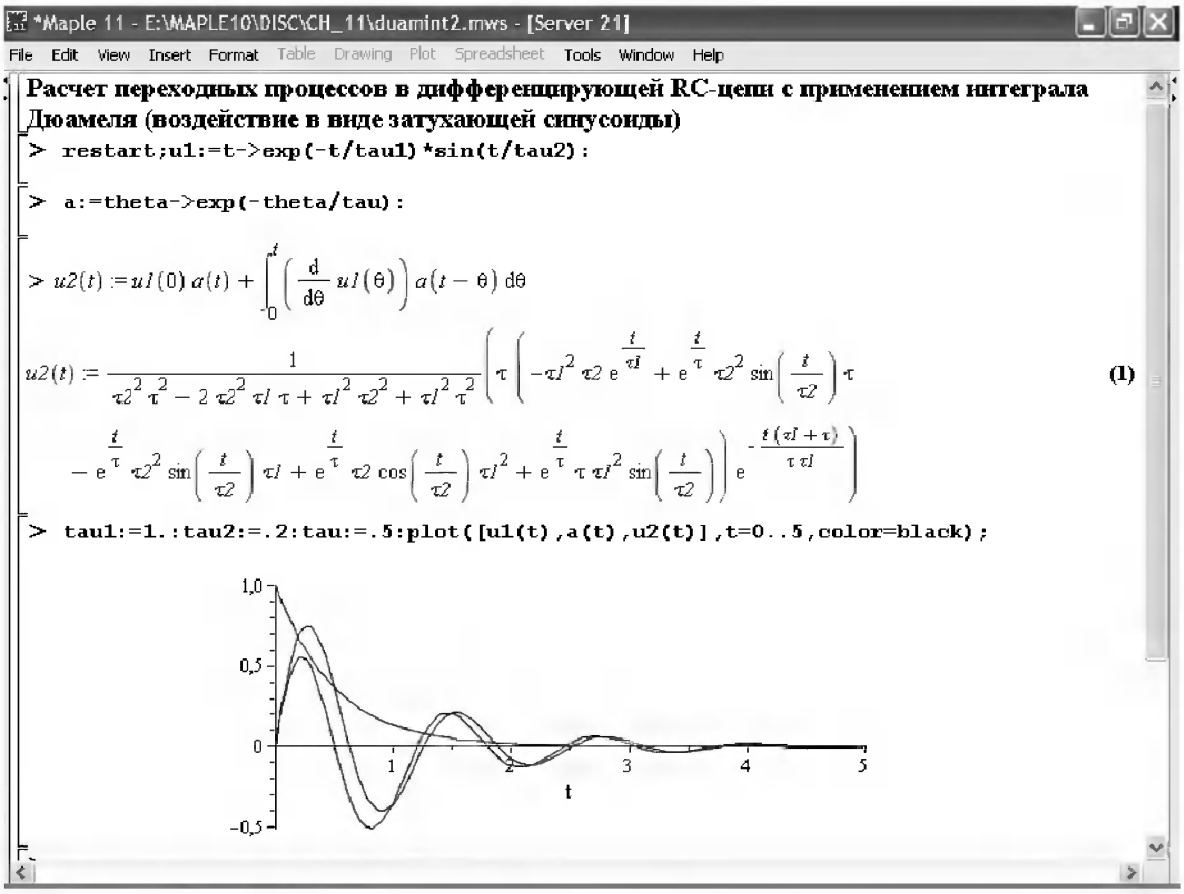


Рис. 15.29. Расчет реакции дифференцирующей цепи на синусоидальный сигнал с экспоненциально уменьшающейся амплитудой

Обратите внимание на то, что выражение для u2(t), получаемое с помощью интеграла Дюамеля, стало намного сложнее. Тем не менее получено как аналитическое выражение для реакции цепи u2(t), так и графики u1(t), a(t) и u2(t). Они показаны внизу графика.

15.3.3. Малосигнальный анализ фильтра-усилителя на операционном усилителе

Теперь рассмотрим проектирование аналогового полосового фильтра-усилителя на операционном усилителе, схема которого приведена на рис. 15.30. Сам операционный усилитель будем считать идеальным.

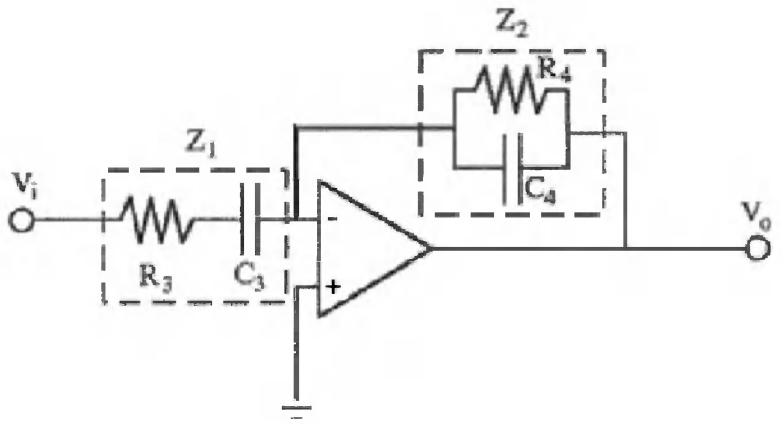


Рис. 15.30. Схема полосового фильтра на интегральном операционном усилителе

Подготовимся к расчету фильтра:

```
> restart:
```

Зададим основные уравнения, описывающие работу усилителя на малом сигнале:

```
> Vo := (-Z2/Z1)* Vi;
```

$$Vo := -\frac{Z2\,Vi}{Z1}$$

```
> Z1 := R3 + 1/(I*omega*C3);
```

$$Z1 := R3 - \frac{I}{\omega C3}$$

```
> Z2 := R4*1/(I*omega*C4) / (R4 + 1/(I*omega*C4));
```

$$Z2 := \frac{-IR4}{\omega C4\left(R4 - \frac{I}{\omega C4}\right)}$$

Введем круговую частоту

```
> omega := 2*Pi*f;
```

$$\omega := 2\pi f$$

Найдем в аналитическом виде коэффициент передачи фильтра и его фазочастотную характеристику как функции от частоты:

```
> gain := abs(evalc(Vo/Vi));
```

$$gain := \left| -\frac{R4\,R3}{4\pi^2 f^2 C4^2 \left(R4^2 + \frac{1}{4\pi^2 f^2 C4^2}\right)\left(R3^2 + \frac{1}{4\pi^2 f^2 C3^2}\right)} - \frac{R4^2}{4\pi^2 f^2 C4 \left(R4^2 + \frac{1}{4\pi^2 f^2 C4^2}\right) C3 \left(R3^2 + \frac{1}{4\pi^2 f^2 C3^2}\right)} + \left(\frac{R4^2\,R3}{4\pi f C4 \left(R4^2 + \frac{1}{4\pi^2 f^2 C4^2}\right)\left(R3^2 + \frac{1}{4\pi^2 f^2 C3^2}\right)} - \frac{R4}{8\pi^3 f^3 C4^2 \left(R4^2 + \frac{1}{4\pi^2 f^2 C4^2}\right) C3 \left(R3^2 + \frac{1}{4\pi^2 f^2 C3^2}\right)} \right) I \right|$$

```
> phase := evalc(op(2,convert(Vo/Vi,polar)));
```

$$phase := \arctan\left(\frac{R4^2\,R3}{2 f C4 \left(R4^2 + \frac{1}{4\pi^2 f^2 C4^2}\right)\left(R3^2 + \frac{1}{4\pi^2 f^2 C3^2}\right)} - \frac{R4}{8 f^3 C4^2 \pi^2 \left(R4^2 + \frac{1}{4\pi^2 f^2 C4^2}\right) C3 \left(R3^2 + \frac{1}{4\pi^2 f^2 C3^2}\right)}, - \frac{R4R3}{4 f^2 C4^2 \pi \left(R4^2 + \frac{1}{4\pi^2 f^2 C4^2}\right)\left(R3^2 + \frac{1}{4\pi^2 f^2 C3^2}\right)} - \frac{R4^2}{4 f^2 C4 \left(R4^2 + \frac{1}{4\pi^2 f^2 C4^2}\right) \pi C3 \left(R3^2 + \frac{1}{4\pi^2 f^2 C3^2}\right)} \right)$$

Эти выражения, несмотря на простоту схемы усилителя, выглядят довольно сложно, что, однако, ничуть не мешает использовать их для выполнения расчетов. Зададим конкретные значения параметров:

```
> R3 := 1000:
> R4 := 3000:
> C3 := 0.08*10^(-6):
> C4 := 0.01*10^(-6):
```

Построим АЧХ фильтра как зависимость коэффициента передачи в децибелах (dB) от частоты f в герцах (Гц):

```
> plot([log10(f), 20*log10(gain), f=10..50000],
color=black,title=`Коэффициент передачи dB как функция от логарифма
частоты f в Гц`);
```

Эта характеристика представлена на рис. 15.31. Здесь полезно обратить внимание на то, что спад усиления на низких и высоких частотах происходит довольно медленно из-за малого порядка фильтра.

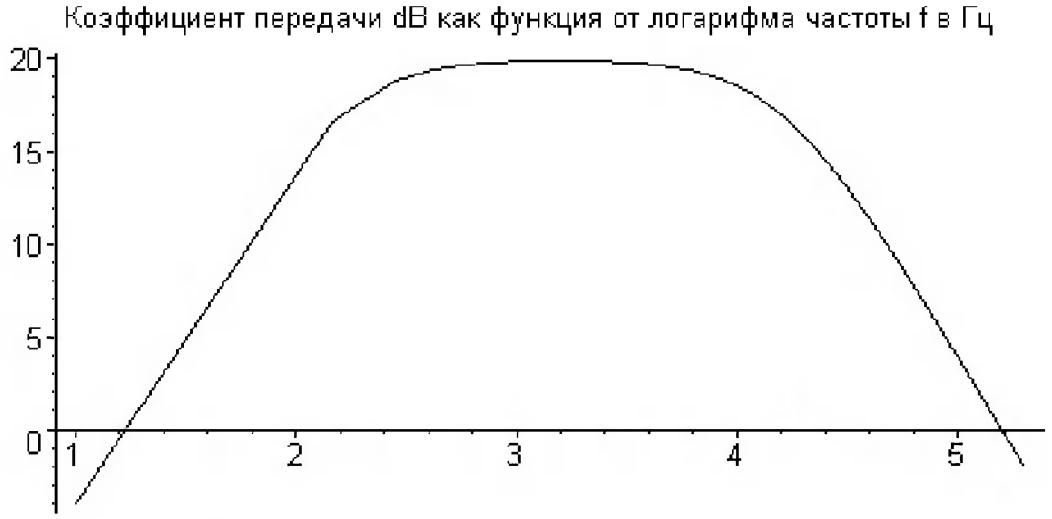


Рис. 15.31. АЧХ фильтра на операционном усилителе

Далее построим фазочастотную характеристику фильтра как зависимость фазы в радианах от частоты f в герцах:

```
> plot ([log10(f), phase, f=10..50000], color=black,
title=`Фаза в радианах как функция логарифма частоты`);
```

Фазочастотная характеристика (ФЧХ) фильтра показана на рис. 15.32.

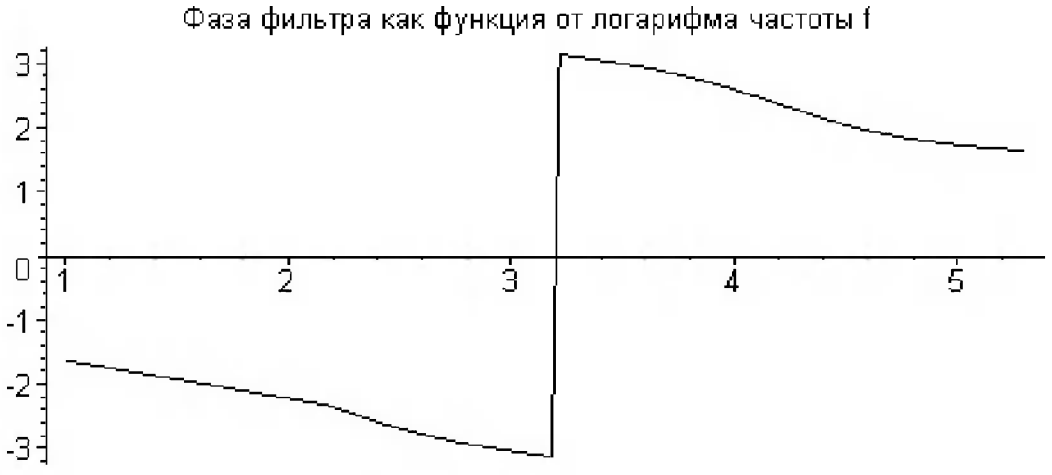


Рис. 15.32. ФЧХ фильтра на операционном усилителе

На ФЧХ фильтра можно заметить характерный разрыв, связанный с превышением фазовым углом граничного значения π. Такой способ представления фазового сдвига общепринят, поскольку его изменения стремятся вписать в диапазон от $-\pi$ до π.

15.3.4. Проектирование цифрового фильтра

Основной недостаток аналоговых активных фильтров, подобных описанному выше, заключается в их малом порядке. Его повышение, за счет применения многих звеньев низкого порядка, ведет к значительному повышению габаритов фильтров и их стоимости. От этого недостатка свободны современные цифровые фильтры, число ячеек которых N даже при однокристальном исполнении может достигать десятков и сотен. Это обеспечивает повышенную частотную селекцию.

Спроектируем фильтр (N+1)-го порядка класса FIR (Finite Impulse Response, или с конечной импульсной характеристикой). Каждая из N ячеек временной задержки фильтра удовлетворяет следующей зависимости выходного сигнала y от входного x вида:

$$y_n = \sum_{k=0}^{N} h_k x_{n-k}.$$

Подключим пакет расширения plots, нужный для графической визуализации проектирования:

```
> restart:with(plots):
Warning, the name changecoords has been redefined
```

Зададим исходные данные для проектирования полосового цифрового фильтра, выделяющего пятую гармонику из входного сигнала в виде зашумленного меандра с частотой 500 Гц:

```
> N := 64:      # Число секций фильтра (на 1 меньше порядка фильтра)
> fs := 10000: # Частота квантования
> fl := 2300:  # Нижняя граничная частота
> fh := 2700:  # Верхняя граничная частота
> m := 10:     # 2^m > N — число точек для анализа
```

Вычислим:

```
> T := 2^m-1;
```

$$T := 1023$$

```
> F1 := evalf(fl/fs);
```

$$F1 := .2300000000$$

```
> F2 := evalf(fh/fs);
```

$$F2 := .2700000000$$

```
> Dirac(0) := 1: # Функция Дирака
> fp1:=2*Pi*F1: fp2:=2*Pi*F2:
```

Зададим характеристику полосового фильтра:

```
> g := (sin(t*fp2)-sin(t*fp1))/(t*Pi);
```

$$g := \frac{\sin(.5400000000\,t\pi) - \sin(.4600000000\,t\pi)}{t\pi}$$

Вычислим FIR коэффициенты для прямоугольного окна фильтра:

```
> C := (n)  ->  limit(g,t=n):h := aray(0..N): N2:=N/2:
> for n from 0 to N2 do h[N2-n]:= evalf(C(n)); h[N2+n] :=
  h[N2-n]; od:
```

Определим массивы входного $x(n)$ и выходного $y(n)$ сигналов:

```
> x := array(-N..T):y := array(0..T):
```

Установим значение `x(n)` равным 0 для времени меньше 0 и 1 для времени t>=0:

```
> for n from -N to -1 do x[n] := 0; od:
> for n from 0 to T do x[n] := Dirac(n); od:
```

Вычислим временную зависимость для выходного сигнала:

```
> for n from 0 to T do y[n] := sum(h[k]*x[n-k],k=0..N); od:
```

Построим график импульсной характеристики фильтра, отражающей его реакцию на сигнал единичной площади с бесконечно малым временем действия:

```
> p := [seq([j/fs,y[j]],j=0..T)]:
> plot(p, time=0..3*N/fs, labels=[time,output], axes=boxed,
xtickmarks=4, title=`Импульсная характеристика
фильтра`,color=black);
```

Он показан на рис. 15.33. Нетрудно заметить, что эта характеристика свидетельствует об узкополосности фильтра, поскольку его частоты `fl` и `fh` различаются несильно. В этом случае полосовой фильтр по своим свойствам приближается к резонансному, хотя само по себе явление резонанса не используется.

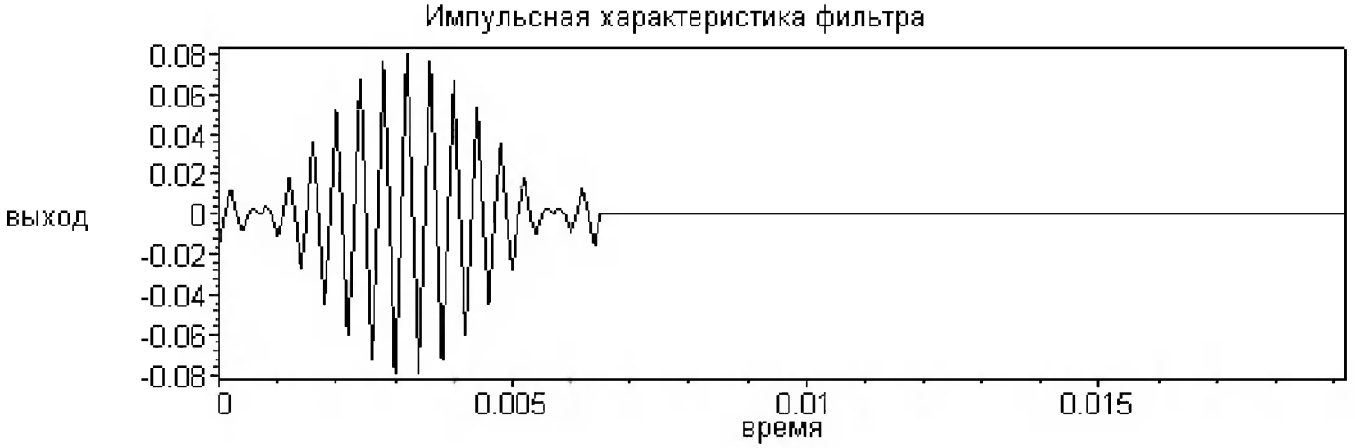


Рис. 15.33. Импульсная характеристика цифрового фильтра

Вычислим АЧХ фильтра, используя прямое преобразование Фурье. Оно, после подготовки обрабатываемых массивов, реализуется функцией FFT:

```
> ro := array(1..T+1):io := array(1..T+1):
> for n from 0 to T do ro[n+1] := y[n]; io[n+1] := 0; od:
> FFT(m,ro,io):
```

Построим график АЧХ фильтра:

```
> p := [seq([j*fs/(T+1),abs(ro[j+1]+io[j+1]*I)],j=0..T/2)]:
> plot(p, frequency=0..fs/2, labels=[frequency,gain], title=`АЧХ
фильтра`,color=black);
```

Он представлен на рис. 15.34. Нетрудно заметить, что и впрямь АЧХ фильтра напоминает АЧХ резонансной цепи – она имеет вид узкого пика. Вы можете легко проверить, что раздвижением частот `fl` и `fh` можно получить АЧХ с довольно плоской вершиной и резкими спадами (говорят, что такая характеристика приближается к прямоугольной).

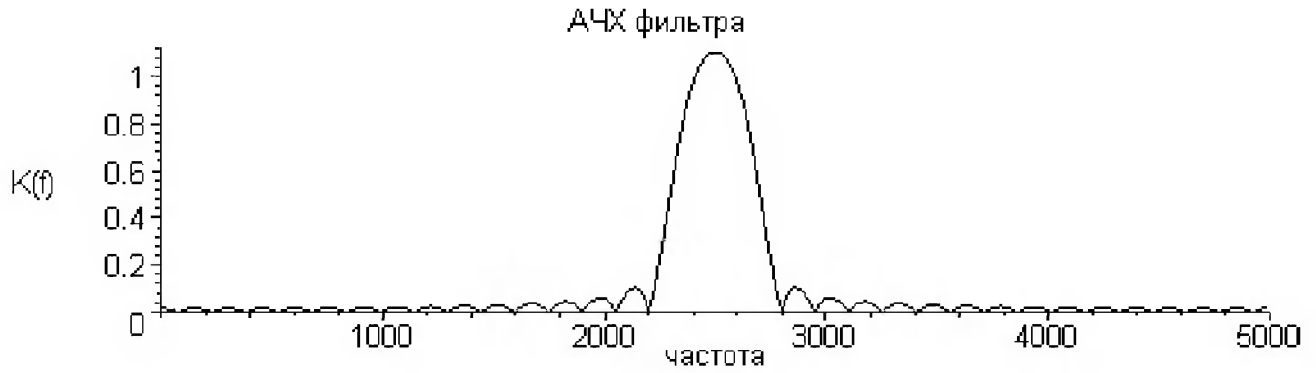


Рис. 15.34. АЧХ цифрового полосового фильтра

Теперь приступим к тестированию фильтра. Зададим входной сигнал в виде зашумленного меандра с частотой 500 Гц и размахом напряжения 2 В:

```
> l := round(fs/2/500):
> for n from 0 by 2*l to T do
>      for n2 from 0 to l-1 do
>           if n+n2 <= T then
>                x[n+n2] := evalf(-1+rand()/10^12-0.5);
>           fi;
>           if n+n2+l <= T then
>                x[n+n2+l] := evalf(1+rand()/10^12-0.5);
>           fi;
>      od:
> od:
```

Временная зависимость синтезированного входного сигнала представлена на рис. 15.35.

Вычислим реакцию фильтра на входной сигнал:

```
> for n from 0 to T do
>      y[n] := sum(h[k]*x[n-k],k=0..N);
> od:
```

Построим график выходного сигнала:

```
> p := [seq([j/fs,x[j]],j=0..T)]:q := [seq([j/fs,y[j]],j=0..T)]:
> plot(p,time=0..T/fs/4,labels=[time,volts],title=`Входной
сигнал`,color=black);
```

Рис. 15.35. Синтезированный входной сигнал

```
> plot(q,time=0..T/fs/4,labels=[time,volts],title=`Выходной
сигнал`,color=black);
```

Временная зависимость выходного сигнала показана на рис. 15.36. Нетрудно заметить, что в конце концов выходной сигнал вырождается в пятую гармонику входного сигнала, но этому предшествует довольно заметный переходной процесс. Он связан с узкополосностью данного фильтра.

Рис. 15.36. Временная зависимость выходного сигнала цифрового фильтра

Вычислим спектры входного и выходного сигналов, подготовив массивы выборок сигналов и применив прямое преобразование Фурье с помощью функции FFT:

```
> ri := array(1..T+1):ii := array(1..T+1):
> for n from 0 to T do
>      ri[n+1] := x[n]*2/T; ii[n+1] := 0;
>      ro[n+1] := y[n]*2/T; io[n+1] := 0;
> od:
> FFT(m,ri,ii):FFT(m,ro,io):
```

Построим график спектра входного сигнала, ограничив масштаб по амплитуде значением 0,5 В:

```
> p := [seq([j*fs/(T+1),abs(ri[j+1]+ii[j+1]*I)],j=0..T/2)]:
> q := [seq([j*fs/(T+1),abs(ro[j+1]+io[j+1]*I)],j=0..T/2)]:
> plot(p, frequency=0..fs/2,y=0..0.5,labels=[частота,V],
title=`Частотный спектр входного сигнала`,color=black);
```

Этот график представлен на рис. 15.37. Из него хорошо видно, что спектральный состав входного сигнала представлен только нечетными гармониками, амплитуда которых убывает по мере роста номера гармоники. Пятая гармоника на частоте 2500 Гц находится посередине полосы пропускания фильтра, ограниченной граничными частотами фильтра 2300 и 2700 Гц. Заметны также беспорядочные спектральные линии шума сигнала в пределах полосы прозрачности фильтра.

Рис. 15.37. Спектр входного сигнала

Теперь построим график спектра выходного сигнала:

```
> plot(q, frequency=0..fs/2,y=0..0.5,labels=[частота,V],
title=`Частотный спектр выходного сигнала`,color=black);
```

Он представлен на рис. 15.38. Хорошо видно эффективное выделение пятой гармоники сигнала и прилегающей к ней узкой полосы шумового спектра.

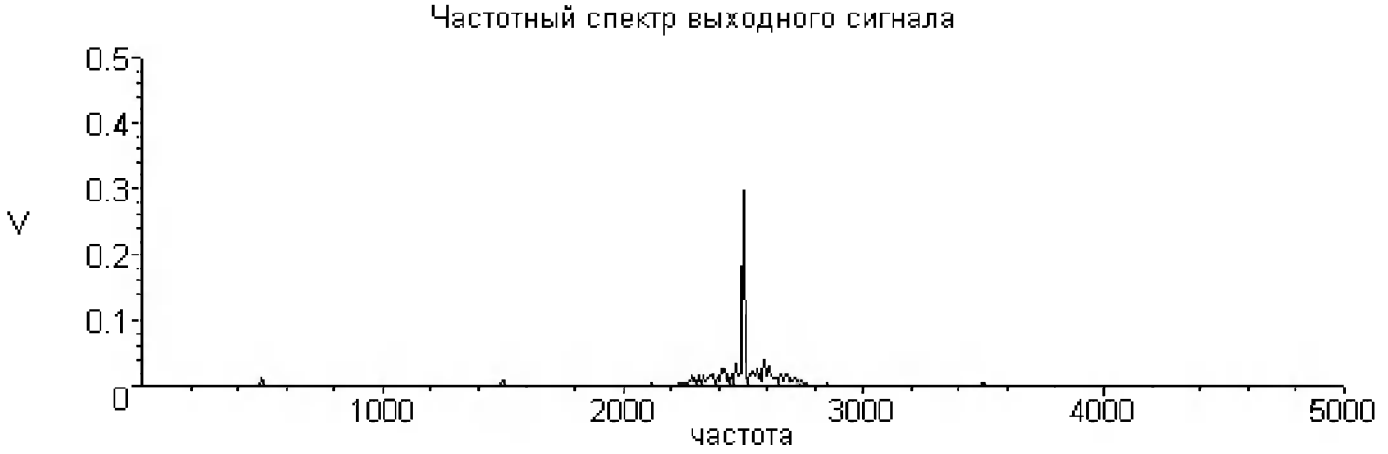


Рис. 15.38. Спектр выходного сигнала цифрового фильтра

Приведенные данные свидетельствуют, что спроектированный фильтр полностью отвечает заданным требованиям и обеспечивает уверенное выделение пятой гармоники зашумленного меандра.

15.3.5. Моделирование цепи на туннельном диоде

А теперь займемся моделированием явно нелинейной цепи. Выполним его для цепи, которая состоит из последовательно включенных источника напряжения Es, резистора Rs, индуктивности L и туннельного диода, имеющих N-образную вольт-амперную характеристику (ВАХ). Туннельный диод обладает емкостью C, что имитируется конденсатором C, подключенным параллельно туннельному диоду.

Пусть ВАХ реального туннельного диода задана выражением:

```
> restart;
> A:=.3: a:=10: B:=1*10^(-8): b:=20:
> Id:=Ud->A*Ud*exp(-a*Ud)+B*(exp(b*Ud-1));
```

$$Id := Ud \to A\,Ud\,\mathbf{e}^{(-a\,Ud)} + B\,\mathbf{e}^{(b\,Ud\,-\,1)}$$

Построим график ВАХ (рис. 15.39):

```
> plot(Id(Ud), Ud=-.02..0.76,color=black);
```

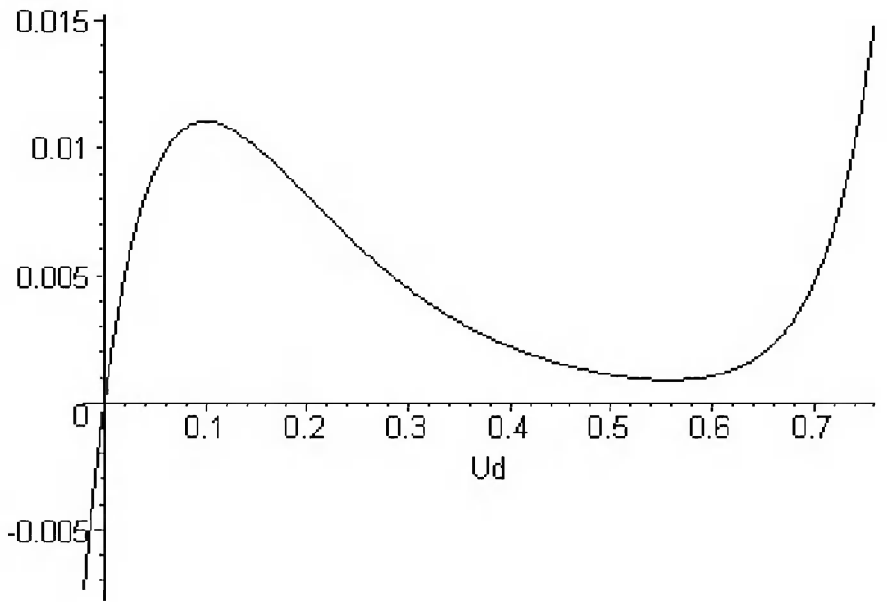


Рис. 15.39. N-образная ВАХ туннельного диода с участком отрицательной дифференциальной проводимости

Работа цепи описывается системой из двух дифференциальных уравнений:

$di/dt = (Es - i(t)*Rs - u(t))/L$,
$du/dt = (i(t) - Id(u(t))/C$.

Пусть задано Es=0,35 В, Rs=15 Ом, $C=10*10^{-12}$, $L=30*10^{-9}$ и максимальное время моделирования $tm=10*10^{-9}$. Итак, задаем исходные данные:

```
> Es:=.35:Rs:=15:C:=10*10^(-12):L:=30*10^(-6):tm:=10*10^(-9):
```

Составим систему дифференциальных уравнений цепи и выполним ее решение с помощью функции `dsolve`:

```
> se:=diff(i(t),t)=(Es-i(t)*Rs-u(t))/L,
     diff(u(t),t)=(i(t)-Id(u(t)))/C;
```

$$se := \frac{\partial}{\partial t}\mathrm{i}(t) = .1166666667\,10^{8} - 500000000\mathrm{i}(t) - \frac{100000000}{3}\mathrm{u}(t),$$

$$\frac{\partial}{\partial}\mathrm{u}(t) = 100000000000\mathrm{i}(t) - .3000000000\,10^{11}\,\mathrm{u}(t)\mathbf{e}^{(-10\mathrm{u}(t))} - 1000\mathbf{e}^{(20\mathrm{u}(t)-1)}$$

```
> F:=dsolve({se,i(0)=0,u(0)=0},{i(t),u(t)},type=numeric,
method=classical, stepsize=10^(-11), output=listprocedure);
```

$F := [t = \mathbf{(proc}(t) \ldots \mathbf{end\ proc)}, \mathrm{u}(t) = \mathbf{(proc}(t) \ldots \mathbf{end\ proc)},$
$\mathrm{i}(t) = \mathbf{(proc}(t) \ldots \mathbf{end\ proc)}]$

Поскольку заведомо известно, что схема имеет малые значения `L` и `C`, мы задали с помощью параметров достаточно малый шаг решения для функции `dsolve` – `stepsize=10^(-11)` с. При больших шагах возможна численная неустойчивость решения, искажающая форму колебаний, получаемую при моделировании. Используя функции `odeplot` и `display` пакета plots, построим графики решения в виде временных зависимостей `u(t)` и `10*i(t)` и линии, соответствующей напряжению `Es` источника питания:

```
> gu:=odeplot(F,[t,u(t)],0..tm,color=black,
     labels=[`t`,`u(t),10*i(t)`]):
> gi:=odeplot(F,[t,10*i(t)],0..tm,color=black):
> ge:=odeplot(F,[t,Es],0..tm,color=red):
> display(gu,gi,ge);
```

Эти зависимости представлены на рис. 15.40. Из них хорошо видно, что цепь создает автоколебания релаксационного типа. Их форма сильно отличается от синусоидальной.

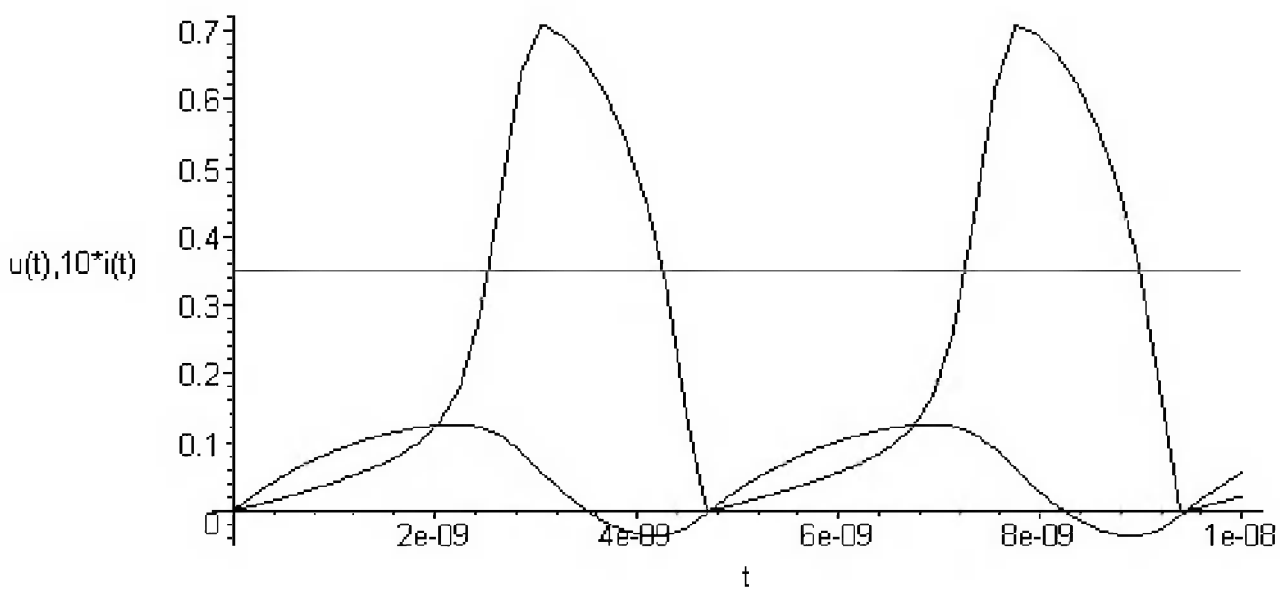


Рис. 15.40. Временные зависимости напряжения на туннельном диоде и тока

Решение можно представить также в виде фазового портрета, построенного на фоне построенных ВАХ и линии нагрузки резистора Rs:

```
> gv:=plot({Id(Ud),(Es-Ud)/Rs},Ud=-.05..0.75,color=black,
                                    labels=[Ud,Id]):
> gpp:=odeplot(F,[u(t),i(t)],0..tm,color=blue):
> display(gv,gpp);
```

Фазовый портрет колебаний показан на рис. 15.41.

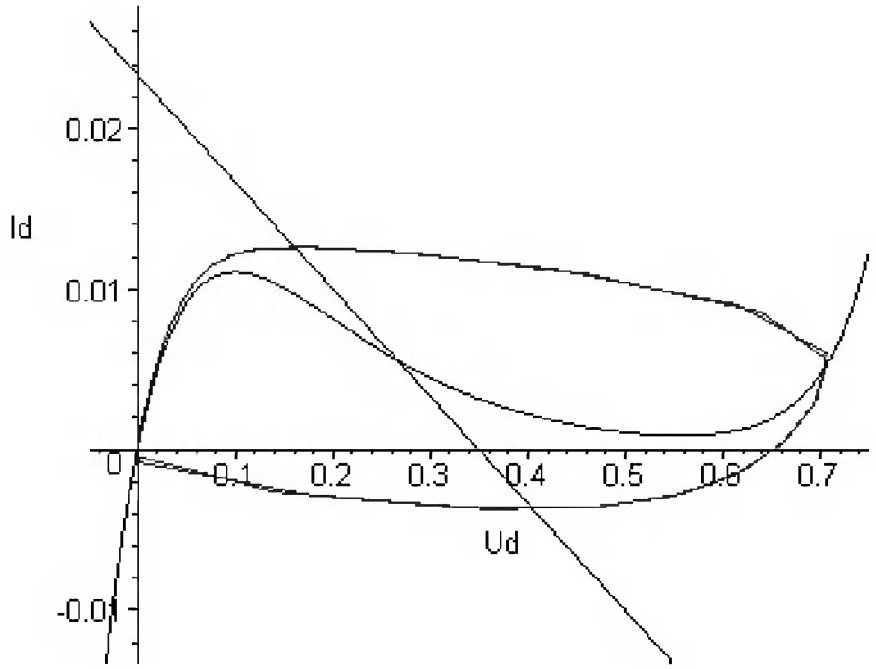


Рис. 15.41. Фазовый портрет колебаний на фоне ВАХ туннельного диода и линии нагрузки резистора Rs

О том, что колебания релаксационные, можно судить по тому, что уже первый цикл колебаний вырождается в замкнутую кривую – предельный цикл, форма которого заметно отличается от эллиптической [46, 81]. Хотя поставленная задача моделирования цепи на туннельном диоде успешно решена, в ходе ее решения мы столкнулись с проблемой обеспечения малого шага по времени при решении системы дифференциальных уравнений, описывающих работу цепи. При неудачном выборе шага можно наблюдать явную неустойчивость решения.

15.3.6. Моделирование детектора амплитудно-модулированного сигнала

Еще один пример, наглядно иллюстрирующий трудности моделирования существенно нелинейных систем и цепей, – детектирование амплитудно-модулированных сигналов. Простейший детектор таких сигналов представляет собой полупроводниковый диод, через который источник сигнала подключается к параллельной RC-цепи, выполняющей роль простого фильтра (без конденсатора C результат детектирования имел бы вид обрезанного снизу сигнала).

Диод имеет резко нелинейную вольтамперную характеристику. Ток через него равен:

$$Id = I0 \cdot (e^{v/0.05} - 1),$$

где v – напряжение на диоде, $I0$ – малый обратный ток диода.

Малое сопротивление диода в прямом направлении порождает жесткость дифференциального уравнения, описывающего работу детектора, и требует применения численных методов решения жестких дифференциальных уравнений. Заметим, что аналитического решения данная задача не имеет ввиду нелинейности дифференциального уравнения, описывающего работу детектора.

Документ, представленный на рис. 15.42, решает задачу моделирования детектора. В нем определено исходное дифференциальное уравнение и содержится его решение при заданных исходных данных – детектируется амплитудно-модулированный сигнал с амплитудой Um=5 В (размерные величины опущены), частотой несущей f=20 кГц, частотой модуляции F=1000 Гц и коэффициентом модуляции m=0.5. Определена вольтамперная характеристика диода при I0=1 мкА и построен ее график. Далее выполнено решение нелинейного дифференциального уравнения при R= 100 Ом и C=1 мкФ с помощью функции `dsolve` и построение графиков исходного сигнала и сигнала на выходе детектора (утолщенной линией).

Обратите внимание на то, что в функции `dsolve` явно указан метод Розенброка – один из лучших методов решения жестких дифференциальных уравнений.

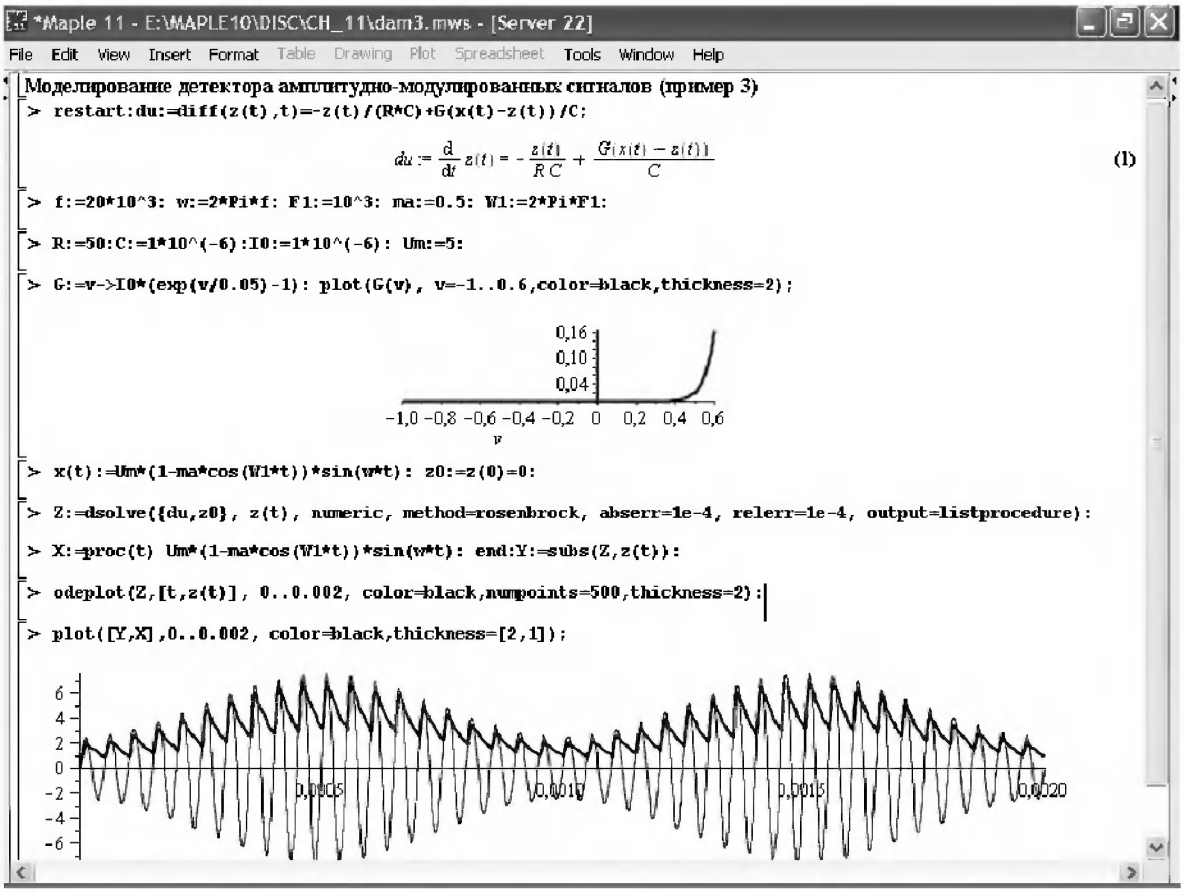


Рис. 15.42. Моделирование детектора амплитудно-модулированного сигнала (пример 1)

Кроме того, во избежание числовой неустойчивости, возможной даже при этом методе, решение задается с заданной абсолютной и относительной погрешностью 10^{-4}. Уменьшение погрешности лучше устраняет числовую неустойчивость, но ведет к увеличению времени моделирования.

15.4. Моделирование систем с заданными граничными условиями

15.4.1. Распределение температуры стержня с запрессованными концами

В некоторых случаях необходим учет заданных, чаще всего постоянных, *граничных условий*. Типичным примером этого является расчет временной и координатной зависимостей температуры нагретого стержня, запрессованного концами в области с постоянной температурой, – рис. 15.43. На нем дана математическая формулировка задачи, задание и решение дифференциального уравнения в частных производных с нулевыми граничными условиями. Температура вдоль стержня при t=0 задана выражением g(x).

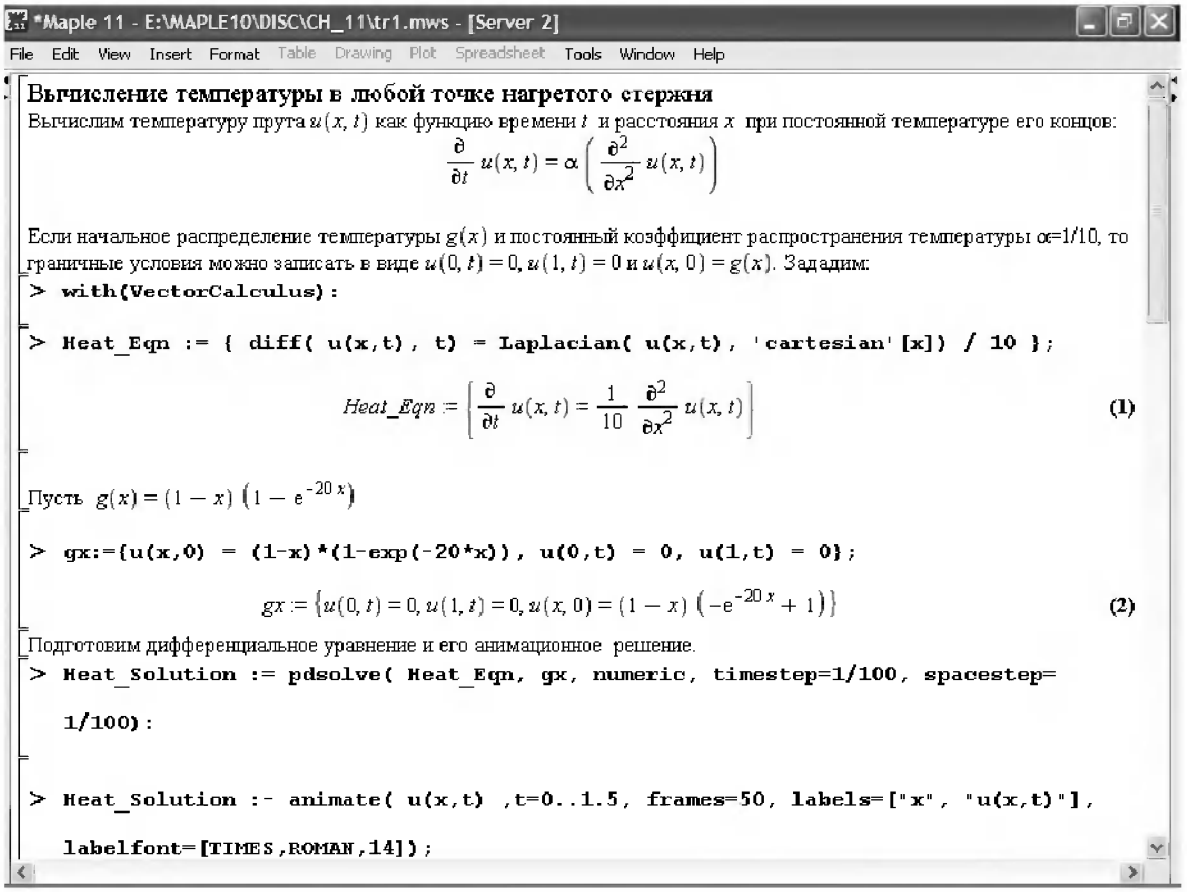


Рис. 15.43. Начало документа, вычисляющего распределение температуры вдоль стержня с нулевой температурой на концах

Рисунок 15.44 показывает результаты моделирования для задачи, представленной на рис. 15.43. Верхний рисунок анимационный и представляет начальный кадр – распределение температуры вдоль оси x при t=0. Если пустить анимацию, можно наблюдать в динамике процесс остывания стержня. Наглядное представление этого процесса в виде трехмерного графика показано ниже. Он представляет собой сплошной набор линий u(x,t) в различные моменты времени t. Нетрудно заметить, что отклонение температуры от 0 падает по мере роста t.

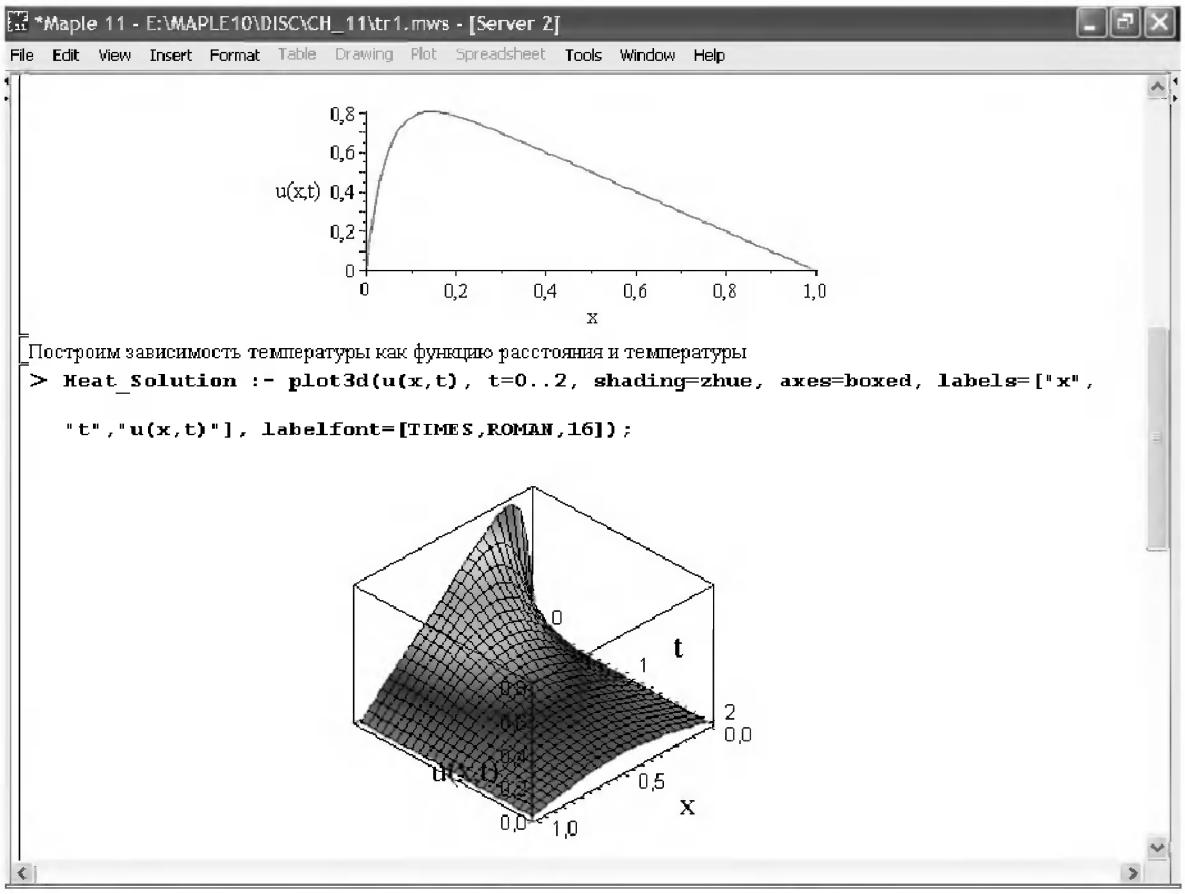


Рис. 15.44. Представление зависимости температуры u(x) в разные моменты времени: сверху – в виде анимационного рисунка, снизу – в виде трехмерного графика

Рисунок 15.45 показывает задание процедуры для вычисления значения температуры в численном виде для заданных x и t, а также дает еще один пример вычисления зависимости u(x,t) и построения анимационного графика этой зависимости. На графике рис. 15.45 представлен конечный кадр анимации.

15.4.2. Моделирование колебаний струны, зажатой на концах

Еще один классический пример решения дифференциального уравнения с заданными граничными условиями – это моделирование колебаний струны, зажатой на концах. Рисунок 15.46 демонстрирует начало документа, выполняющего такое

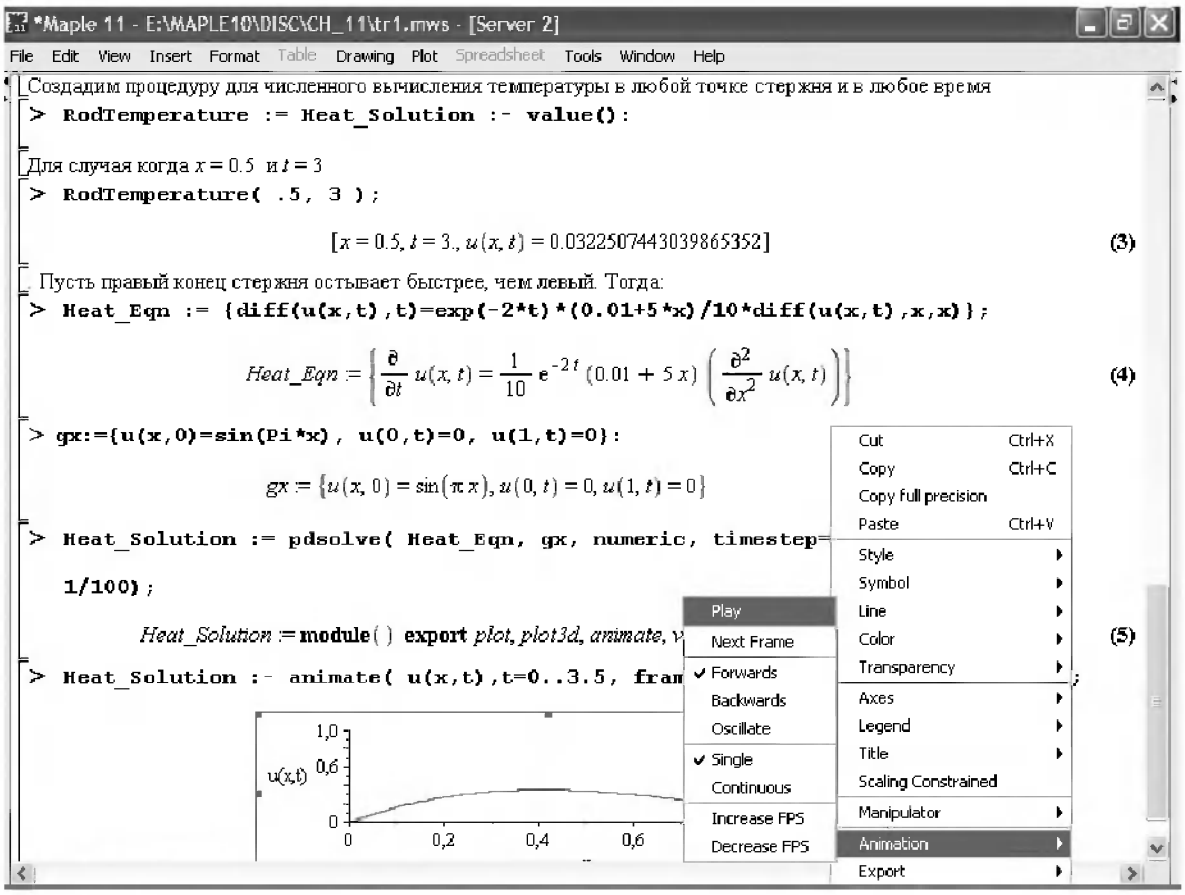


Рис. 15.45. Конец документа, представленного рис. 15.43 и 15.44

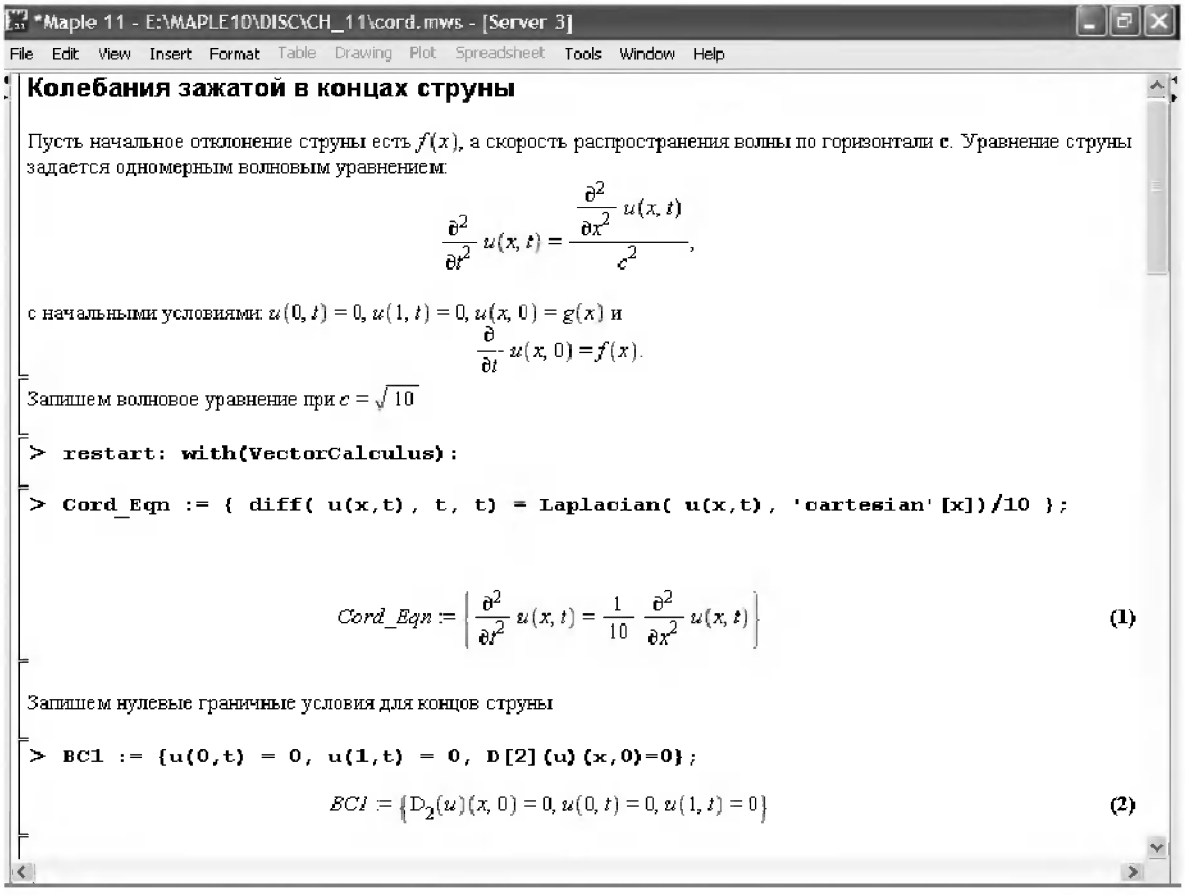


Рис. 15.46. Начало документа моделирования колебаний струны

моделирование. На нем представлены формулировка задачи, задание дифференциального уравнения и граничных условий для его решения.

На рис. 15.47 показан первый случай моделирования – струна оттянута в середине, так что распределение ее отклонения от расстояния x имеет характер вначале нарастающей линейно, а затем линейно уменьшающейся зависимости. Анимационный кадр второй по счету показывает, что после отпускания струны в центре появляется плоский участок, который расширяется и перемещается вниз. Формируется один период колебаний (положительный и отрицательный полупериоды).

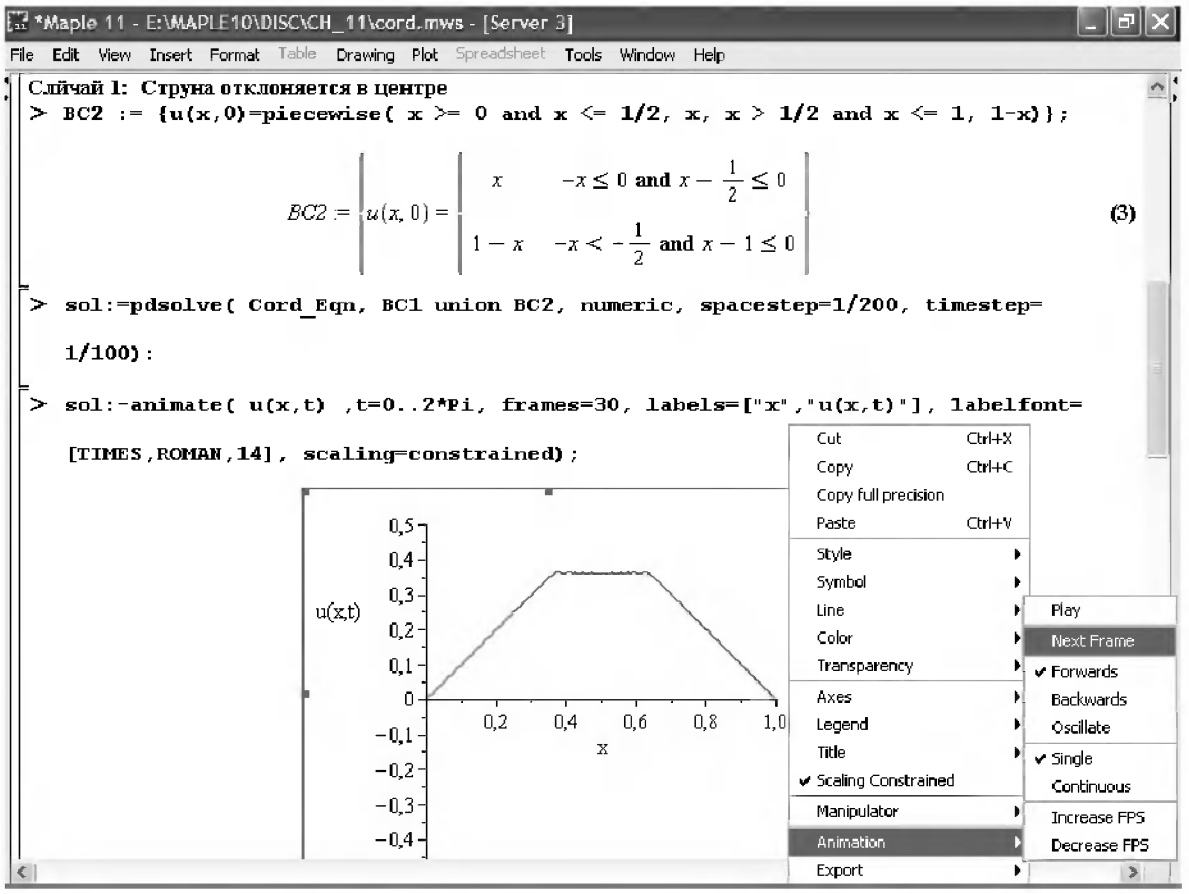


Рис. 15.47. Моделирование колебаний струны, оттянутой вверх посередине, после ее отпускания

Рисунок 15.48 показывает второй пример моделирований. На этот раз струна деформирована по синусоидальному закону, так что на ней укладываются три периода синусоиды. С момента начала моделирования можно наблюдать ее колебания, в ходе которых амплитуда синусоиды периодически то уменьшается, то увеличивается – режим стоячих волн. На рисунке представлен конечный кадр анимации.

Эту модель можно использовать и для моделирования колебания двух струн с более сложным характером начальной деформации. Такой случай представлен на рис. 15.49. Здесь представлен промежуточный кадр анимации.

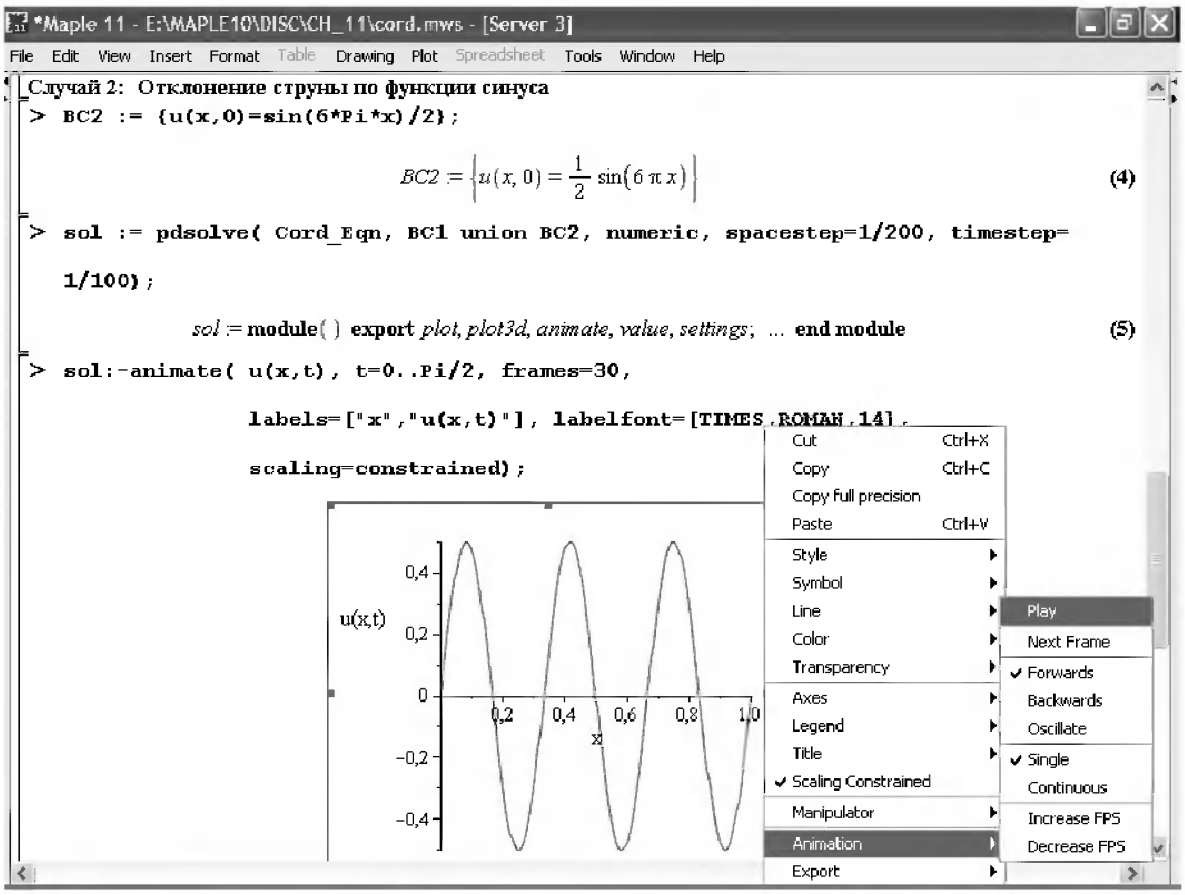


Рис. 15.48. Моделирование колебаний струны, деформированной по синусоидальному закону

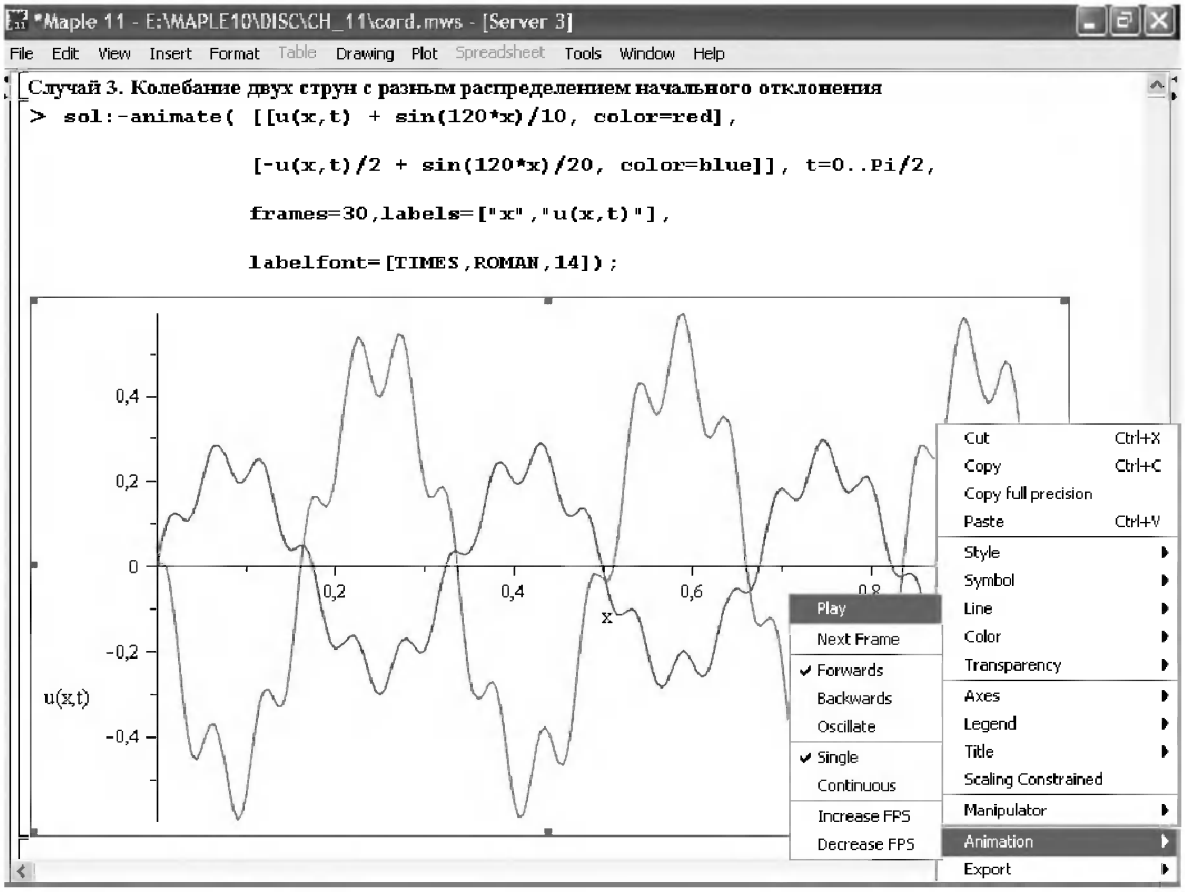


Рис. 15.49. Пример моделирования колебаний двух струн

15.5. Моделирование в системе Maple + MATLAB

15.5.1. Выделение сигнала на фоне шумов

Среди небольшого числа доступных функций системы MATLAB [98–118] в пакете Matlab нельзя не выделить особо функции быстрого прямого и обратного преобразований Фурье (БПФ). Покажем возможность применения БПФ на ставшем классическим примере – выделении спектра полезного сигнала на фоне сильных помех. Зададим некоторый двухчастотный сигнал, имеющий 1500 точек отсчета:

```
> num := 1500:
Time := [seq(.03*t, t=1..num)]:
data := [seq((3.6*cos(Time[t]) + cos(6*Time[t])), t=1..num)]:
plots[pointplot](zip((x,y)->[x,y],Time,data), style=line);
```

График сигнала представлен на рис. 15.50.

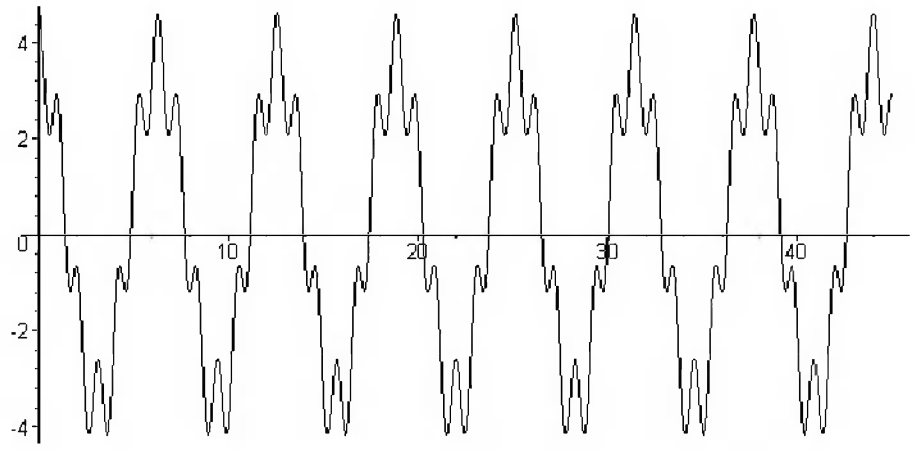


Рис. 15.50. График исходного сигнала

Теперь с помощью генератора случайных чисел наложим на этот сигнал сильный «шум» (слово «шум» взято в кавычки, поскольку речь идет о математическом моделировании шума, а не о реальном шуме физической природы):

```
> tol := 10000:
r := rand(0..tol):
noisy_data := [seq(r()/(tol)*data[t], t=1..num)]:
plots[pointplot](zip((x,y)->[x,y],Time,noisy_data), style=line);
```

Нетрудно заметить, что теперь форма сигнала настолько замаскирована шумом (рис. 15.51), что можно лишь с трудом догадываться, что сигнал имеет периодическую составляющую малой амплитуды. Эта высокочастотная составляющая сигнала скрыта шумом.

Подвергнем полученный сигнал (в виде временной зависимости) прямому преобразованию Фурье, реализованному функцией `fft`:

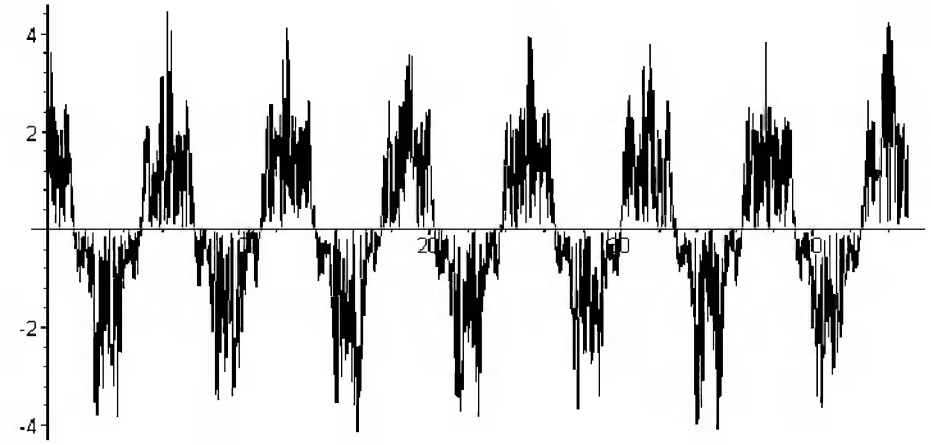


Рис. 15.51. Временная зависимость сигнала с шумом

```
> ft := fft(noisy_data):
> VectorOptions(ft, datatype);
```

$$complex_8$$

Эта операция переводит задачу из временного представления сигнала в частотное, что позволяет использовать частотные методы анализа сигнала. Выделим, к примеру, действительную и мнимую части элементов вектора `ft` и проверим его размер:

```
> real_part := map(Re, ft):
imag_part := map(Im, ft):
> dimensions(ft);
```

$$[1500]$$

Теперь выполним обычные операции вычисления спектра и зададим построение графика частотного спектра мощности сигнала:

```
> setvar("FT", ft);setvar("n", num);
> evalM("result = FT.*conj(FT)/n");
> pwr := getvar("result"):
> VectorOptions(pwr, datatype);
```

$$float_8$$

```
> pwr_list := convert(pwr, list):
> pwr_points := [seq([(t-1)/Time[num], pwr_list[t]], t=1..num/2)]:
> plots[pointplot](pwr_points, style=line);
```

Спектр сигнала представлен на рис. 15.52.

Из него отчетливо видно, что сигнал представлен двумя частотными составляющими с разной амплитудой. Они четко выделяются.

15.5.2. Моделирование линейного осциллятора

Теперь рассмотрим организацию совместной работы систем Maple 9 и MATLAB 6.5 SP1 (это широко распространенная, хотя и не новейшая версия данной системы) на примере моделирования механического осциллятора (маятника).

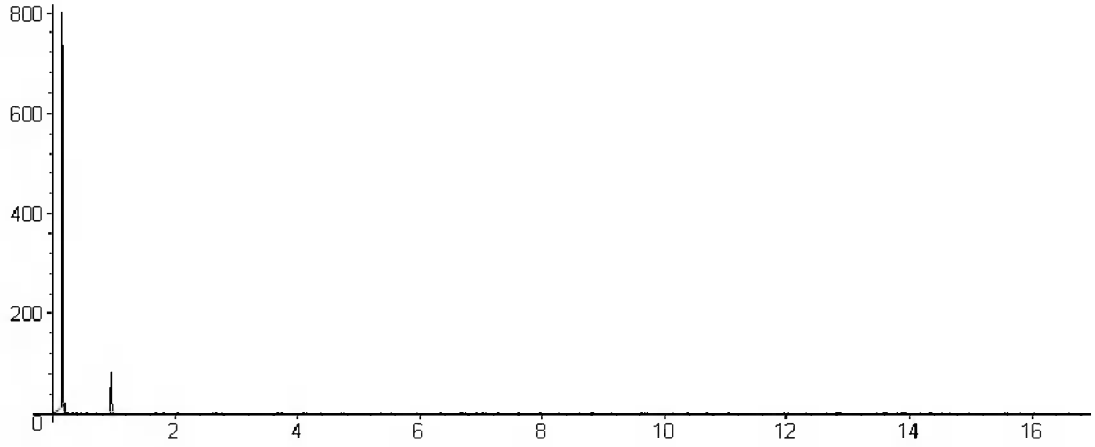


Рис. 15.52. Спектр сигнала рис. 15.51

Рисунок 15.53 показывает документ Maple 9.5, в котором решается эта задача. Уравнение маятника записывается средствами Maple в виде файла oscil.m в формате M-файлов системы MATLAB. Записываются также системные переменные, задающие массу маятника M, его затухание C и упругость K. Затем с помощью функции ode45 (решение ОДУ методом Рунге-Кутта–Фельберга порядка 4–5) находится временная зависимость отклонения маятника. Она построена внизу рисунка и отражает типичные затухающие синусоидальные колебания с заданными граничными условиями.

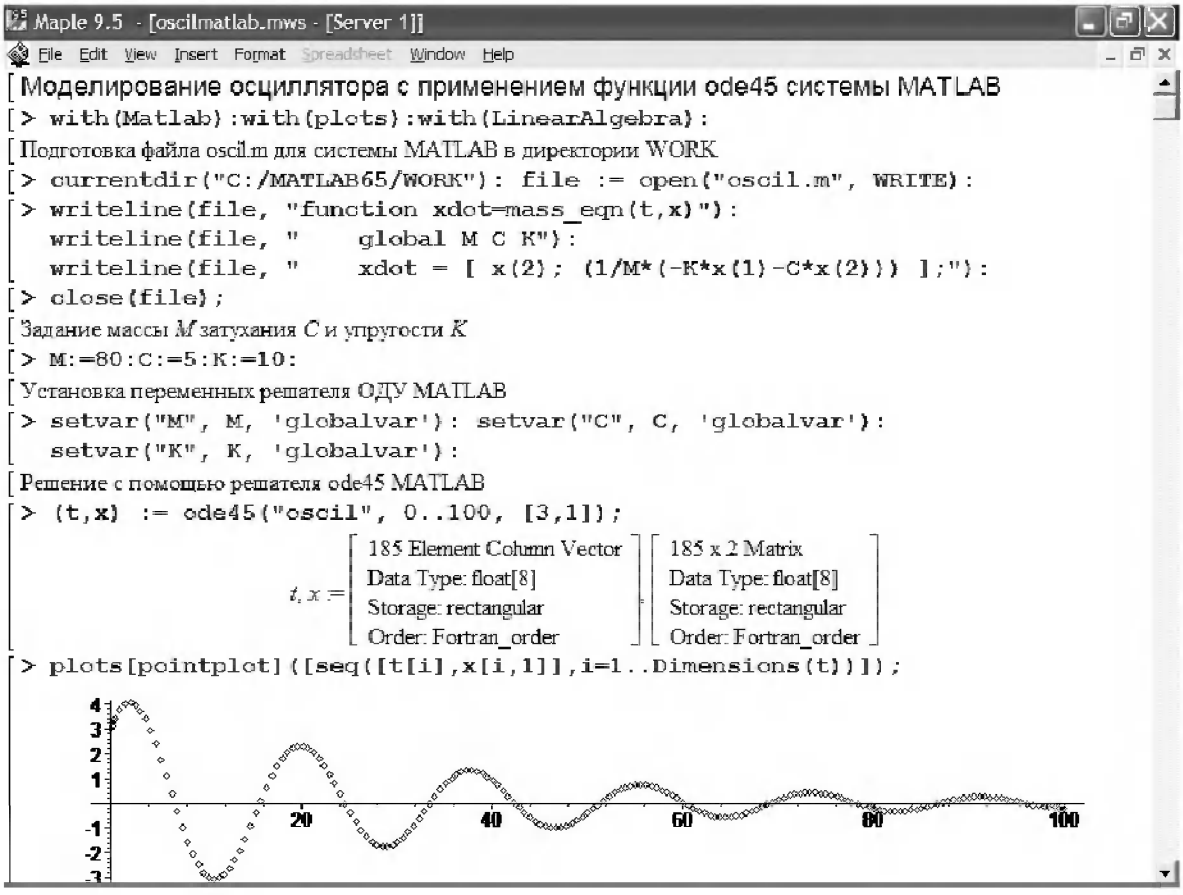


Рис. 15.53. Пример моделирования механического осциллятора в системе Maple 9.5 + MATLAB 6.5

Разумеется, для такой относительно простой задачи привлечение подобной мощной и громоздкой системы, как MATLAB, имеет только познавательный смысл. Эта задача, что было показано выше, легко решается средствами системы Maple. Однако технология совместного использования систем Maple и MATLAB на этом примере хорошо видна и может применяться пользователями для решения существенно более сложных задач математического моделирования.

15.6. Моделирование *RLC*-цепи с применением Maplets-интерфейса

15.6.1. Подготовка процедуры моделирования и тестового примера

Теперь рассмотрим пример на моделирование последовательной *RLC*-цепи, подключенной к источнику напряжения с заданной произвольно временной зависимостью $v(t)$. Наша задача заключается в нахождении тока $i(t)$ из решения системы из двух дифференциальных уравнений заряда:

$$C\frac{dq(t)}{dt}+R\frac{dq(t)}{dt}+\frac{1}{C}q(t)=v(t),\ i(t)=\frac{dq(t)}{dt},$$

где $q(t)$ – временная зависимость заряда конденсатора C и $i(t)$ – искомая временная зависимость тока в цепи.

Maple-процедура lrc , позволяющая вычислять i(t) по этой системе дифференциальных уравнений, представлена ниже:

```
> restart;
> lrc := proc(L, R, C, q0, i0, tf, v)
  local de, ics, sol, q, i, p;
  de  :=L*diff(q(t),t,t)+ R*diff(q(t),t)+(1/C)*q(t)= v;
  ics :=q(0) = q0,D(q)(0)= i0;
  sol :=dsolve( {de, ics},q(t), range=0..tf,numeric);
  plots[odeplot](sol,[[t,v,color=red],[t, diff(q(t),t),
  color=blue]],
  t=0..tf,legend=["v(t)","i(t)"],numpoints=1000);
end proc:
```

Подготовим тестовый пример. Пусть L=250 mH, C=500 mF, R=100 mOhm, v(t)=sin(10*t)*exp(t/2), при нулевых начальных условиях и интервале времени от 0 до 5 обращение к процедуре lrc имеет вид

```
> lrc(.25,.1,.5, 0,0, 5, sin(10*t)*exp(-t/2));
```

и ведет к построению графика переходных процессов – v(t) и i(t), показанного на рис. 15.54.

Нетрудно заметить, что переходные процессы достаточно сложны, хотя и вполне понятны читателю, разбирающемуся в радиотехнических цепях [161, 162].

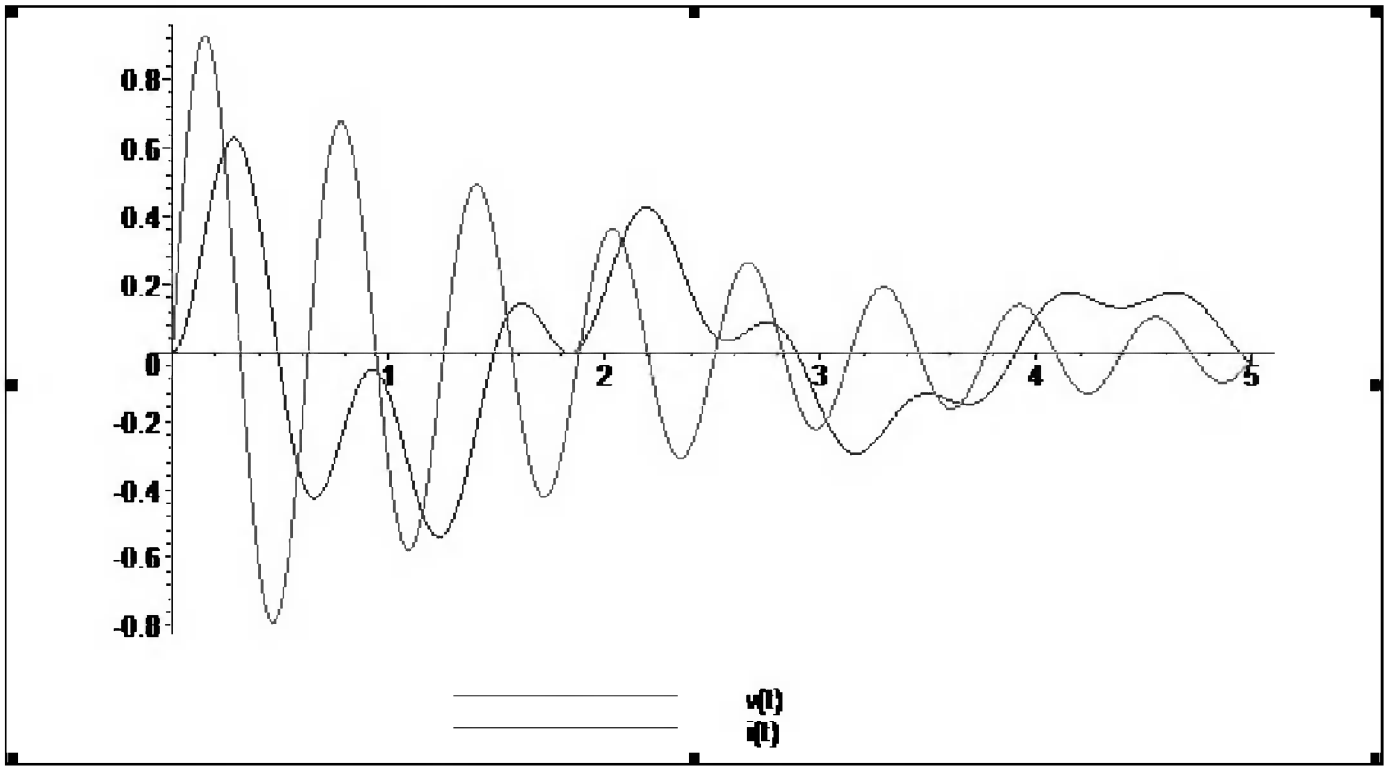


Рис. 15.54. Временные зависимости v(t) и i(t) при моделировании LCR-цепи

15.6.2. Подготовка окна Maplets-интерфейса

Теперь зададимся целью построить окно Maplets-интерфейса, имеющего следующие детали:

- поле для задания индуктивности L в mH;
- слайдеры для задания резистивности R в mOhm и емкости C в mF;
- поля для задания начальных значений $q0$ и $i0$ и конечного времени t;
- поле для задания временной зависимости $i(t)$ по умолчанию sin(10*t);
- кнопки **Plot** для построения графиков временных зависимостей $v(t)$ и $i(t)$ и **Close** для закрытия окна;
- подокно для отображения графиков временных зависимостей $v(t)$ и $i(t)$.

Поскольку построение Maplets-интерфейса уже было подробно описано, приведем процедуру lrc_maplet, реализующую эти возможности:

```
> lrc_maplet := proc()
local OPTIONS, COMMAND, WINDOW, MAPLET,LINE1,LINE2,LINE3, LINE4,
LINE5,       LINE6, L, R, C, q0, i0, tf, v;
use Maplets, Maplets[Elements] in L,R,C,q0,i0,tf:= 1/10,1/10,1/
10,0,0,10;
   v      := sin(10*t);
   OPTIONS:= title="RLC Circuit Simulator";
   COMMAND:= Evaluate( function="lrc_simulate" );
   LINE1:= "L(mH):",TextBox[L_](value=L*1000,
   onchange=COMMAND );
   LINE2  := "R (mOhm): ",Slider[R_]
```

```
   ( value=R*1000,lower=0,upper=1*1000, majorticks=100,
   minorticks=10,filled=true,onchange=COMMAND);
   LINE3  := "C (mF): ", Slider[C_]( value=C*1000, lower=0,
   upper=1*1000,majorticks=100, minorticks=10,filled=true,
   onchange=COMMAND);
   LINE4  := "q0: ",TextBox[q0_]( value=q0,onchange=COMMAND),
   "i0: ", TextBox[i0_]( value=i0, onchange=COMMAND ),"tf: ",
   TextBox[tf_]( value=tf, onchange=COMMAND );
   LINE5   := "v(t): ",TextBox[v_](value=v,onchange=COMMAND),
   Button("Plot", COMMAND), Button("Close", Shutdown());
   LINE6   := Plotter[p_]();
   WINDOW  := Window[W_](OPTIONS,[[LINE1],[LINE2],[LINE3],
   [LINE4],[LINE5],[LINE6]]);
   MAPLET  := Maplet(WINDOW);  Display(MAPLET);
end use;
end proc:
```

15.6.3. Организация связи между процедурой моделирования и Maplets-интерфейсом

Следующая процедура служит для связи между процедурой моделирования RLC-цепи и процедурой задания Maplets-окна:

```
> lrc_simulate := proc()
local L, R, C, q0, i0, tf, v, p;
use Maplets[Tools] in
   L  := Get(  L_(value) :: algebraic, corrections=true );
   R  := Get(  R_(value) :: algebraic, corrections=true );
   C  := Get(  C_(value) :: algebraic, corrections=true );
   L, R, C := (L, R, C)/1000; # преобразование mH -> H, etc.
   q0 := Get( q0_(value) :: algebraic, corrections=true );
   i0 := Get( i0_(value) :: algebraic, corrections=true );
   tf := Get( tf_(value) :: algebraic, corrections=true );
   v  := Get(  v_(value) :: algebraic, corrections=true );
   p := lrc(L, R, C, q0, i0, tf, v);
   Set( p_(value) = p );
end use;
end proc:
```

В эту процедуру включены проверки на алгебраичность вводимых с Maplets-окна параметров.

15.6.4. Моделирование RLC-цепи в окне Maplets-интерфейса

Теперь все готово к началу моделирования *RLC*-цепи с применением Maplets-интерфейсного окна. Для этого достаточно исполнить команду

```
> lrc_maplet();
```

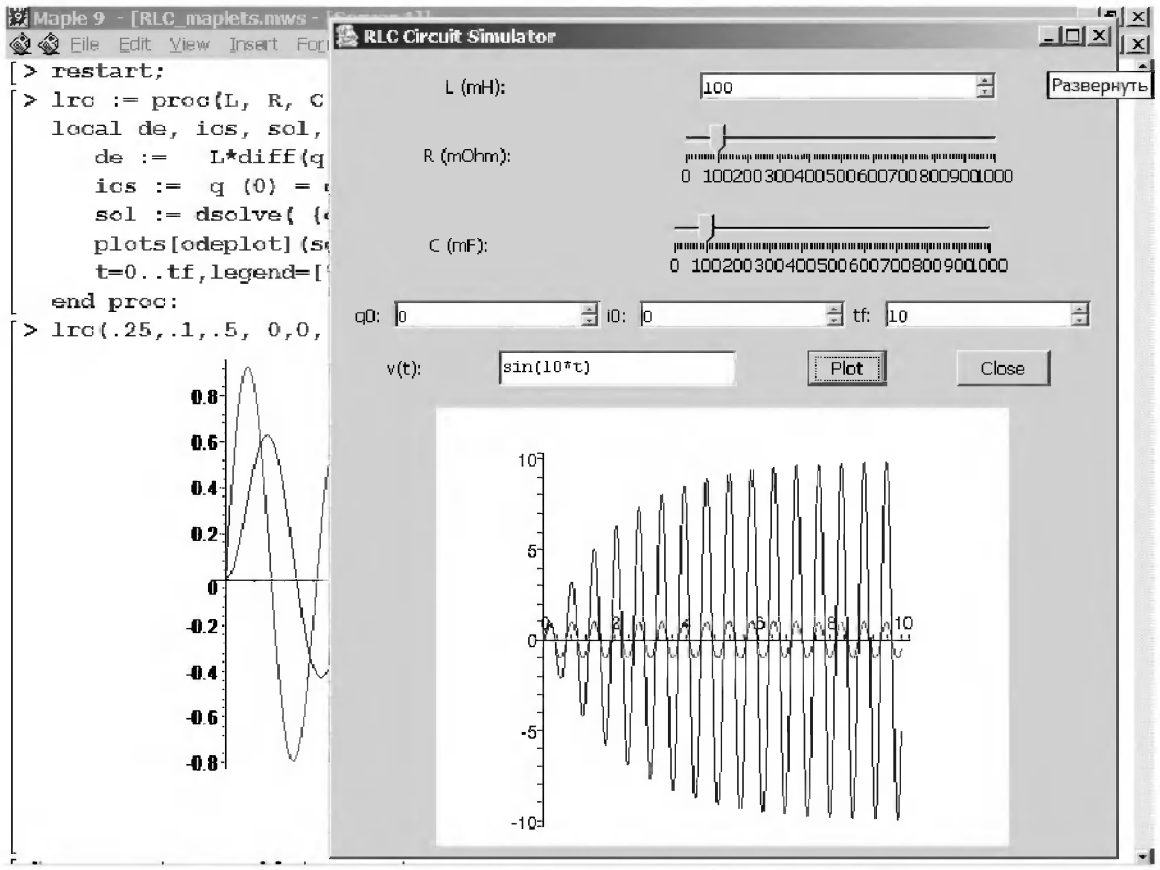


Рис. 15.55. Моделирование RLC-цепи с параметрами, заданными по умолчанию

Появится окно, представленное на рис. 15.55, поначалу с пустым подокном вывода графиков. Графики, показанные на рис. 15.55, появятся после активизации кнопки `Plot`.

При параметрах, взятых по умолчанию, частота собственных колебаний RLC-контура близка к частоте синусоидального воздействия, и наблюдаются нарастающие, почти синусоидальные колебания тока – случай сам по себе интересный, хотя и хорошо известный.

А теперь зададим в окне данные для тестового примера. Для этого изменим значения L, C (R остается прежним) и конечное время tf, а также временную зависимость $v(t)$, добавив в нее экспоненциальный член. Запустив моделирование кнопкой **Plot**, получим новый рис. 15.56. Сравнив его с тестовым примером (рис. 15.55), убеждаемся в полной идентичности расчетных переходных процессов.

Следует отметить, что кнопка **Plot** должна нажиматься только при изменении параметров, вводимых в полях. При перемещении слайдеров для R и C перестройка графиков происходит автоматически. Это позволяет наглядно оценивать переходные процессы при плавном изменении этих параметров. На рис. 15.57 показан случай, когда движком слайдера значительно уменьшена емкость C, что привело к близости частот синусоидальной компоненты входного сигнала и собственной частоты контура. В итоге получен еще один интересный вариант переходного процесса – вначале амплитуда ставших почти синусоидальными колебаний тока нарастает, но затем падает (из-за экспоненциального уменьшения входного напряжения).

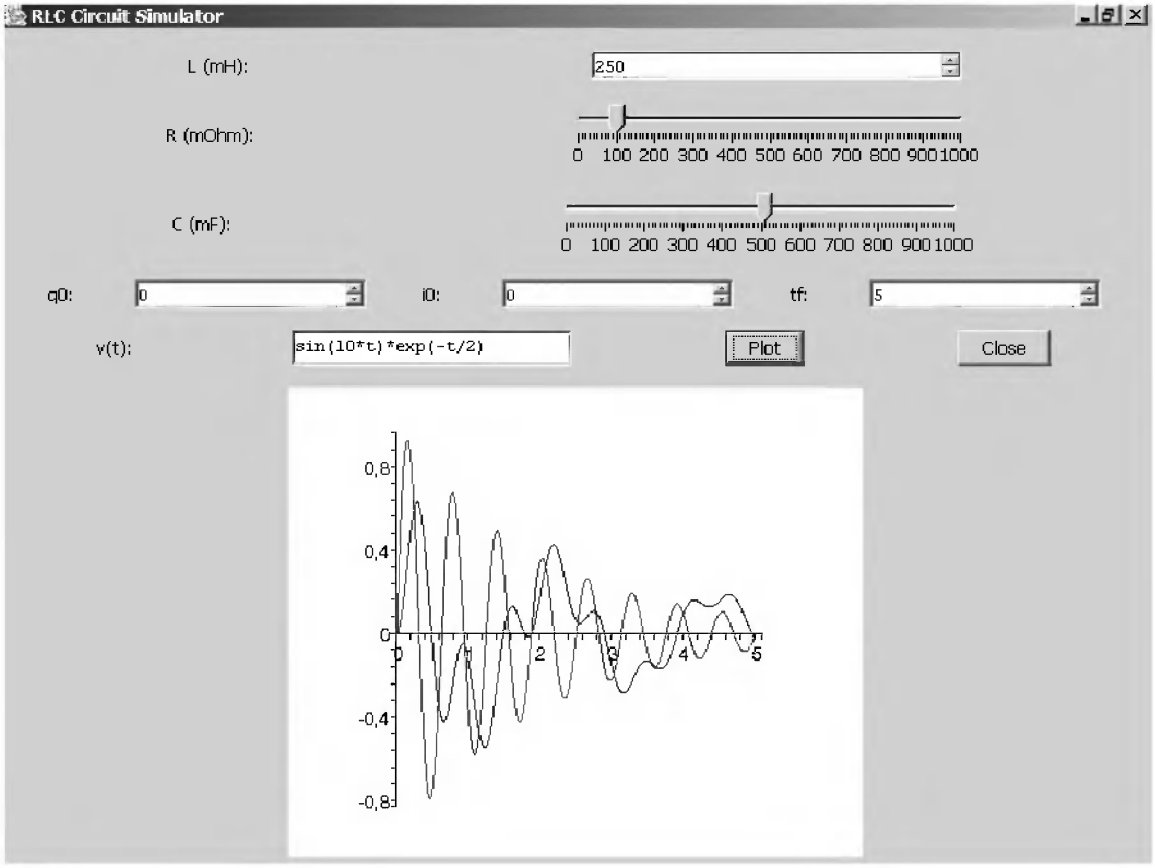


Рис. 15.56. Моделирование RLC-цепи в Maplets-окне с параметрами тестового примера

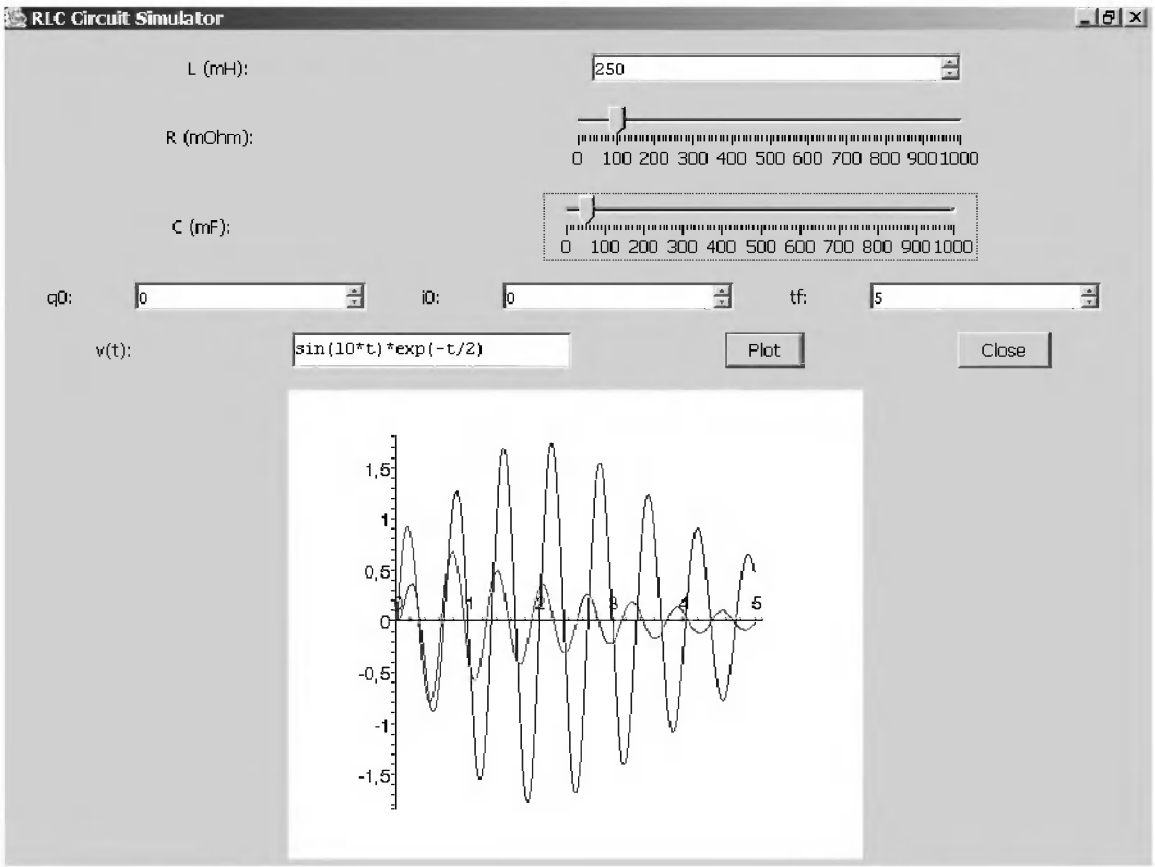


Рис. 15.57. Моделирование RLC-цепи в Maplets-окне с уменьшенным значением емкости C

Нетрудно заметить, что моделирование *RLC*-цепи в интерфейсном Maplets-окне весьма наглядно. По этому и описанным ранее примерам читатель может легко конструировать свои Maplets-окна и готовить в Maple 9 программные утилиты вполне профессионального качества.

15.7. Инженерные методы спектрального анализа в СКМ Mathematica 4/5

В решении задач электро- и радиотехники важное место занимают инженерные методы анализа сигналов, например реакции нелинейных цепей и устройств, таких как усилители, на синусоидальное воздействие. Нелинейность цепей ведет к искажению синусоидального входного сигнала и появлению в спектре выходного сигнала новых гармоник. Ниже описана реализация некоторых методов инженерного спектрального анализа средствами СКА Mathematica. Особое внимание уделено визуализации этих методов.

15.7.1. Метод пяти ординат

Метод пяти ординат – это инженерный метод расчета коэффициента нелинейных искажений в системах со слабой нелинейностью. Суть метода такова. Пусть на некоторую систему с передаточной характеристикой u(x), где x – входное воздействие, действует гармонический сигнал вида x[t]=X0+Cos[ωt], где X0 – постоянная составляющая сигнала, A – амплитуда сигнала, ω – угловая частота. Интервал измерения x от X0–A до X0+A можно разбить на 4 одинаковых подынтервала с шириной 0.5*A и таким образом получить пять равноотстоящих отсчетов входного сигнала. По ним вычисляются пять ординат функции u[x], а уже по ним – амплитуды 4 гармоник и постоянная составляющая выходного сигнала y[t]. Это позволяет найти коэффициент нелинейных искажений kg. Итак, схема реализации метода следующая:

x(t)=X0+A*cos(ω*t) → u(x) → yi → kg.

Ниже представлено задание трех видов зависимости u(x):

```
Clear[u]
u[x_] = N[-Tanh[x - 2] + √ (x + 10) / 10 ];
uu[x_]=N[x*Exp[-x/5]];
uu[x_]=N[1-Tanh[x-2]^3];
```

Только одна из этих зависимостей u[x_] используется в расчетах. Замените имя другой зависимости на u, например записав для второй зависимости вместо uu только u. Тогда вы сможете просмотреть реализацию метода для второй зависимости. Вы можете также задать любую свою зависимость.

Теперь зададим постоянную составляющую входного сигнала и его амплитуду:

```
X0:=2;A:=1.5;
```

Вектор из пяти равноотстоящих отсчетов в Mathematica можно получить следующим образом:

```
d=Table[u[X0+A*(3-i)/2],{i,5}]
  {0.256747,0.49401,1.09545,1.69581,1.92984}
```

Построим график u[x] и отсчетов (рис. 15.58):

```
<<Graphics`Graphics`
DisplayTogether[Plot[u[x],{x,0,5},PlotRange→{0,2},PlotStyle→{Hue[.7]}],
GeneralizedBarChart[{{X0+A,d[[1]],.05},{X0+A/
2,d[[2]],.05},{X0,d[[3]],.05},{X0-A/2,d[[4]],.05},{X0-
A,d[[5]],.05}},PlotRange→{{0,5},{0,2}}]]
```

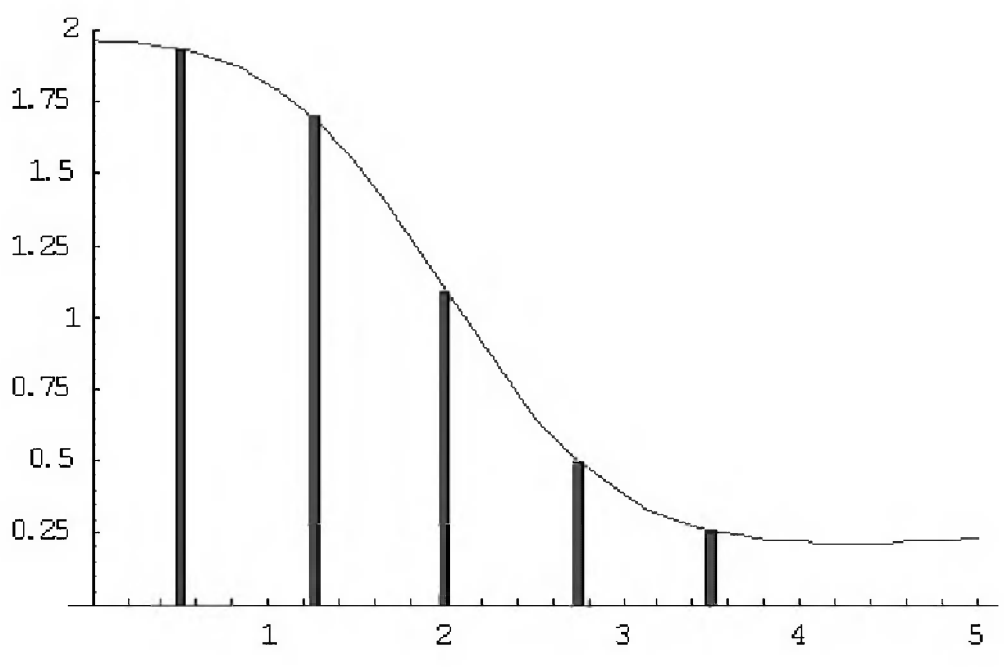


Рис. 15.58. Зависимость u[x]

Записав для каждой ординаты тригонометрический ряд Фурье, можно получить систему из пяти линейных уравнений, решение которой дает постоянную составляющую зависимости y[t] – y0 и амплитуды первых четырех гармоник y1, y2,y3 и y4. Ниже представлен их расчет по известным готовым формулам:

```
y0=(d[[1]]+d[[5]]+2*d[[2]]+2*d[[4]])/6
  1.09437
y1=(d[[1]]-d[[5]]+d[[2]]-d[[4]])/3
  -0.958299
y2=(d[[1]]+d[[5]]-2*d[[3]])/4
  -0.00107504
y3=(d[[1]]-d[[5]]-2*d[[2]]+2*d[[4]])/6
  0.12175
y4=(d[[1]]+d[[5]]-4*d[[2]]-4*d[[4]]+6*d[[3]])/12
  -1.3194 × 10^-6
```

Теперь можно вычислить коэффициент гармоник:

```
Kg = Sqrt[y2^2 + y3^2 + y4^2] / Abs[y1]
  0.127053
```

Для получения коэффициента гармоник в процентах вычисленное значение умножается на 100:

```
Kg*100
 12.7053
```

Полученные расчетом постоянную составляющую сигнала y0 и амплитуды гармоник y1, y2, y3 и y4 можно использовать для синтеза зависимости y[t] или y[x] в обобщенном виде (ω*t=x):

```
y[x_]=y0+y1*Cos[x]+y2*Cos[2*x]+y3*Cos[3*x]+y4*Cos[4*x];
```

Это позволит построить график, на котором одновременно построены точная зависимость y[x], полученная как u[X0+A*Sin[x]], и приближенная зависимость y[x], полученная с помощью ряда Фурье. Ниже дано построение совмещенных графиков этих зависимостей (рис. 15.59):

```
Plot[{u[X0+A*Cos[x]],y[x]},{x,0,4π},PlotRange→{0,2},
PlotStyle→{Hue[.7],Hue[0]}]
```

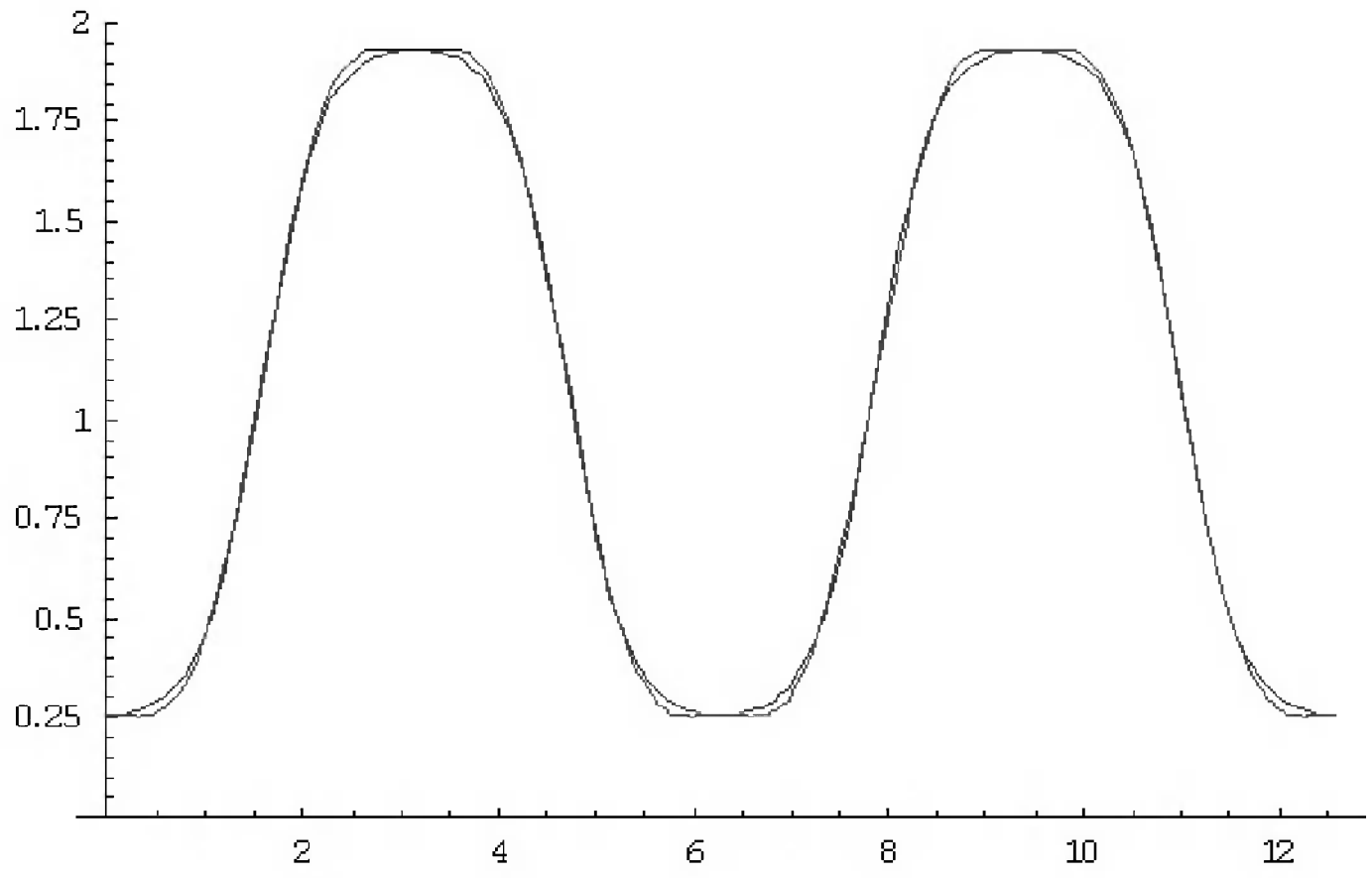


Рис. 15.59. Выходной сигнал и его синтезированная модель

Мы также можем вычислить и представить графически относительную погрешность приближения реальной y[x] ее тригонометрическим рядом и построить график этой погрешности в интервале x от 0 до 2π (то есть в пределах периода косинусоиды на входе нелинейной системы). Для этого воспользуемся командами:

```
δ[x_]:=(u[X0+A*Cos[x]]-y[x])/A
Plot[δ[x],{x,0,2* π}]
```

Полученный график погрешности представлен на рис. 15.60.

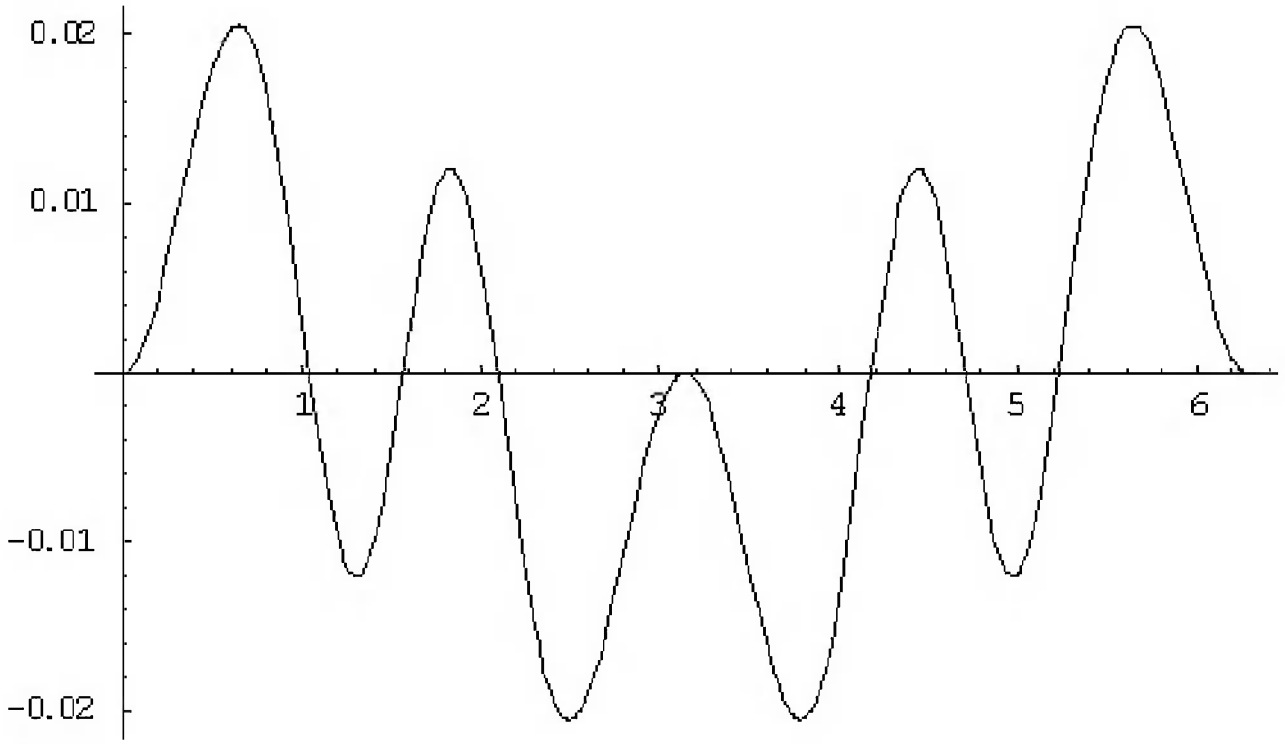


Рис. 15.60. График погрешности спектрального анализа методом пяти ординат

Теперь рассмотрим новую модификацию этого метода, позволяющую строить зависимости коэффициента гармоник от амплитуды входного воздействия и величины постоянной составляющей входного сигнала. Для этого будем задавать передаточную характеристику произвольным по числу набором координат ее точек (xi,ui). Равномерность расположения узловых точек при этом не обязательна. Ниже представлена реально снятая передаточная характеристика каскада на мощном полевом транзисторе (x – напряжение на затворе, u – напряжение на стоке). Она задана массивом даных **data**:

```
data={{-2,48},{3,46.2},{7,38.9},{10,31},{13,23},{16,17},
{20,13}, {25,12.2}};
```

Используя функцию интерполяции Interpolation, представим передаточную характеристику в виде непрерывной функции, аппроксимируемой полиномом пятой степени (эксперименты с другой степенью полинома поощряются):

```
u:=Interpolation[data,InterpolationOrder→5]
```

Построим график этой зависимости вместе с исходными точками (рис. 15.61):

```
DisplayTogether[{Plot[u[x],{x,-
2,25}],ListPlot[data,PlotStyle→{PointSize[.02]}]},PlotRange→{0,50}]
```

Теперь зададим функцию пользователя – процедуру для вычисления коэффициента гармоник методом пяти ординат для произвольно заданных X0 и A. Поскольку использована интерполяция для зависимости u[x], то расположение пяти отсчетов для вычисления амплитуд гармоник не связано с узловыми точками

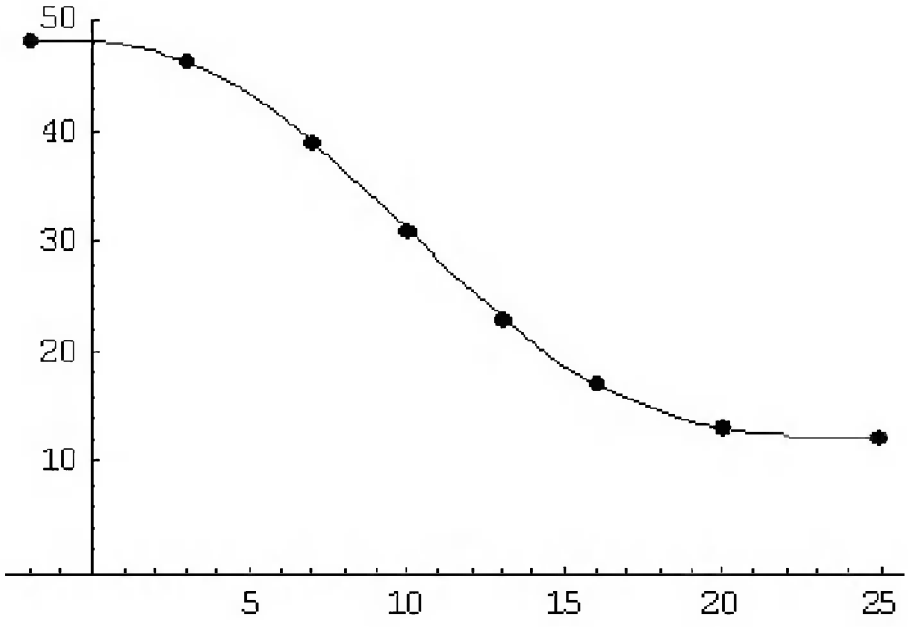


Рис. 15.61. Аппроксимация передаточной функции полиномом

этой зависимости. Ниже представлена процедура, задающая реализацию метода 5 ординат для данного случая:

```
kg[X0_, A_] := Block[{d, y0, y2, y1, y3, y4},
                     d = Table[u[X0 + A* (3 - i) / 2], {i, 5}];
                     y0 = (d[[1]] + d[[5]] + 2*d[[2]] + 2*d[[4]]) / 6;
                     y1 = (d[[1]] - d[[5]] + d[[2]] -d[[4]]) / 3;
                     y2 = (d[[1]] + d[[5]] - 2*d[[3]]) / 4;
                     y3 = (d[[1]] - d[[5]] - 2*d[[2]] + 2*d[[4]]) / 6;
                     y4 = (d[[1]] + d[[5]] - 4*d[[2]] - 4*d[[4]] + 6*d[[3]]) / 12;
                     Sqrt[y2^2 + y3^2 + y4^2] / Abs[y1]                    ]
```

Ее можно использовать для вычисления коэффициента гармоник

```
{kg[10,5],kg[10,8],kg[12,6]}
 {0.0318136,0.0771222,0.0914766}
```

и построения графика зависимости kg от A для разных X0. Пример построения такого графика для X0=10, 11 и 12 и A от 0 до 10 представлен ниже (рис. 15.62):

```
Plot[{kg[8,A],kg[9,A],kg[10,A]},{A,0,10},PlotStyle→{Hue[1],
Hue[.6],Hue[.75]},PlotLegend→{"X0=8","X0=9","X0=10"},
LegendPosition→{1,0}]
```

15.7.2. Спектральный анализ методом Берга

В резонансных усилителях радиопередающих устройств широко используются гармонические сигналы с отсечкой. Отсечка задается углом θ, который характеризует ту часть периода $[0,2\pi]$, во время которой сигнал проходит через нелинейный

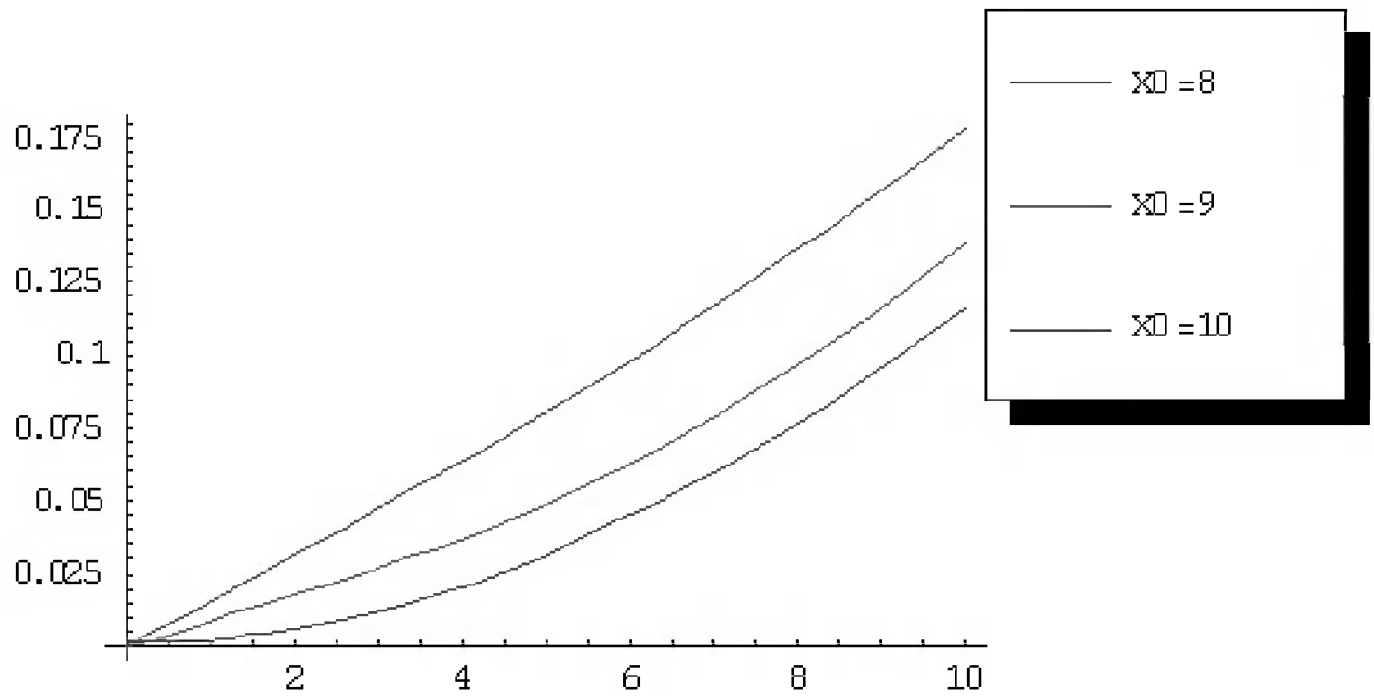


Рис. 15.62. Зависимость kg от A для разных X0

усилитель (или умножитель частоты). Ниже представлена форма косинусоидального сигнала с углом отсечки около $\pi/3$ (рис. 15.63):

```
<<Graphics`FilledPlot`
f[x_]:=If[Cos[x]>1/2,Cos[x]-1/2,0]
FilledPlot[{f[x],0}, {x, -2 π, 2 π}]
```

Функция **FindRoot** позволяет легко найти значение угла θ при заданной амплитуде A отсеченного косинусоидального импульса из решения уравнения Cos[θ]==A. Например, если A=0.5, то:

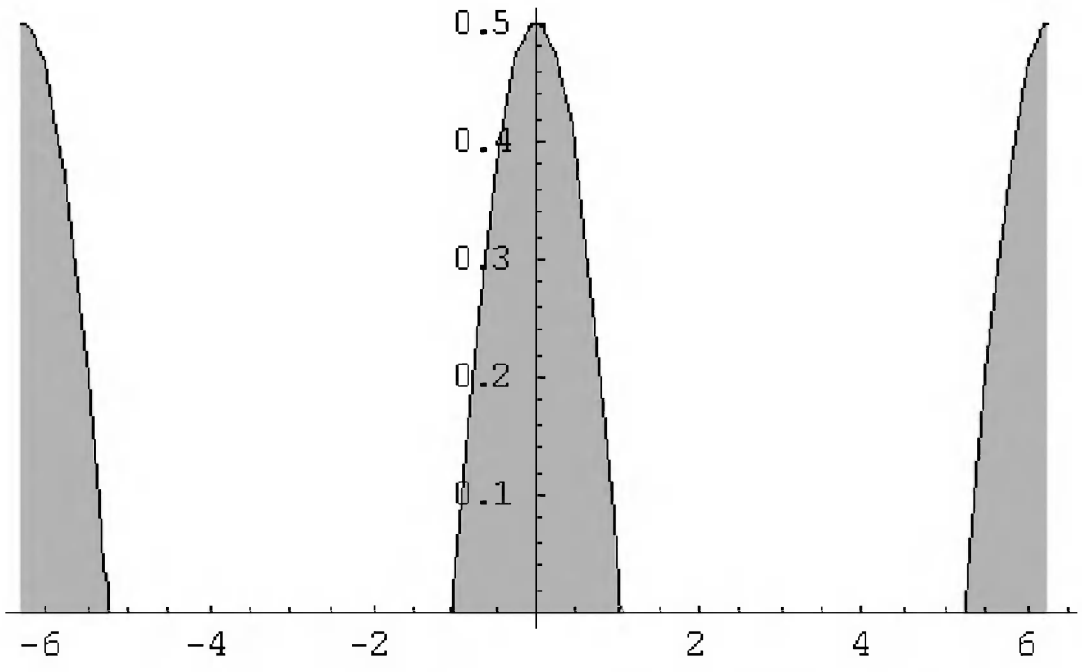


Рис. 15.63. Верхушки косинусоидального сигнала

```
FindRoot[Cos[q]==.5,{q,1}]
```

$$\left\{\frac{\pi}{3} \to 1.0472\right\}$$

Если fm – амплитуда отсеченной косинусоидальной функции (например, тока активного прибора), то отношение амплитуды n-ой гармоники к fm, то есть приведенная к интервалу [0,1] относительная амплитуда, выражается известными коэффициентами Берга:

$$\alpha 0[\theta\_] := \frac{\mathbf{Sin}[\theta] - \theta * \mathbf{Cos}[\theta]}{\pi * (1 - \mathbf{Cos}[\theta])}$$

$$\alpha 1[\theta\_] := \frac{\theta - \mathbf{Sin}[\theta] * \mathbf{Cos}[\theta]}{\pi * (1 - \mathbf{Cos}[\theta])}$$

$$\alpha n[\theta\_, n\_] := 2 * \frac{\mathbf{Sin}[n*\theta] * \mathbf{Cos}[\theta] - n * \mathbf{Cos}[n*\theta] * \mathbf{Sin}[\theta]}{\pi * n * (n^2 - 1) * (1 - \mathbf{Cos}[\theta])}$$

Выразим их единым выражением α[θ,n]:

```
α12[q_,n_]:=If[n==1,a1[q],a0[q]]
α[q_,n_]:=If[n>1,N[an[q,n]],N[a12[q,n]]]
```

Приведем пример вычисления коэффициентов Берга для постоянной составляющей отсеченной косинусоиды и трех первых гармоник при θ=π/3 (или 180/3=60 градусов):

```
θ:=π/3;{α[θ,0],α[θ,1],α[θ,2],α[θ,3]}
 {0.217996,0.391002,0.275664,0.137832}
```

Построим графики зависимости α[θ,n]от θ при n=0,1,2 и 3 (рис. 15.64):

```
<<Graphics`Legend`
Plot[{α[θ,0],α[θ,1],α[θ,2],α[θ,3]},{θ,0,π},PlotStyle→{Hue[1],
Hue[.6],Hue[.75],Hue[.8]},PlotLegend→{"n=0","n=1","n=2","n=3"},
LegendPosition->{1,-.3}]
```

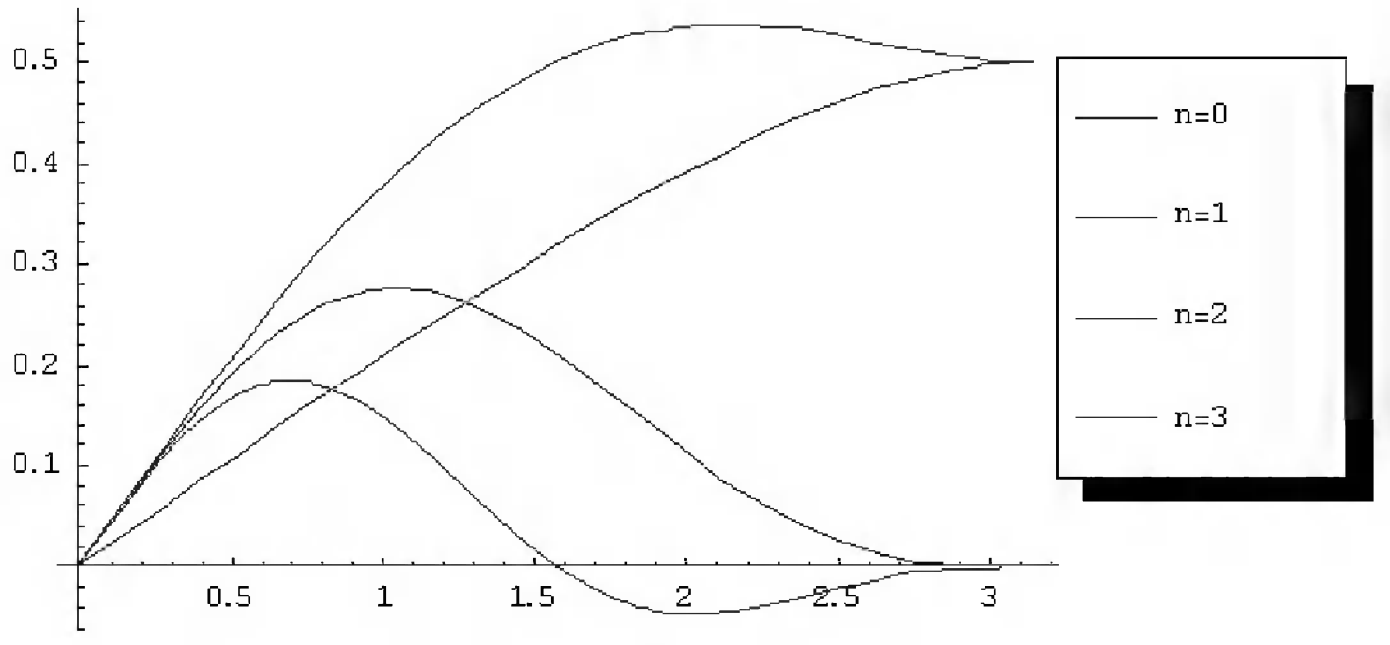


Рис. 15.64. Зависимости коэффициентов Берга от угла отсечки

Зависимости α[θ,n] служат основанием для расчета резонансных высокочастотных усилителей мощности, а также умножителей частоты. Например, из них видно, что для построения удвоителя частоты угол отсечки должен составлять примерно 1 рад, или около 60 градусов. Именно в этом случае относительная амплитуда второй гармоники α[θ,2] максимальна.

Насколько точно гармонический ряд, вычисленный по коэффициентам Берга, описывает усеченную косинусоиду? Об этом можно судить по результатам синтеза функции f(t) для заданных n гармоник. Ниже представлена формула для такого синтеза:

$$y[t\_, \theta\_, n\_] := \alpha[\theta, 0] + \sum_{k=1}^{n} \alpha[\theta, k] * Cos[k * t]$$

Теперь можно построить график для y(t) и 2*f(t) с приведенной единичной амплитудой (рис. 15.65):

```
A:=.5;θ:=π/3;
```

$$y[t\_, \theta\_, n\_] := A * \left(\alpha[\theta, 0] + \sum_{k=1}^{n} \alpha[\theta, k] * Cos[k * t] \right)$$

```
Plot[{y[t,θ,5],y[t,θ,10],f[t]},{t,-
2*π,2*π},PlotStyle→{Hue[1],Hue[.45],Hue[.7]},PlotRange→
{-.1,.55},PlotLegend→{"n=5","n=10","f(t)"},LegendPosition→
{1,-.3},LegendSize→{0.4,1}]
```

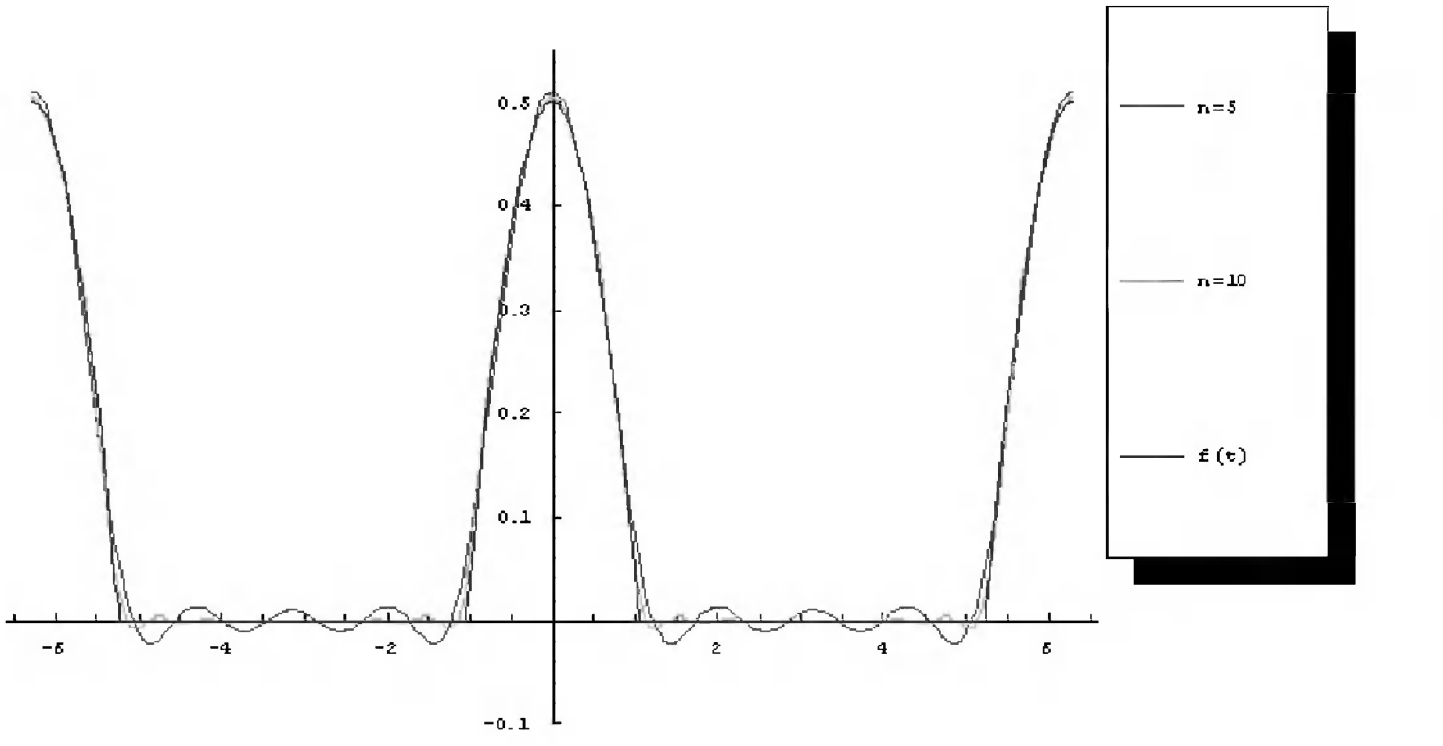


Рис. 15.65. Синтез усеченных косинусоид рядами Фурье

График функции y (t) для 5 и 10 гармоник наглядно показывает, что уже при 10 гармониках функция y(t) практически аналогична усеченной косинусоиде 2*f[t] с амплитудой, приведенной к 1. Таким образом, коэффициенты Берга хорошо описывают гармонический состав усеченной косинусоиды.

15.7.3. Задание сигналов различной формы

Многие сигналы можно моделировать с помощью различных комбинаций элементарных функций, в основном тригонометрических. Поскольку такие функции периодические, то это упрощает моделирование периодических сигналов. Ниже представлены примеры получения с помощью элементарных функций различных сигналов (рис. 15.66), смоделированные в СКМ Mathematica 4/5:

```
s1[t_]:=Sin[t];
s2[t_]:=Abs[Sin[t]];
s3[t_]:=Tan[Cos[t]];
s4[t_]:=Sec[Sin[t]];
s5[t_]:=Sign[Sin[t]];
s6[t_]:=ArcSin[Sin[t]];
s7[t_]:=ArcTan[Tan[t]];
s8[t_]:=2*Csch[Sec[t]];
 Plot[{s1[t]+8.5,Abs[Sin[t]]+6,s3[t]+4,s4[t]+1,s5[t],s6[t]-2,s7[t]-
5.5,s8[t]-8},{t,-10,10},PlotRange→{-10,10},PlotStyle→{Hue[1]}]
```

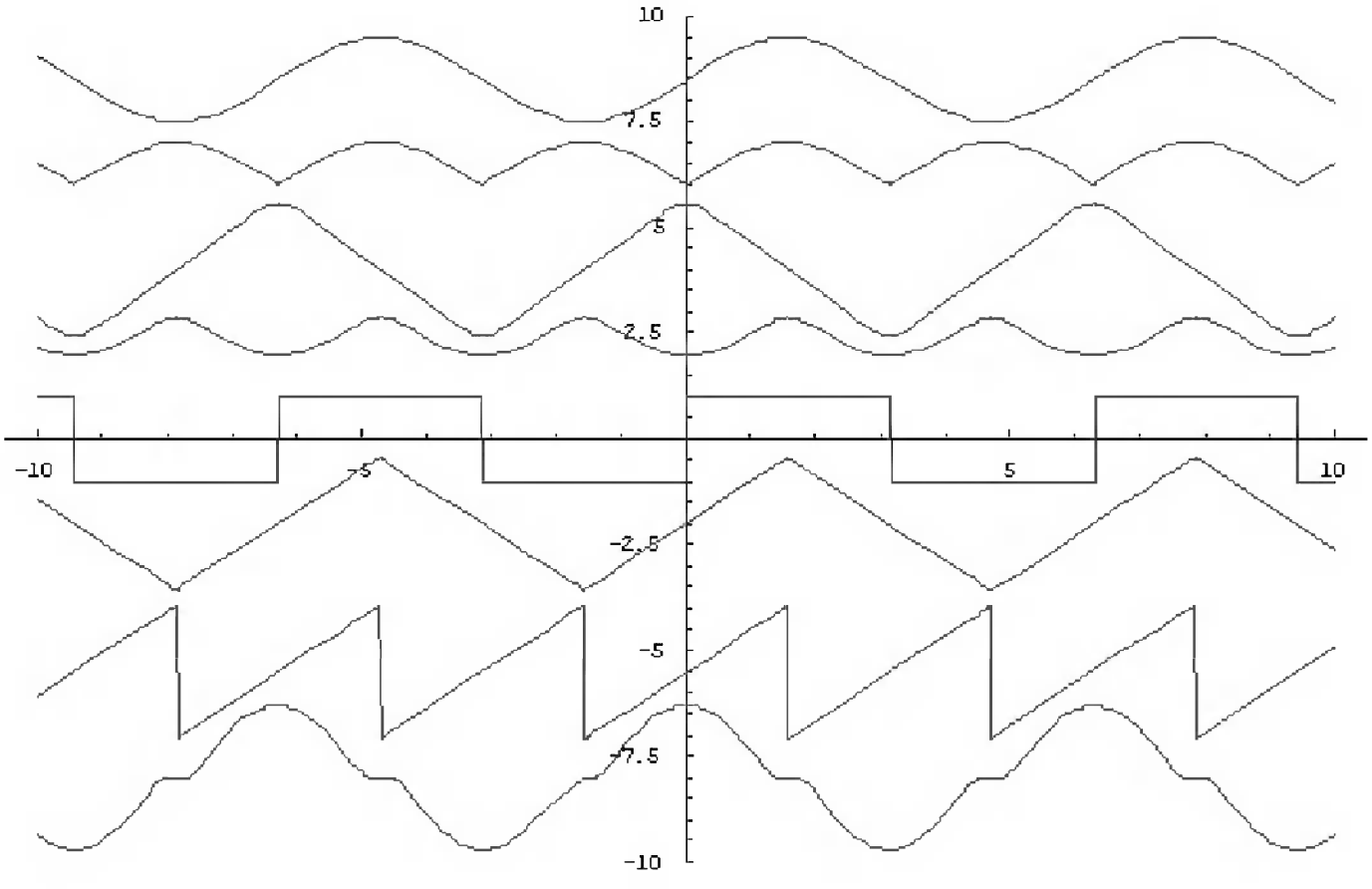


Рис. 15.66. Сигналы, полученные комбинацией элементарных функций

Важным достоинством моделирования сигналов с помощью аналитических функций является возможность применения для спектрального анализа функций пакета расширения Calculus, в частности функции FourierTrigSeries.

Функции с условиями сравнения (такие как **Round**, **Floor** и **Sign**) открывают интересные возможности в задании сигналов с разрывами. Ряд таких примеров приведен ниже.

Задание ступенчатого сигнала (рис. 15.67):

```
f1[t_]:=Round[t]
Plot[f1[t],{t,-2,2},PlotStyle→{Hue[1]}]
```

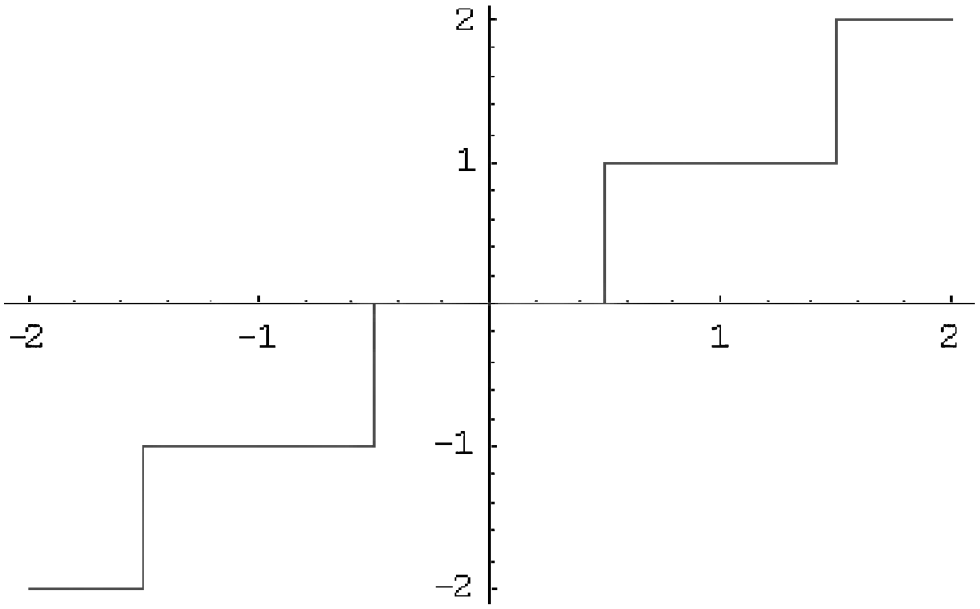


Рис. 15.67. Ступенчатый сигнал

Задание пилообразного линейно-нарастающего сигнала (рис. 15.68):

```
f2[t_]:=t-Round[t]
Plot[f2[t],{t,-2,2},PlotStyle→{Hue[*1]}]
```

Создание прямоугольного сигнала – меандра (рис. 15.69):

```
f3[t_]:=Sign[t-Round[t]]
Plot[f3[t],{t,-2,2},PlotStyle→{Hue[1]}]
```

Создание одиночного импульса отрицательной полярности (рис. 15.70):

```
f4[t_]:=Sign[(t^2)-Round[t]]
Plot[f4[t],{t,-2,2},PlotStyle→{Hue[1]}]
```

Нередко сигналы, полученные с помощью таких функций, можно использовать для спектрального анализа на основе аналитического вычисления коэффициентов Фурье.

Функция If предоставляет интересные возможности для моделирования сигналов с разрывами. Ряд примеров этого дан ниже.

Создание одиночного перепада (рис. 15.71):

```
F5[t_]:=If[t>1,1,0]
Plot[f5[t],{t,-2,2},PlotStyle→{Hue[1]}]
```

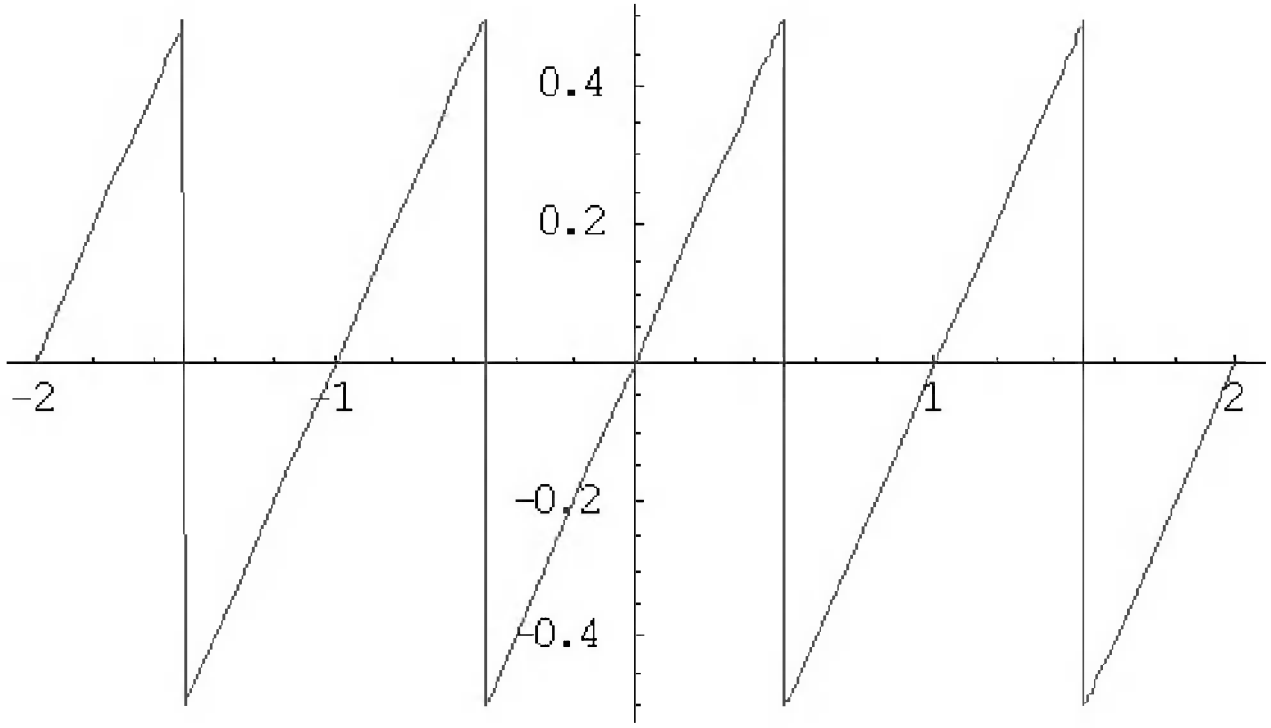


Рис. 15.68. Пилообразный линейно-нарастающий сигнал

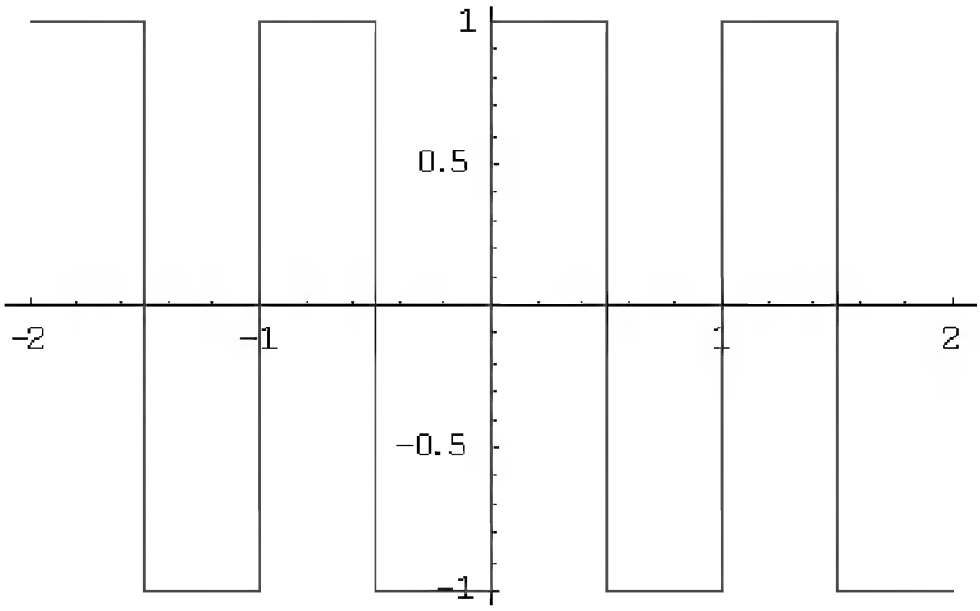


Рис. 15.69. Прямоугольный сигнал – меандр

Создание единичного прямоугольного импульса (рис. 15.72):

```
  f6[t_]:=If[(0>t) ||(t>1),0,1]
Plot[f6[t],{t,-2,2},PlotStyle→{Hue[1]}]
```

Создание одиночного пилообразного импульса (рис. 15.73):

```
f7[t_]:=If[(0>t)||(t>1.5),0,t]
Plot[f7[t],{t,-2,2},PlotStyle→{Hue[1]}]
```

Создание пачки синусоидальных колебаний (рис. 15.74):

```
f8[t_]:=If[(-0.5>t)||(t>1.5),0,Sin[4*π*t]]
Plot[f8[t],{t,-2,2},PlotStyle→{Hue[1]}]
```

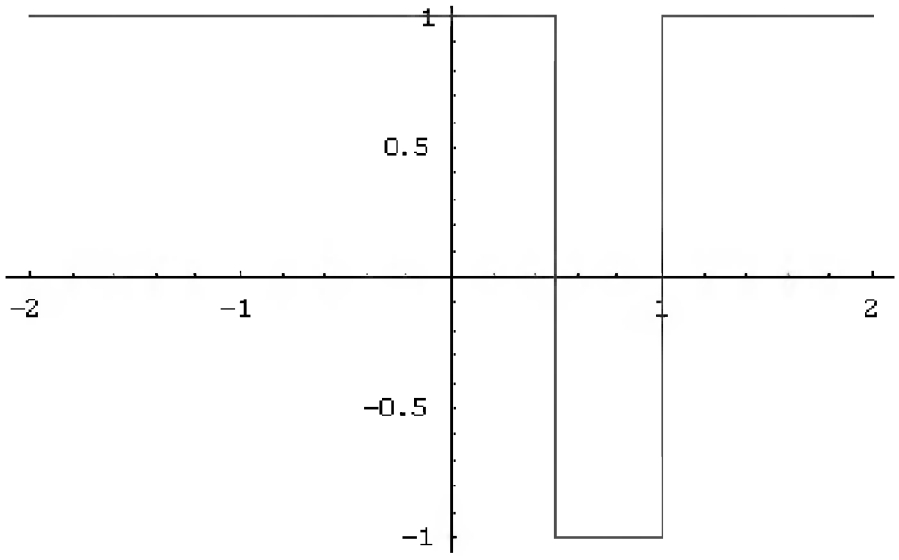


Рис. 15.70. Одиночный импульс отрицательной полярности

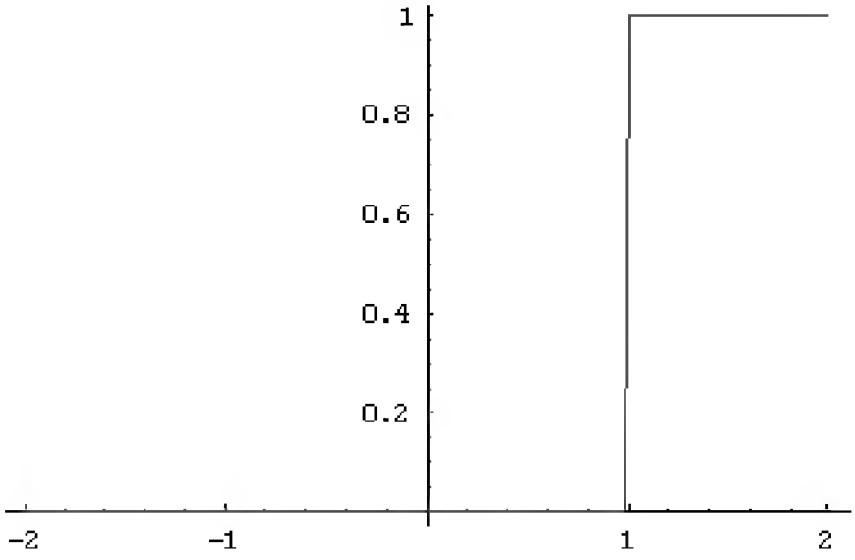


Рис. 15.71. Одиночный перепад

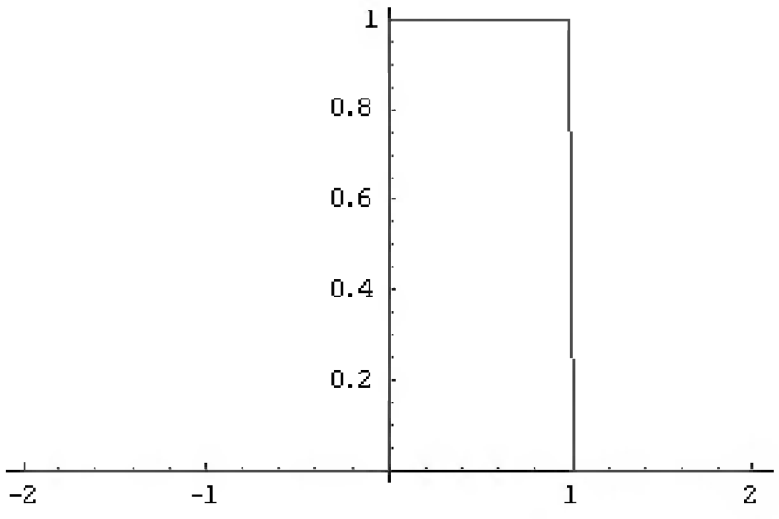


Рис. 15.72. Единичный прямоугольный импульс

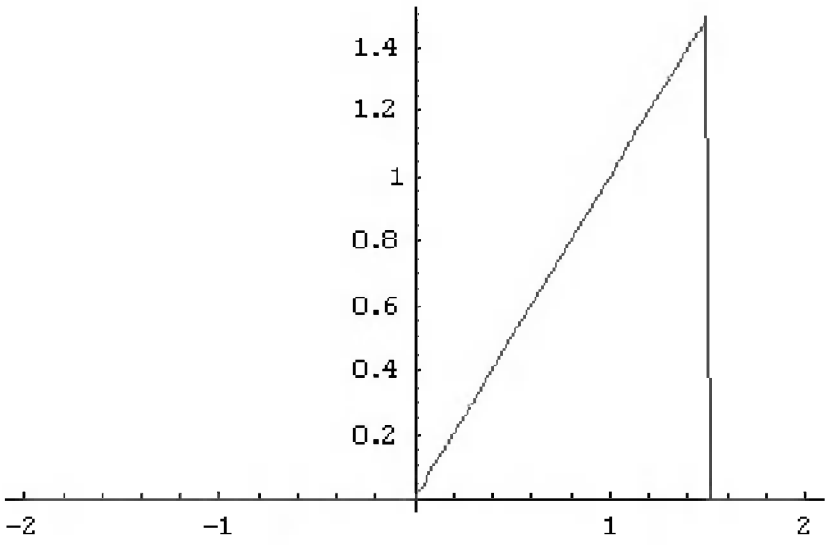


Рис. 15.73. Одиночный пилообразный импульс

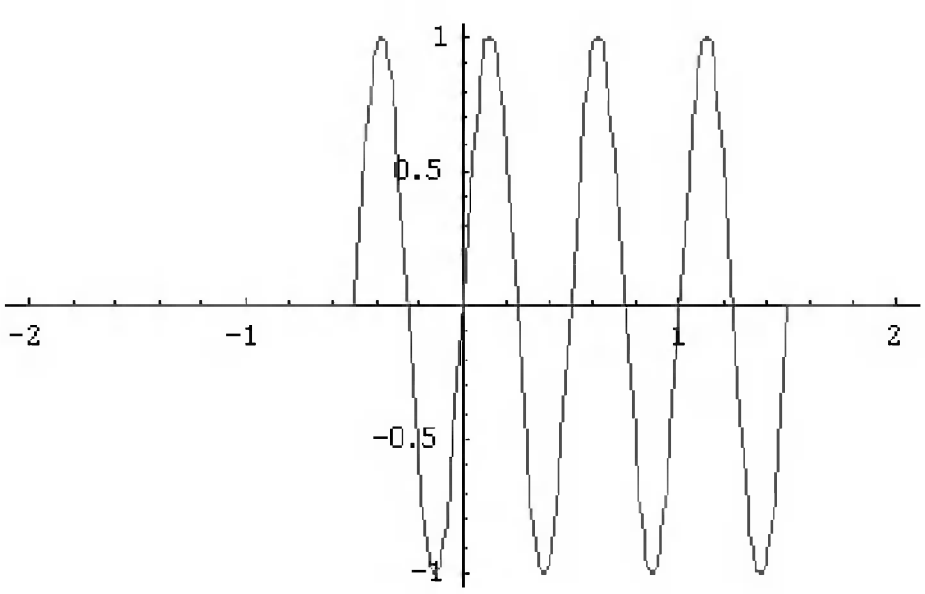


Рис. 15.74. Создание пачки синусоидальных колебаний

Главным недостатком моделей сигналов, полученных с помощью функции If, является невозможность аналитического вычисления интегралов Фурье. Более того, даже численное получение таких интегралов нередко оказывается невозможным. Однако приближенное вычисление интегралов, например методом прямоугольников, вполне возможно.

15.7.4. Спектральный анализ и синтез сигналов

В 1807 году Фурье нашел и обосновал метод вычисления коэффициентов тригонометрического ряда, при котором такой ряд был способен приближать *любую* периодическую функцию. Правда, при выполнении условий, известных как условия Дирихле для функции $y(x)$ на промежутке $(-\pi, \pi)$:

- функция непрерывна или имеет конечное число разрывов первого рода;
- промежуток $(-\pi, \pi)$ можно разбить на конечное число таких промежутков, на которых функция меняется монотонно.

Здесь важно отметить, что условия Дирихле выполняются для практически всех существующих сигналов, поэтому в дальнейшем мы будем опускать ссылки на них. Рядом Фурье для функции, удовлетворяющей условиям Дирихле, является ряд:

$$y(t) = \frac{a_0}{2} + \sum_{k=1}^{N} a_k \sin(2\pi k f_1 t) + b_k \cos(2\pi k f_1 t), \tag{15.1}$$

коэффициенты которого находятся по формулам Эйлера–Фурье:

$$a_k = \frac{1}{\pi} \int_{-\pi}^{\pi} y(x)\cos(kx)dx \tag{15.2}$$

и

$$b_k = \frac{1}{\pi} \int_{-\pi}^{\pi} y(x)\sin(kx)dx. \tag{15.3}$$

Строгое доказательство того, что ряд (15.1) может приближать произвольную функцию, базируется на элементарных тригонометрических преобразованиях и понятии ортогональности набора функций, образующих этот ряд:

$1, \cos(x), \sin(x), \cos(2x), \sin(2x), \ldots, \cos(nx), \sin(nx), \ldots$

Ортогональность означает, что интеграл от произведения двух любых различных функций этого (возможно, и иного) набора функций в промежутке от 0 до 2π равен нулю. Само доказательство, довольно громоздкое, можно найти в учебниках по высшей математике, например в [21].

Важными сферами применения рядов Фурье являются расчеты радиотехнических устройств. В них обычно периодические сигналы представляют как функции времени $y(t)$ на отрезке $[0, T]$ с периодом $T = 1/f_1$, где f_1 – частота первой гармоники периодического сигнала. В этом случае ряд Фурье, после несложных преобразований, записывается в виде:

$$y(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} (a_k \cos(2\pi k f_1 t) + b_k \sin(2\pi k f_1 t)), \tag{15.4}$$

где

$$a_k = \frac{2}{T} \int_0^T y(t)\cos(2\pi k f_1 t)dt \tag{15.5}$$

и

$$b_k = \frac{2}{T} \int_0^T y(t)\sin(2\pi k f_1 t)dt. \tag{15.6}$$

Тогда коэффициенты a_k и b_k описывают косинусную и синусную составляющие k-ой гармоники сигнала с периодом T и частотой $f_1 = 1/T$. Часто используется иная форма ряда Фурье, упрощающая его синтез [23]:

$$y(t) = \frac{a_0}{2} + \sum_{k=1}^{N} M_k \cos(2\pi k f_1 t + \varphi_k), \qquad (15.7)$$

где амплитуды гармоник M_k и их фазы φ_k определяются выражениями:

$$M_k = \sqrt{a_k^2 + b_k^2} \qquad (15.8a)$$

и

$$\varphi_k = -\arctan(b_k/a_k). \qquad (15.8b)$$

В дальнейшем будут приведены формулы, позволяющие вычислять коэффициенты Фурье (либо амплитуды и фазы гармоник) для любой функции $y(t)$. Это является задачей спектрального анализа. Однако вначале мы рассмотрим обратную задачу – задачу синтеза зависимости $y(t)$ путем вычисления ряда Фурье с ограниченным числом членов.

15.7.5. Дискретный Фурье-анализ и спектр периодических функций

Предположим, что некоторая функция (или сигнал) задана рядом равноотстоящих дискретных отсчетов с числом N, то есть $y1, y2,...,yN$. В этом случае у нас нет никаких оснований считать, что в промежутках между узлами значения функции не постоянны. Если же они постоянны, то интегралы при расчете коэффициентов Фурье могут вычисляться простейшим методом прямоугольников:

$$a_k = \frac{2}{N}\sum_{i=1}^{N} y_i \cos\left(\frac{2\pi n i}{N}\right) \quad \text{и} \quad b_k = \frac{2}{N}\sum_{i=1}^{N} y_i \sin\left(\frac{2\pi n i}{N}\right). \qquad (15.9)$$

Детальный анализ, выходящий за рамки данной книги, показывает, что приведенные формулы для коэффициентов Фурье являются единственными теоретически обоснованными формулами приближенного вычисления коэффициентов Фурье [22, 23]. Для произвольных функций они обеспечивают минимум среднеквадратической погрешности.

Зависимости амплитуд и фаз гармоник от частоты получили название амплитудного и фазового *спектров* сигнала. Для периодических колебаний такой спектр является дискретным, то есть состоящим из отдельных частот – гармоник. Его удобно представлять вертикальными отрезками прямых, длина которых определяет значение амплитуды или фазы той или иной гармоники.

Разложение функции на гармонические составляющие, то есть вычисление коэффициентов Фурье, принято называть *спектральным анализом*. А воссоздание функции, представленной рядом Фурье, называют *спектральным синтезом*.

Гармонику с $k = 1$ называют основной, или *первой*, *гармоникой* сигнала. Она задает его частоту повторения f_1. Остальные гармоники называют *высшими*, их частоты равны $f_k = k \cdot f_1$, где $k = 2, 3, 4, ...$Таким образом, спектр периодических сигналов, представимых рядом Фурье, *дискретный* – он содержит набор фиксированных частот f_k, где $k = 1, 2, 3, ...$

15.7.6. Быстрое преобразование Фурье (БПФ)

Для преодоления вычислительных трудностей, связанных с интегрированием в ходе ППФ и ОПФ быстроизменяющихся функций, были предложены методы быстрого преобразования Фурье (БПФ, или, в англоязычной транскрипции, FFT). Они используют специальную технику комбинации отсчетов функций, помноженных на осциллирующие множители, и учитывают периодичность значений тригонометрических функций. Алгоритмы БПФ не уменьшают погрешности вычислений при заданном числе гармоник, но позволяют резко уменьшить время спектрального анализа и синтеза – особенно если число временных отсчетов $y_i(t)$ кратно 2^N, где N – целое число.

Описание алгоритмов БПФ в данной книге опущено в силу двух причин – его громоздкости и отнесения к сугубо численным методам. Такое описание можно найти в [159, 160].

> ***Внимание!*** *В основе БПФ лежат прореживание по частоте и пирамидальный алгоритм, исключающий повторные вычисления периодически повторяющихся членов тригонометрического ряда Фурье. БПФ-алгоритм выполняется за ~ N•logN операций, где N – число отсчетов сигнала.*

15.7.7. Спектральный синтез сигналов средствами Mathematica 4/5

Практически в любом справочнике по высшей математике можно найти таблицы с разложениями в тригонометрические ряды Фурье простых сигналов, полученные применением формул разделов 15.6.4 и 15.6.5. Ниже даны примеры прямого синтеза таких сигналов по известным разложениям.

Гармонический синтез меандра (рис. 15.75):

```
meandr[k_, j_, N_] := 4/π Sum[1/i * Sin[2 * i * j * π / N], {i, 1, k, 2}]
meandr[10,j,N]
```

$$\frac{1}{\pi}\left(4\left(\text{Sin}\left[\frac{2\,j\,\pi}{N}\right]+\frac{1}{3}\,\text{Sin}\left[\frac{6\,j\,\pi}{N}\right]+\frac{1}{5}\,\text{Sin}\left[\frac{10\,j\,\pi}{N}\right]+\frac{1}{7}\,\text{Sin}\left[\frac{14\,j\,\pi}{N}\right]+\frac{1}{9}\,\text{Sin}\left[\frac{18\,j\,\pi}{N}\right]\right)\right)$$

```
Plot[{meandr[5,j,200],meandr[20,j,200]},{j,1,400},
PlotStyle→{Hue[0.75],Hue[1]},PlotRange→{-1.2,1.2}]
```

Гармонический синтез разнополярных прямоугольных импульсов (рис. 15.76):

```
sqr[k_, j_, N_, a_] :=
  4/π Sum[1/i * (Cos[i * a]) * Sin[2 * i * j * π / N], {i, 1, k, 2}]
```

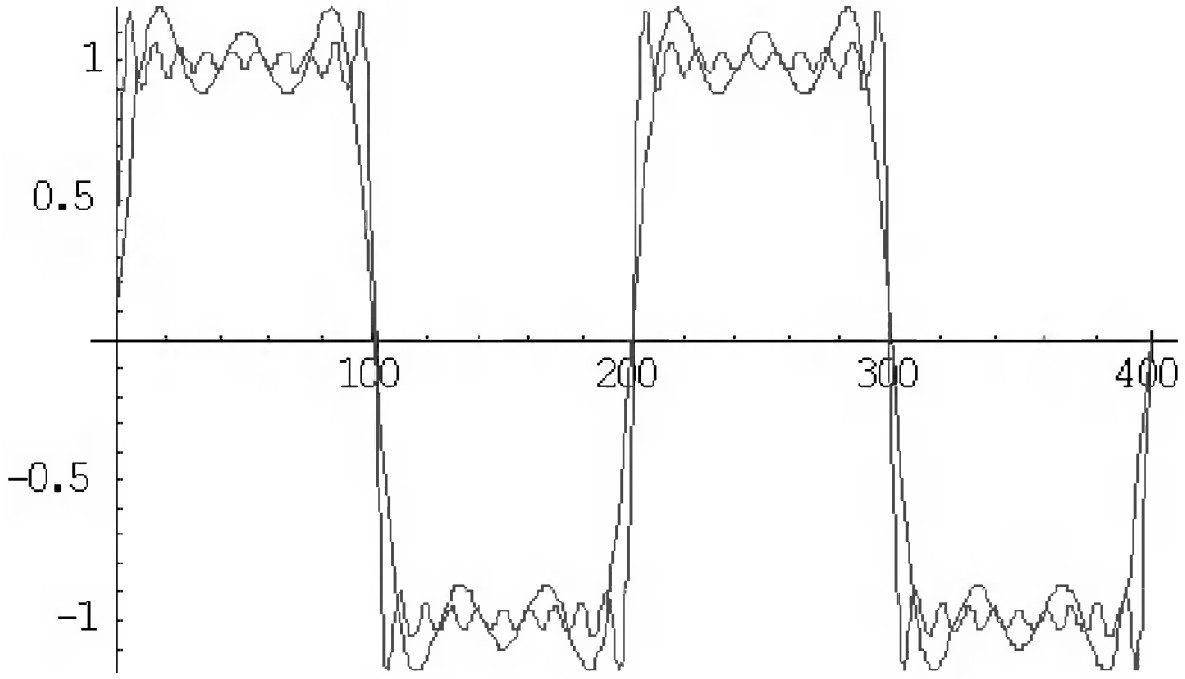


Рис. 15.75. Гармонический синтез прямоугольных импульсов – меандра

```
sqr[10,j,N,a]
```

$$\frac{1}{\pi}\left(4\left(\text{Cos}[a]\,\text{Sin}\left[\frac{2\,j\,\pi}{N}\right]+\frac{1}{3}\,\text{Cos}[3\,a]\,\text{Sin}\left[\frac{6\,j\,\pi}{N}\right]+\frac{1}{5}\,\text{Cos}[5\,a]\right.\right.$$
$$\left.\left.\text{Sin}\left[\frac{10\,j\,\pi}{N}\right]+\frac{1}{7}\,\text{Cos}[7\,a]\,\text{Sin}\left[\frac{14\,j\,\pi}{N}\right]+\frac{1}{9}\,\text{Cos}[9\,a]\,\text{Sin}\left[\frac{18\,j\,\pi}{N}\right]\right)\right)$$

```
Plot[{sqr[5,j,200,1],sqr[20,j,200,1]},{j,1,400},PlotStyle→
{Hue[0.75],Hue[1]},PlotRange→{-1.2,1.2}]
```

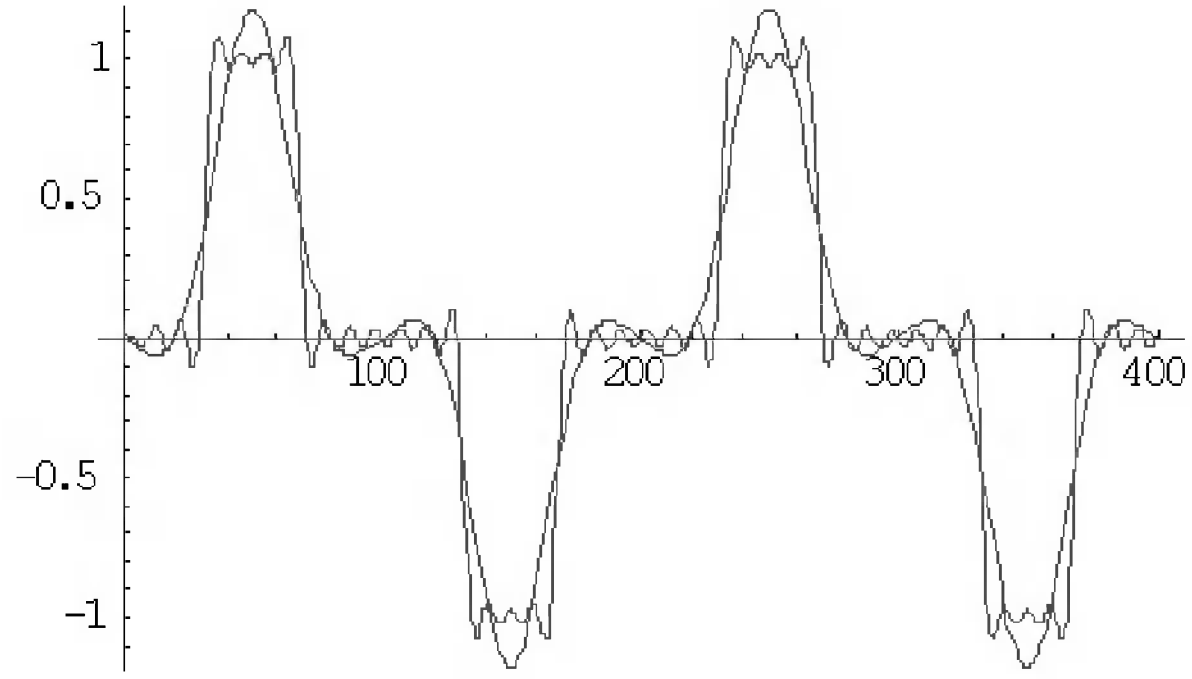


Рис. 15.76. Гармонический синтез разнополярных прямоугольных импульсов

Гармонический синтез треугольного импульса (рис. 15.77):

```
triag[k_,j_,N_]:=Sum[(-1/(i^2))*Cos[2*i*j*π/N],{i,1,k,2}]
triag[10,j,N]
```

$$-\text{Cos}\left[\frac{2\,j\,\pi}{N}\right]-\frac{1}{9}\text{Cos}\left[\frac{6\,j\,\pi}{N}\right]-\frac{1}{25}\text{Cos}\left[\frac{10\,j\,\pi}{N}\right]-\frac{1}{49}\text{Cos}\left[\frac{14\,j\,\pi}{N}\right]-\frac{1}{81}\text{Cos}\left[\frac{18\,j\,\pi}{N}\right]$$

```
Plot[{triag[5,j,200],triag[20,j,200]},{j,1,400},PlotStyle→
{Hue[0.75],Hue[1]},PlotRange→{-1.2,1.2}]
```

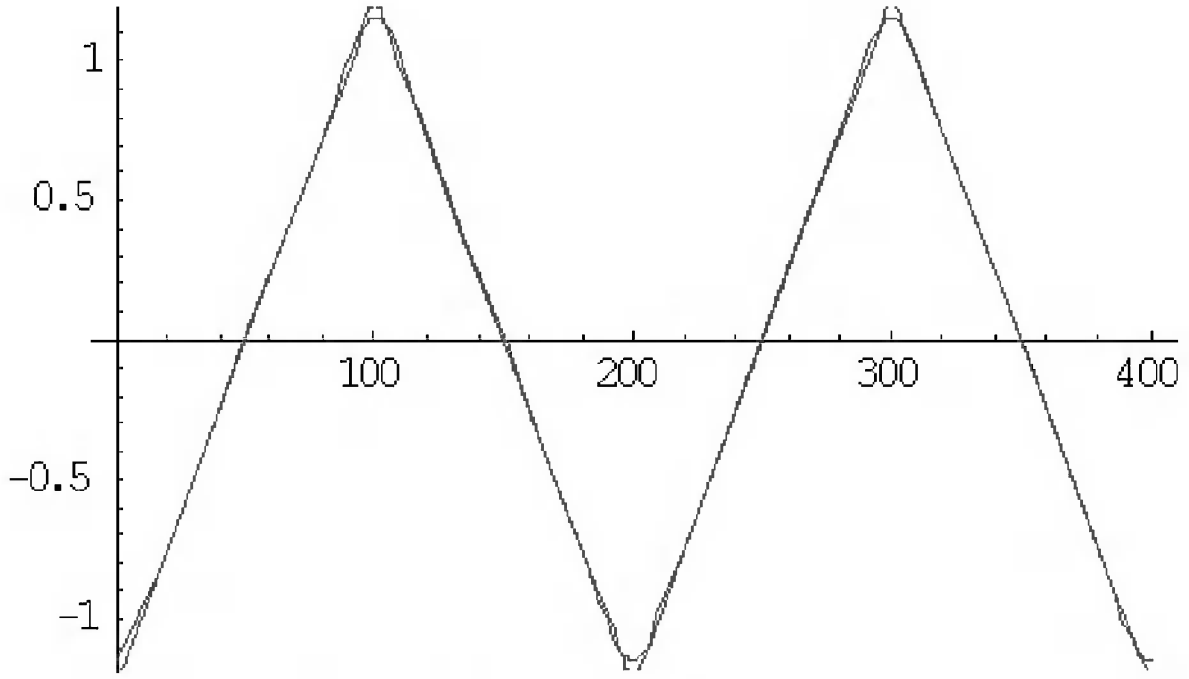


Рис. 15.77. Гармонический синтез треугольных колебаний

Гармонический синтез пилообразных колебаний (рис. 15.78):

```
saw[k_,j_,N_]:=(1/2)*Sum[(-1/i)*Sin[2*i*j*π/N],{i,1,k}]
saw[10,j,N]
```

$$\frac{1}{2}\left(-\text{Sin}\left[\frac{2\,j\,\pi}{N}\right]-\frac{1}{2}\text{Sin}\left[\frac{4\,j\,\pi}{N}\right]-\frac{1}{3}\text{Sin}\left[\frac{6\,j\,\pi}{N}\right]-\frac{1}{4}\text{Sin}\left[\frac{8\,j\,\pi}{N}\right]-\frac{1}{5}\text{Sin}\left[\frac{10\,j\,\pi}{N}\right]-\frac{1}{6}\text{Sin}\left[\frac{12\,j\,\pi}{N}\right]-\frac{1}{7}\text{Sin}\left[\frac{14\,j\,\pi}{N}\right]-\frac{1}{8}\text{Sin}\left[\frac{16\,j\,\pi}{N}\right]-\frac{1}{9}\text{Sin}\left[\frac{18\,j\,\pi}{N}\right]-\frac{1}{10}\text{Sin}\left[\frac{20\,j\,\pi}{N}\right]\right)$$

```
Plot[{saw[5,j,200],saw[20,j,200]},{j,1,400},PlotStyle→
{Hue[0.75],Hue[1]},PlotRange→{-1,1}]
```

Гармонический синтез приплюснутой синусоиды (рис. 15.79):

```
f1[k_,j_,N_]:=Sum[1/(i^2)*Sin[2*i*j*π/N],{i,1,k,2}]
 f1[10,j,N]
```

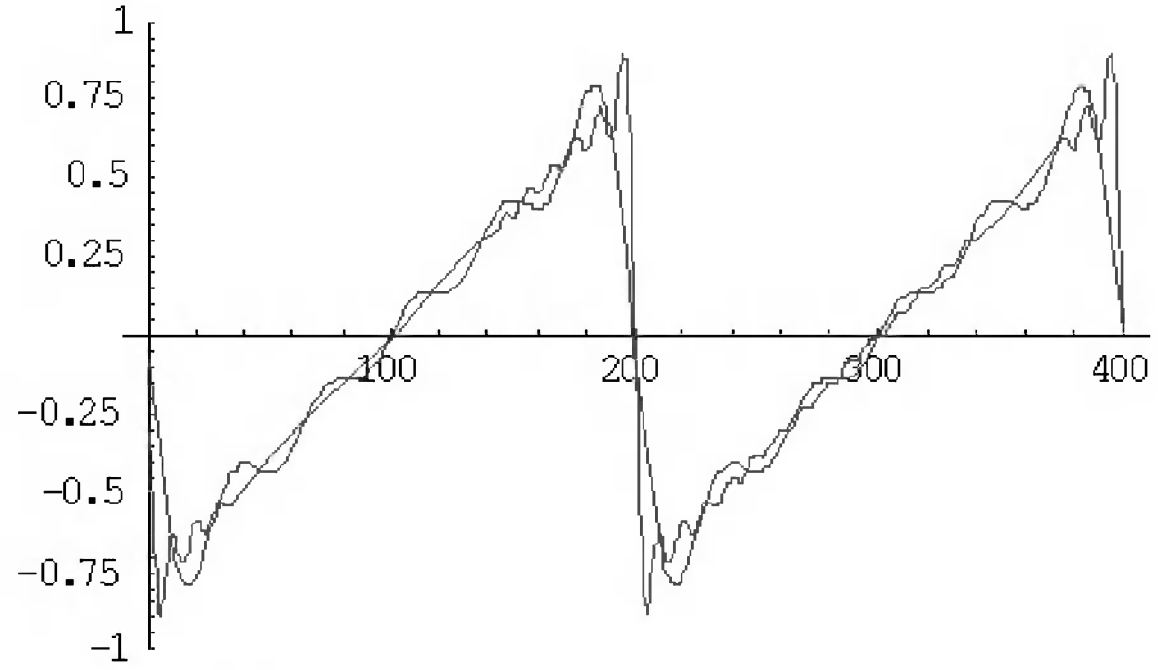


Рис. 15.78. Гармонический синтез пилообразных колебаний

$$\text{Sin}\left[\frac{2\,j\,\pi}{N}\right] + \frac{1}{9}\,\text{Sin}\left[\frac{6\,j\,\pi}{N}\right] + \frac{1}{25}\,\text{Sin}\left[\frac{10\,j\,\pi}{N}\right] + \frac{1}{49}\,\text{Sin}\left[\frac{14\,j\,\pi}{N}\right] + \frac{1}{81}\,\text{Sin}\left[\frac{18\,j\,\pi}{N}\right]$$

```
Plot[{f1[2,j,200],f1[5,j,200]},{j,1,400},PlotStyle→
{Hue[0.75],Hue[1]},PlotRange→{-1,1}]
```

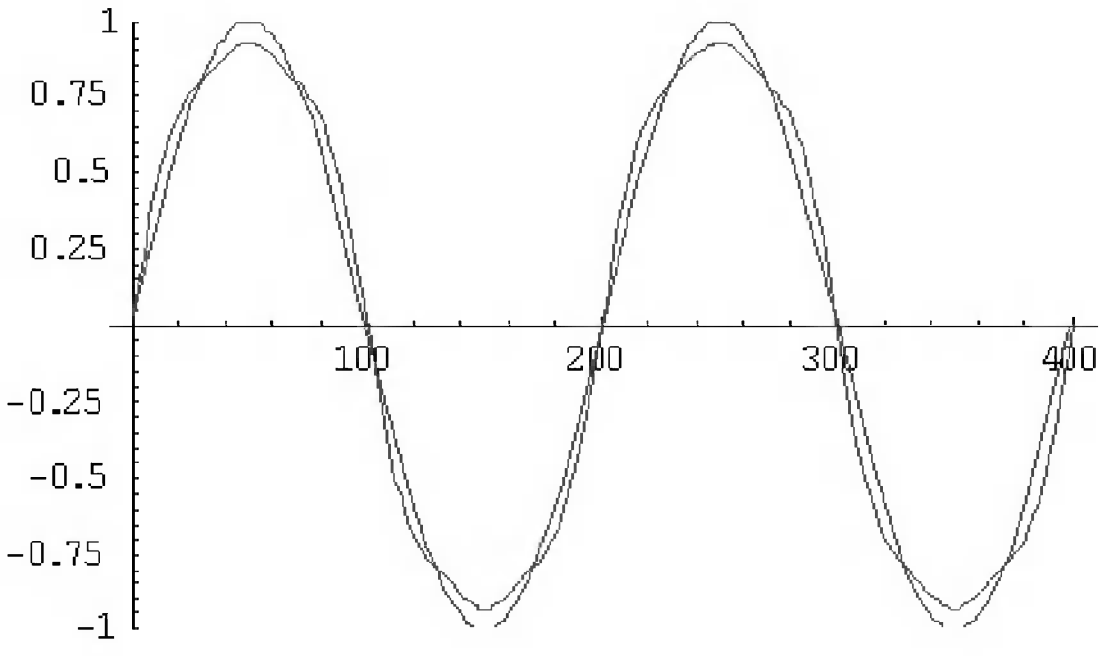


Рис. 15.79. Гармонический синтез приплюснутой синусоиды

Гармонический синтез «выпрямленной» синусоиды (рис. 15.80):

```
f2[k_,j_,N_]:=(2/π)+(4/π)*Sum[((-1)^Floor[(i+2)/2])*Cos[2*i*j*π/N]/
((i-1)*(i+1)),{i,2,k,2}]
f2[10,j,N]
```

$$\frac{2}{\pi}+\frac{1}{\pi}\left(4\left(\frac{1}{3}\text{Cos}\left[\frac{4\,j\,\pi}{N}\right]-\frac{1}{15}\text{Cos}\left[\frac{8\,j\,\pi}{N}\right]+\frac{1}{35}\text{Cos}\left[\frac{12\,j\,\pi}{N}\right]-\frac{1}{63}\text{Cos}\left[\frac{16\,j\,\pi}{N}\right]+\frac{1}{99}\text{Cos}\left[\frac{20\,j\,\pi}{N}\right]\right)\right)$$

```
Plot[{f3[5,j,200],f3[50,j,200]},{j,1,400},PlotStyle→
{Hue[0.75],Hue[1]},PlotRange→{-.1,1}]
```

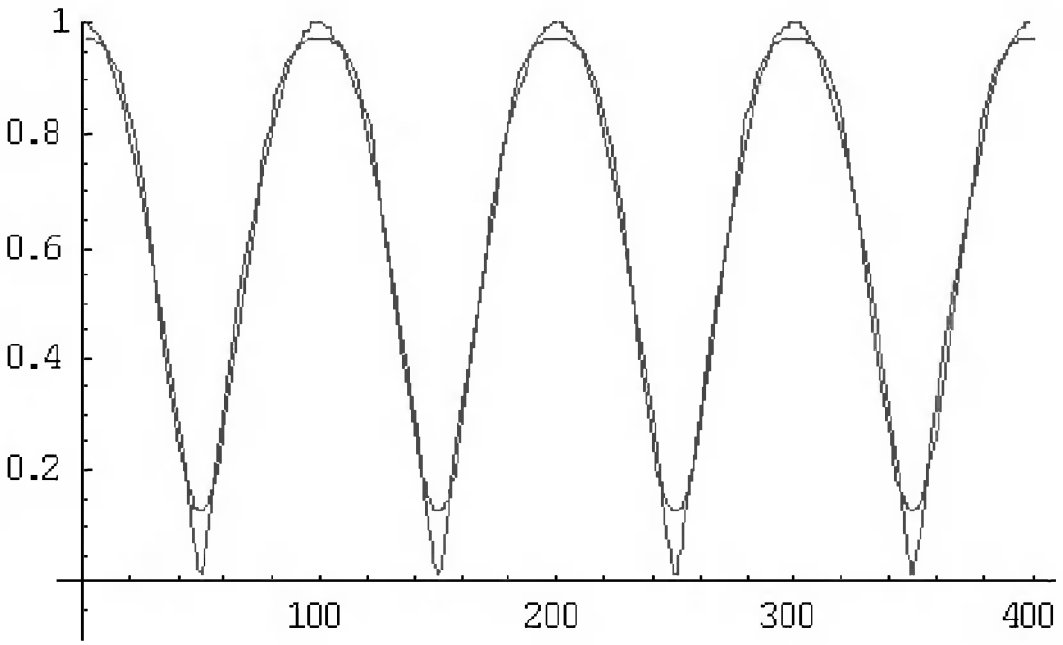


Рис. 15.80. Гармонический синтез «выпрямленной» синусоиды

15.7.8. Спектральный анализ и синтез сигналов, заданных аналитически

Пакет расширения Calculus СКМ Mathematica 4/5 содержит ряд функций спектрального анализа для аналитически заданных сигналов. Ниже представлены примеры применения функции FourierTrigSeries, дающей разложение в тригонометрический ряд Фурье таких периодических сигналов. Для загрузки пакета используется команда

```
<<Calculus`FourierTransform`
```

Синтез треугольных колебаний:

```
FourierTrigSeries[t,t,5]
```

$$\frac{\text{Sin}[2\,\pi\,t]}{\pi}-\frac{\text{Sin}[4\,\pi\,t]}{2\,\pi}+\frac{\text{Sin}[6\,\pi\,t]}{3\,\pi}-\frac{\text{Sin}[8\,\pi\,t]}{4\,\pi}+\frac{\text{Sin}[10\,\pi\,t]}{5\,\pi}$$

```
Plot[{%,t-Round[t]},{t,-2,2},PlotStyle→{Hue[0.75],Hue[1]}]
```

Синтез прямоугольных колебаний – меандра:

```
FourierTrigSeries[Sign[Sin[t]],t,5]
```

$$\frac{4\,\mathrm{Sin}[2\,\pi\,t]}{\pi}+\frac{4\,\mathrm{Sin}[6\,\pi\,t]}{3\,\pi}+\frac{4\,\mathrm{Sin}[10\,\pi\,t]}{5\,\pi}$$

```
Plot[{%,Sign[Sin[2*π*t]]},{t,-2,2},PlotStyle→{Hue[0.75],Hue[1]}]
```

Синтез выпрямленной синусоиды:

```
FourierTrigSeries[Abs[Sin[π*t]],t,3]
```

$$\frac{2}{\pi}-\frac{4\,\mathrm{Cos}[2\,\pi\,t]}{3\,\pi}-\frac{4\,\mathrm{Cos}[4\,\pi\,t]}{15\,\pi}-\frac{4\,\mathrm{Cos}[6\,\pi\,t]}{35\,\pi}$$

```
Plot[{%,Abs[Sin[π*t]]},{t,-2,2},PlotStyle→{Hue[0.75],Hue[1]}]
```

Поскольку графики этих примеров подобны ранее приведенным, они не приводятся. Вы можете просмотреть их самостоятельно. К сожалению, с сигналами, заданными с помощью функции If, функции пакета Calculus *не работают*.

15.7.9. Цифровая фильтрация зашумленного сигнала

Одно из самых распространенных применений спектрального анализа и синтеза – выделение сигнала на фоне шумов. Покажем, как это делается в системе Mathematica. Для этого создадим достаточно сложный сигнал, показанный на рис. 15.81:

```
y[i_]:=Sin[π*(i-1)/64]*(1-Exp[-32/i])
g1:=ListPlot[Table[y[n],{n,256}],PlotJoined→True,PlotStyle→{Hue[1]}];g1
```

Вектор **data** задает 256 точек этого сигнала со смесью сильного шума, моделируемого с помощью генератора случайных чисел (рис. 15.82):

```
data = Table[ N[y[n]+ .5*(Random[ ] – 1/2)], {n, 256} ] ;
ListPlot[data]
```

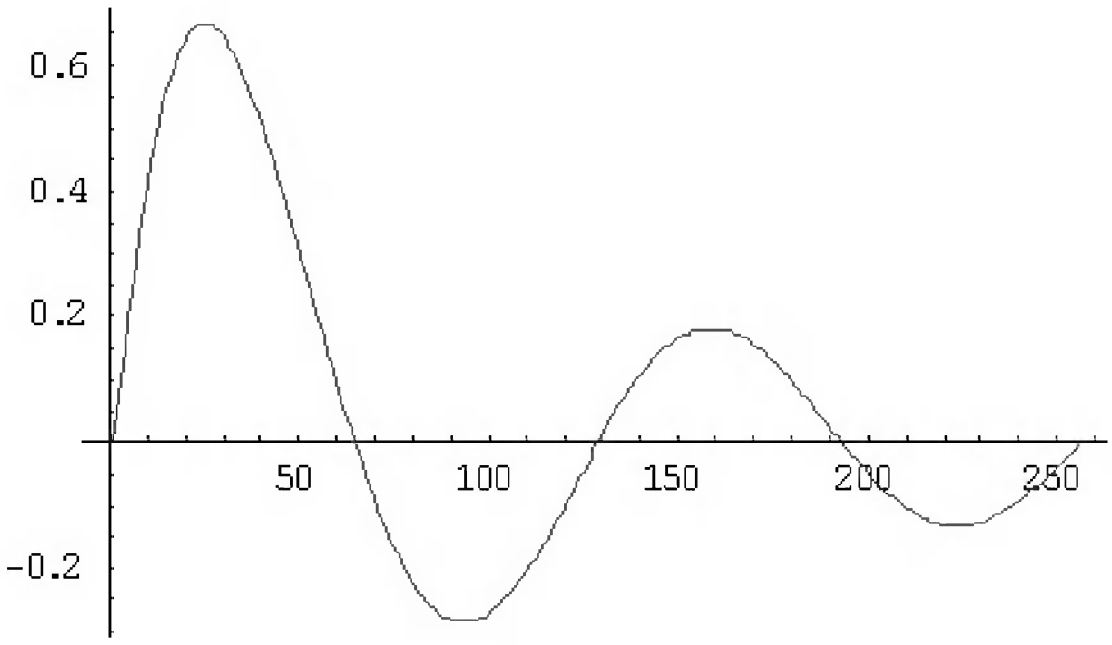


Рис. 15.81. Исходный сигнал

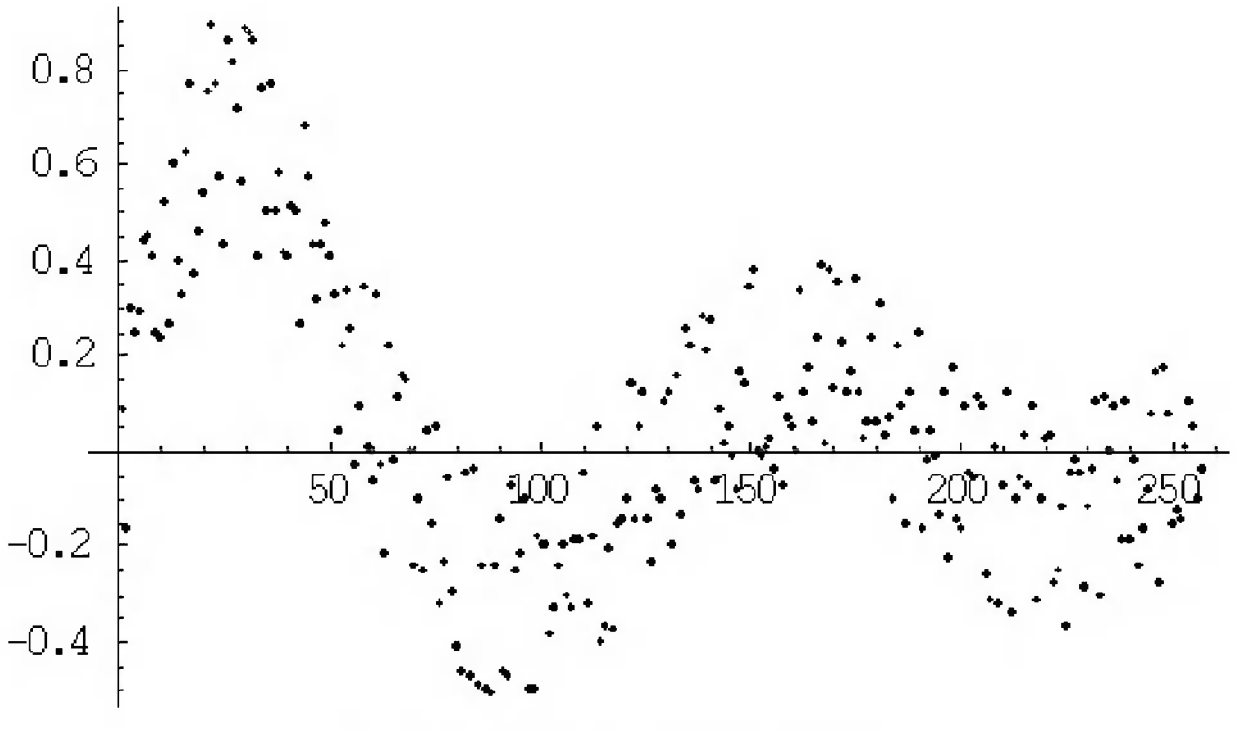


Рис. 15.82. Зашумленный сигнал

Шум настолько маскирует сигнал, что детали формы сигнала становятся незаметными. Построим фильтрующую функцию и ее частотную характеристику (рис. 15.83) для теоретического цифрового фильтра:

```
kern = Table[Exp[-n^2/100.], {n, -128, 127}] ;ListPlot[kern,
PlotRange -> All]
```

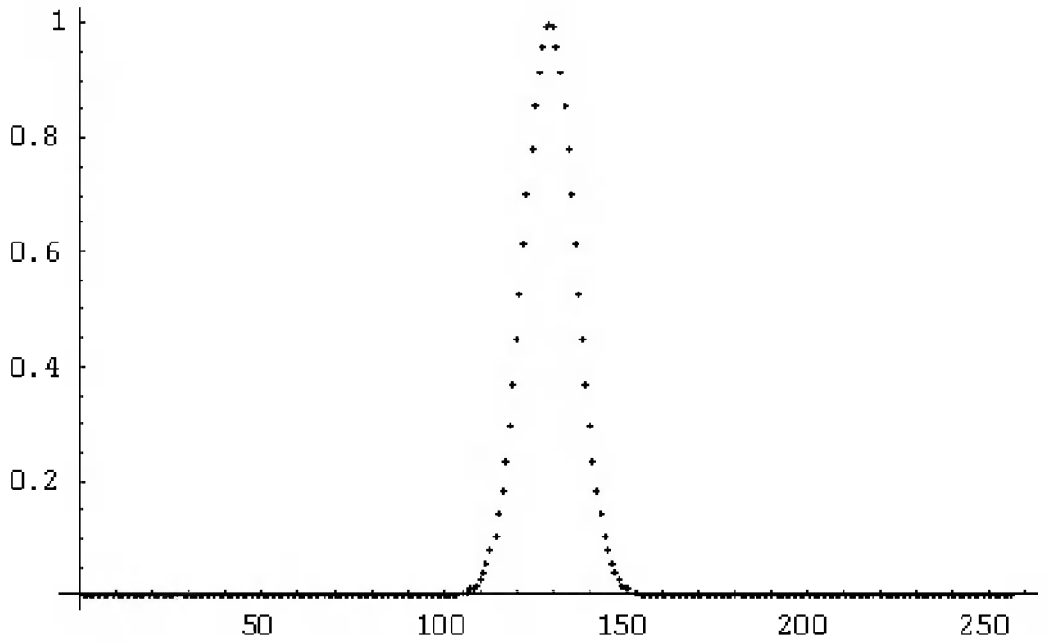


Рис. 15.83. Фильтрующая функция и ее частотная характеристика

Проведем трансформацию вектора фильтрации и получим его преобразованную частотную характеристику (рис. 15.84):

```
kern = RotateLeft[kern, 128]/Apply[Plus, kern] ;
ListPlot[kern,PlotRange→All]
```

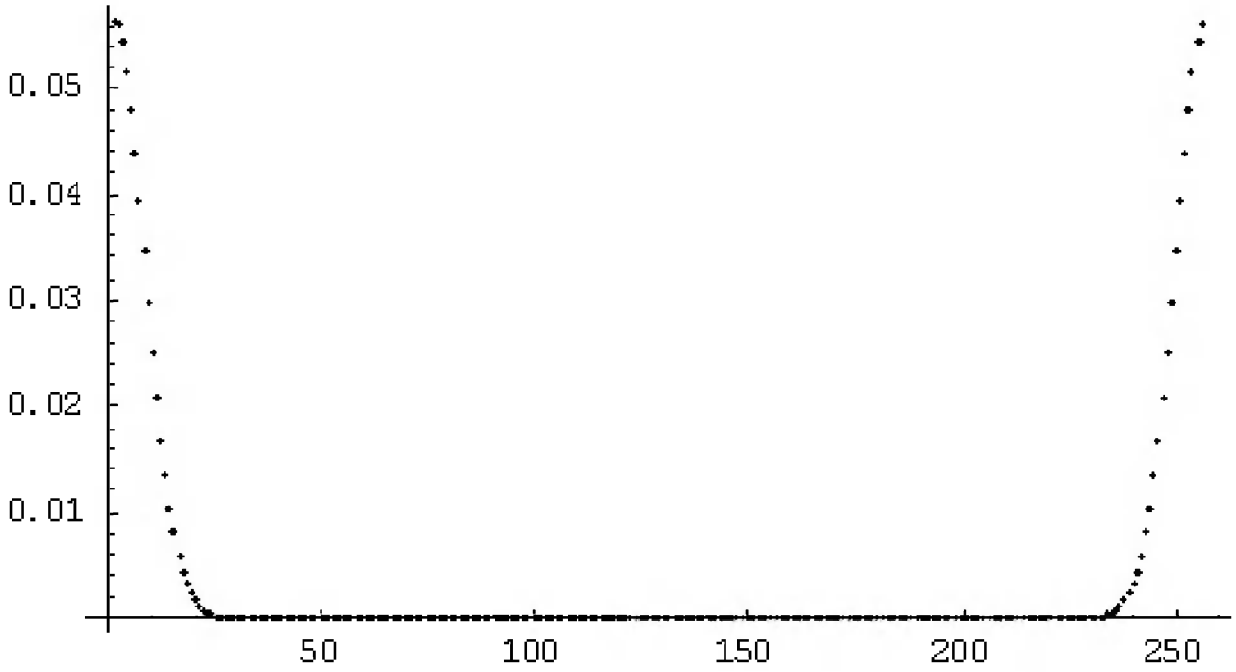


Рис. 15.84. Преобразованная частотная характеристика фильтра

Для получения отфильтрованного сигнала надо осуществить обратное преобразование Фурье для произведения масштабирующего множителя, вектора прямого преобразования Фурье **Fourier[data]** и вектора прямого преобразования Фурье фильтрующей функции **Fourier[kern]**:

```
conv = InverseFourier[Sqrt[256] Fourier[data] Fourier[kern] ] ;
```

Теперь можно построить графики зашумленного и выходного сигналов (рис. 15.85):

```
g2:=ListPlot[conv]
Show[{g1,g2}]
```

Нетрудно отметить превосходную степень фильтрации сигнала от шума и хорошее соответствие формы сигнала на выходе фильтра в форме исходного сигнала.

15.7.10. Спектральный анализ и синтез сигналов с линейной интерполяцией между узлами

Средства пакета расширения Calculus системы Mathematica 4/5 содержат функции для вычисления тригонометрического ряда Фурье для периодических функций, представленных на отрезке времени от $-T/2$ до $+T/2$. Это крайне неудобно для практических вычислений, когда заданная функция $y(t)$ определена на ин-

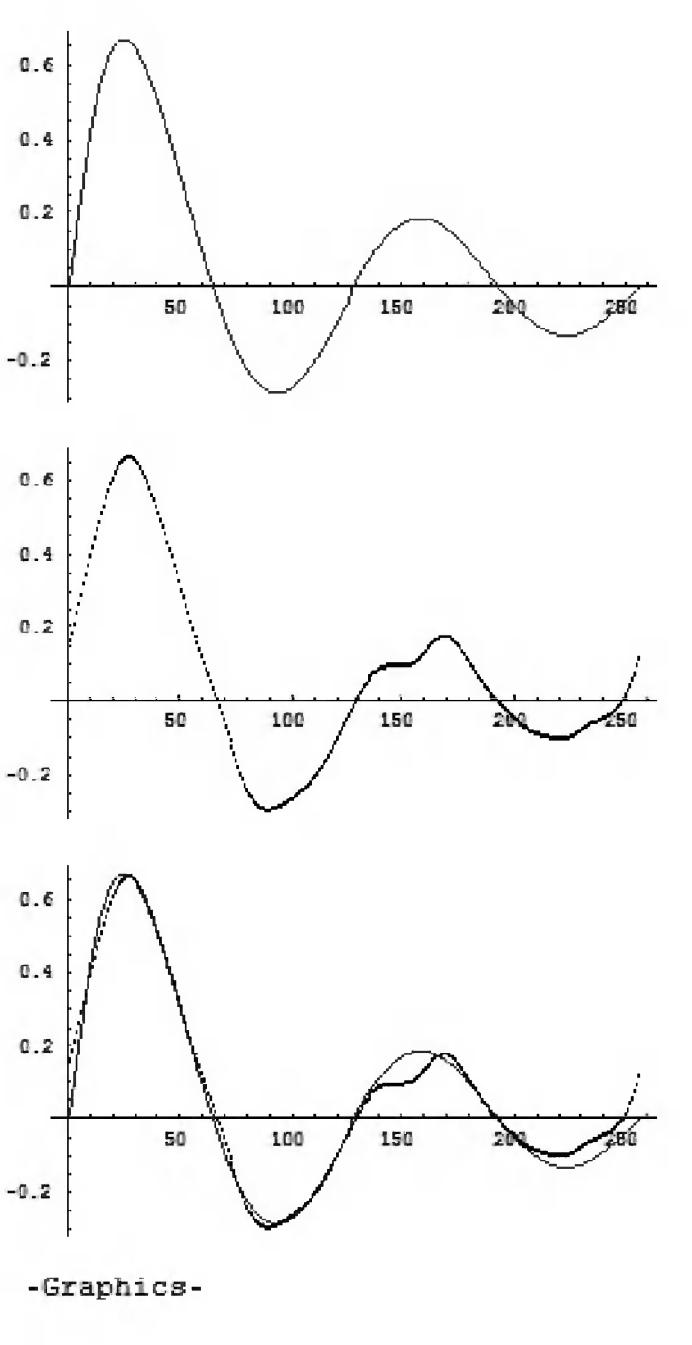


Рис. 15.85. Графики зашумленного и выходного сигналов

тервале [0,T], причем она может быть непериодической. Кроме того, нередко такая функция задана таблицей своих отсчетов. Если таких отсчетов Ni, то при классическом спектральном анализе, на основе теоремы отсчетов, для представления функции можно использовать не более Ni/2 гармоник. Это резко снижает точность представления y(t) тригонометрическим рядом Фурье, что, в частности, связано со значительным проявлением эффекта Гиббса.

Предлагаемый ниже метод обеспечивает высокоточное представление сложных сигналов, заданных в табличной форме и допускающих линейную (или иную) аппроксимацию. Суть метода очень проста: сигнал y(t) заменяется его непрерывной на отрезке [0,T] аппроксимацией, в частности кусочно-линейной. Это позволяет найти произвольно большое число гармоник, что повышает точность спектрального представления y(t) и снижает проявление эффекта Гиббса, обусловленного малым конечным числом гармоник.

Пусть сигнал представлен вектором его значений:

```
Yi={0,0,1,1,1,0,0,0,0,.5,1,1,1,.5,.5,0,.25,.25,0,0}
  {0,0,1,1,1,0,0,0,0,0.5,1,1,1,0.5,0.5,0,0.25,0.25,0,0}
```

Найдем число элементов вектора Yi:

```
Ni=Length[Yi];
```

Выполним линейную интерполяцию без трансформации масштаба времени:

```
yy=Interpolation[Yi,InterpolationOrder→1];
```

График зависимости yy(i+1) при изменении i от 0 до (Ni–1), представленный на рис. 15.86, строится командой

```
Plot[yy[i+1],{i,0,Ni-1}]
```

Для получения временной зависимости в интервале времени [0,T] зададим интервал времени, на котором определен сигнал T, и нужное нам число гармоник M:

```
T=4*10^-6;M=30;
```

Еще раз отметим, что при данном методе число гармоник может значительно превышать N/2, то есть предел, налагаемый теоремой об отсчетах (у нас она известна как теорема Котельникова). Выполним кусочную линейную интерполяцию сигнала с приведением интервала интерполяции к интервалу времени [0,T]:

```
dti=T/Ni;N1=2*M;dt=T/N1;p=2*π*dt/T;p1=2*π/T;
y[t_]:=If[t<T-dti,yy[1-dti+t/dti],0]
```

Если нужна разрывно-ступенчатая аппроксимация y[t], переформатируйте следущую ячейку под входную.

```
y[t_]:=If[t<-dti+T,yy[1-dti+Floor[t/dti]],0]
```

Специфика интерполяции связана с тем, что функция **Interpolation** дает интерполяцию на отрезке [1,Ni]. Введенная функция пользователя y[t] определяет сигнал на отрезке [0,T]. Представленная ниже команда строит график сигнала на этом отрезке (рис. 15.86):

```
Plot[y[t],{t,0,T},PlotStyle→Hue[1]]
```

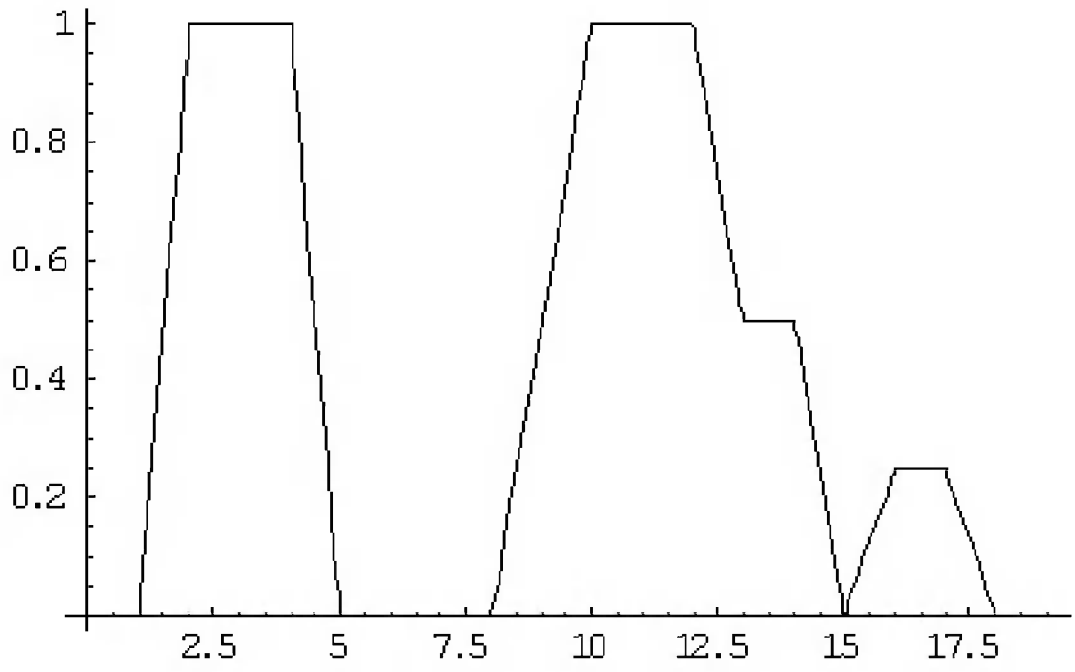


Рис. 15.86. График исходного сигнала на заданном отрезке времени

Когда проивольная функция задана лишь своими отсчетами, расчет косинусных и синусных коэффициентов прямо по их интегральным представлениям нерационален, поскольку не дает никаких преимуществ перед вычислением коэффициентов по формуле интегрирования методом прямоугольников. Ниже представлен такой расчет, причем для ускорения вычислений в общем случае нулевые отсчеты функции (если таковые есть) не обрабатываются:

$$A[k\_] = \sum_{i=0}^{N1-1} If[y[i*dt] == 0,\ 0,\ y[i*dt]*Cos[p*k*i]];$$

```
B[k_] = Sum[If[y[i*dt] == 0, 0, y[i*dt]*Sin[p*k*i]], {i, 0, N1-1}];
A[1]
 -5.22359
B[1]
 0.850285
```

По найденным косинусным и синусным коэффициентам ряда Фурье можно найти амплитуды Mg[k] и фазы Phg[k] гармоник:

```
m=2;
Mg[k_]=Abs[A[k]+i*B[k]]*(2/N1)*(Sin[Pi*f1*k*dt]/(Pi*f1*k*dt))^m;
 Phg[k_]=If[A[k]==0,0,-Arg[A[k]+i*B[k]]];
 Mg[1]
  0.17625
 Phg[1]
   -2.98023
```

Показатель степени m позволяет реализовать три метода расчета амплитуды гармоник:

- m=0 – приближенный (в этом случае целесообразно отказаться от возводимого в степень множителя);
- m=1 – точный метод прямоугольников;
- m=2 – точный метод трапеций (для кусочно-линейного представления y(t) кусочно-линейная).

Амплитудно-частотную и фазочастотную характеристики спектра можно построить, используя команды:

```
<<Graphics`Graphics`
LabeledListPlot[Table[Mg[k],{k,1,M}],PlotRange→{0,.5}]
…Graphics…
LabeledListPlot[Table[Phg[k],{k,1,M}],PlotRange->{-π,π}]
```

Их графики не приводятся, и заинтересованный читатель может вывести их самостоятельно, используя приведенные команды.

Зададим функцию пользователя, вычисляющую мгновенное значение сигнала по заданным M гармоникам:

```
F[t_] = (A[0] / N1) + Sum[If[Mg[k] == 0, 0, Mg[k]*Cos[p1*k*t + Phg[k]]], {k, 1, M}];
```

Нетрудно получить аналитическое выражение для ряда Фурье зависимости F(t):

```
N[F[t]];
```

Ввиду громоздкости вывод этого выражения заблокирован – вы можете убрать точку с запятой в конце строки ввода и просмотреть это выражение. Для ускорения вычислений при вычислении F(t) исключаются гармоники с нулевой амплитудой, если таковые есть.

О том, насколько точно приближение исходной функции тригонометрическим рядом для M гармоник, позволяет судить совмещенный график этих функций (рис. 15.87), который строится командой

```
Plot[{y[t],F[t]},{t,0,T},PlotStyle→{Hue[1],Hue[.65]}]
```

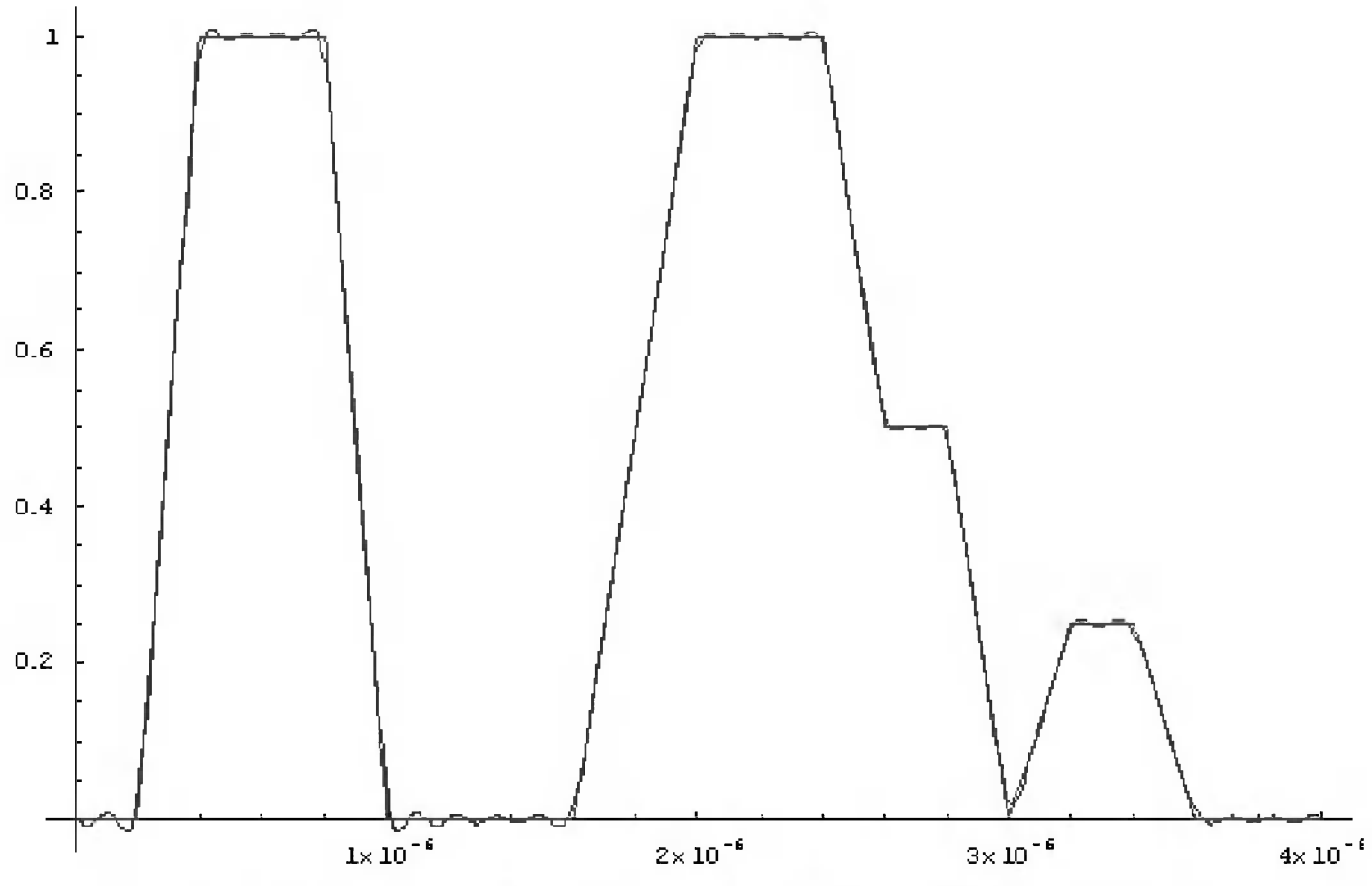


Рис. 15.87. Исходный и синтезированный рядом Фурье сигналы

Мы можем использовать описанный метод для оценки искажения синтезированного сигнала каким-либо устройством, например фильтром низких частот. Зададим амплитудно-частотную АЧХ и фазочастотную ФЧХ характеристики устройства (усилителя, фильтра и т. д.), искажающего сигнал:

```
fh:=5000000;K0:=5;f1:=1/T;
Af[f_]=K0/(1+(f/fh)^2)^2;
Pf[f_]=2*(ArcTan[1-f/fh]-Pi/4);
```

Построим график АЧХ (верхняя линия) и ФЧХ (нижняя линия):

```
Plot[{Af[f],Pf[f]},{f,0,10000000},PlotStyle→{Hue[1],
Hue[.65]}]
```

Соответствующие характеристки представлены на рис. 15.88.

Запишем выражение для гармонического синтеза сигнала, учитывая его искажения (амплитуда каждой гармоники умножается на значение Af на ее частоте k*f1, а к фазовому сдвигу гармоники прибавляется фазовый сдвиг Pf на частоте гармоники).

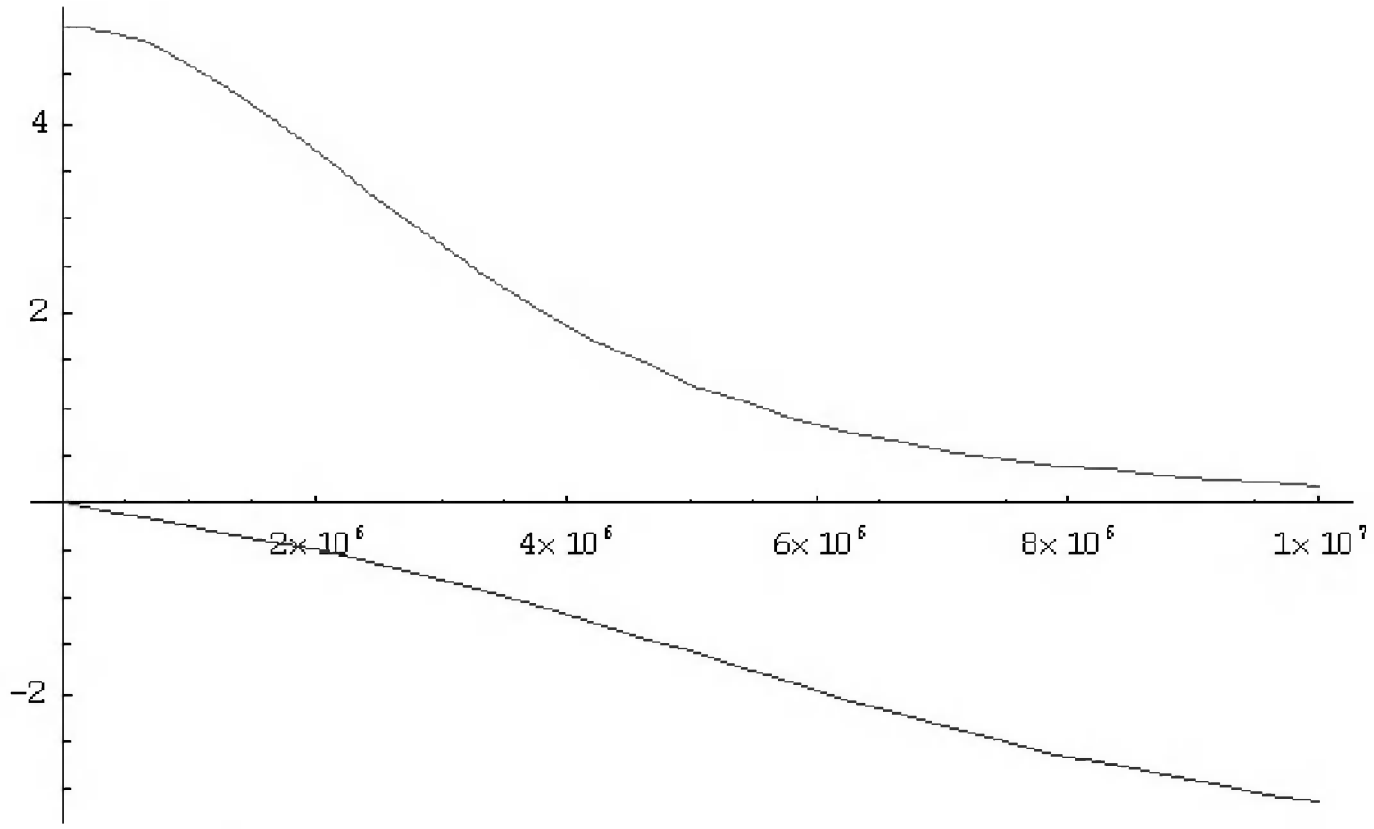


Рис. 15.88. АЧХ (сверху) и ФЧХ (снизу) устройства, искажающего сигнал

```
Ff[t_] = A[0] * Af[0] / N1 +
    Sum[If[Mg[k] == 0, 0, Af[k * f1] * Mg[k] *
        Cos[p1 * k * t + Phg[k] + Pf[k * f1]]], {k, 1, M}];
```

Теперь можно построить исходную временную зависимость сигнала и временную зависимость сигнала после прохождения им искажающей цепи (рис. 15.89):

```
Plot[{y[t],Ff[t]/K0},{t,0,T},PlotStyle→{Hue[1],Hue[.65]}]
```

15.8. Специальные вопросы спектрального анализа и синтеза

15.8.1. Функции БПФ системы Mathcad 12

Не все СКМ имеют функции, реализующие БПФ. Отчасти это связано с тем, что высокая эффективность БПФ обеспечивается только при числе отсчетов функции, кратном двум в целой степени. Если отсчетов меньше, то чаще всего их просто дополняют до числа, кратного двум в целой степени, путем добавления нулевых отсчетов. Достаточно эффективные средства БПФ давно имеют СКМ Mathcad и MATLAB, тогда как в мощных системах символьной математики

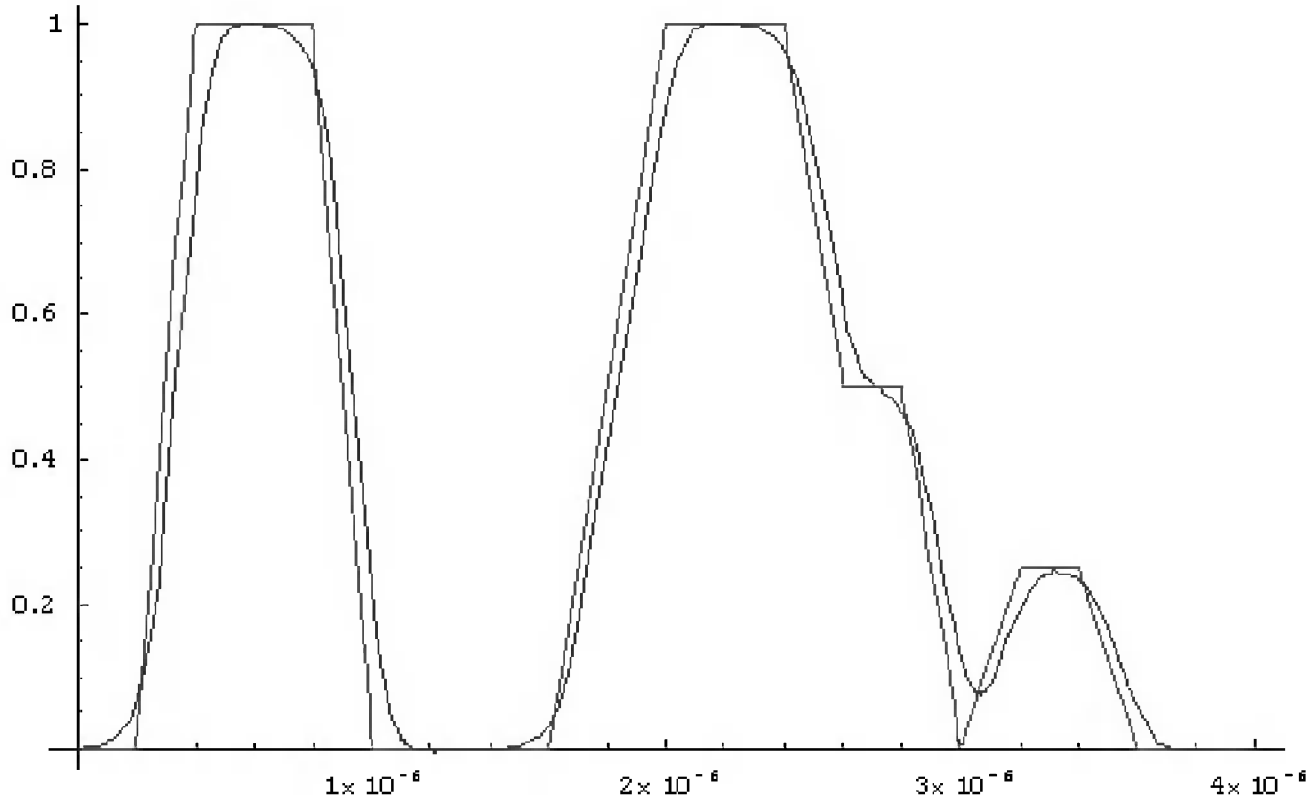


Рис. 15.89. Пример расчета искажений сигнала после его прохождения через искажающее устройство с АЧХ и ФЧХ, представленными на рис. 15.89

Mathematica и Maple такие функции появились лишь в последних версиях. Наконец (см. конец главы 1) бурное развитие программных средств БПФ наблюдается в современных цифровых осциллографах и анализаторах спектра [190–196].

Функция `fft(v)` системы Mathcad выполняет БПФ для данных, представленных действительными числами – значениями исходного вектора `v`. Он должен иметь точно 2^m составляющих, где `m` – целое число. В противном случае выводится сообщение об ошибке – неверном размере вектора. Если число элементов вектора `v` все же отличается от целой степени двух, то его можно дополнить нулями до этой величины. Элементы вектора, возвращаемого функцией `fft(v)`, соответствуют формуле

$$C_j = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} v_k e^{2\pi i (j/n)k}. \qquad (15.10)$$

Здесь n – число элементов вектора $\mathbf{v}$, i – мнимая единица, k – индекс суммирования (от 0 до $n-1$), j – номер гармоники (от 0 до $n/2$). Эти элементы вектора соответствуют следующим частотам:

$$f_j = \frac{j}{n} \cdot f_s.$$

Здесь f_s – частота квантования сигнала, который подвергается БПФ. Элементы вектора, возвращаемого функцией `fff(v)`, – в общем случае комплексные числа, даже если сигнал представлен вещественными отсчетами.

Прямое преобразование Фурье по существу означает перевод временной зависимости в ее частотный спектр. А обратное преобразование Фурье переводит частотный спектр вновь во временную зависимость.

Функция `ifft(v)` реализует обратное (инверсное) преобразование Фурье для вектора `v` с комплексными элементами. Вектор `v` здесь должен иметь `1+2m+1` элементов. В противном случае выдается сообщение об ошибке. Функция `ifft(v)` вначале создает вектор `w`, комплексно-сопряженный с `v`, и затем присоединяет его к вектору `v`. После этого вычисляется вектор `d` с элементами, рассчитанными по формуле

$$d_j = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} w_k e^{-2\pi i(j/n)k}. \qquad (15.11)$$

Функции `fft(v)` и `ifft(v)` дают точные (в пределах погрешности численных расчетов) обращения. При этом `ifft(fft(v))=v`, что можно использовать для проверки преобразований.

Функция `cfft(A)` аналогична предыдущей, но реализует прямое преобразование Фурье для вектора `A` с комплексными элементами. Если `A` – матрица, реализуется двумерное преобразование. Введение функции `fft(V)` обусловлено тем, что преобразование для векторов с действительными элементами реализуется по более быстрому алгоритму (БПФ) и занимает меньше времени. В этом случае более прост и ввод исходных данных.

Функция `icfft(B)` выполняет обратное преобразование Фурье по полному алгоритму, при котором как исходный, так и результирующий векторы или матрицы содержат элементы с комплексными значениями.

15.8.2. Примеры выполнения БПФ в Mathcad 12

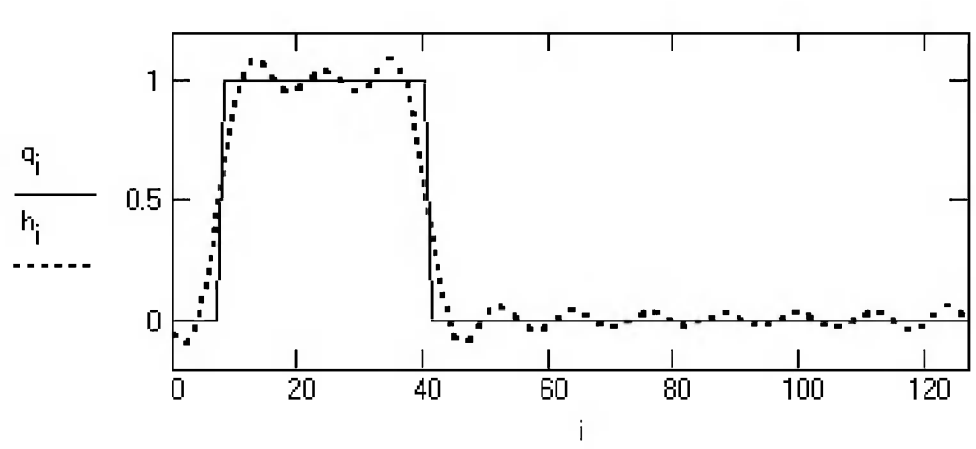
Для проверки функций БПФ можно задать некоторый вектор из 2^m действительных или комплексных элементов. Проведя прямое преобразование, получим новый вектор. Затем над ним проведем обратное преобразование. Можно заметить, что полученный таким двукратным преобразованием вектор полностью совпадает с исходным. Читатель может легко проверить эту методику тестирования функций самостоятельно. Мы же перейдем к более сложным примерам.

Техника проведения БПФ на примере разложения прямоугольного импульса и последующего его синтеза с помощью ряда Фурье с ограниченным (k = 10) числом гармоник демонстрируется на рис. 15.90. Здесь исходный вектор задан элементами действительного типа, поэтому используются функции `fft` и `ifft`.

На рис. 15.91 представлено построение АЧХ и ФЧХ спектра прямоугольного импульса (см. документ рис. 15.90) для случая, когда прямое БПФ реализовано функцией `fft`. Обратите внимание на то, что для вычисления амплитуд гармоник используется оператор вычисления модуля, а для вычисления фаз гармоник – функция `arg`.

ПРИМЕНЕНИЕ БПФ ДЛЯ СПЕКТРАЛЬНОГО РАЗЛОЖЕНИЯ И СИНТЕЗА ИМПУЛЬСА

$i := 0..127$	$q_i := 0$	Задание сигнала в виде прямоугольного импульса
$i := 8..40$	$q_i := 1$	
$f := fft(q)$	Прямое БПФ	
$k := 10$	Задание максимального числа гармоник	
$j := 0..64$	$g_j := if(j \le k, f_j, 0)$	Ограничение числа гармоник
$h := ifft(g)$	Обратное БПФ	
$i := 0..127$	Построение графиков исходого и преобразованного сигналов	



Характерные колебательные процессы на графике преобразованного сигнала иллюстрируют эффект Гиббса

Рис. 15.90. Применение БПФ для спектрального разложения и синтеза прямоугольного импульса

При вычислении АЧХ и ФЧХ использован график типа error, причем за вторую линию взят 0. При использовании функции `fft` спектр ограничен числом гармоник, вдвое меньшим, чем число отсчетов сигнала – в нашем случае прямоугольного импульса. Это ограничение хорошо видно на рис. 15.91. АЧХ и ФЧХ спектра имеют обычный вид.

А на рис. 15.92 показаны АЧХ и ФЧХ того же импульса, но полученные с помощью функции cfft. На этот раз можно заметить два принципиально важных отличия: выходной вектор имеет то же число отсчетов, что и входной, то есть вдвое больше, чем при использовании функции fft; АЧХ и ФЧХ спектра имеют двойную длину и представлены как обычным своим видом в левой части графика, так и их зеркальным отражением в правой части графика.

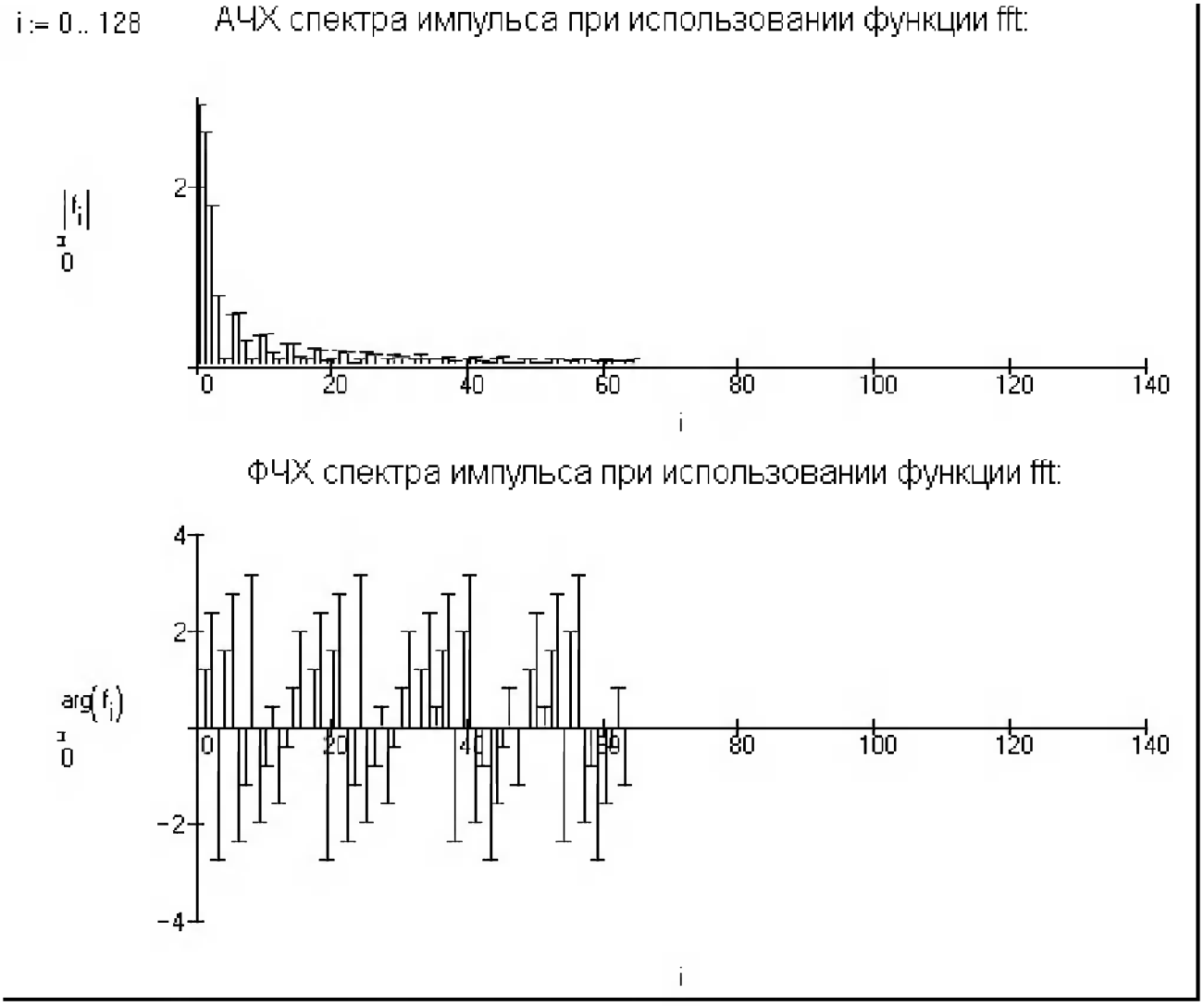


Рис. 15.91. АЧХ (сверху) и ФЧХ (снизу) спектра прямоугольного импульса (рис. 15.99) при использовании функции fft для прямого БПФ

15.8.3. Альтернативные преобразования Фурье

Рассмотренные выше функции основаны на обычных формулах преобразований Фурье. Однако существуют и *альтернативные формы* такого преобразования, две из которых показаны ниже:

$$F(v)=\frac{1}{n}\sum_{\tau=1}^{n} f(\tau)e^{-2\pi i\tau(v/n)},$$

$$f(\tau)=\sum_{v=1}^{n} F(v)e^{2\pi iv(\tau/n)}.$$

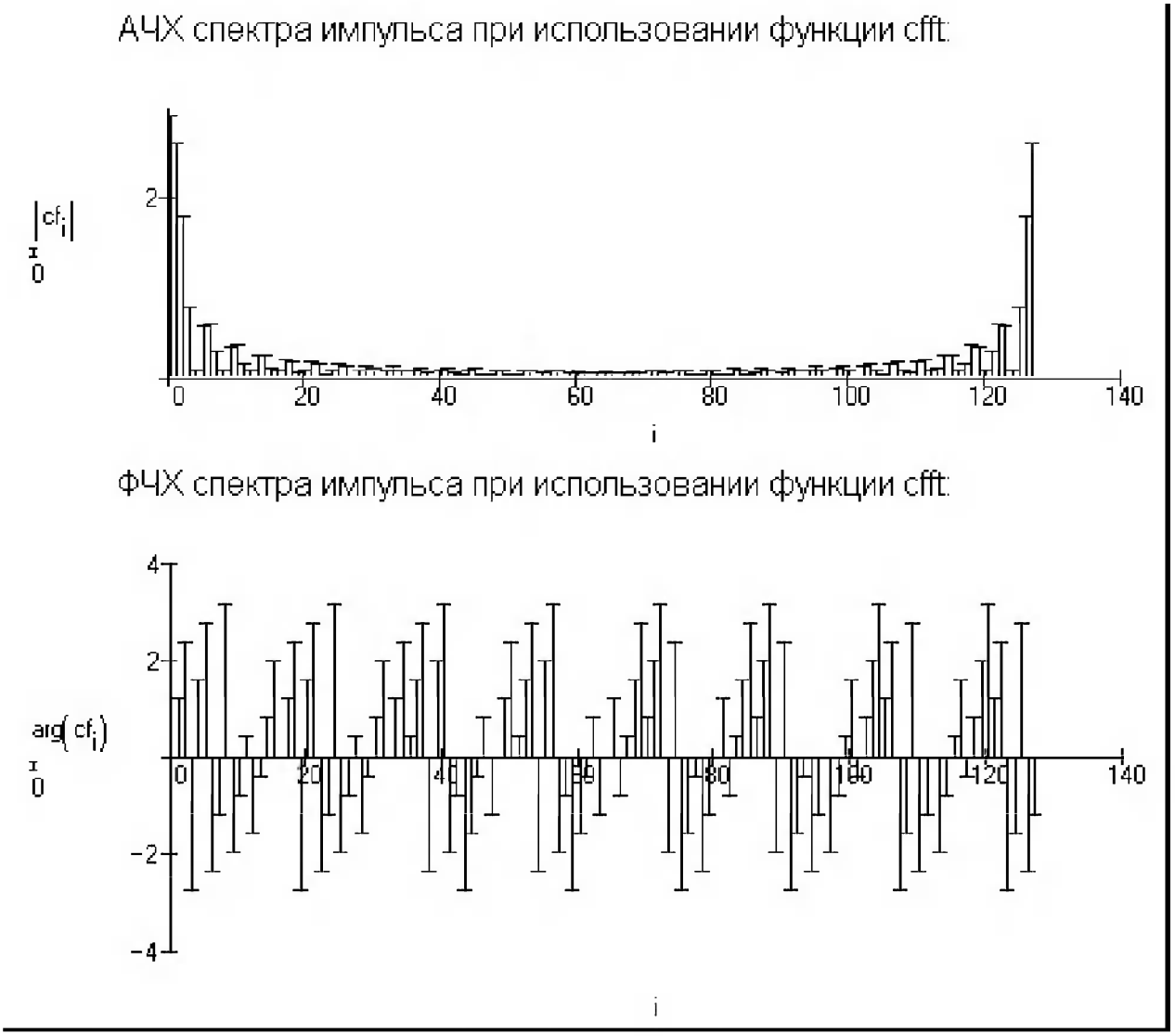


Рис. 15.92. АЧХ (сверху) и ФЧХ (снизу) спектра прямоугольного импульса (рис. 7.15) при использовании функции cfft для прямого БПФ

Вместо множителя $1/\sqrt{n}$ перед обоими выражениями перед первым выражением стоит множитель $1/n$, а перед вторым – 1. Знак «минус» перед показателем степени имеется только в первой формуле (его нет во второй).

Если в функциях преобразования Фурье системы Mathcad в качестве входного параметра задана матрица, то реализуется двумерное обратное преобразование Фурье. В ранних версиях Mathcad эта возможность отсутствовала, но могла быть реализована (хотя и более сложным путем) с помощью одномерного БПФ.

15.8.4. Эффект Гиббса

Мы уже столько раз упоминали эффект Гиббса, возникающий при спектральном синтезе (моделировании) сигналов, что пора разобраться с причинами его возникновения и обсудить способы борьбы с ним. Поскольку пульсации эффекта Гиббса наиболее явно проявляются в моменты скачков анализируемой функции, ограничимся рассмотрением случая представления скачка

$$f(x)=\begin{cases}0, & x<0\\ 1, & x\geq 0\end{cases}$$

ограниченным спектром. Ограничение спектра можно учесть, введя при обратном преобразовании Фурье умножение $F(\omega)$ на прямоугольное частотное окно

$$W(\omega)=\begin{cases}1, & |\omega|\leq\gamma\\ 0, & |\omega|>\gamma\end{cases}.$$

Это окно задает резкое ограничение спектра. Опуская формальные математические детали вывода, которые можно найти в [144, 145, 158], найдем, что в этом случае

$$f(x)=\frac{1}{2}+\frac{1}{\pi}Si(\gamma x),$$

где функция $Si(z)$ известна как интегральный синус

$$Si(z)=\int_0^z\frac{\sin(t)}{t}dt.$$

Задание функции $Si(z)$ и построение зависимости $f(x)$ для данного случая представлены на рис. 15.93.

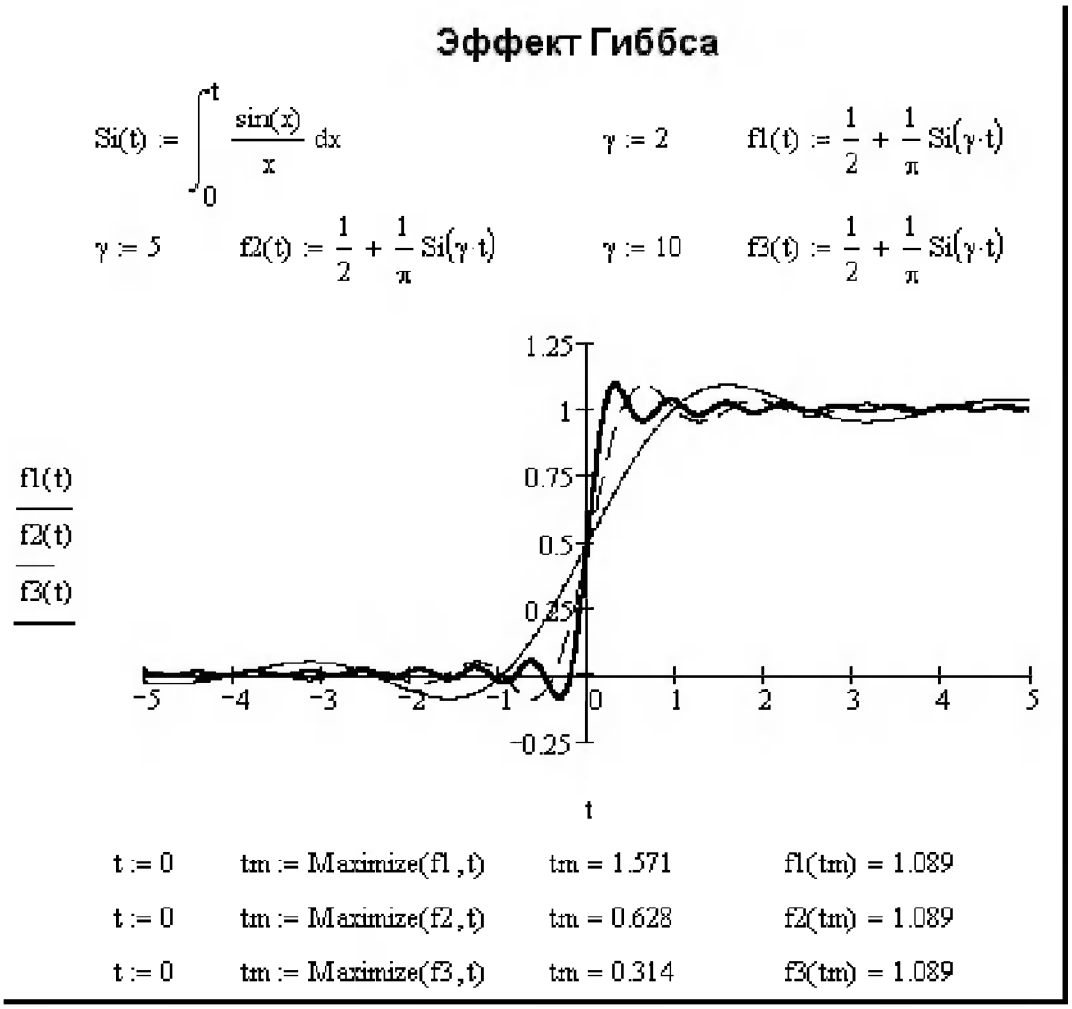


Рис. 15.93. Иллюстрация к возникновению эффекта Гиббса

Итак, как следует из рассмотренного, эффект Гиббса существует, увы, как теоретически обоснованная реальность. И связан он прежде всего с неудачно подобранной (или просто поневоле существующей) формой частотного окна, резко ограничивающего число используемых при спектральном синтезе гармоник (частот), – прямоугольного.

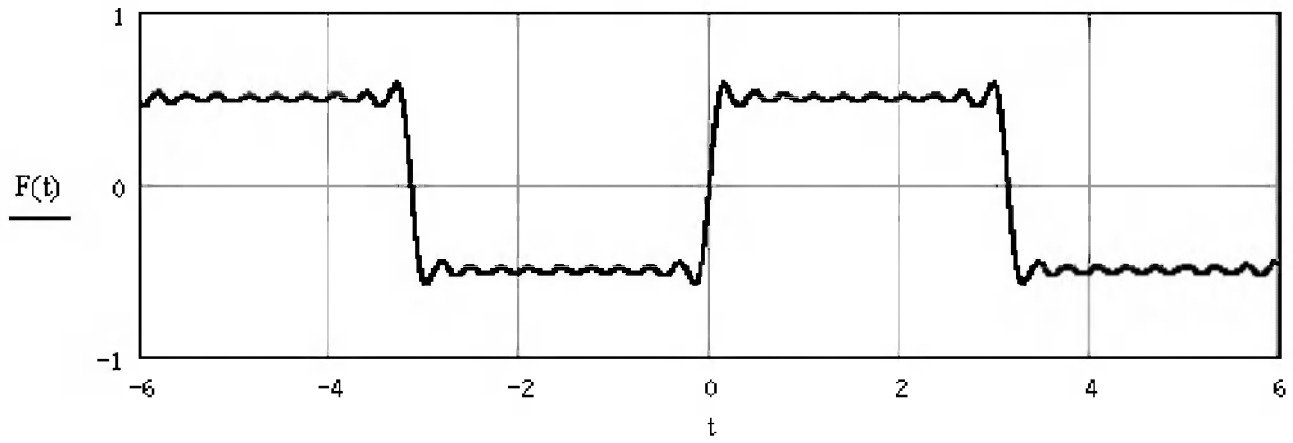
15.8.5. Способы подавления эффекта Гиббса

Одним из давно известных методов подавления эффекта Гиббса является применение сигма-множителей, уменьшающих амплитуды гармоник по мере роста их номера *n* при приближенном вычислении коэффициентов Фурье методом прямоугольников – рис. 15.94. На представленном документе даны ссылки на работы, в которых предложены методы борьбы с эффектом Гиббса.

ПОДАВЛЕНИЕ ЭФФЕКТА ГИББСА

Эффект Гиббса - возникновение осцилляций на кривой, аппроксимирующей зависимость рядом Фурье, в местах изломов аппроксимируемой зависимости.

$N := 10$ $\quad n := 1..N$ $\quad F(t) := \frac{2}{x} \cdot \sum_{n} \frac{\sin[(2 \cdot n - 1) \cdot t]}{2 \cdot n - 1}$ $\quad t := -6, -5.95..6$



Для устранения осцилляций в местах разрыва непрерывности функции вводятся множители, уменьшающие коэффициенты ряда Фурье при больших n : сигма- множитель Ланцоша (Ланцош К. Практические методы прикладного анализа. М.: Физматгиз, 1961.-524 с.) и множитель Фейера (Савелова Т.И. О регуляризации при помощи операторов типа Фейера. Журнал вычислительной математики и математической физики, 1978, т. 18, № 3, с. 582-588).

Рис. 15.94. Синтез меандра по 10 гармоникам

Реализация метода сигма-множителей представлена рис. 15.95. Нетрудно заметить, что эффект Гиббса резко ослабляется, но небольшие его проявления все же остаются.

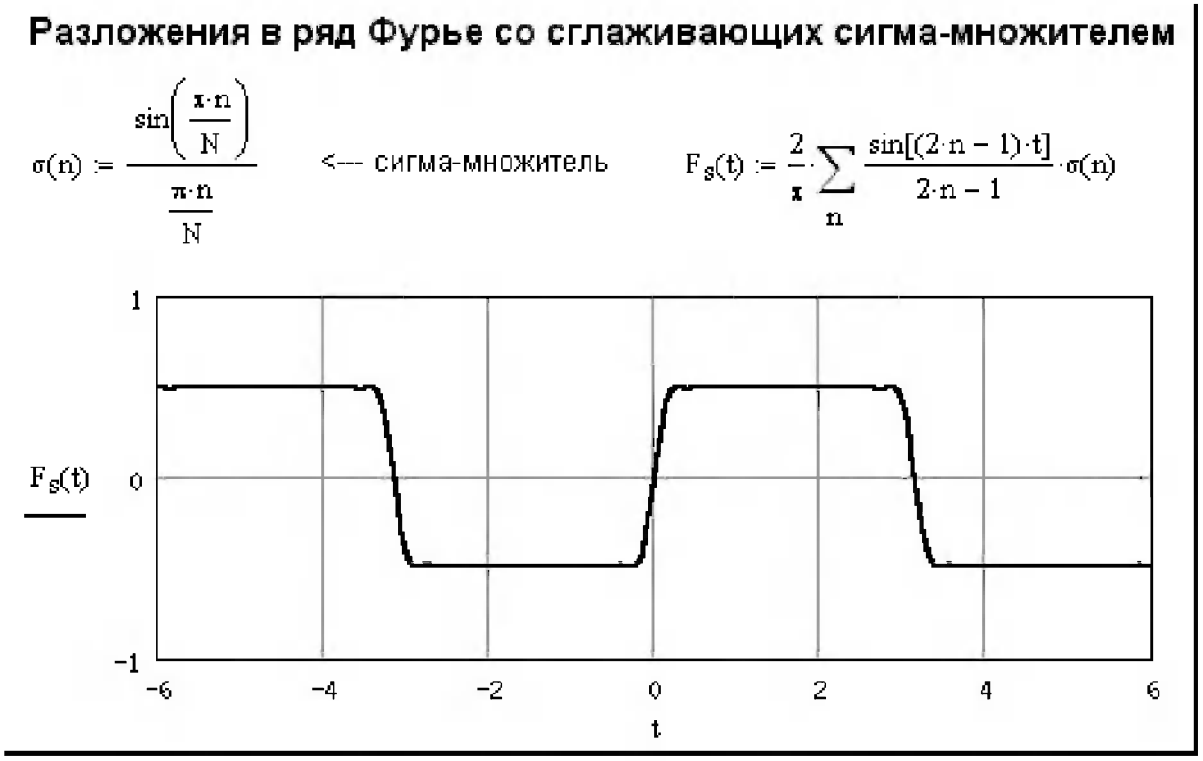


Рис. 15.95. Синтез меандра по 10 гармоникам с сигма-множителями

Другой метод борьбы с эффектом Гиббса при использовании синтеза рядом Фурье заключается в применении множителей Фейера. Их определение и применение представлены на рис. 15.96.

Этот метод обеспечивает практически полное подавление эффекта Гиббса. Но его недостатком является несколько скошенная вершина импульсов.

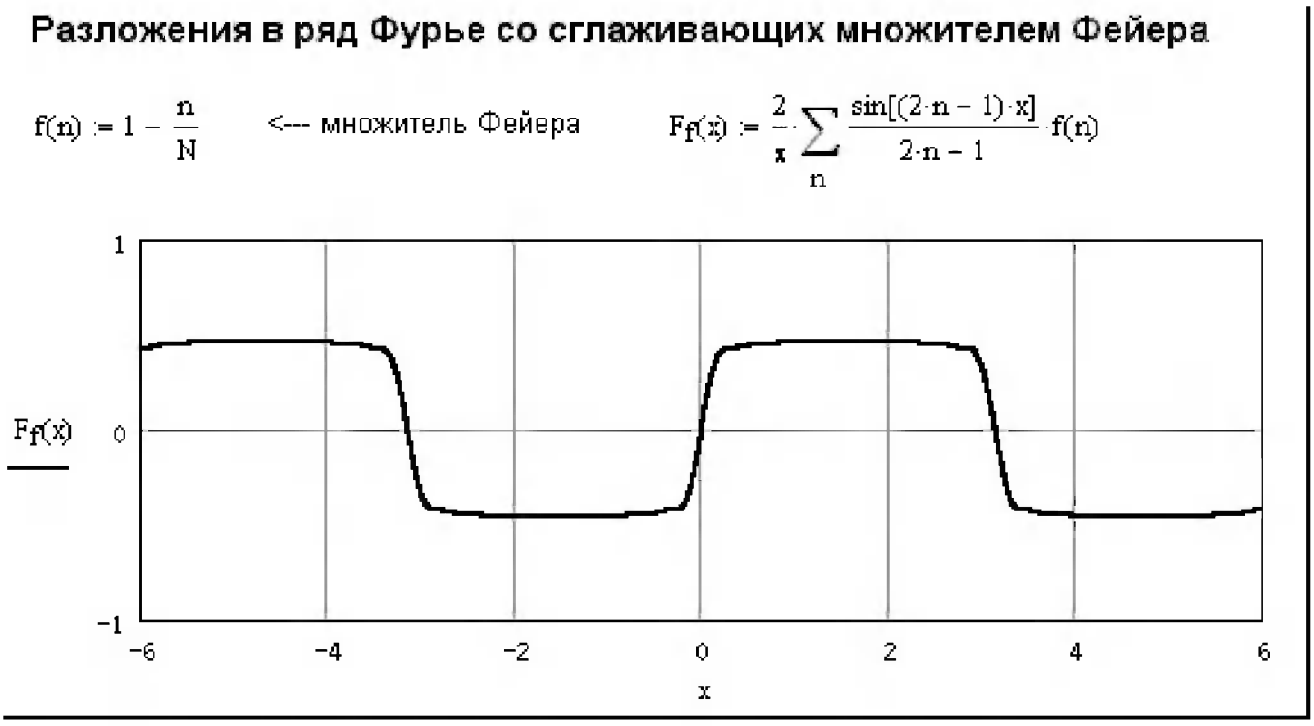


Рис. 15.96. Синтез меандра по 10 гармоникам с множителями Фейера

15.8.6. Восстановление сигнала по базису Котельникова

При цифровой обработке сигналов часто производят их *выборку* (вырезку) в определенные моменты времени (рис. 15.97 сверху). Они могут равномерно или неравномерно отстоять друг от друга. Сигнал может быть периодическим (с периодом T), но может быть и однократным, или непериодическим.

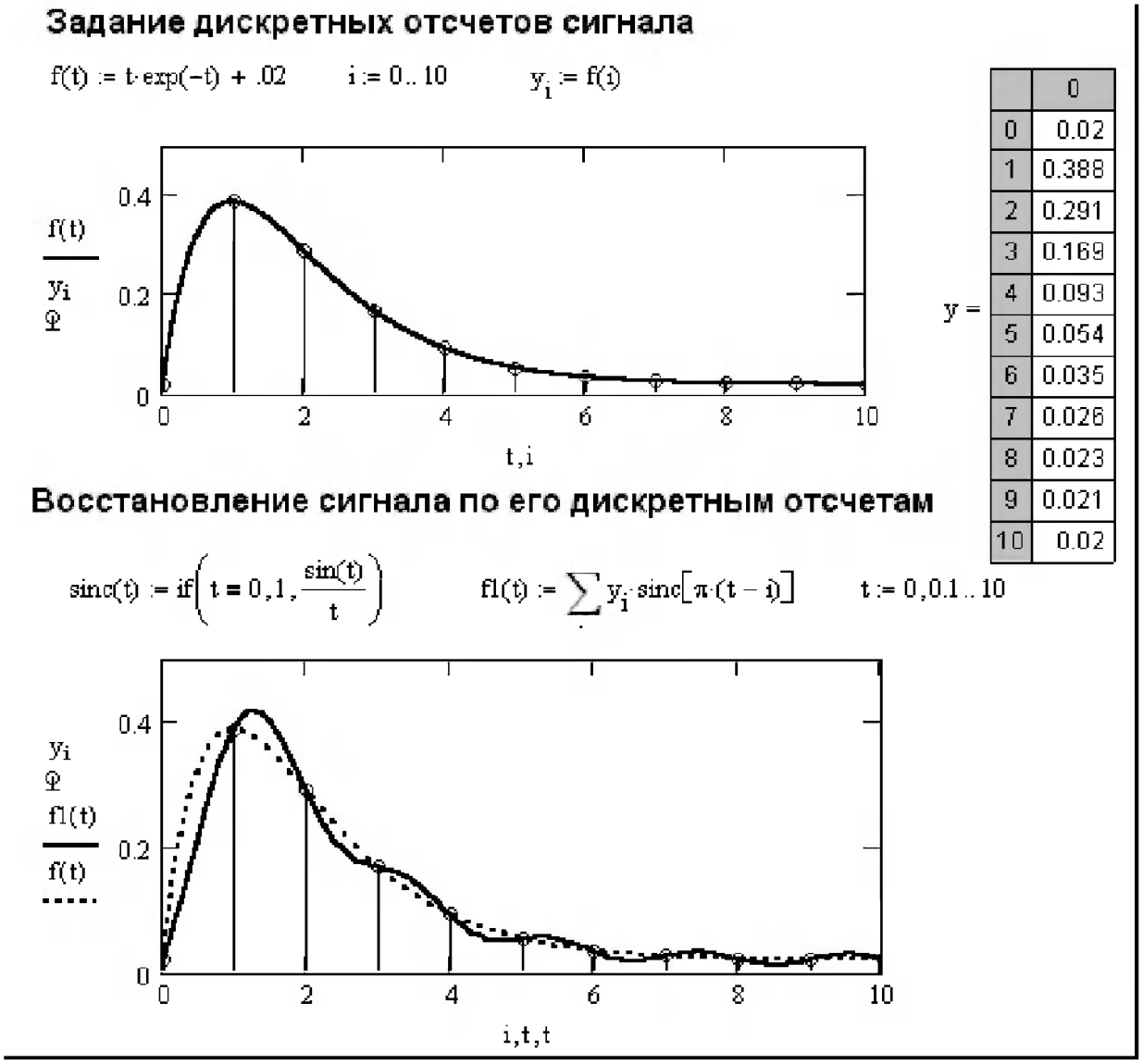


Рис. 15.97. Задание непрерывного сигнала его выборками и его восстановление на основе теоремы Котельникова

Выборку электрических сигналов и их представление в виде чисел или кодов конечной разрядности выполняют так называемые *аналого-цифровые преобразователи* – АЦП. В результате на выходе АЦП мы имеем дискретный сигнал, представленный потоком чисел (кодов). Главные показатели АЦП – это их разрядность (число уровней квантования, обычно выражаемое в двоичном виде) и скорость выполнения преобразований (число операций в секунду).

Обратное преобразование цифровой информации в аналоговую выполняют *цифро-аналоговые преобразователи* – ЦАП.

Как часто надо делать равномерные выборки произвольного сигнала, чтобы после преобразования в цифровую форму, а затем снова в аналоговую была сохранена форма сигнала? Ответ на этот важный вопрос дает теорема об отсчетах, или теорема Котельникова (за рубежом именуема также теоремой Найквиста): «Если спектр сигнала $e(t)$ ограничен высшей частотой $f_в$, то он без потери информации может быть представлен дискретными отсчетами с числом, равным $2 \cdot f_в$». При этом сигнал восстанавливается по его отсчетам $e(k \cdot dt)$, следующим с интервалом времени dt, с помощью фильтра низких частот, реализующего восстановление по формуле

$$e(t) = \sum_{k=-\infty}^{\infty} e(k \cdot dt) \frac{\sin(\pi(t - k \cdot dt)/dt)}{\pi(t - k \cdot dt)/dt}. \qquad (15.12)$$

Для восстановления непрерывного сигнала по его выборкам достаточно располагать функцией $\mathrm{sinc}(x)=\sin(x)/x$ с учетом ее особого значения $\mathrm{sinc}(x)=1$ при $x=0$. Такая функция есть в системе Mathcad 11/12, но ее нет в предшествующих версиях системы Mathcad. Однако, как показано на рис. 15.97 (часть документа снизу), такую функцию несложно задать с помощью функции `if`. Заодно на рис. 15.97 снизу показано восстановление сигнала по его отсчетам на основе теоремы Котельникова. Примечательно, что функция $\mathrm{sinc}(x)=\sin(x)/x$ имеет максимум при $x=0$ и колебательно спадает в обе стороны, что чем-то напоминает поведение некоторых вейвлетов (см. следующую главу).

Рисунок 15.97 показывает, что даже при небольшом числе отсчетов (в нашем случае их 11) восстановление сигнала (или, можно сказать, его интерполяция) происходит вполне прилично, хотя и неидеально. Интерполирующая кривая получается довольно плавной и непрерывной и точно проходит через точки исходных отсчетов. Увы, злополучный эффект Гиббса (колебания интерполирующей кривой, не присущие исходному сигналу) и здесь имеет место. Его причиной, как всегда, является ограниченное число выборок. При увеличении числа выборок точность восстановления можно существенно улучшить, что часто и делается на практике.

15.9. Оконное преобразование Фурье

15.9.1. Ограничения и недостатки преобразования Фурье

С позиций точного представления преобразованием Фурье произвольных сигналов и функций можно отметить целый ряд его недостатков:

- неприменимость к анализу нестационарных сигналов;
- преобразование Фурье даже для одной заданной частоты требует знания сигнала не только в прошлом, но и в будущем, что является теоретической абстракцией;

- в условиях практически неизбежного ограничения числа гармоник или спектра колебаний точное восстановление сигнала после прямого и обратного преобразований Фурье теоретически (и тем более практически) невозможно, в частности из-за появления эффекта Гиббса;
- базисной функцией при разложении в ряд Фурье является гармоническое (синусоидальное) колебание, которое математически определено в интервале времени от $-\infty$ до $+\infty$ и имеет неизменные во времени параметры;
- численное интегрирование во временной области от $-\infty$ до $+\infty$ при прямом преобразовании Фурье (ППФ) и от $-\infty$ до $+\infty$ в частотной области при обратном преобразовании Фурье (ОПФ) встречает большие вычислительные трудности;
- отдельные особенности сигнала (например, разрывы или пики) вызывают незначительные изменения частотного образа сигнала во всем интервале частот от $-\infty$ до $+\infty$, которые «размазываются» по всей частотной оси, что делает их обнаружение по спектру практически невозможным;
- ясно, что такая плавная базисная функция, как синусоида, в принципе вообще не может представлять перепады сигналов с бесконечной крутизной, хотя такие сигналы (например, прямоугольные импульсы) применяются весьма широко;
- единственным приспособлением к представлению быстрых изменений сигналов, таких как пики или перепады, является резкое увеличение числа гармоник, которые оказывают влияние на форму сигнала и за пределами локальных особенностей сигнала;
- по составу высших составляющих спектра практически невозможно оценить местоположение особенностей на временной зависимости сигнала и их характер;
- для нестационарных сигналов (а таковых сейчас большинство) трудности ППФ и ОПФ (и, соответственно, быстрого преобразования Фурье – БПФ) многократно возрастают.

Небольшие разрывы (ступеньки) на синусоидальном или любом плавно изменяющемся сигнале трудно обнаружить в его Фурье-спектре, ибо они создают множество высших гармоник очень малой амплитуды. Спектр таких сигналов содержит едва заметные высокочастотные составляющие спектра, по которым распознать локальную особенность сигнала и тем более ее место и характер практически невозможно. Составляющие спектра особенности как бы размазаны по оси частот.

15.9.2. Кратковременное (оконное) преобразование Фурье

Проблемы спектрального анализа и синтеза сигналов, ограниченных во времени, частично решаются переходом к так называемому *кратковременному*, или *оконному*, *преобразованию Фурье*. Идея этого преобразования очень проста: временной

интервал существования сигнала разбивается на ряд промежутков – временных окон. В каждом промежутке вычисляется свое преобразование Фурье. Если в каком-то окне существовали частотные составляющие некоторого сигнала, то они будут присутствовать в спектре. А если нет – будут отсутствовать. Таким образом, можно перейти к частотно-временному представлению сигналов, которое является особым разделом техники обработки сигналов.

Кратковременное (оконное) преобразование выполняется с использованием выражения

$$A(\omega) = \int_{-\infty}^{\infty} y(t) \cdot w(t-b) \cdot e^{-i\omega t} dt. \qquad (15.13)$$

Здесь, в отличие от интеграла Фурье, функция $y(t)$ под знаком интеграла дополнительно умножается на оконную функцию $w(t-b)$. Параметр b окна задает его сдвиг на временной оси. Обычно задается ряд фиксированных значений b в пределах полного окна. Например, для простейшего *прямоугольного окна* функция $w(t-b)$ в пределах окна дает 1, а за пределами окна просмотра – 0. При этом для каждого окна мы получаем свой набор комплексных амплитуд сигнала в частотной области.

Естественно, что поскольку каждое окно охватывает небольшой участок по времени, точность описания локальных изменений сигнала может быть повышена. Часто используются окна Гаусса или иные окна, обеспечивающие малые искажения спектра из-за граничных явлений и уменьшающие проявление эффекта Гиббса.

Казалось бы, раз оконное преобразование Фурье дает нам частотно-временное представление сигналов, то достаточно им и ограничиться. И не нужно было открывать вейвлет-преобразования, описанные в главе 16.

Однако ситуация не так проста! Она упирается в известный принцип неопределенности Гейзенберга. Согласно ему, невозможно получить одновременно высокое частотное и высокое временное разрешения. Выбирая окно с малой шириной по времени, мы получаем высокое временное разрешение, но низкое частотное разрешение. Взяв окно с большой шириной во времени, получаем хорошее разрешение по частоте, но плохое – во времени. Оконное преобразование оперирует с окнами, имеющими одинаковую ширину, а потому данное противоречие для него неразрешимо.

15.9.3. Функции оконного спектрального анализа в пакете Signal Processing СКМ Mathcad

Пакет расширения Signal Processing популярной системы Mathcad имеет ряд функций оконного (короткого) спектрального анализа. Мы не рассматриваем этот пакет полностью, но отметим применение функций оконного спектрального анализа. Они позволяют разбить сигнал на диапазоны (окна) как без перекрытия, так и с перекрытием и выполнить спектральный анализ следующего типа:

- `pspectrum(x,n,r[,w])` – расчет средней спектральной мощности сигнала x;
- `cspectrum(x,n,r[,w])` – расчет кросс-спектра сигнала x;
- `coherence(x,y,n,r[,w])` – расчет когерентности сигналов;
- `snr (x,y,n,r[,w])` – расчет отношения сигнал/шум для векторов x и y.

В этих функциях x и y – векторы с комплексными или вещественными элементами, n – число поддиапазонов входного сигнала (лежит в пределах от 1 до длины вектора x), r – фактор перекрытия поддиапазонов (от 0 до 1) и w – код окна, выбираемый следующим образом:

1) `rectangular` – прямоугольное окно;
2) `tapreg rectangular` – окно типа трапеции;
3) `triangular` – треугольное окно;
4) `hanning` – окно Хэннинга;
5) `hamming` – окно Хэмминга;
6) `blackman` – окно Блэкмана.

Свойства окон различного типа описаны в [190, 191]. Прямоугольное окно имеет наилучшее частотное разрешение, но плохое амплитудное разрешение. Другие окна улучшают амплитудное разрешение и отличаются уровнем боковых лепестков.

15.9.4. Спектральный анализ с помощью функций CFFT и pspectrum

Поскольку мощность сигнала пропорциональна его уровню, то построение спектра спектральной мощности (СПМ) сигналов с помощью функции `pspectrum` позволяет эффективно отсеивать составляющие спектра с малым уровнем – например, боковые лепестки радиоимпульсов. Это наглядно иллюстрирует рис. 15.98, на котором задано построение радиоимпульса (пачки синусоидальных колебаний) и построены спектры, вычисляемые функциями `CFFT` и `pspectrum`.

Нетрудно заметить, что боковые лепестки у основных спектральных линий, хорошо заметные в спектре, полученном функцией `CFFT`, практически отсутствуют в спектре, созданном с помощью функции `pspectrum`.

15.9.5. Спектры на основе оконного преобразования Фурье

Существенным недостатком спектров, полученных при обычном преобразовании Фурье, является их малая информативность. Локальные особенности сигналов, например короткие всплески или провалы, разрывы и ступеньки и т. д., ведут к появлению в спектре высших гармоник с малой амплитудой, размазанных по всей частотной оси. Определить по ним характер локальных особенностей довольно трудно.

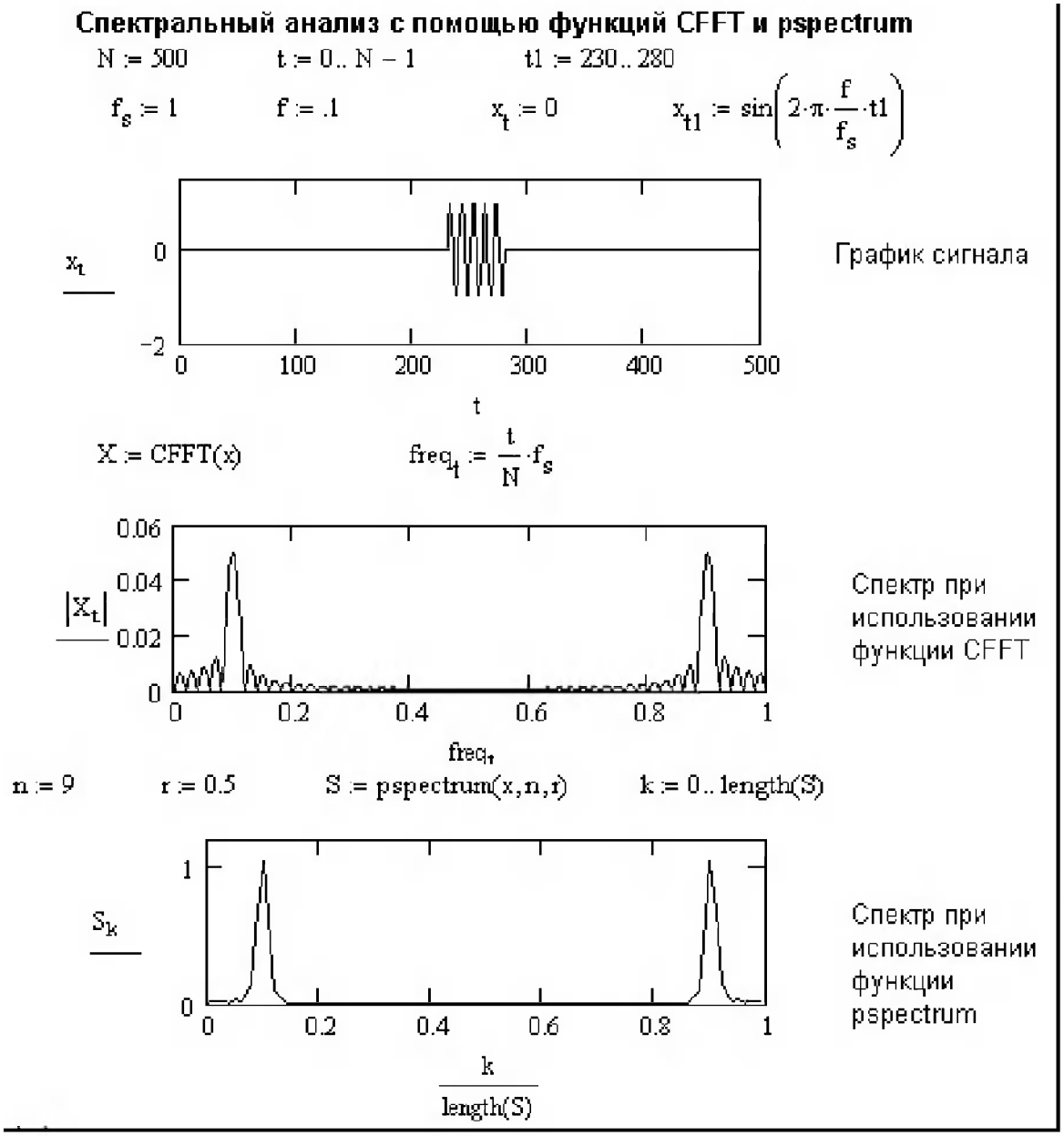


Рис. 15.98. Построение спектров радиоимпульса с помощью функций CFFT и pspectrum

Одним из методов улучшенной визуализации спектра является *короткое (оконное) преобразование Фурье*. Оно реализуется функцией stft(x[,n][,s][,w]), где:

- x – вектор данных действительных или комплексных;
- n – число частот преобразования (по умолчанию 64);
- s – число пропущенных при преобразовании периодов частоты дискретизации (по умолчанию n/2);
- w – индекс окна или вектор с его коэффициентами.

Окно можно задавать или вектором его коэффициентов, или индексом (см. раздел 15.9.3). Формула для короткого оконного преобразования дана сверху на рис. 15.99 для сигнала, который представляет собой две пачки синусоидальных колебаний с разной частотой.

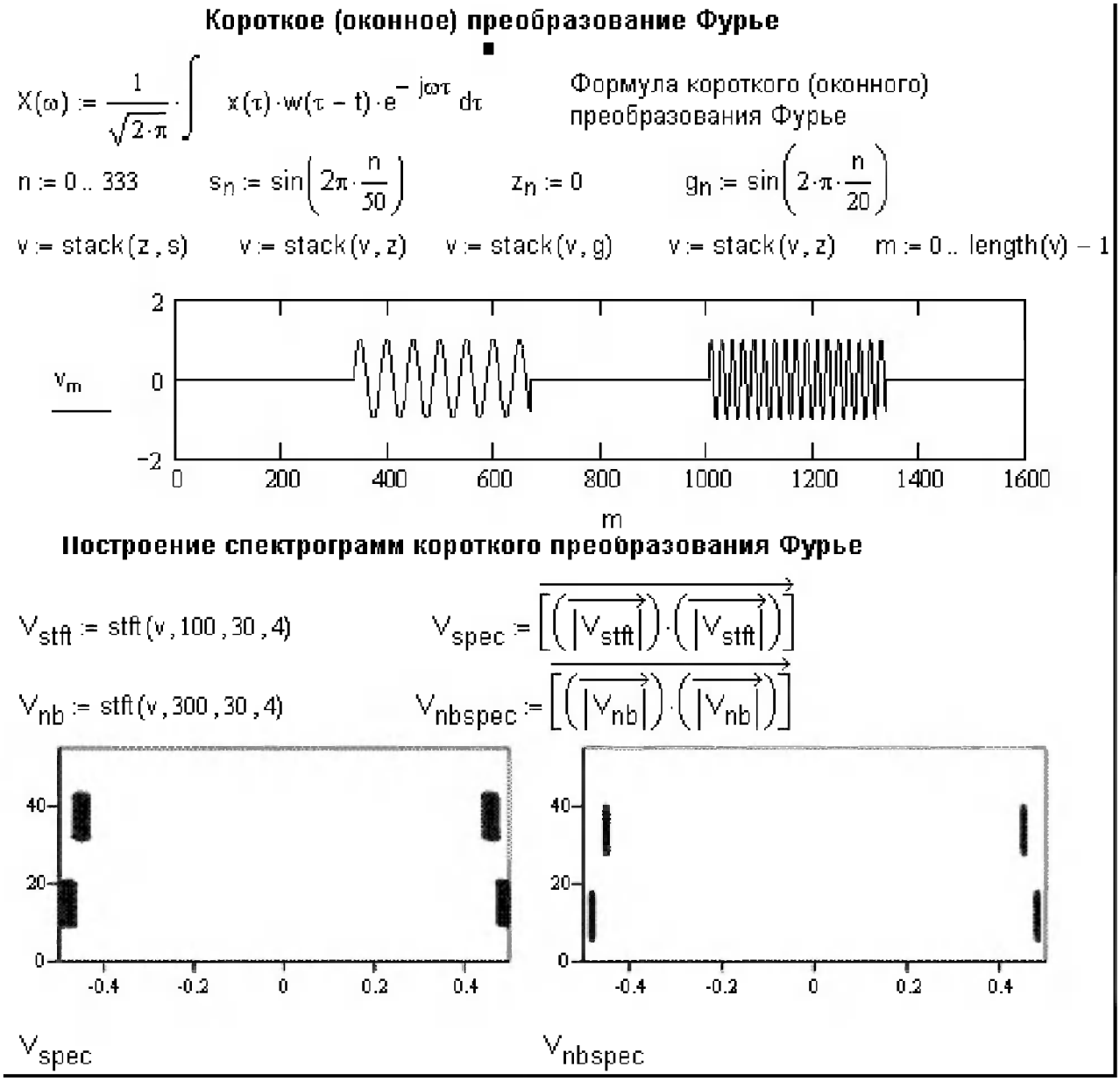


Рис. 15.99. Короткое (оконное) преобразование Фурье для двух пачек синусоидальных колебаний

В отличие от обычного преобразования Фурье, функция x(τ) умножается на окно w(τ–t), значение которого (в первом приближении) равно 1 в пределах окна и 0 за его пределами. Это окно перемещается по временной оси без перекрытия, и спектр по методу Фурье вычисляется в пределах каждого окна. Таким образом, получается набор спектрограмм в плоскости время–частота – спектрограммы представлены ниже для окон разной ширины. На спектрограммах можно четко выделить спектральные линии для обеих пачек синусоидального сигнала. Отчетливо выделяются их длительность и длительность пауз между пачками. Спектрограммы строят современные анализаторы спектра реального времени, кратко описанные в конце главы 1.

Глава 16

Вейвлеты и вейвлет-преобразования

Вейвлеты (всплески, или короткие волны) – это новый базис для представления произвольных зависимостей, как абсолютно точного, так и приближенного [162–168]. В отличие от рядов Фурье, компонентами которых являются синусоиды, определенные в пределах от $-\infty$ до $+\infty$, вейвлет-функции определены на конечном интервале и отличаются многообразием форм вейвлет-образующих функций. Они позволяют приближать и обрабатывать нестационарные сигналы. Ниже описаны основы вейвлет-технологии обработки зависимостей и сигналов и примеры их реализации с помощью СКМ.

16.1. Пример вейвлет-преобразований с применением вейвлета Хаара

В литературе по вейвлетам [163–168, 178] широко используется понятие *носителя* функции. Носителем функции фактически является область ее определения. Например, если $f(x)$ определена при $x \in [a,b]$ и $f(x) = 0$ при $a < x$ и $x > b$, а $b - a$ невелико, то говорят, что функция $f(x)$ имеет *компактный носитель*. Если $a = -\infty$ или $b = +\infty$ или имеет место и то и другое, то функция компактного носителя не имеет.

Прежде чем мы рассмотрим вейвлет-преобразования детально, остановимся на наглядном представлении этих преобразований [178]. Воспользуемся для этого самым простым *вейвлетом Хаара* – рис. 16.1. Несмотря на свою простоту и ряд недостатков, он имеет компактный носитель, относится к ортогональным вейвлетам и обеспечивает теоретическую возможность точной декомпозиции и синтеза любого сигнала.

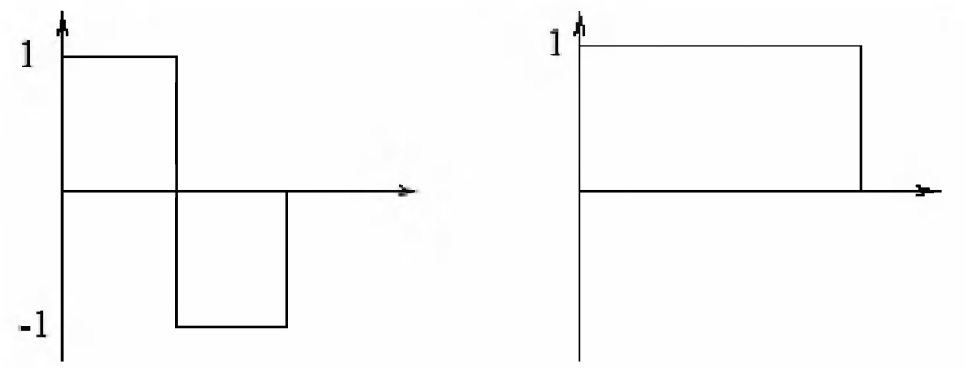


Рис. 16.1. Вейвлет Хаара (слева – аппроксимирующая функция, а справа – детализирующая функция)

Вначале выполним *декомпозицию* сигнала. Для этого используются две функции вейвлетов Хаара. Одна – это *аппроксимирующая функция*, которая у вейвлета Хаара просто равна 1 на всем компактном носителе. А вот другая – *детализирующая функция* – имеет значение +1 на первой половине длины носителя и –1 на второй половине (словом, на компактном носителе размещен один период меандра).

Аппроксимирующая функция в данном случае указывает на усреднение значений сигнала, а детализирующая задает порядок применения приращений сигнала, причем ее значение +1 означает сложение, а –1 – вычитание.

Пусть имеется сигнал, представленный целочисленными компонентами вектора [9 7 3 5]. Это могут быть, например, значения пикселей некоторой подстроки изображения. Разрешение в этом случае равно 4 (таково число элементов вектора), и поначалу обрабатывается весь вектор.

Перейдем к более грубому (вдвое меньшему) разрешению 2, для чего поделим компактный носитель вейвлета вдвое и используем наш вейвлет дважды. Это значит, что мы должны вычислить среднее из каждой пары компонентов разделенного вдвое вектора сигнала. Получим вектор [8 4] с двумя детализирующими коэффициентами [1 –1]. Они представляют половинки от приращений уровня относительно среднего значения, то есть (9 – 7)/2 = 1 и (3 – 5)/2 = –1.

Прибавив и отняв +1 от первого компонента вектора огрубленного сигнала – числа 8, получим компоненты 9 и 7. Аналогично, прибавив и отняв –1 от второго компонента вектора огрубленного сигнала 4, получим 3 и 5, то есть вторую пару компонентов исходного вектора.

Продолжим огрублять сигнал вдвое и перейдем к разрешению 1. Наш вектор превратится в [6] с детализирующим коэффициентом 16. Его прибавление и вычитание дадут вектор [8 4].

Итак, для декомпозиции (разложения) исходного сигнала имеем:

Разрешение	Аппроксимирующие коэффициенты	Детализирующие коэффициенты
4	[9 7 3 5]	
2	[8 4]	[1 -1]
1	[6]	[2]

Таким образом, для представления сигнала достаточно хранить его грубое значение 6 и детализирующие коэффициенты 2, 1 и –1. Операции с ними задаются видом вейвлета Хаара. Например, на уровне разрешения 1 он представляется двумя функциями – аппроксимирующей с уровнем 1 и детализирующей с уровнем +1 на первой половине периода и –1 на второй половине периода (именно это задает вначале сложение, а затем вычитание детализирующего коэффициента). В итоге, осуществляя композицию сигнала, мы точно восстанавливаем его значение, используя последний (самый грубый) аппроксимирующий коэффициент и ряд детализирующих коэффициентов.

Процедуры изменения разрешения вдвое в ходе композиции и декомпозиции реализуют так называемый *диадический* метод. Он является разновидностью более общего *кратномасштабного* метода и лежит в основе устранения избыточности, свойственной непрерывным вейвлет-преобразованиям (см. ниже).

Коэффициенты вейвлет-представления для большинства сигналов – часто существенно меньшие числа, чем представления отсчетов сигналов. Для реальных сигналов многие коэффициенты по уровню оказываются настолько малыми, что

их можно отбросить. Это означает возможность значительного сокращения объема информации о сигнале, выполнения его компрессии и очистки от шумов. Добавьте к этому, что сейчас есть множество куда более ценных и интересных вейвлетов, чем вейвлет Хаара, что дает обширный выбор базисных функций, как для точного, так и для приближенного представления любых сигналов.

Теоретически доказано, что точное представление могут давать только так называемые *ортогональные вейвлеты*. Об определении *ортогональности* и о многих иных свойствах вейвлетов можно прочесть в [162–168].

16.2. Прямое вейвлет-преобразование

На основании понятия о векторном пространстве общепринятым в теории сигналов подходом к анализу зависимостей $s(t)$ стало их представление в виде взвешенной суммы простых составляющих – базисных функций $\Psi_k(t)$, помноженных на коэффициенты C_k:

$$s(t) = \sum_k C_k \psi_k(t). \tag{16.1}$$

Так как базисные функции $\Psi_k(t)$ предполагаются заданными как функции вполне определенного вида, то только коэффициенты C_k содержат информацию о конкретном сигнале. Таким образом, можно говорить о возможности представления произвольных сигналов на основе рядов (16.1) с различными *базисными функциями*, или просто *базисами*. К примеру, ряд Фурье использует в качестве базисных функций синусоиды и косинусоиды.

Термин *вейвлет*, введенный впервые специалистом по сейсмографии Морле (J. Morlet), в переводе с английского wavelet означает «короткая, или маленькая, волна». Такие волны распространяются в объеме Земли при различных ударных воздействиях, например взрывах или землетрясениях. У нас термин wavelet изначально переводили как «всплеск», «выброс» и т. д. [165, 166], что менее удачно, поскольку большинство вейвлетов имеет временные зависимости с ярко выраженной колебательной компонентой (как и волны).

Вейвлеты характеризуются своим временным и частотным образами. Временной образ определяется некоторой *psi-функцией* времени $\psi(t)$. А частотный образ определяется ее *Фурье-образом* $\hat{\psi}(\omega) = F(\omega)$, который задает огибающую спектра вейвлета. Фурье-образ определяется выражением

$$F(\omega) = \int_{-\infty}^{\infty} \hat{\psi}(t) e^{-i\omega t} dt. \tag{16.2}$$

Прямое непрерывное вейвлет-преобразование (ПВП) означает разложение произвольного входного сигнала на принципиально новый базис в виде совокупности волновых пакетов – вейвлетов, – которые характеризуются четырьмя принципиально важными свойствами:

- имеют вид коротких, локализованных во времени (или в пространстве) волновых пакетов с нулевым значением интеграла;
- обладают возможностью сдвига по времени;

- способны к масштабированию (сжатию/растяжению);
- имеют ограниченный (или локальный) частотный спектр.

Этот базис может быть ортогональным, что заметно облегчает анализ, дает возможность реконструкции сигналов и позволяет реализовать алгоритмы быстрых вейвлет-преобразований. Однако есть ряд вейвлетов, которые свойствами ортогональности не обладают, но которые тем не менее практически полезны – например, в задачах анализа и идентификации локальных особенностей сигналов и функций.

Одна из основополагающих идей вейвлет-представления сигналов заключается в разбивке приближения к сигналу на две составляющие – грубую (аппроксимирующую) и утонченную (детализирующую) – с последующим их уточнением итерационными методами. Каждый шаг такого уточнения соответствует определенному уровню декомпозиции и реставрации сигнала. Это возможно как во временной, так и в частотной областях представления сигналов вейвлетами.

В основе непрерывного вейвлет-преобразования НВП (или CWT – Continue Wavelet Transform) лежит использование двух непрерывных и интегрируемых по всей оси t (или x) функций:

- вейвлет-функция psi $\psi(t)$ с нулевым значением интеграла ($\int_{-\infty}^{\infty} \psi(t)dt = 0$), определяющая детали сигнала и порождающая детализирующие коэффициенты;
- маштабирующая, или скейлинг-функция, phi $\varphi(t)$ с единичным значением интеграла ($\int_{-\infty}^{\infty} \varphi(t)dt = 1$), определяющая грубое приближение (аппроксимацию) сигнала и порождающая коэффициенты аппроксимации.

Аппроксимирующие phi-функции $\varphi(t)$ присущи далеко не всем вейвлетам, а только тем, которые относятся к ортогональным. Такие вейвлеты мы рассмотрим в дальнейшем, а пока остановимся только на свойствах детализирующей psi-функции $\psi(t)$ и на приближении ими локальных участков сигналов $s(t)$.

Psi-функция $\psi(t)$ создается на основе той или иной *базисной функции* $\psi_0(t)$, которая, как и $\psi(t)$, определяет тип вейвлета. Базисная функция должна удовлетворять всем тем требованиям, которые были отмечены для psi-функции $\psi(t)$. Она должна обеспечивать выполнение двух основных операций:

- смещение по оси времени t – $\psi_0(t-b)$ при $b \in \boldsymbol{R}$;
- масштабирование – $a^{-1/2}\psi_0\left(\frac{t}{a}\right)$ при $a > 0$ и $a \in \boldsymbol{R}^+ - \{0\}$.

Параметр a задает ширину этого пакета, а b – его положение. В ряде литературных источников вместо явного указания времени t используется аргумент x, а вместо параметров a и b используются имеющие тот же смысл иные обозначения. Нетрудно убедиться в том, что следующее выражение задает сразу два этих свойства функции $\psi(t)$:

$$\psi(t) \equiv \psi(a,b,t) = a^{-1/2}\psi_0\left(\frac{t-b}{a}\right). \qquad (16.3)$$

Итак, для заданных a и b функция $\psi(t)$ и есть *вейвлет*. Вейвлеты, обозначаемые как $\psi(t)$, иногда называют «материнскими вейвлетами», поскольку они порождают целый ряд вейвлетов определенного рода.

Вейвлеты являются вещественными функциями времени t и колеблются вокруг оси t (или x). Параметр b в (16.3) задает положение вейвлетов, а параметр a – их масштаб. Коэффициент $a^{-1/2}$ обеспечивает нормирование энергии при изменении масштаба. О вейвлетах, локализованных в пространстве (или во времени), говорят, что они имеют *компактный носитель*.

Применительно к сигналам как функциям времени параметр $b \in \mathbf{R}$ задает положение вейвлета на временной оси, а параметр a – его масштабирование по времени. Поскольку параметр масштаба a реально может быть только положительным и его нельзя брать равным нулю, то считается, что $a \in \mathbf{R}^+ - \{0\}$. В дальнейшем мы будем опускать выражение $-\{0\}$, означающее исключение значения $a = 0$.

На рис. 16.2 показано построение вейвлета, известного под названием «мексиканская шляпа». Здесь «шляпа» представлена перевернутой сменой знака у исходной временной функции. Для вычисления и построения графиков этого вейвлета использована популярная СКМ Mathcad.

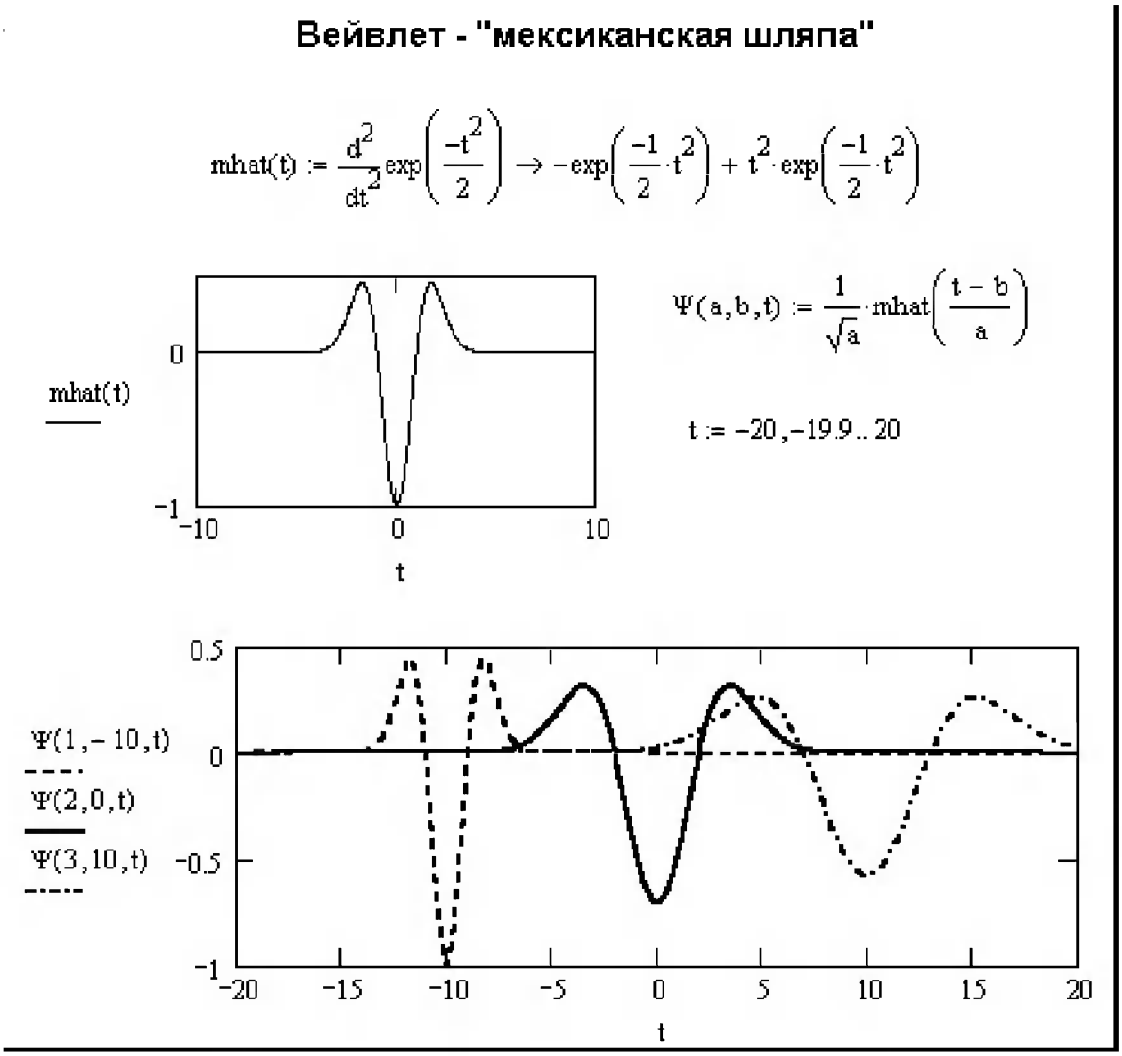


Рис. 16.2. Иллюстрация к сдвигу и масштабированию вейвлета типа «мексиканская шляпа»

На рис. 16.2 представлены базисная функция данного вейвлета и функция $\Psi(a,b,t)$ для разных a и b, что иллюстрирует сдвиг вейвлета и его масштабирование. В Mathcad для обеспечения изменений a и b функцию $\Psi(a,b,t)$ приходится задавать в более полном виде, чем $\Psi(t)$, в котором зависимость Ψ от a и b лишь подразумевается – см. (16.3).

В частотной области малые значения a соответствуют высоким частотам, а большие – низким частотам. Таким образом, операция задания окна, используемая в оконном преобразовании Фурье, как бы заложена в самой базисной функции вейвлетов. Это создает предпосылки их приспособления (адаптации) к сигналам, которые могут быть представлены совокупностью вейвлетов.

Для сигналов с конечной энергией *прямое непрерывное вейвлет-преобразование (ПНВП)* сигнала $s(t)$ задается, по формальной аналогии с преобразованием Фурье, путем вычисления *вейвлет-коэффициентов* по формуле

$$C(a,b)=\langle s(t),\psi(a,b,t)\rangle=\int_{-\infty}^{\infty} s(t)a^{-1/2}\psi\left(\frac{t-b}{a}\right)dt, \qquad (16.4a)$$

где обозначение $\langle ..., ... \rangle$ означает скалярное произведение соответствующих сомножителей. С учетом ограниченной области определения сигналов и $a,b \in \mathbf{R}$, $a \neq 0$:

$$C(a,b)=\int_{R} s(t)a^{-1/2}\psi\left(\frac{t-b}{a}\right)dt. \qquad (16.4b)$$

Итак, вейвлет-коэффициенты определяются интегральным значением скалярного произведения сигнала на вейвлет-функцию заданного вида. Выражение (16.4b) используется как основное для функции прямого непрерывного вейвлет-преобразования в пакете Wavelet Toolbox матричной системы MATLAB. Это один из самых известных и мощных пакетов по вейвлетам.

> ***Внимание!*** *При вейвлет-преобразовании выбор типов вейвлетов намного более обширен, чем при преобразовании Фурье. В качестве вейвлет-функций могут использоваться ортогональные и биортогональные непериодические функции, функции, имеющие глобальный экстремум и быстрое затухание на бесконечности и т. д. Основные требования к этим функциям обсуждались. Все это дает обширные возможности для представления различных сигналов.*

16.3. Вейвлет-спектрограммы

Зависимость уровня вейвлет-коэффициентов отображают вейвлет-спектрограммы. Они являются важнейшим продуктом вейвлет-анализа сигналов и прекрасным дополнением к обычным спектрограммам на основе оконного преобразования Фурье. Технику построения вейвлет-спектрограмм иллюстрирует документ системы Mathcad, показанный на рис. 16.3.

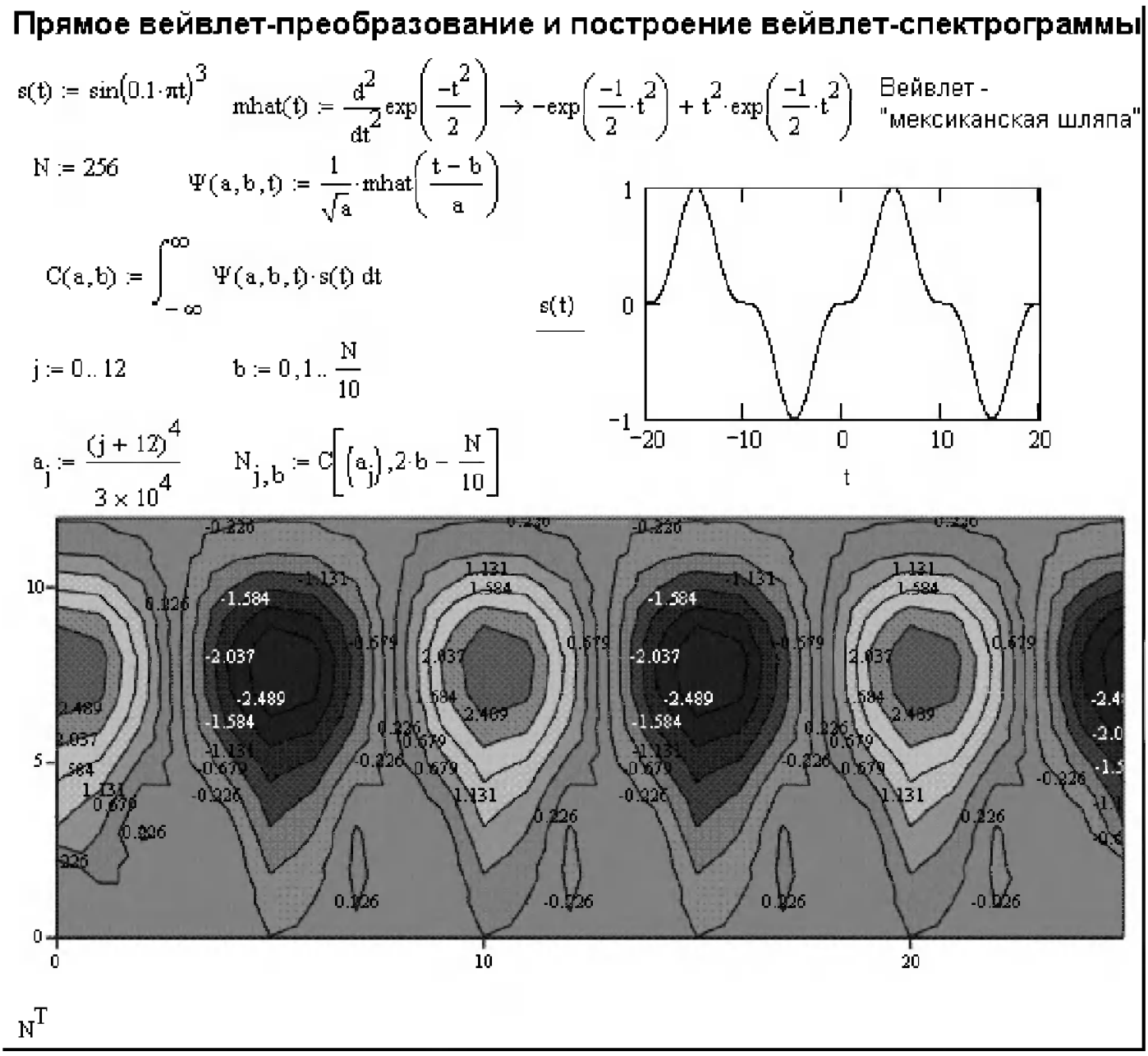


Рис. 16.3. Пример прямого вейвлет-преобразования сигнала и построения его вейвлет-спектрограммы в системе Mathcad

Вейвлет-коэффициенты здесь вычисляются по формуле (16.4b), что неэффективно и требует больших затрат времени. В пакете Signal Processing системы MATLAB есть более мощные средства построения вейвлет-спектрограмм – рис. 16.4. Вейвлет-спектрограммы порой выделяют такие особенности сигналов, которые вообще незаметны на графиках сигналов и на Фурье-спектрограммах. Например, вейвлет-спектрограмма выделяет разрывы синусоидальной функции с уровнем в тысячную долю амплитуды (на графике функции и на обычной спектрограмме такие разрывы вообще не видны).

Для знакомства с техникой прямого вейвлет-преобразования удобна хорошо известная у нас СКМ Mathcad, общение пользователя с которой происходит с помощью математически ориентированного языка общения и визуального програм-

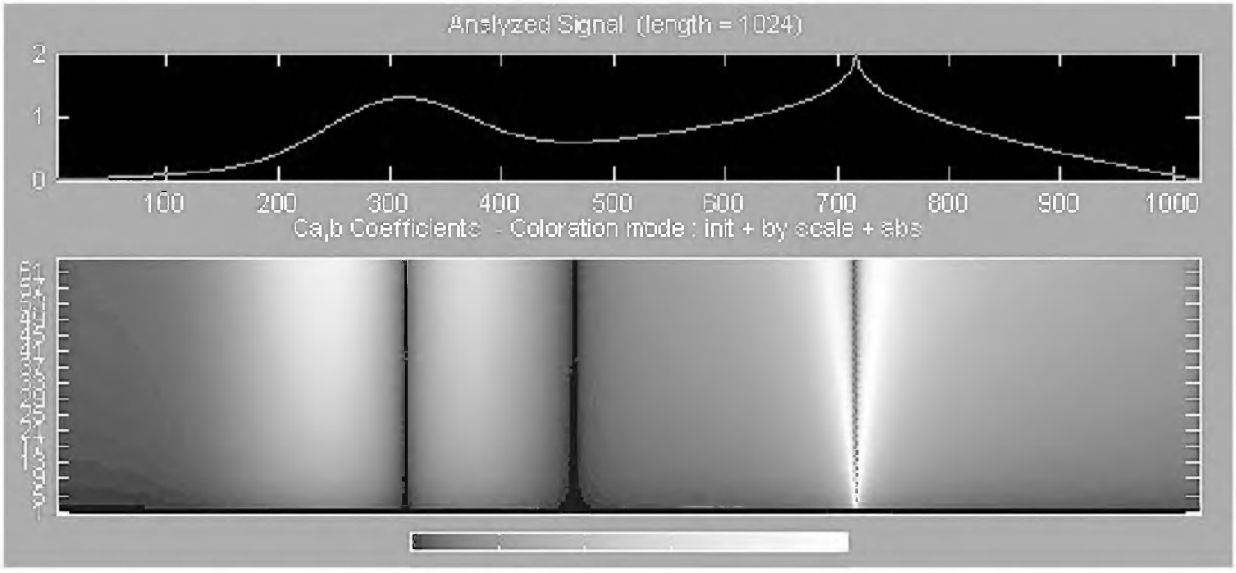


Рис. 16.4. Сигнал с особенностями и его вейвлет-спектрограмма, полученная в СКМ MATLAB

мирования. Это единственная из СКМ, у которой функции вейвлет-преобразования включены в ядро системы.

> ***Внимание!*** *Вейвлет-анализ сигналов открывает принципиально новые возможности в детальном анализе тонких особенностей сигналов. Это особенно важно для звуковых сигналов и сигналов изображения, где именно такие особенности подчас определяют качество их воспроизведения. Биология, картография, медицина, акустика, астрономия и космос – все это именно те области, где применение вейвлетов способно привести к новым открытиям путем выявления характерных особенностей сигналов и изображений, малозаметных на временных зависимостях сигналов и на их спектрах Фурье. Однако пользоваться формально построенными спектрограммами, без тщательного изучения причин возникновения тех или иных их особенностей, недопустимо, ибо может привести к «лжеоткрытиям».*

16.4. Обратное непрерывное вейвлет-преобразование

Обратное непрерывное вейвлет-преобразование (ОНВП) осуществляется по формуле реконструкции во временной области, которая имеет ряд форм, зависящих как от математической стилистики записи, так и (что более важно) от определения областей существования сигнала. В работе [163] формула реконструкции для произвольной функции f представлена в виде:

$$f = C_{\psi}^{-1} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \frac{dadb}{a^2} \langle f, \psi^{a,b} \rangle \psi^{a,b}, \tag{16.5}$$

где

$$\psi^{a,b}(x) = |a|^{-1/2} \psi\left(\frac{x-b}{a}\right). \tag{16.6}$$

Обратите внимание на несколько непривычную запись интеграла в выражении для f, которая часто используется в работе [163] и в других работах западных авторов. Непривычно записаны и параметры масштаба и сдвига (их индексы не надо отождествлять со степенями). Здесь параметры сдвига и сжатия меняются непрерывно вдоль R с ограничением $a \neq 0$. Постоянная C_{ψ} в (16.5) зависит только от ψ и определяется как

$$C_{\psi} = 2\pi \int_{-\infty}^{\infty} d\xi |\hat{\psi}(\xi)|^2 |\xi|^{-1} = 2\pi \int_{-\infty}^{\infty} \frac{|\hat{\psi}(\omega)|^2}{|\omega|^{-1}} d\omega. \tag{16.7}$$

Если ψ является вещественной функцией, то оказывается справедливым следующее, более строгое чем (16.7), выражение:

$$C_{\psi} = 2\pi \int_{0}^{\infty} d\xi |\hat{\psi}(\xi)|^2 |\xi|^{-1} = 2\pi \int_{-\infty}^{0} d\xi |\hat{\psi}(\xi)|^2 |\xi|^{-1} < \infty.$$

В этом случае:

$$f = C_{\psi}^{-1} \int_{0}^{\infty} \frac{da}{a^2} \int_{-\infty}^{\infty} db \langle f, \psi^{a,b} \rangle \psi^{a,b}.$$

Читатель, заинтересовавшийся подобными математическими выкладками, может обратиться к работам [163–166], где приведен еще целый ряд формул реконструкции.

В практическом аспекте интерес представляют лишь те формулы реконструкции, которые используются в конкретных программных инструментальных средствах, реализующих прямое и обратное вейвлет-преобразования. Так, ниже представлена формула реконструкции сигнала $s(t)$ в том виде, который использован в пакете расширения системы MATLAB – Wavelet Toolbox:

$$s(t) = \frac{1}{K_{\psi}} \int_{R^+} \int_{R} C(a,b) a^{-1/2} \psi\left(\frac{t-b}{a}\right) \frac{dadb}{a^2}, \tag{16.8}$$

где K_{ψ} – константа, определяемая функцией ψ. Нетрудно понять, что здесь $K_{\psi} = C_{\psi}$, и несколько иначе обозначены пределы интегрирования (с учетом области **R** и исключения отрицательных и нулевых значений параметра масштаба a, которые для целей практики не интересны).

Основной задачей теории вейвлет-преобразований является доказательство того, что прямое и обратное вейвлет-преобразования способны обеспечить *рекон-*

струкцию сигнала, причем точную или хотя бы приближенную, локальную или для сигнала в целом на заданном промежутке времени. Учитывая нулевое значение интеграла для функции $\psi(t)$ и то, что не все вейвлеты являются ортогональными, можно допустить, что эти преобразования не всегда способны восстановить любой сигнал в целом. Именно это и строго доказано в работах [163–166].

Мы уже отмечали, что вейвлет-анализ не использует амплитудно-частотную область для визуального представления спектров сигналов, как это имеет место при спектральном анализе Фурье. Вместо нее используется область времени (точнее, сдвига) – масштаб (см. выше).

Теперь мы можем наглядно отобразить различные виды представлений сигналов в ходе тех или иных их преобразований – рис. 16.5. Здесь показаны виды представления сигналов не только с помощью вейвлет-преобразования, но и с помощью других видов преобразования сигналов, описанных в предшествующей главе.

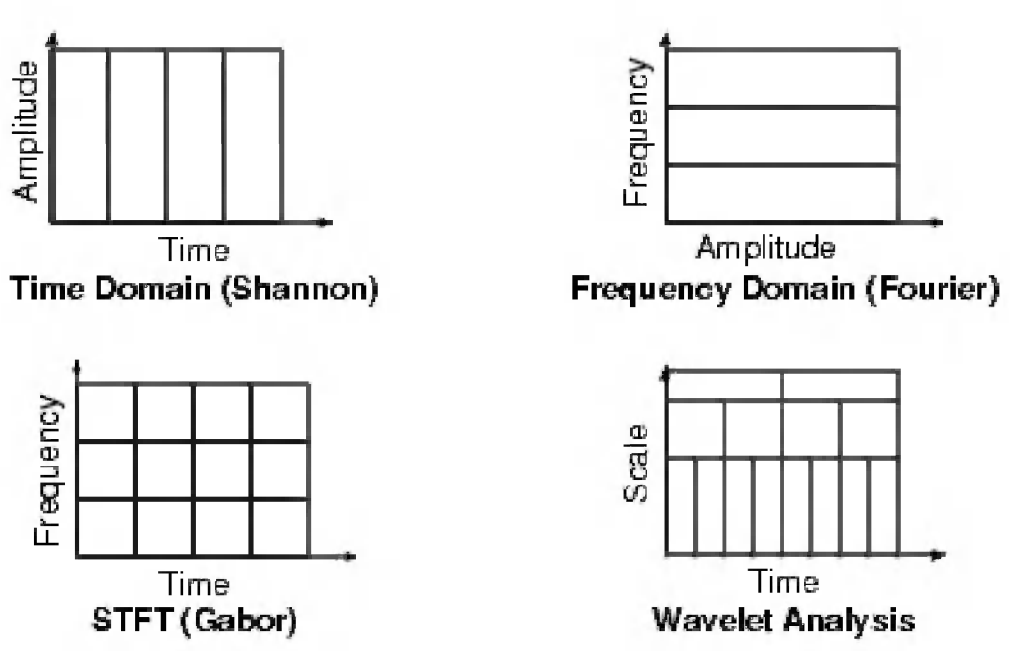


Рис. 16.5. Различные представления сигналов

На рис. 16.5 представлены следующие наиболее известные формы представления сигналов:

- Time Domain – временное представление (по Шеннону);
- Frequency Domain – частотное представление (по Фурье);
- STFT – кратковременное (оконное) быстрое преобразование Фурье;
- Wavelet – вейвлет-преобразование.

Нетрудно заметить, что вейвлет-преобразование отличается наиболее сложной и гибкой структурой представления сигналов в плоскости «Масштаб–время» (Scale-Time). Это открывает возможности более полного и тонкого вейвлет-анализа сигналов, по сравнению с другими известными видами их анализа. При этом особенности сигналов привязаны к временной шкале.

16.5. Примеры вейвлет-преобразований в СКМ Mathcad

В ядро систем Mathcad (начиная с версии Mathcad 8) включены следующие две функции для вейвлет-преобразований:

- `wave(V)` – дискретное (диадное) вейвлет-преобразование действительных чисел (вектор `V` должен содержать `2n` действительных значений, где `n` – целое число);
- `iwave(V)` – обратное вейвлет-преобразование относительно преобразования `wave` (`V` – вектор, с числом элементов `2n`).

В Mathcad используются ортогональные вейвлеты Добеши четвертого порядка [163, 164].

Рассмотрим пример применения этих функций, показанный на рис. 16.5. Здесь для получения вейвлет-коэффициентов выполнено прямое вейвлет-преобразование импульса с крутыми нарастанием и спадом и линейно растущей вершиной. Такой сигнал можно рассматривать как суперпозицию прямоугольного и пилообразного сигналов. На рис. 16.6 вновь построены пять вейвлет-коэффициентов, представляющих декомпозицию сигнала.

Тут любопытно отметить, что, несмотря на отсутствие линейных трендов у самих вейвлет-коэффициентов, линейно-нарастающая часть импульса приближается превосходно, без малейших намеков на ступенчатость – рис. 16.7. Здесь показано восстановление сигнала для трех уровней $L = 3$, 6 и 8.

Поскольку вейвлеты Добеши относятся к ортогональным, можно ожидать возможности точного восстановления произвольного сигнала. Разумеется, мы не можем воспроизвести весь бесконечный набор произвольных сигналов. Но приведем наглядный пример – вейвлет-преобразование для 512 точек сложного нестационарного сигнала, представляющего собой смесь импульса с наклонной спадающей верхушкой, меандра с переменной частотой, синусоиды с нарастающей частотой и шумовой компоненты, созданной генератором случайных чисел (функция rnd в системе Mathcad). Этот пример дан на рис. 16.8.

На этом рисунке показан как график реконструированного сигнала, так и исходный сигнал. Нетрудно заметить, что, несмотря на сложный и даже случайный (недетерминированный) характер исходного сигнала, связанный с наличием шумовой компоненты и потому исключающий его точное аналитическое описание, графики исходного и восстановленного сигналов практически совпадают, и их

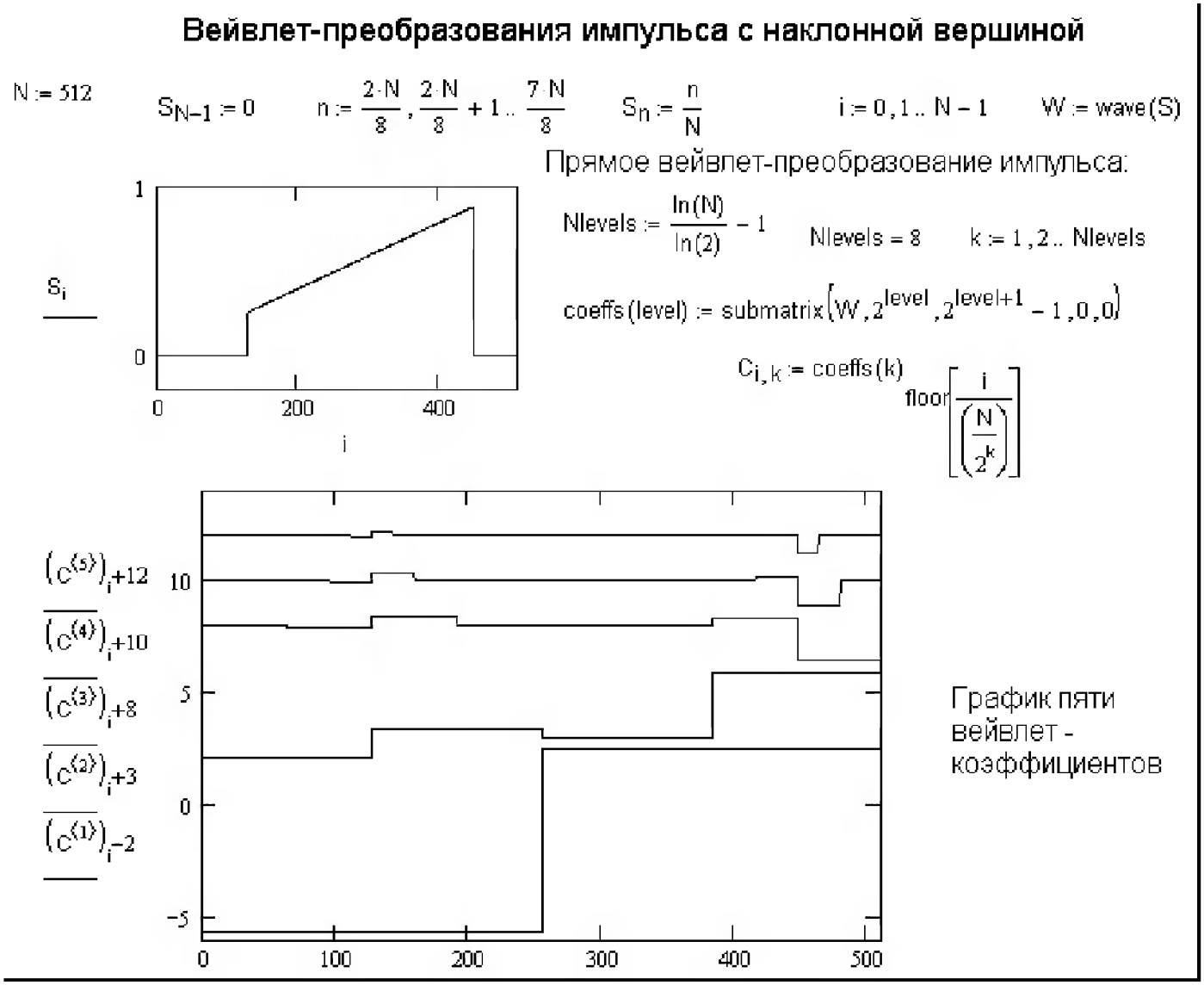


Рис. 16.6. Пример декомпозиции импульса с линейно-нарастающей вершиной (начало документа)

для разделения пришлось искусственно сдвинуть относительно друг друга. Совпадают и графики на небольшом отрезке времени (снизу рис. 16.7). Этому не стоит удивляться – вычисленная максимальная среднеквадратическая погрешность ничтожно мала – менее $6 \cdot 10^{-15}$.

В заключение надо отметить, что в данных функциях системы Mathcad точной реконструкции соответствует максимальный уровень реконструкции, а не нулевой (к примеру, как в большинстве других СКМ). Это обстоятельство непринципиально, но при практическом использовании описанных функций о нем просто надо помнить.

Восстановление сигнала с использованием
L первых вейвлет - коэффициентов

L := 4 W1 := W j := 2^L.. N – 1 $W1_j$:= 0 S4 := iwave(W1)

L := 6 W1 := W j := 2^L.. N – 1 $W1_j$:= 0 S6 := iwave(W1)

L := 8 W1 := W j := 2^L.. N – 1 $W1_j$:= 0 S8 := iwave(W1)

График исходного радиоимпульса и импульса,
синтезированного с помощью вейвлет - преобразования

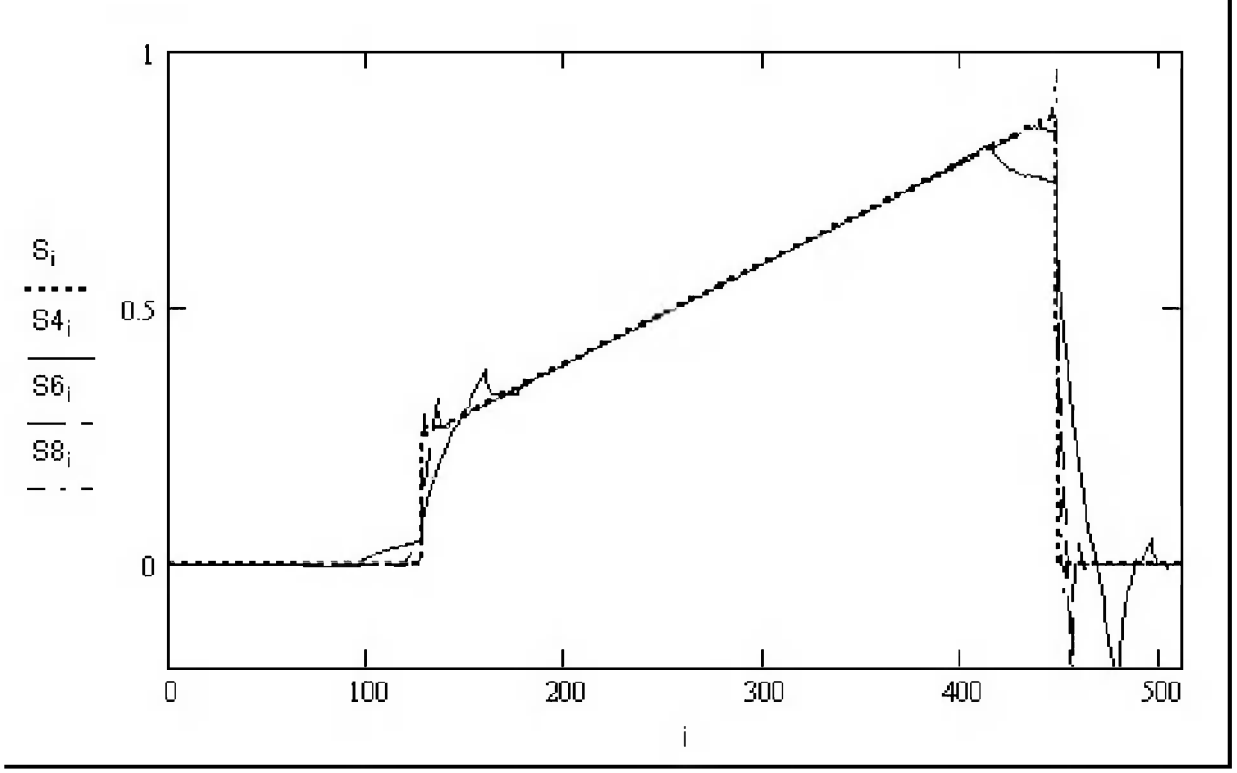


Рис. 16.7. Пример реставрации импульса с линейно-нарастающей вершиной (конец документа)

16.6. Средства СКМ для вейвлет-преобразований

Вейвлеты – новейшее направление в науке и технике. Естественно, что они поддерживаются соответствующими средствами современных СКМ и СКА. Но пока единственной СКМ, в которой функции прямого и обратного дискретных вейвлет-преобразований включены в ядро, является система Mathcad. Кроме того, система имеет довольно мощный пакет расширения по вейвлетам Wavelet Explorer. Этот пакет поставляется как отдельно, так и в составе некоторых расширенных версий систем Mathcad, например Mathcad 11 Enterprise Edition и Mathcad 12/13/14. Достаточно полное описание пакета Wavelet Explorer можно найти в [46, 168].

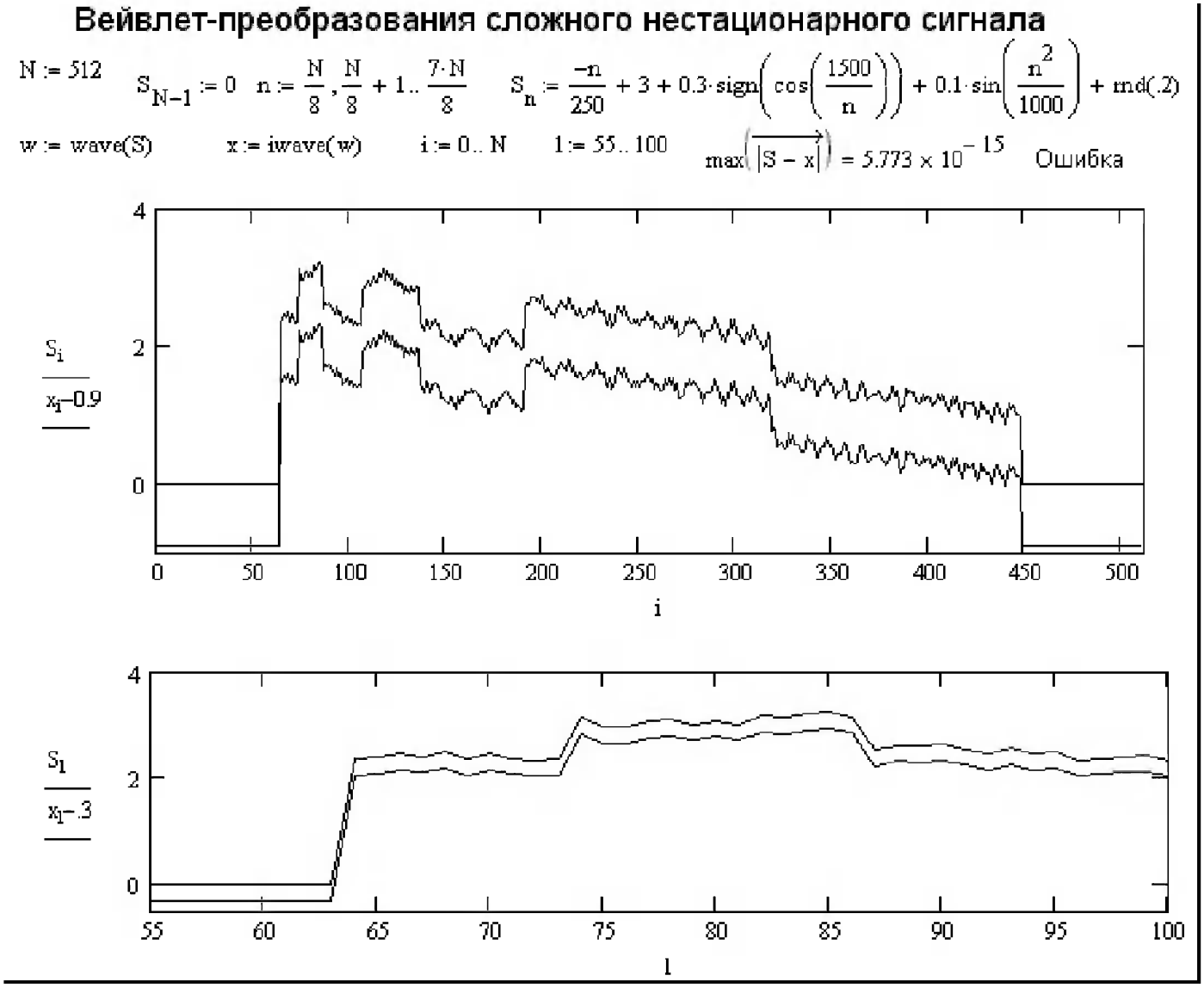


Рис. 16.8. Вейвлет-преобразования для сложного сигнала

СКМ Mathematica 4/5/6 также имеет пакет расширения по вейвлетам – Wavelet Explorer [81]. Но одним из самых мощных и распространенных пакетов расширения по вейвлетам является пакет Wavelet Toolbox [46], входящий в состав программного комплекса MATLAB+Simulink (последняя реализация MATLAB R2008a). Пакет содержит определения свыше двух десятков классов вейвлетов, около трехсот функций вейвлет-преобразований и обработки сигналов и изображений, средства визуализации вейвлет-спектров, средства вевлет-анализа и синтеза сигналов и множество примеров применения этой технологии, в том числе созданные на основе графического интерфейса пользователя GUI этой системы.

Большими возможностями в реализации вейвлет-технологии обладает программа WAVELAB 8.02, созданная в конце 1999 года в Стэнфордском университете (www-stat.stanford.edu/~wavelab\). По тем временам эта программа превосхо-

дила стандартный пакет Wavelet Toolbox системы MATLAB. Эта программа сделана как пакет расширения, включаемый в инструментальный набор Toolbox системы MATLAB 5.*. К сожалению, с новыми реализациями MATLAB данный пакет не работает из-за многочисленных неточностей в записи файлов.

Полное описание всех пакетов расширения по вейвлетам выходит за рамки объема, да и тематики данной книги – вейвлеты скорее относятся к численным методам обработки данных, чем к компьютерной алгебре. В связи с этим ограничимся приведенными выше сведениями.

Список литературы

1. Дьяконов В. П. Компьютерная математика. Теория и практика. – М.: Нолидж, 2001.
2. Дьяконов В., Новиков Ю. и Рычков В. Компьютер для студента: Самоучитель. – СПб.: Питер, 2000.
3. Глушков В. М., Бондарчук В. Г., Гривченко Т. А. Аналитик – алгоритмический язык для описания процессов с использованием аналитических преобразований // Кибернетика. – 1971. – № 3.
4. Биркгоф Г., Барти Т. Современная компьютерная алгебра. – М.: Мир, 1976.
5. Барри Саймон. Символьная математика: новые времена – новые формы // PC Magazine. – 1992. – № 5.
6. Акритас А. Основы компьютерной алгебры / Пер. с англ. Е. В. Панкратьева. – М.: Мир, 1994.
7. Дэвенпорт Дж., Сирэ И., Турнье Э. Компьютерная алгебра. Системы и алгоритмы алгебраических вычислений. – М.: Мир, 1991.
8. Кук Д., Бейз Г. Компьютерная математика. – М.: Наука, 1990.
9. Кнут Д. Искусство программирования: в 3 т. – М.: Вильямс, 2000.
10. Грехем Р., Кнут Д., Паташник О. Конкретная математика. Основания информатики. – М.: Мир, 1998.
11. Самсонов Б. Б., Плохов Е. М., Филоненков А. И. Компьютерная математика (основания информатики). – Ростов-на-Дону: Феникс, 2002.
12. Душин В. К. Теоретические основы информационных процессов и систем: Учебник. – М.: Издательско-торговая корпорация «Дашков и К°», 2003.
13. Дьяконов В. П. Расчет нелинейных и импульсных устройств на программируемых микрокалькуляторах. – М.: Радио и связь, 1984.
14. Дьяконов В. П. Справочник по расчетам на микрокалькуляторах. 3-е изд., доп. и перераб. – М.: Наука, Физматлит, 1989.
15. Дьяконов В. П. Современные зарубежные микрокалькуляторы. – М.: СОЛОН-Пресс, 2002.
16. Дьяконов В. П. Справочник по алгоритмам и программам на языке Бейсик для персональных ЭВМ. – М.: Наука, Физматлит, 1987.
17. Дьяконов В. П. Применение персональных ЭВМ и программирование на языке Бейсик. – М.: Радио и связь, 1989.
18. Dyakonov V. P., Yemelchenkov E. P., Munerman V. I., Samoilova T. A. The Revolutionary Guide to QBASIC. – UK.: Wrox Press, 1996.
19. Дьяконов В. П. Язык программирования ЛОГО. – М.: Радио и связь, 1991.
20. Дьяконов В. П. Форт-системы программирования персональных ЭВМ. – М.: Наука, Физматлит, 1992.
21. Дьяконов В. Как выбрать математическую систему? // Монитор-Аспект. – 1993. – № 2.
22. Дьяконов В., Пеньков А. Современные математические системы // PC WEEK. – 1996. – № 43 (67).
23. Дьяконов В. П. Общедоступные математические САПР для персональных компьютеров класса IBM PC // Программные продукты и системы. – 1993. – № 1.
24. Дьяконов В. П. Компьютерные математические системы в образовании // Информационные технологии. – 1997. – № 4.
25. Дьяконов В. П . Справочник по применению системы Eureka. – М.: Наука, Физматлит, 1993.

26. Дьяконов В. П. Mercury – отличная система для всех // Монитор-Аспект. – 1995. – № 5.
27. Дьяконов В. П. MathCAD прорывается в Windows! // Монитор-Аспект. – 1993. – № 2.
28. Дьяконов В. П. Расширяемые системы для численных расчетов MatLAB // Монитор-Аспект. – 1993. – № 2.
29. Дьяконов В. Derive – жемчужина символьной математики // Монитор-Аспект. – 1993. – № 2.
30. Дьяконов В., Бирюков С. Derive в России // Монитор. – 1995. – № 3.
31. Дьяконов В. П. Mathematica 2.0 под MS-DOS и под Windows // Монитор-Аспект. – 1993. – № 2.
32. Дьяконов В. П. Mathematica 2.1 для Windows: от слов к делу! – Монитор-Аспект. – 1995. – № 4.
33. Дьяконов В. П. Mathematica 2.2.2 – на пути к совершенству // Монитор-Аспект. – 1996. – № 6.
34. Дьяконов В. П. Maple V – мощь и интеллект компьютерной алгебры! // Монитор-Аспект. – 1993. – № 2.
35. Дьяконов В. П., Пеньков А. А. Моделирование транзисторных преобразователей напряжения с последовательным контуром // Электричество. – 1998. – № 12.
36. Дьяконов В. П., Абраменкова И. В. Техника визуализации учебных и научных задач с применением систем класса MathCAD // Информационные технологии. – 1998. – № 11.
37. Дьяконов В. П. Система MathCAD: Справочник. – М.: Радио и связь, 1993.
38. Дьяконов В. П. Справочник по MathCAD PLUS 6.0 PRO. – М.: СК-ПРЕСС, 1997.
39. Дьяконов В. П. Справочник по MathCAD 7.0 PRO. – М.: СК-ПРЕСС, 1998.
40. Дьяконов В. П., Абраменкова И. В. MathCAD 7.0 в математике, в физике и в Internet. – М.: Нолидж, 1998.
41. Дьяконов В. П., Абраменкова И. В. MathCAD 8.0 в математике, в физике и в Internet. – М.: Нолидж, 1999.
42. Дьяконов В. П. MathCAD 8/2000: Специальный справочник. – СПб.: Питер, 2000.
43. Дьяконов В. П. MathCAD 2000: Учебный курс. – СПб.: Питер, 2000.
44. Дьяконов В. П. MathCAD 2001: Специальный справочник. – СПб.: Питер, 2000.
45. Дьяконов В. П. MathCAD 2001i/11: Энциклопедия. – М.: СОЛОН-Пресс, 2004.
46. Дьяконов В. П. MathCAD 8–12 для студентов. – М.: СОЛОН-Пресс, 2005.
47. Дьяконов В. П. MathCAD 11/12/13 в математике: Справочник. – М.: Горячая линия. Телеком, 2007.
48. Очков В. Ф. MathCAD 7 Pro для студентов и инженеров. – М.: Компьютер Press, 1998/1999.
49. Очков В. Ф. Советы пользователям MathCAD. – М.: Издательство МЭИ, 2001.
50. Очков В. Ф. MathCAD 14 для студентов, инженеров и конструкторов. – СПб.: БХВ-Петербург, 2007.
51. MathCAD 6.0 PLUS. Финансовые, инженерные и научные расчеты в среде Windows 95 / Пер. с англ. – М.: Филинъ, 1996.
52. Плис А. И., Сливина Н. А. MathCAD: математический практикум для экономистов и инженеров. – М.: Финансы и статистика, 1999.
53. Херхагер М., Партолль Х. MathCAD 2000: Полное руководство. – К.: БХВ, 2001.
54. Сдвижков О. А. MathCAD 2000. Введение в компьютерную математику. – М.: Издательско-торговая корпорация «Дашков и К°», 2002.
55. Поршнев С. В. Компьютерное моделирование физических процессов с использованием пакета MathCAD. – М.: Горячая линия – Телеком, 2002.

56. Глушаков С. В., Жакин И. А., Хачиров Т. С. Математическое моделирование: Учебный курс. – Харьков.: Фолио; М.: АСТ, 2001.

57. Каганов В. И. Радиотехника + компьютер + MathCAD. – М.: Горячая линия – Телеком, 2001.

58. Черняк А. А., Новиков В. А., Мельников О. И., Кузнецов А. В. Математика для экономистов на базе MathCAD. – СПб.: БХВ-Петербург, 2003.

59. Макаров Е. Г. Инженерные расчеты в MathCAD: Учебный курс. – СПб.: Питер, 2003.

60. Черняк А. А., Черняк Ж. А., Доманова Ю. А. Высшая математика на базе MathCAD: Общий курс; Учебное пособие. – СПб.: БХВ-Петербург, 2004.

61. Кирьянов Д. В. Самоучитель MathCAD 12. – СПб.: БХВ-Петербург, 2004.

62. MathCAD 13. User's Guide. Mathsoft Engineering&Education Inc. – 2005.

63. Дьяконов В. П. Математическая система Maple V R3/R4/R5. – М.: Солон, 1998.

64. Дьяконов В. П. Maple 6: Учебный курс. – СПб.: Питер, 2001.

65. Дьяконов В. П. Maple 7: Учебный курс. – СПб.: Питер, 2002.

66. Дьяконов В. П. Maple 8 в математике, физике и образовании. – М.: СОЛОН-Пресс, 2003.

67. Дьяконов В. П. Maple 9 в математике, физике и образовании. – М.: СОЛОН-Пресс, 2004.

68. Дьяконов В. П. Maple 9.5/10 в математике, физике и образовании. – М.: СОЛОН-Пресс, 2006.

69. Говорухин В. Н., Цибулин В. Г. Введение в Maple V. Математический пакет для всех. – М.: Мир, 1997.

70. Васильев А. Н. Maple 8: Самоучитель. – М.: Изд. дом «Вильямс», 2003.

71. Прохоров Г. В., Леденев М. А., Колбеев В. В. Пакет символьных вычислений Maple V. – М.: Петит, 1997.

72. Манзон Б. М. Maple V Power Edition. – М.: Филинъ, 1998.

73. Матросов А. В. Maple 6. Решение задач высшей математики и механики. – СПб.: БХВ-Петербург, 2003.

74. Говорухин В., Цибулин В. Компьютер в математическом исследовании. – СПб.: Питер, 2001.

75. Голоскоков. Уравнения математической физики. Решение задач в системе Maple: Учебник для вузов. – СПб.: Питер, 2004.

76. Char B. W. Maple 7. Learning Guide. Waterloo Maple Inc. – 2001.

77. Monagan M. B., Geddes K. O., Heal K. M. and other. Maple 7. Programming Guide. Waterloo Maple Inc. – 2001 .

78. Дьяконов В. П. Системы символьной математики Mathematica 2 и Mathematica 3. – М.: СК ПРЕСС, 1998.

79. Дьяконов В. П. Mathematica 4 с пакетами расширения. – М.: Нолидж, 2000.

80. Дьяконов В. П. Mathematica 4: Учебный курс. – СПб.: Питер, 2001.

81. Дьяконов В. П. Mathematica 4.1/4.2/5 в в математических и научно-технических расчетах. – М.: СОЛОН-Пресс, 2004.

82. Половко А. М. Mathematica для студентов. – СПб.: БХВ-СанктПетербург, 2007.

83. Воробьев В. М. Введение в систему «Математика»: Учебное пособие. – М.: Финансы и статистика, 1998.

84. Воробьев Е. М. Введение в систему символьных, графических и численных вычислений «МАТЕМАТИКА-5». – М.: ДИАЛОГ-МИФИ, 2005.

85. Wolfram S. The Mathematica Book. 4-th ed. WOLFRAM MEDIA/Cambridge University Press. – 1999.

86. Rich A. D., Stoutemyer D. R. Inside the DERIVE Computer Algebra System. England: The International Derive Journal. – V. 1. – 1994. – № 1.

87. Berry J. S., Graham E., Watkins A. J. (University of Plymouth). Learning Mathematics through DERIVE. Ellis Horwood. – 1993.

88. Bohm Josef. Teaching Mathematics with DERIVE. – Great Britain: Chartwell-Bratt Ltd, 1992.

89. Brian H. Denton. Learning Linear Algebra Through DERIVE. Ellis Horwood. – 1994.

90. Дьяконов В. П. Справочник по применению системы Derive.: – М.: Наука, Физматлит, 1996.

91. Дьяконов В. П. Справочник по системе символьной математики Derive. – М.: СК-ПРЕСС, 1998.

92. Дьяконов В. П. Системы компьютерной алгебры DERIVE: Самоучитель и руководство пользователя. – М.: СОЛОН-Пресс, 2002.

93. Лобанова О. В. Практикум по решению задач в математической системе Derive. – М.: Финансы и статистика, 1999.

94. Руководство пользователя DERIVE. Математический помощник на ПК / Русский пер. МНИИТЦ «Скан». – Soft Warehouse, Inc. – 1993.

95. DERIVE. Математический помощник на ПК / Русский пер. МНИИТЦ «Скан». – Soft Warehouse, Inc. – 1993.

96. Хювейн Э., Сеппянен И. Мир Лиспа: В 2 т. – М.: Мир, 1990.

97. Oevel W., Postel F., Ruscher G., Wehmeier S. Das MuPAD-Tutorium. – Paderborn: SciFace Software GmbH&Co. KG, 1998.

98. Дьяконов В. П. Справочник по применению системы PC MatLAB. – М.: Наука, Физматлит, 1993.

99. Дьяконов В. П., Абраменкова И. В. MATLAB 5.0/5.3. Система символьной математики. – М.: Нолидж, 1999.

100. Дьяконов В. П., Абраменкова И. В., Круглов В. В. MATLAB 5 с пакетами расширений. – М.: Нолидж, 2001.

101. Дьяконов В. П. MATLAB: Учебный курс. – СПб.: ПИТЕР, 2001.

102. Дьяконов В. П. MATLAB 6: Учебный курс. – СПб.: ПИТЕР, 2001.

103. Дьяконов В. П. Simulink 4: Специальный справочник. – СПб.: ПИТЕР, 2002.

104. Дьяконов В. П., Круглов В. В. Математические пакеты расширения MATLAB: Специальный справочник. – СПб.: ПИТЕР, 2001.

105. Дьяконов В. П., Круглов В. В. MATLAB. Анализ, идентификация и моделирование систем: Специальный справочник. – СПб.: ПИТЕР, 2002.

106. Дьяконов В. П., Абраменкова И. В. MATLAB. Обработка сигналов и изображений: Специальный справочник. – СПб.: ПИТЕР, 2002.

107. Дьяконов В. П. MATLAB 6/6.1/6.5 + Simulink 4/5. Основы применения: Полное руководство пользователя. – М.: СОЛОН-Пресс, 2002.

108. Дьяконов В. П. MATLAB 6/6.1/6.5 + Simulink 4/5 в математике и моделировании: Полное руководство пользователя. – М.: СОЛОН-Пресс, 2003.

109. Дьяконов В. П. MATLAB 6/6.1/6.5 + Simulink 4/5. Обработка сигналов и изображений: Полное руководство пользователя. – М.: СОЛОН-Пресс, 2004.

110. Потемкин В. Г. Система MATLAB: Справочное пособие. – М.: Диалог-МИФИ, 1997.

111. Потемкин В. Г. MATLAB 5 для студентов. – М.: Диалог-МИФИ, 1998.

112. Потемкин В. Г. Система инженерных и научных расчетов MATLAB 5.x. Т. 1, 2. – М.: Диалог-МИФИ, 1999.

113. Потемкин В. Г. Вычисления в среде MATLAB. – М.: ДИАЛОГ-МИФИ, 2004.

114. Кетков Ю. Л., Кетков А. Ю., Шульц М. М. MATLAB 6.x: программирование численных методов. – СПб.: БХВ-Петербург, 2004.

115. Рудаков П. И., Сафонов В. И. Обработка сигналов и изображений. MATLAB 5.x. / Под общ. ред. В. Г. Потемкина. – М.: ДИАЛОГ-МИФИ, 2000.

116. Мартынов Н. Н., Иванов А. П. MATLAB 5.x. Вычисления, визуализация, программирование. – М.: КУДИЦ-ОБРАЗ, 2000.

117. Лазарев Ю. Ф. MatLAB 5.X (серия «Библиотека студента»). – Киев: Издательская группа BHV, 2000.

118. Чен К., Джиблин П., Ирвинг А. MATLAB в математических исследованиях. – М.: Мир, 2001.

119. Боровиков В. П., Боровиков И. П. STATISTICA. Статистический анализ и обработка данных в среде Windows. – М.: Филинъ, 1998.

120. Боровиков В. П. Популярное введение в программу STATISTICA. – М.: Компьютер Пресс, 1998.

121. Калаичев А. П. Методы и средства анализа данных в среде Windows STADIA 6.0. – М.: Информатика и компьютеры, 1998.

122. Математический энциклопедический словарь / Под ред. Ю. В. Прохорова. – М.: Советская энциклопедия, 1988.

123. Бронштеин И. Н., Семендяев К. А. Справочник по математике для инженеров и учащихся втузов. – М.: Наука, Физматлит, 1980.

124. Корн Г., Корн Т. Справочник по математике для научных работников и инженеров. – М.: Наука, 1973.

125. Справочник по специальным функциям с формулами, графиками и математическими таблицами / Под ред. М. Абрамовица и И. Стиган. – М.: Наука, Физматлит, 1979.

126. Ильин В. А., Позняк Э. Г. Основы математического анализа: В 2 ч. 6-е изд. – М.: Физматлит, 2001.

127. Гантмахер Ф. Теория матриц. – М.: Наука, Физматлит, 1988.

128. Ильин В. А., Позняк Э. Г. Линейная алгебра. – М.: Физматлит, 2001.

129. Фаддеев Д. К., Фаддеева В. Н. Вычислительные методы линейной алгебры. 3-е изд. – СПб.: Лань, 2002.

130. Бабенко К. И. Основы численного анализа. – М.: Наука, Физматлит, 1986.

131. Марчук Г. И. Методы вычислительной математики. – М.: Наука, Физматлит, 1989.

132. Бахвалов Н. С., Жидков Н. П., Кобельков Г. М. Численные методы. – М.: Наука, Физматлит, 1987.

133. Трауб Дж. Итерационные методы решения уравнений. – М.: Мир, 1985.

134. Дэннис Дж., Шнабель Р. Численные методы безусловной оптимизации и решения нелинейных уравнений / Пер. с англ., под ред. Ю. Г. Евтушенко. – М.: Мир, 1988.

135. Иванов В. В. Методы вычислений на ЭВМ: Справочное пособие. – Киев: Наукова думка, 1986.

136. Абиев Р. Ш. Вычислительная гидродинамика и теплообмен. Введение в метод конечных разностей: Учебное пособие. – СПб.: Изд-во НИИ химии СПбГУ, 2002.

137. Холоднов В. А., Дьяконов В. П., Иванова Е. Н., Кирьянова Л. С. Математическое моделирование и оптимизация химико-технологических процессов: Практическое руководство. – СПб.: АНО НПО «Профессионал», 2003.

138. Банди Б. Основы линейного программирования. – М.: Радио и связь, 1989.
139. Тюрин Ю. Н., Макаров А. А. Статистический анализ данных на компьютере / Под ред. В. Э. Фигурнова. – М.: ИНФРА-М, 1998.
140. Львовский Е. Н. Статистические методы построения эмпирических формул. – М.: Высшая школа, 1988.
141. Дюк В. Обработка данных на ПК в примерах. – СПб.: Питер, 1997.
142. Королюк В. С., Портенко Н. И., Скороход А. В., Турбин А. Ф. Справочник по теории вероятности и математической статистике. – М.: Наука, Физматлит, 1985.
143. Воднев В. Т., Наумович А. Ф., Наумович Н. Ф. Основные математические формулы. – Минск: Высшая школа, 1988.
144. Толстов Г. П. Ряды Фурье. – М.: Наука, Физматлит, 1980.
145. Жуков А. И. Метод Фурье в вычислительной математике. – М.: Физматлит, 1992.
146. Ганеев Р. М. Математические модели в задачах обработки сигналов: Справочное пособие. – М.: Горячая линия – Телеком, 2002.
147. Носач В. В. Решение задач аппроксимации с помощью персональных компьютеров. – М.: МИКАП, 1994.
148. Марпл-мл. С. Л. Цифровой спектральный анализ и его приложения. – М.: Мир, 1990.
149. Коршунов Ю. М., Бобиков А. И. Цифровые сглаживающие и преобразующие системы. – М.: Энергия, 1969.
150. Ивахненко А. Г., Лапа В. Г. Предсказание случайных процессов. – Киев: Наукова думка, 1971.
151. Гмошинский В. Г., Флиорент Г. И. Теоретические основы инженерного прогнозирования. – М.: Наука, 1973.
152. Абраменкова И. В., Круглов В. В., Дли М. И. Мультимодельный метод прогнозирования процессов с переменной структурой. – М.: Физматлит, 2003.
153. Дьяконов В. П. Intel. Новейшие информационные технологии. Достижения и люди. – М.: СОЛОН-Пресс, 2004.
154. Бокс Дж., Дженкинс Г. Анализ временных рядов. – М.: Мир, 1974.
155. Роджерс Д., Адамс Дж. Математические основы машинной графики. – М.: Мир, 2001.
156. Топчеев Ю. И. Атлас для проектирования систем автоматического регулирования. – М.: Машиностроение, 1989.
157. Брейсуэлл Б. Преобразование Хартли. – М.: Мир, 1990.
158. Ланцош К. Практические методы прикладного анализа. – М.: Физматлит, 1961.
159. Блейхут Р. Быстрые алгоритмы цифровой обработки сигналов. – М.: Мир, 1989.
160. Прокис Дж. Цифровая связь. – М.: Радио и связь, 2000.
161. Зернов Н. В., Карпов В. Г. Теория радиотехнических цепей. –Л.: Энергия, 1972.
162. Баскаков С. И. Радиотехнические цепи и сигналы: Учебник для вузов по специальности «Радиотехника». –М.: Высшая школа, 2000.
163. Добеши И. Десять лекций по вейвлетам / Пер. с англ. Е. В. Мищенко; под ред. А. П. Петухова. – М.: РХД, 2001.
164. Чуи К. Введение в вэйвлеты / Пер. с англ., под ред. Я. М. Жилейкина. – М.: Мир, 2001.
165. Воробьев В. И., Грибунин В. Г. Теория и практика вейвлет-преобразований. – СПб.: ВУС, 1999.

166. Новиков И. Я., Стечкин С. Б. Основные конструкции всплесков. Фундаментальная и прикладная математика. Т. 3, вып. 4. – М., 1997.

167. Дьяконов В. П., Абраменкова И. В. MATLAB. Обработка сигналов и изображений. – СПб.: Питер, 2002.

168. Дьяконов В. П. Вейвлеты. От теории к практике. 2-е. изд. – М.: СОЛОН-Пресс, 2004.

169. Дьяконов В. П., Абраменкова И. В., Пеньков А. А. Новые информационные технологии: Учебное пособие для вузов. – М.: СОЛОН-Пресс, 2004.

170. Дьяконов В. П. Mathematica 5.1/5.2/6.0. Программирование и математические вычисления. – М.: ДМК-Пресс, 2008.

171. Сергиенко А. Б. Цифровая обработка сигналов. – СПб.: Питер, 2002.

172. Цисарь И. Ф., Нейман В. Г. Компьютерное моделирование экономики. – М.: Диалог-МИФИ, 2002.

173. Сдвижков О.А. MathCAD-2000: Введение в компьютерную математику. – М.: Издательско-торговая корпорация «Дашков и К°», 2002.

174. Математические методы и модели в экономике, финансах, бизнесе: Учеб. пособие для вузов. – М.: ЮНИТИ-ДАНА, 2001.

175. Экономико-математические методы и прикладные модели: Учебное пособие для вузов / В. В. Федосеев, А. Н. Гармаш, Д. М. Дайтибегов и др.; под ред. В. В. Федосеева. – М.: ЮНИТИ, 2001.

176. Шредер Ш. Фракталы, хаос, степенные законы. Миниатюры из бесконечного рая / Пер. с англ., под ред. А. В. Борисова. – М.: R&C Dynamics, 2001.

177. Spiegel, Murray R. Mathematical Handbook of Formulas and Tables. – New York: McGraw Hill Book Company, 1968.

178. Столниц Э., ДеРоуз Т., Салезин Д. Вейвлеты в компьютерной графике. Теория и приложения / Пер. с англ., под ред. Е. В. Мищенко. – М.: P&C Dynamic, 2002.

179. Системы компьютерной математики и их приложения: Материалы международной конференции. Вып. 9. – Смоленск: СмолГУ, 2008.

180. Richard J. Gaylord, Kazume Nishidate. Modeling Nature: Cellular Automata Simulations with Mathematica. TELOS/Springer-Verlag. – 1996.

181. Marvin L. DeJong. Mathematica for Calculus-Based Physics. Addison-Wesley. – 1999.

182. Michael Trott. The Mathematica GuideBook for Programming. Springer-Verlag. – 2004.

183. Michael Trott. The Mathematica GuideBook for Graphics. Springer-Verlag. – 2004.

184. Michael Trott. The Mathematica GuideBook for Symbolics. Springer-Verlag. – 2006.

185. Michael Trott. The Mathematica GuideBook for Numerics. Springer-Verlag. – 2006.

186. Абиев Р. Ш. Вычислительная гидродинамика и теплообмен. Введение в метод конечных разностей: Учебное пособие. – СПб.: Изд-во НИИ химии СПбГУ, 2002.

187. Кристалинский Р. Е. Приближенные аналитические методы решения задач механики деформируемого тела. – Смоленск: СмолГУ, 2007.

188. Кристалинский Р. Е., Кристалинский В. Р. Преобразования Фурье и Лапласа в системах компьютерной математики. – М.: Горячая линия – Телеком, 2006.

189. Половко А. М., Бутусов П. Н. Интерполяция. Методы и компьютерные технологии их реализации. – СПб.: БХВ-Петербург, 2004.

190. Дьяконов В. П. Современная осциллография и осциллографы. – М.: СОЛОН-Пресс, 2005.

191. Афонский А. А., Дьяконов В. П. Измерительные приборы и массовые электронные измерения / Под ред. В. П. Дьяконова. – М.: СОЛОН-Пресс, 2007.

192. Дьяконов В. П. Совместная работа генераторов произвольных функций Tektronix AFG3000 с осциллографами TDS1000B/2000B // Контрольно-измерительные приборы и системы. – 2007. – № 3.

193. Дьяконов В. П. Работа цифровых осциллографов TDS1000B/2000B с системой компьютерной математики MATLAB // Схемотехника. – 2007. – № 7, 8.

194. Раушер Кристоф. Основы спектрального анализа. ROHDE&SCHWARZ. – М.: Горячая линия – Телеком, 2006.

195. Основы анализа спектра в реальном масштабе времени // Tektronix, www.tektonix.com/rsa.

196. Анализаторы спектра реального времени. Анализаторы спектра реального времени серии RSA6100A с частотным диапазоном от 6,2 ГГц до 14 ГГц // Tektronix, www.tektonix.com/rsa.

197. Кирсанов М. Н. Графы в Maple. Задачи, алгоритмы, программы. – М.: Физматлит. – 2007.

198. Maple Getting Started Guide. Maplesoft of division Maple Inc. – Canada, 2007.

199. Monagan M. B., Geddes K. O., Heal K. M. and other. Maple Introductory Programming Guide. Maplesoft of division Maple Inc. – Canada, 2007.

200. Monagan M. B., Geddes K. O., Heal K. M. and other. Maple Advanced Programming Guide. Maplesoft of division Maple Inc. – Canada, 2007.

201. Васильев А. Н. Mathematica. Практический курс с примерами решения прикладных задач. – Киев: Век; СПб.: КОРОНА-ВЕК. – 2007.

202. Дьяконов В. П. MATLAB 7.*/R2006/R2007. Самоучитель. – М.: ДМК-Пресс. – 2008.

203. Дьяконов В. П. SIMULINK 5, 6, 7. Самоучитель. – М.: ДМК-Пресс. – 2008.

Алфавитный указатель

D

E

F

G

К

Л

М

H

O

P

Книги издательства «ДМК Пресс» можно заказать в торгово-издательском холдинге «АЛЬЯНС-КНИГА» наложенным платежом, выслав открытку или письмо по почтовому адресу: **123242, Москва, а/я 20** или по электронному адресу: **orders@alians-kniga.ru**.

При оформлении заказа следует указать адрес (полностью), по которому должны быть высланы книги; фамилию, имя и отчество получателя. Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в Internet-магазине: **www.alians-kniga.ru**.

Оптовые закупки: тел. **(495) 258-91-94, 258-91-95**; электронный адрес **books@alians-kniga.ru**.

Дьяконов Владимир Павлович

Энциклопедия компьютерной алгебры

Главный редактор *Мовчан Д. А.*
dm@dmk-press.ru
Корректор *Синяева Г. И.*
Верстка *Чаннова А. А.*
Дизайн обложки *Мовчан А. Г.*

Гарнитура «Петербург». Печать офсетная.
Усл. печ. л. 118,5. Тираж 1000 экз.
№
Издательство ДМК Пресс
Web-сайт издательства: www.dmk-press.ru
Internet-магазин: www.alians-kniga.ru