Numerical Methods and Software

David Kahaner
National Institute of Standards & Technology
(National Bureau of Standards)

Cleve Moler
Ardent Corporation

Stephen Nash
George Mason University

Prentice-Hall International, Inc.

Д.Каханер, К.Моулер, С.Нэш

Численные методы и программное обеспечение

Перевод с английского
под редакцией Х. Д. ИКРАМОВА

Москва «Мир» 1998

УДК 519.852.6
ББК 22.193
К31

Каханер Д., Моулер К., Нэш С.
К31 Численные методы и математическое обеспечение: Пер. с англ.– М.: Мир, 1998.– 575 с., ил.

ISBN 5-03-002432-8

Книга американских специалистов, представляющая собой расширенный и существенно переработанный вариант известной книги Форсайт Дж., Малькольм М., Моулер К. Машинные методы математических вычислений, М.: Мир, 1980. Несмотря на измененный авторский состав, в книге сохранен неформальный стиль изложения, помогающий читателю освоить современное программное обеспечение по численным методам.

Для специалистов по вычислительной и прикладной математике, аспирантов и студентов, изучающих численные методы и программное обеспечение ЭВМ.

ББК 22.193

РФФИ

Издание осуществлено при поддержке Российского фонда фундаментальных исследований по проекту № 97-01-14004

Редакция литературы по математическим наукам

ISBN 5-03-002432-8 (русск.)

ISBN 0-13-626672-X (англ.)

Предисловие редактора перевода

Предлагаемая книга является переработанной и расширенной версией учебника Дж. Форсайта, М. Малькольма и К. Моулера «Машинные методы математических вычислений». Русский перевод последнего, выпущенный в 1980 г. издательством «Мир», приобрел известную популярность у советского читателя и в ряде вузов, где преподается численный анализ, стал в то время основой вычислительного практикума.

Почему потребовалось обновление книги, авторы подробно объясняют в своем предисловии. Мы же отметим, что переработка и существенное увеличение объема не ухудшили, как это нередко бывает, изначально удачную книгу. Она сохранила «философию и неформальный стиль» прежней, как о том говорят авторы, по-прежнему хорошо и ясно написана и дает теперь значительно больше информации – для читателя, которому эта информация нужна (соответствующие параграфы помечены звездочкой). Появились и новые привлекательные особенности, например прекрасные исторические заметки о знаменитых вычислителях, таких, как Ньютон, Фурье, Гаусс, фон Нейман и др.

Для читателя, который знаком с прежней книгой, отметим еще одно важное изменение: тексты программ вынесены на дискету, прилагаемую к каждому экземпляру (оригинала).

Все три автора книги – известные американские специалисты по численному анализу. Однако о Кливе Моулере хочется сказать более подробно. Это не только первоклассный алгебраист и соавтор популярнейших учебников (двух уже названных и книги Форсайт Дж., Молер К. Численное решение систем линейных алгебраических уравнений, М.: Мир, 1969). Широкое признание Моулеру принесло участие в разработке знаменитых алгебраических программных пакетов EISPACK и LINPACK, и еще большее – проектирование и реализация популярнейшей системы математических и инженерных вычислений MATLAB. Эта система

стала в американских университетах стандартным средством поддержки курсов по численному анализу, теории управления, обработке сигналов и др. К настоящему времени в США изданы около четырех десятков учебников, рассчитанных на использование MATLAB'а.

В заключение я хотел бы поблагодарить коллег, согласившихся сотрудничать со мной в переводе этой книги – А. Б. Кукаркина, А. Б. Кучерова, Е. С. Николаева, С. И. Орлика, М. В. Уфимцева и Г. С. Икрамова.

Х. Д. Икрамов.
e-mail:ikramov@cmc.msk.su

От авторов

Круг людей, профессионально занимающихся численным анализом и подготовкой математического программного обеспечения, не широк. Тем не менее в нем есть, к счастью, мужчины и женщины, сознающие необходимость разработки алгоритмов и программ как можно более высокого качества и передачи их в другие сферы научного сообщества. Многие из этих одаренных людей с энтузиазмом поддерживали нас словом и делом при написании этой книги. Ее главы и программы, прилагаемые к книге, прошли тщательную проверку специалистов. Мы хотели бы особо отметить чувство признательности, которое испытываем по отношению к Жавьеру Берналу, Джеймсу Блу, Тому Буту, Ральфу Байерсу, Джорджу Бирну, Паулю Каламаи, Альфреду Карассо, Джанет Доналдсон, Грэму Фэрвезеру, Фреду Фритчу, Майклу Грини, Марку Джонсону, Уильяму Кахану, Брюсу Келлогу, Рэндаллу Левеку, Даниэлю Луазье, Джорджу Марсалья, Джону Нэшу, Роджеру Пинкхэму, Филипу Радиновичу, Берту Расту, Роберту Шнабелю, С. Декстер Сазерленд, Паулю Шварцтрауберу, Скипу Томпсону, Лейну Уотсону и Х. Джозефу Виверу в связи с замечаниями и предложениями, которыми они с нами поделились. Мы старались как можно полнее учесть эти предложения, но, разумеется, только мы несем ответственность за все ошибки, упущения и несогласованности, оставшиеся в книге. Особо мы должны поблагодарить Джеральда Канделу, помогавшего нам в подготовке ряда рисунков. Наконец, мы хотели бы выразить свою благодарность руководству Центра вычислительной и прикладной математики Национального института стандартов и технологии (Национального бюро стандартов), в особенности Полу Богсу, Бертону Колвину, Гленну Ингрэму и Фрэнсис Салливэн, создавшим нечто вроде питательной среды, в которой математическое обеспечение нашло себе законное место.

Главным стимулом к написанию этого учебника было желание освежить и расширить материал книги Форсайта Дж., Малькольма М., Моулера К. Машинные методы математических вычислений (русский перевод: М.: Мир, 1980.– *Ред.*). Нам по душе философия и неформальный стиль этой книги, а также то, что в нее включены программы. Первоначально мы намеревались ограничиться ее пересмотром, однако, учитывая кончину Форсайта и уход Малькольма в другие сферы деятельности, в конце концов решили начать заново. Мы получили

разрешение издателя заимствовать некоторые фрагменты исходной книги. На практике это свелось к тому, что было сохранено несколько задач. Основной текст почти полностью обновлен, хотя, разумеется, мы вдохновлялись оригиналом. Мы хотели бы сказать, что считаем себя в долгу перед всеми, кто писал на эту тему прежде нас; может быть, нам удалось изложить некоторые вопросы чуть яснее или чуть основательнее.

Наконец, благодарим наши многострадальные семьи за поддержку и терпение.

Дэвид Каханер
Клив Моулер
Стивен Нэш

Глава 1
Введение

1.1. Для чего нужна новая книга?

Компьютеры быстро меняются. На смену традиционной универсальной ЭВМ, последовательно выполняющей скалярные операции, в настоящее время приходит многообразие различных архитектур. Такие векторные компьютеры, как Cray 1, CDC Cyber 205 или ETA 10, могут манипулировать с массивами, вычислять скалярные произведения и производить одновременно несколько операций, и все эти действия подчинены одной машинной команде. Современные мультипроцессоры объединяют до нескольких тысяч сильно связанных компьютеров, работающих более или менее независимо над различными аспектами единой задачи. Множество исследователей пытаются освоить преимущества, предоставляемые этими новыми машинными архитектурами.

Способы, посредством которых научные работники и инженеры общаются с компьютерами, также меняются. Медленные линии связи пользователей и удаленных ЭВМ уступают место персональным компьютерам и рабочим станциям с типичной для них высокой скоростью передачи данных. Это в свою очередь стимулирует развитие приложений, характеризующихся режимом диалога и интенсивным использованием графики при относительно небольшом объеме программирования.

Параллельно с этими событиями появляются все более совершенные численные методы. Вследствие быстрого прогресса возникает зазор между вычислительными методами, которым обучают физиков, химиков и инженеров в соответствующих курсах, и программным обеспечением, с которым будет иметь дело действующий специалист. Нужен учебник, находящийся на уровне современного численного анализа, учитывающий достижения вычислительной и коммуникационной техники и в то же время пригодный для вводного курса. Эта книга предназначена именно для такой цели. Мы приведем сейчас четыре простых примера. Если в них встретятся какие-либо незнакомые вам термины, можете быть уверены, что они будут разъяснены в последующих главах.

(1) Пусть нужно провести гладкую кривую через заданные точки, координаты которых известны с достаточной точностью. Часто для этой цели рекомендуют применение *сплайнов*, однако лучшие на настоящий момент программы используют *эрмитовы кубические кривые*. При выполняемом в диалоговом режиме проектировании (таком, как в системах машинного проектирования или компьютеризованного производства) хорошим выбором являются также *кривые Безье*.

(2) Программы, обрабатывающие матрицы (например, программы решения линейных систем), можно организовать таким образом, что они будут очень эффективны и на скалярных, и на векторных компьютерах. Для этого соответствующие алгоритмы нужно реализовать с помощью типовых операций нижнего уровня, так называемых *базисных подпрограмм линейной алгебры* (BLAS – Basic Linear Algebra Subroutines).

(3) Для вычисления определенных интегралов рекомендуют *адаптивные* методы, однако интенсивно используемые программные пакеты, включающие в себя *формулы Кронрода*, дают значительно лучшие результаты.

(4) В современном программном обеспечении для численного интегрирования обыкновенных дифференциальных уравнений *методы Рунге–Кутты* большей частью вытесняются *многозначными* методами.

В этой книге обсуждаются вопросы, принадлежащие, на наш взгляд, к числу наиболее важных для научного работника, которому предстоит заниматься вычислениями. Среди них много стандартных: решение систем линейных уравнений, интерполирование, быстрое преобразование Фурье, вычисление интегралов и численное интегрирование дифференциальных уравнений, оптимизация и моделирование. В каждом случае мы пытались отразить в своем изложении то, чем реально занимаются профессионалы в соответствующей области. Это иллюстрируется приведенными выше примерами. Кроме того, в любой главе можно найти самые современные идеи и понятия; их столько, сколько мы посчитали полезным включить в книгу. Например, в главе о линейных уравнениях описаны столбцово-ориентированные алгоритмы, в главе об оптимизации объясняется использование квазиньютоновых методов, в главе о моделировании введены фракталы, и т. д.

Следуя приводимому ниже плану, книгу можно использовать для односеместрового курса по численным методам. Все главы организованы таким образом, чтобы как можно быстрее подвести учащегося к работе с подпрограммами. В каждой главе основные идеи излагаются в самом начале так, чтобы параграф, описывающий программы, мог быть удобным финишным пунктом. Последующие параграфы содержат более сложный материал и могут использоваться выборочно, в зависимости от имеющегося учебного времени.

План односеместрового курса

Глава 1

Глава 2 (§ 1–4): машинная арифметика и ошибки вычислений

Глава 3 (§ 1–3): решение хранимых систем линейных уравнений

Глава 4 (§ 1–9): полиномиальная и кусочно-полиномиальная интерполяция

Глава 5 (§ 1–7): квадратуры и адаптивные квадратуры

Глава 6 (§ 1–5): решение линейных задач метода наименьших квадратов

Глава 7 (§ 1–3): решение одного нелинейного уравнения

Глава 10 (§ 1–6): случайные числа (если позволяет время)

Вовсе не обязательно включать в курс все главы или придерживаться такого же их порядка, как в книге. В частности, после изложения материала первых семи глав остальные четыре главы можно упорядочить произвольно. Более подробные сведения о соподчиненности глав дает рис. 1.1.

Наш план односеместрового курса покрывает лишь наиболее важные вопросы глав 1–7. Для двухсеместрового курса лектор может дополнительно привлечь основной материал остальных четырех глав или ограничиться меньшим числом тем, но излагать их более подробно.

Некоторые параграфы помечены знаком *; они требуют несколько более высокого уровня математической подготовленности читателя. Эти параграфы можно опустить, не нарушая непрерывности изложения. Даже с такими купюрами основного материала будет достаточно для двухсеместрового курса. Мы умышленно опустили некоторые важные вопросы: либо потому, что они слишком сложны для разъяснения в одной главе (дифференциальные уравнения с частными производными), либо потому, что они составляют предмет более специализированного интереса (собственные значения).

Эта книга предназначена для студентов естественнонаучных и технических специальностей. По типу она представляет собой нечто

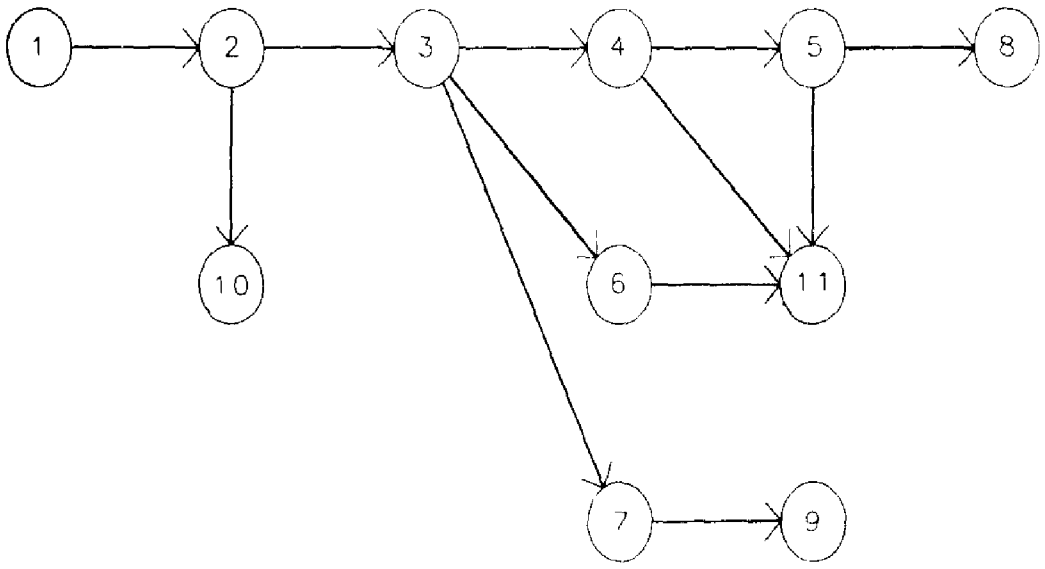


Рис. 1.1. Соподчиненность глав.
Замечания:
(1) § 6 гл. 9 требует сведений из гл. 6;
(2) § 12 гл. 8 требует сведений из гл. 7.

среднее между рецептурным справочником и учебником численного анализа. Предполагается, что читатель имеет за плечами два года обучения высшей математике. Сюда должны входить дифференциальное и интегральное исчисление, а также кое-что из теории матриц и дифференциальных уравнений. Кроме того, предполагается, что читатель имеет доступ к компьютеру и некоторые первоначальные познания в языке Фортран. По завершении курса, основанного на этой книге, учащийся

(1) будет понимать возможности и ограничения, связанные с некоторыми методами научных вычислений;

(2) будет иметь опыт работы с более чем дюжиной высокоавторитетных подпрограмм, реализующих эти методы;

(3) усовершенствует свои навыки программирования на Фортране, которые станут вполне достаточными для реальных приложений.

1.1.1. Комментарии по поводу гибкого диска, прилагаемого к книге

К учебнику прилагается 5.25-дюймовый гибкий диск[1]. На него записаны тексты всех обсуждаемых в книге фортранных программ, а также вспомогательные программы и связанные с задачами файлы данных. Диск содержит файл READ-ME.NMS с подробным объяснением того, как организованы файлы. Формат диска – двусторонний, 360 килобайт, – допускает чтение на любом персональном компьютере с операционной системой MS-DOS, версия 2 или 3 (например, на IBM PC/XT или PC/AT). Вопросы, относящиеся к программам, можно направлять по следующим адресам:

David Kahaner
NBS, Center for Applied
Mathematics
Technology Building, Room A 151
Gaithersburg, MD 20899

Stephen Nash
George Mason University
Operations Research
Department
Fairfax, VA 22030

За дополнительную плату можно приобрести те же программы, записанные в ином формате, или версии программ для другой разрядности. Всю необходимую информацию можно получить у авторов.

1.2. Подпрограммы

Очень важную часть этой книги составляет набор фортранных подпрограмм. В сущности, книгу вполне можно было бы рассматривать как расширенное «руководство пользователя» по этим подпрограммам. Они отнюдь не являются простыми иллюстративными программами, а, напротив, адекватно отражают уровень современного математического обеспечения. Вместе они составляют полезную программную библиоте-

[1] Речь идет об английском издании книги. Соответствующая информация о русском издании приводится в предисловии редактора перевода. – *Прим. перев.*

ку. Для обсуждаемых в книге разделов численного анализа существуют хорошо написанные и документированные подпрограммы, вследствие своей робастности и удобства в обращении получившие всемирное распространение. Они входят в известные, используемые всем научным сообществом собрания подпрограмм, такие, как Linpack, Quadpack, Minpack, SLATEC и др. Мы выбрали те из них, которые, на наш взгляд, больше всего подходят для включения в книгу. Кроме того, *мы совладали с искушением написать свои собственные версии подпрограмм.* Таким образом, студент приобретет двойной выигрыш: получит доступ к некоторым из лучших имеющихся подпрограмм и испытает удовлетворение от сознания, что эти подпрограммы встретятся ему снова в будущей профессиональной деятельности. Единственное, что мы себе позволили, это (а) стандартизовать документацию каждой подпрограммы в соответствии с соглашениями, принятыми для библиотеки SLATEC министерства энергетики (эта библиотека широко используется многими государственными лабораториями США); (б) в тех случаях, когда практика подсказывает, что какая-то особенно популярная подпрограмма существенно выиграет от простого интерфейса с пользователем, надстроить такой интерфейс над оригиналом. Главные мотивы, которыми мы руководствовались, положив эти подпрограммы в основу учебника заключались в том, чтобы (а) не изобретать колесо: не создавать заново программ для хорошо разработанных математических задач; (б) содействовать распространению высококачественного программного обеспечения, тщательно спроектированного, документированного и проверенного (см. [Gaffney, 1987]).

Приведем перечень подпрограмм, включенных в книгу. Более полную информацию об их первоисточниках можно получить, пользуясь списком литературы в конце книги.

Системы линейных уравнений

SGEFS:

подпрограмма для решения системы $Ax = b$ с оценкой достигнутой точности взята из библиотеки SLATEC. Представляет собой драйвер для подпрограмм SGECO и SGESL из пакета Linpack.

Интерполяция

PCHEZ,

PCHEV:

подпрограммы для операций со сплайнами и визуально привлекательными кусочно-полиномиальными функциями, в частности для интерполирования. Эти подпрограммы являются удобными драйверами для пакета PCHIP, входящего в состав библиотеки SLATEC.

Квадратуры

Q1DA:

подпрограмма для адаптивного вычисления одномерных интегралов, включая интегралы с особенностями. Это удобный драйвер для под-

программы QK15 из пакета Quadpack.

QK15:

подпрограмма для быстрого оценивания одномерных интегралов без использования адаптации. Применяется 15-точечная квадратурная формула Гаусса–Кронрода. Взята из пакета Quadpack.

PCHQA:

подпрограмма для интегрирования одномерного массива данных. Это удобный драйвер для пакета PCHIP, входящего в состав библиотеки SLATEC.

Линейная задача наименьших квадратов

SQRLS:

подпрограмма для решения системы линейных уравнений методом наименьших квадратов. Это удобный драйвер для подпрограмм из библиотеки SLATEC. Последние в свою очередь вызывают подпрограммы из пакета Linpack.

Решение нелинейных уравнений

FZERO:

подпрограмма для решения одного уравнения с одним неизвестным $f(x) = 0$. Взята из библиотеки SLATEC.

SNSQE:

подпрограмма для решения системы n уравнений с n неизвестными $f(x) = 0$. Это удобный драйвер для подпрограмм из пакета Minpack. Взята из библиотеки SLATEC.

Обыкновенные дифференциальные уравнения

SDRIV2:

подпрограмма для решения системы $y' = f(t, y)$, включая случай жестких уравнений. Это удобный драйвер для пакета SDRIV из библиотеки программ Национальной лаборатории в Лос-Аламосе.

Оптимизация и нелинейная задача наименьших квадратов

FMIN:

подпрограмма для решения задачи одномерной минимизации [Brent, 1973].

UNCMIN:

подпрограмма для решения задачи многомерной минимизации. Это удобный драйвер для подпрограммы из статьи [Schnabel, Koontz, Weiss, 1982].

Случайные числа и моделирование

UNI,

RNOR:

подпрограммы для генерирования случайных чисел с равномерным или нормальным распределением, взяты из библиотеки SLATEC.

Тригонометрические приближения и быстрое преобразование Фурье
EZFFTF,
EZFFTB,
CFFTF,
CFFTB,
CFFT2D:
удобные подпрограммы вещественного и комплексного быстрого преобразования Фурье (одно- и двумерного), взяты из библиотеки SLATEC.

Многие из этих подпрограмм весьма сложны, и для того, чтобы описать во всех деталях, как и почему они работают, потребовалось бы гораздо больше времени, чем может затратить на материал подобного типа большинство студентов (см., например, следующий параграф). Следовательно, необходимо рассматривать такие подпрограммы как черные или, может быть, серые ящики. Это согласуется с существующей тенденцией мыслить модулями, системами или преобразованиями, отображающими вход в выход в соответствии с некоторыми четко определенными правилами. Большинство программистов именно таким образом подходит к стандартным функциям (SIN, EXP и т. д.). Последние воспринимаются как нечто, заведомо работающее правильно. По мере того как алгоритмы решения задач совершенствуются, они, во-первых, превращаются в примитивы языка вместо того, чтобы оставаться подпрограммами, добавляемыми к основному тексту; во-вторых, для достижения большей эффективности они реализуются аппаратно; в-третьих, они исчезают из элементарных учебников. Именно так обстоит дело с алгоритмами машинной арифметики с плавающей точкой (т. е. арифметики «вещественной», в отличие от арифметики целых чисел) и вычисления стандартных функций. Это же во все большей степени происходит с алгоритмами генерирования случайных чисел, быстрого преобразования Фурье и решения систем линейных уравнений.

Для более сложных алгоритмов недостаточно просто объяснить, как обращаться к соответствующей подпрограмме, и сказать, что она «обычно работает». В численных методах всегда имеется бездна ловушек. Студента нужно предупредить о них, чтобы он мог правильно диагностировать симптомы численного нездоровья. Это требует определенного уровня понимания используемых методов; см. в связи со сказанным пример отрицательных эффектов плохо сконструированного датчика случайных чисел в гл. 10.

Нам часто приходится консультировать научных работников, желающих изучить исходный текст подпрограммы, чтобы модифицировать ее для некоторых специальных целей. Однако подобное желание далеко не всегда оправданно. Обычно хорошо сконструированная подпрограмма сама способна справиться с большинством возникающих ситуаций, и изменять ее как заблагорассудится излишне и неразумно. Более правильное решение состоит в том, чтобы посоветоваться с

местным экспертом: он может предложить лучшую альтернативу. По этой причине, а также из-за сложности подпрограммы не приводим в книге листинги, а ограничиваемся документацией подпрограмм. Мы надеемся таким образом поощрить осторожный подход к модификациям. На диске, вложенном в кармашек в конце учебника, имеются полные программы, а также примеры их использования. Мы рассчитываем, что читатели перепишут их в свои компьютеры, чтобы иметь возможность проработать задачи, обсуждаемые в отдельных главах.

Заметим, что для выявления ошибок в программах может потребоваться много времени. Однако программы из этой книги имеют широкое хождение, и все ставшие известными ошибки в них уже исправлены. Авторы попытаются исправить любые новые ошибки, о которых им будет сообщено. Наконец, подчеркнем, что совершенствование алгоритмов происходит быстрыми темпами. Поэтому советуем читателям регулярно справляться в библиотеках по месту своей работы о поступлении новых подпрограмм.

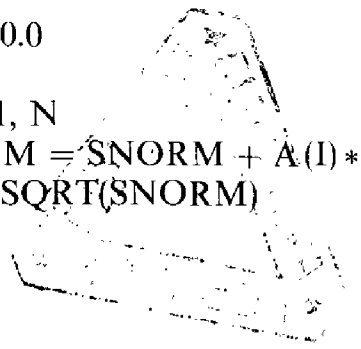
1.3. Математическое обеспечение. Пример: квадратный корень из суммы квадратов

При разработке программ, пригодных для включения в интенсивно эксплуатируемую библиотеку или достаточно надежных для того, чтобы использоваться на борту космического корабля, возникает немало вопросов. Для иллюстрации некоторых из них рассмотрим задачу вычисления *евклидовой нормы* (или длины) массива $a = (a_1, \ldots, a_n)$

$$||a||_2 = \left(\sum_{i=1}^{n} a_i^2 \right)^{1/2}.$$

Вычисления евклидовой нормы требуют многие подпрограммы этой книги. Оно представляется несложной задачей, и реализующая его функция Фортрана могла бы выглядеть таким образом:

```
      REAL FUNCTION SNORM (N, A)
C
C Вычисляет квадратный корень из суммы квадратов элементов
C массива A длины N (версия 1)
      REAL A(N)
      INTEGER N, I
C
      SNORM = 0.0
C
      DO 20 I = 1, N
   20    SNORM = SNORM + A(I)**2
      SNORM = SQRT(SNORM)
      RETURN
      END
```



Функция SNORM представляет собой непосредственную реализацию формулы, определяющей величину $||a||_2$, но она работает не всегда. Даже для не очень большого числа a_i его квадрат a_i^2 может оказаться слишком большим для представления в компьютере, и выполнение программы прекратится досрочно: произойдет *переполнение*. Этот вопрос мы обсудим более подробно в следующей главе. С другой стороны, если число a_i близко к нулю, его квадрат может быть слишком мал для нетривиального представления в машине; такая ситуация определяется термином *исчезновение порядка*. При исчезновении порядка во многих машинах малое число автоматически заменяется нулем. Если некоторые из остальных элементов a_i достаточно велики, то значение нормы $||a||_2$ будет правильным, но если все компоненты вектора a малы, то функция SNORM выдает значение нуль.

Один из способов разрешения этой проблемы состоит в том, чтобы найти наибольшую компоненту вектора a и разделить на нее все компоненты. У масштабированного вектора наибольшая по модулю компонента равна 1, и вычисление нормы можно теперь провести без осложнений. Сказанное иллюстрируется приводимым ниже программным вектором. Заметим, что программа работает даже в том случае, если $a = 0$ или $n = 0$.

```
      REAL FUNCTION SNORM (N, A)
C
C Вычисляет квадратный корень из суммы квадратов элементов
C массива A длины N (версия 2)
C Предполагается, что при исчезновении порядка число заменяется
C нулем.
C
      REAL A (*), AMAX
      INTEGER N, I
C
      SNORM = 0.0
      IF (N.LE.O) RETURN
      AMAX = 0.0
      DO 10 I = 1, N
   10     AMAX = MAX (AMAX, ABS (A,(I)))
C
      IF (AMAX.LE.0.0) RETURN
      DO 20 I = 1, N
   20     SNORM = SNORM + (A(I)/AMAX)**2
      SNORM = AMAX*SQRT(SNORM)
      RETURN
      END
```

Эта функция более надежна, но ее тоже можно улучшить. Если компоненты вектора a не слишком велики и не слишком малы, то в масштабировании нет нужды и данная программа будет производить

вдвое больше вычислений, чем это необходимо. Более сложный алгоритм, учитывающий эту проблему, равно как и некоторые другие, предложен в [Lawson, 1978] и реализован подпрограммой SNRM2 в упоминавшемся выше пакете Linpack. Подпрограмма SNRM2 имеется на диске в конце учебника. Нет необходимости подробно изучать эту подпрограмму для того, чтобы усвоить основной тезис: составление эффективных и надежных программ даже для простых задач требует тщательного продумывания, а также понимания особенностей машинной арифметики.

1.4. Переносимость

Переносимой называют программу, которая без каких-либо изменений способна работать на машинах различных моделей и при этом дает идентичные результаты. Такова цель, но достичь ее удается редко. Программа на Фортране, которую вы написали для одного компьютера, может подвести, если вы попытаетесь воспользоваться ею на другом.

(1) Программа может не пройти трансляцию. Хотя Фортран возник еще в 50-х годах, точное описание языка до недавнего времени отсутствовало. Каждый производитель вычислительной техники поставлял на рынок трансляторы со своими особенностями или с различным синтаксисом.

Все программы этой книги написаны на Фортране-77, современном и тщательно определенном диалекте языка Фортран. Всякая фирма, ориентирующаяся на производство компьютеров для научных расчетов, предусматривает комплектацию их транслятором с Фортрана-77. Хотя допускаются расширения и дополнительные возможности, транслятор обязан отмечать все нестандартные обороты программного текста. Таким образом, можно, казалось бы, надеяться на переносимость программ. Однако даже программа, написанная с соблюдением синтаксиса Фортрана-77, не гарантирована от сбоя при трансляции. Например, стандартом не определены границы области машинно-представимых чисел. Поэтому оператор X = 1.E50 будет воспринят на CDC Cyber, но отторгнут на VAX 11/780, поскольку VAX Fortran, как правило, не допускает значений, превосходящих число, приблизительно равное 10^{38}. Точно так же не стандартизованы максимальные длины массивов: они определяются конструктивными ограничениями и организацией системного программного обеспечения. Некоторые трансляторы с Фортрана для персональных компьютеров IBM требуют, чтобы длины массивов были меньше, чем 64 К = 64 · 1024 = 65536 байтов (1 байт = 8 двоичных разрядов)[1]. Обычно это означает, что порядок квадратной матрицы

[1] Обозначение 64 К для числа 64 1024 родилось в компьютерную эру и было навеяно использованием двоичных чисел для адресации памяти. Однако само по себе приближение К = 1024 ≈ 1000 = кило имело по крайней мере одно более раннее применение, а именно в кулинарном деле. Для изготовления слоеных

A (I, J) должен быть меньше 128, если каждый элемент занимает 4 байта. Иногда программа не проходит трансляцию по нестандартным причинам: в трансляторе имеется дефект или определение языка недостаточно точно. Эти проблемы не тривиальны. В одной из фирм, специализирующихся на продаже программных библиотек, было подсчитано, что почти 40% продукции фирмы, написанной на «стандартном» Фортране, вызывает осложнения при трансляции на одной или даже нескольких из числа поддерживаемых фирмой моделей. Подпрограммы, прилагаемые к этой книге, прошли проверку на различных компьютерах: VAX, CDC Cyber, Sun, а также на персональных компьютерах IBM.

(2) Программа может пройти трансляцию, но при исполнении досрочно прекратить работу или дойти до конца, но дать не те результаты. Различия в конструкциях компьютеров отражаются в фундаментальных различиях точности. Так, аппаратура машины Cray 1 обеспечивает примерно 12 верных десятичных знаков при выполнении оператора X = SQRT (7.), но на компьютере Sperry Univac мы получим лишь около 7 знаков. Разумеется, на любом компьютере точность вычислений можно изменять в широких пределах, но это достигается ценой существенного замедления, поскольку требует применения специальных сложных подпрограмм. Задача составления переносимых программ еще больше усложняется тем обстоятельством, что во многих программах используются машинно-зависимые константы. Среди машинно-зависимых параметров, встречающихся в математическом обеспечении, наиболее распространенным является машинный эпсилон. Это число, определяемое в гл. 2, связано с точностью арифметики с плавающей точкой.

1.5. Конструирование программ: обработка ошибок

Практика показывает, что при пользовании программами неизбежно происходят ошибки. Очевидными примерами могут быть запросы на вычисление квадратного корня из отрицательного числа или тангенса от аргумента $\pi/2$. Робастные программы должны проверять приемлемость значений своих входных параметров. Например, невозможно найти точку пересечения двух параллельных прямых, и если программа получила такое требование, она должна выдать сообщение об ошибке. В ходе вычислений могут возникать ошибки и другого рода; скажем, подпрограмма не в состоянии решить задачу с заданной точностью. Конструирование механизма, сообщающего пользователям об ошибках, является делом одновременно скучным и трудным; скучным потому, что большинство пользователей все же не делает ошибок, и у разработчиков

пирожных раскатывают тесто, наносят на него масло, складывают пополам и повторяют эту процедуру обычно от 8 до 10 раз. В результате получается $2^8 - 2^{10}$ слоев. Во Франции слоеное тесто используется как основа для пирожного «наполеон» (иначе *"mille-feuille"* – «тысячелистник»).

алгоритма нет желания продумывать все комбинации необычных ситуаций, которые могут встретиться, а трудным потому, что механизм должен быть простым, эффективным и гибким.

Простейший способ работы с ошибками состоит в их игнорировании: программа, в которой обнаружена ошибка, или прекращает свое выполнение, или продолжает его с теми неправильными данными, какими располагает. Для некоторых видов ошибок существуют априорные решения, позволяющие продолжить вычисления, например присваивание $\mathrm{tg}\,\pi/2$ в качестве значения фиксированного большого числа или взятие абсолютной величины от аргумента функции SQRT. В некоторых случаях программа выдает сообщение типа «Ошибка: квадратный корень из отрицательного числа», а затем возобновляет работу или прекращает ее. Более хитроумные программы могут выдавать *код ошибки*, который расшифровывается с помощью приведенного в документации списка возможных сообщений об ошибках. Но этот способ годится не для всякой программы: как бы вы посмотрели, например, на вычисление квадратного корня путем вызова подпрограммы SQRT (X, IERROR)?

Среда, в которой вы работаете, определяет вашу позицию по отношению к ошибкам. Кроме того, существуют разные типы ошибок: серьезные ошибки (когда программа не способна решить, как ей следует продолжать работу; такие ошибки могут указывать на фундаментальный изъян в самой задаче), предупреждения (например, о том, что решение задачи не единственно) или просто информация (скажем, решение задачи существует, но тривиально). Для простых программ с одной–двумя подпрограммами выводимые на печать сообщения об ошибках вполне приемлемы. Но если мы имеем дело с большой программой, состоящей из тысяч модулей, то напечатанные сообщения могут остаться незамеченными, особенно когда они невелики и теряются в массе других выдач. Вы также едва ли захотели бы увидеть предупредительное сообщение посреди элегантно форматированного табулированного материала или в центре графика.

Крайне огорчительно для пользователя, запустив большую программу, получить сообщение типа «Недопустимые аргументы» и при этом не знать, чем оно вызвано или как его отменить. Среди профессионалов существует общее согласие в том, что коды ошибок – удобное, но, возможно, недостаточно гибкое средство и что подпрограммы не должны генерировать сообщения об ошибках посредством операторов WRITE или PRINT, разве что имеется входной параметр, позволяющий отказаться от печати.

В большинстве крупных собраний программ делается попытка согласованного подхода к обработке ошибок. Типичным примером может служить XERROR [Jones et al., 1983] – пакет подпрограмм, разработанный для библиотеки SLATEC. Некоторые из этих подпрограмм используются в данном учебнике. Идея этого пакета заключается в том, что обработка всех ошибок должна производиться одной

программой, которая является единственным местом, где содержатся операторы WRITE. Поэтому библиотека SLATEC, насчитывающая более чем 200 000 строк на Фортране, содержит лишь два оператора WRITE, и оба они находятся в пакете XERROR. Недостаток такого подхода – необходимость дополнительных программ.

Классификация ошибок (например, фатальные и нефатальные ошибки, предупреждения и т. п.) осуществляется разработчиком программной библиотеки, но при этом пользователю предоставляются некоторые средства управления. Так, пользователь может решить, что нефатальная ошибка, скажем «Запрошена слишком высокая точность», для его конкретной ситуации является фатальной или что не следует печатать предупреждения. Предусматривается также накапливание сообщений об ошибках в избранных файлах, чтобы все сообщения находились в удобном месте. Поскольку текст сообщений об ошибках готовится разработчиком библиотеки, в него может быть вложена специфическая проблемно-зависимая информация, которую трудно передать с помощью кодов ошибок. Некоторые разработчики используют и XERROR, и код ошибки.

1.6. Конструирование программ: рабочая память

Многие из программ этой книги требуют дополнительной памяти для промежуточных вычислений. Такую память часто называют *рабочей* или *временной*. Если размер этой памяти фиксирован, то достаточно внутренних описаний. Если же он зависит от входных параметров, скажем от числа заданных точек или порядка матрицы, то должен быть принят механизм, обеспечивающий нужное пространство. Существуют три распространенных подхода.

(1) Можно потребовать, чтобы в вызываемой программе был описан массив с фиксированными размерами. Программа MAT, обрабатывающая матрицы, может иметь внутренний массив с описанием REAL ATEMP (500), ограничивающий максимальный порядок решаемой задачи. Подобный способ вполне эффективен, если предусмотрено указание ошибки в случае обращения к MAT с чрезмерно большой задачей. Недостатки состоят в том, что все пользователи расплачиваются затратами памяти, определяемыми наибольшей допустимой задачей, и что при необходимости решать большие задачи в программу MAT нужно вносить изменения. В своих собственных программах это сделать несложно, но на большинстве вычислительных установок библиотечные подпрограммы транслируются заранее и доступны пользователю именно в такой форме. Чтобы предупредить многократное копирование, вычислительные центры обычно не предоставляют пользователям исходных текстов. Если программная библиотека защищена авторскими правами, ее копирование может быть вообще недопустимым.

(2) В некоторых координированных собраниях программ, например

в библиотеке PORT[1], моделируется *динамическое распределение памяти.* В служебной программе общего назначения описывается большой массив W. Когда некоторая подпрограмма библиотеки, скажем MAT, нуждается в рабочей памяти, она обращается к распределителю памяти с запросом на требуемое количество. Распределитель сообщает MAT адрес в W начала блока неиспользуемого пространства подходящей длины. Прежде чем подпрограмма MAT вернет управление пользователю, она должна вызвать перераспределитель с тем, чтобы ненужное более рабочее пространство могло использоваться другими программами. Основные недостатки этого метода таковы:

(а) каждый несет потери в памяти, занимаемой большим массивом W;

(б) для управления распределением памяти необходимы дополнительные программы;

(в) программы библиотеки вступают в сложную связь, что затрудняет изъятие отдельных программ.

С другой стороны, поскольку рабочий массив велик, маловероятно, чтобы конкретная задача могла выйти за рамки имеющегося пространства. Кроме того, в большом программном проекте, состоящем из многих подпрограмм, управление рабочей памятью в едином месте, скорее всего, будет более эффективным, чем если бы каждая подпрограмма решала эту проблему самостоятельно.

(3) Подпрограмма, нуждающаяся в рабочем пространстве, такая, как наша подпрограмма MAT, может потребовать, чтобы в *вызывающей* программе был описан массив W с соответствующими размерами, который бы передавался MAT как параметр. Внутри MAT этот массив не описывается или имеет фиктивное описание, скажем REAL A(*), что позволяет использовать MAT для задач различного порядка. При таком подходе MAT может требовать и наличия в последовательности вызова параметра LENW – точного числа, указанного пользователем в операторе DIMENSION для W. Подпрограмма может сверить размер входной задачи со значением LENW, чтобы удостовериться, что рабочий массив достаточно велик. (Напомним, что в Фортране нет автоматического средства, с помощью которого вызываемая программа могла бы определить длину переданного ей массива.) Недостатки этого подхода – увеличение количества аргументов в последовательности вызова, а также опасность того, что при записи в память MAT выйдет за границу массива W и тем самым уничтожит нужную информацию (это произойдет, если пользователь занизил размерность массива W и в то же время неправильно задал значение LENW). Достоинствами же являются общность подхода и независимость различных программ. Большинство математических подпрограмм используют эту технику: несколько примеров вы найдете в данном учебнике.

---

[1] The PORT Mathematical Subroutine Library, AT & T Bell Laboratories, 600 Mountain Ave., Murray Hill, NJ 07974.

1.7. Историческая справка: Бэкус и язык Фортран

Как показывают университетские учебные планы по информатике, обучение программированию сейчас модно вести на примере языков Паскаль, Си, APL или еще каких-либо «структурированных» языков. Фортрану времени почти не уделяется; его обычно характеризуют как устаревший и неструктурированный язык. Однако Фортран был первым языком высокого уровня; разработанный в середине 50-х годов группой сотрудников фирмы IBM под руководством Джона Бэкуса, он все еще остается языком научных вычислений, тогда как все другие языки играют менее важную роль.

Со времен второй мировой войны и до 1954 года практически все программы писались (главным образом, женщинами) в машинных кодах или на языке ассемблера. Программисты считали свою работу творческим процессом, родственным искусству, и много ума и изобретательности было вложено в преодоление трудностей программирования на компьютерах той эпохи. Уже существовало несколько языков полуавтоматического программирования, такие, как Autocode, Speedcoding и Dual, но они были не очень эффективны. Правда, это не имело особого значения, поскольку недостатки в логике циклов и условий, а также в адресной арифметике терялись на фоне огромного количества времени, затрачиваемого при программном моделировании операций с плавающей точкой. Появление компьютера IBM 704 со встроенными устройствами для плавающей точки и индексирования радикально изменило ситуацию; неэффективность программ теперь нечем было прикрыть. По этой причине программисты скептически относились к возможностям систем автоматизации программирования, что привело Бэкуса к убеждению: единственный способ внедрить такую систему – это добиться, чтобы она переводила научные программы умеренной сложности, составленные на исходном языке, в объектные программы, уступающие по скорости соответствующим вручную написанным программам не более чем в два раза. И сейчас ему кажется, что современные языки страдают от того, что слишком сильно акцентируются вопросы архитектуры языка и недостаточно – проблема генерирования эффективных программ.

Фортран никогда не проектировался в собственном смысле этого слова. Бэкус утверждает: «Мы не считали проектирование языка трудной задачей, а скорее просто прологом к истинной проблеме: созданию компилятора, который бы давал эффективные программы... Мы ничего не знали о многих вопросах, которые впоследствии были признаны важными, например о блочных структурах, условных выражениях, описаниях типов, – нам казалось, что, после того как выработаны понятия оператора присваивания, переменной с индексами и оператора DO, остающиеся проблемы языкового проектирования тривиальны» [Backus, 1979]. Фортран игнорировал пробелы; он и сейчас их игнорирует. Эта черта языка неоднократно критиковалась и все же была

сохранена, так как при наборе текстов много проблем связано с трудностью распознавания и подсчета количества пробелов. Разработчики полагали, что Фортран «по существу ликвидирует программирование в машинных кодах и отладку», поэтому недостаточное внимание было уделено выявлению синтаксических ошибок.

Поначалу Фортрану был оказан сдержанный прием, но уже к 1958 г. более половины компьютеров IBM 704 использовали его в течение 50 и более процентов своего рабочего времени, а для нескольких установок этот процент превысил 80. В настоящее время все большие научные вычислительные центры поддерживают Фортран – значительно усовершенствованный, но все же напоминающий оригинал – в качестве основного языка научных расчетов. Например, американский Национальный центр атмосферных исследований (NCAR – National Center for Atmospheric Research) в Боулдере, штат Колорадо, является «исключительно фортранным учреждением» (слова Дж. Адамса, председателя комитета ANSI Fortran и консультанта NCAR). В Ливерморской Национальной лаборатории имени Лоренса (LLNL) в Калифорнии «почти все вычисления, производимые на больших машинах, делаются на Фортране; это по-прежнему самый лучший язык для физиков» (С. Хендриксон, помощник директора LLNL по расчетам для систем обороны). Бэкус полагает, что «структурное программирование может рассматриваться как скромная попытка внести некоторый порядок в хаотический мир операторов... Новые направления в программировании когда-нибудь приведут к обретению гораздо большей интеллектуальной и вычислительной мощи. Фортран хорош для некоторых приложений. Однако если говорить о нем как о языке будущего, то нам следовало потрудиться гораздо лучше. К сожалению, мы этого не сделали».

Сделанные к настоящему моменту вложения в программное обеспечение на Фортране гарантируют, что язык будет оставаться на плаву, может быть, даже до начала следующего столетия. Но в нем происходят изменения. Комитет по Фортрану 8X готовит стандарт для новой версии языка; он появится, вероятно, где-то около 1990 г. Планируются расширения языка, которые увеличат его возможности в отношении обработки массивов и графики, двух слабых мест Фортрана-77. Другие расширения предназначены для того, чтобы усилить вычислительные качества языка, ввести в него конструкцию производных типов данных и придать его синтаксису более современный вид. Циники заявляют, что через тридцать лет язык научных вычислений будет очень сильно отличаться от всего, что мы имеем сейчас, но по-прежнему будет называться Фортраном.

1.8. Другие полезные источники информации

В любом учебнике такого типа, как наш, можно охватить лишь малую часть имеющегося материала. Быть может, в некоторых случаях вы сочтете, что какое-то другое пособие излагает тот или иной вопрос

лучше, чем мы, или что в книге повышенного уровня трактовка предмета глубже. В конечном счете наиболее современное изложение дают журнальные статьи и монографии. В этом параграфе перечисляются некоторые конкретные источники с краткими описаниями, включающими их в рассматриваемый контекст. Более полный список литературы см. в конце книги.

Глава 2. Машинная арифметика

1. The IEEE Floating-Point Standard. Стандарт машинной арифметики, устанавливающий разрядность и правила округления, регулирующий вычисление специальных функций и многие другие вопросы.

2. J. H. Wilkinson. Rounding Errors in Algebraic Computations. Обсуждение способов анализа ошибок округления в приложении к различным вычислительным задачам.

3. M. Abramowitz, I. A. Stegun. Handbook of Mathematical Functions[1]. Всеобъемлющее собрание математических таблиц, тождеств и прочей информации, относящейся к специальным функциям и константам.

4. M. R. Roy. A History of Computing Technology.

Глава 3. Системы линейных уравнений

Глава 6. Приближение данных методом наименьших квадратов

1. J. J. Dongarra et al. The Linpack Users' Guide. Руководство для пользователя пакета подпрограмм по линейной алгебре Linpack.

2. G. H. Golub, C. Van Loan. Matrix Computations[1]. Всеохватывающее обсуждение методов вычислительной линейной алгебры.

3. A. George, J. W. Liu. Computer solution of large sparse positive definite systems[1]. Описание прямых методов в применении к разреженным матрицам.

4. R. S. Varga. Matrix Iterative Analysis[1]. Обсуждение итерационных методов, особенно полезное для систем, возникающих при численном решении дифференциальных уравнений.

5. C. Lawson, R. Hanson. Solving Least Squares Problems[2].

Глава 4. Интерполяция

1. C. deBoor. A Practical Guide to Splines. Книга содержит тщательно продуманное и в то же время доступно написанное изложение теории, а также много примеров и программ.

---

[1] Имеется русский перевод; см. список литературы в конце книги.– *Прим. перев.*

[2] Вместо этой старой и не переведенной на русский язык книги рекомендуем читателю сравнительно недавний перевод: Хейгеман Л., Янг Д. Прикладные итерационные методы. М.: Мир, 1986.– *Прим. перев.*

2. Davis P. Interpolation and Approximation. В действительности это не учебник по численным методам, однако здесь даны теоретические основы большинства форм интерполяции.

Глава 5. Квадратуры

1. P. J. Davis, P. Rabinowitz. Methods of Numerical Integration. Это наиболее полный обзор данной области, который, кроме того, содержит наилучшую библиографию и много численных примеров.

2. R. Piessens et al. Quadpack: A Subroutine Package for Automatic Integration. Руководство для пользователя и пояснения к пакету подпрограмм одномерного интегрирования Quadpack.

Глава 7. Нелинейные уравнения

1. J. M. Ortega, W. C. Rheinboldt. Iterative Solution of Nonlinear Equations in Several Variables[1]. Обстоятельное обсуждение теории и методов.

2. J. J. More, B. S. Garbow, K. E. Hillstrom. User Guide for Minpack-1. Руководство для пользователя подпрограммы SNSQE.

Глава 8. Обыкновенные дифференциальные уравнения

1. L. Lapidus, J. Seinfeld. Numerical Solution of Ordinary Differential Equations.

2. J. Lambert. Computational Methods in Ordinary Differential Equations.

Обе эти книги содержат массу подробностей относительно математической и вычислительной сторон вопроса.

Глава 9. Оптимизация и нелинейный метод наименьших квадратов

1. R. P. Brent. Algorithms for Minimization Without Derivatives.

2. P. E. Gill, W. Murray, M. H. Wright. Practical Optimization[1].

Глава 10. Случайные числа и моделирование

1. D. E. Knuth. The Art of Computer Programming: Volume 2, Seminumerical Algorithms[1]. Подробное обсуждение алгоритмов генерирования случайных чисел.

2. R. Rubinstein. Simulation and Monte Carlo Method. В книге есть алгоритмы и хорошая библиография.

Глава 11. Тригонометрические приближения и быстрое преобразование Фурье

1. H. J. Weaver. Applications of Discrete and Continuous Fourier Analysis.

---

[1] Имеется русский перевод; см. список литературы в конце книги.– *Прим. перев.*

2. A. Rosenfeld, A. Kak. Digital Picture Processing.
3. L. Ludeman. Fundamentals of Digital Signal Processing.
4. P. Bloomfield. Fourier Analysis of Time Series: An Introduction.

1.9. Задачи

1.1. Во многих областях прикладной математики важную роль играет так называемая *функция ошибок*. Она определяется интегральным представлением

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2)\, dt.$$

Этот интеграл нельзя выразить через более элементарные функции.

Каждая глава книги содержит задачу, связывающую материал этой главы с тем или иным аспектом функции ошибок. Мы начнем с таких заданий:

(а) Достаньте таблицу функции $\operatorname{erf}(x)$ и найдите значение $\operatorname{erf}(1.0)$.

(б) Имеется ли на вашей машине функция или подпрограмма, вычисляющая $\operatorname{erf}(x)$? Если да, то выясните, как пользоваться ею, вычислите значение $\operatorname{erf}(1.0)$ и сравните его со значением, найденным в таблице. (На прилагаемый к книге гибкий диск записана программа, вычисляющая $\operatorname{erf}(x)$.)

(в) Как удается подпрограмме вычислять значения функции $\operatorname{erf}(x)$, если эту функцию нельзя выразить с помощью более элементарных функций?

1.2. В 250-м году до новой эры греческий математик Архимед находил приближенное значение числа π следующим образом. Он рассматривал круг диаметра 1, имеющий, следовательно, длину окружности π. Внутрь круга Архимед вписывал квадрат (см. рис. 1.2). Периметр квадрата меньше длины окружности и дает, таким образом, оценку снизу для π. Далее Архимед рассматривал вписанный правильный восьмиугольник, 16-угольник и т. д. Каждый раз число сторон вписанного правильного многоугольника удваивалось и в результате получались все лучшие оценки для π. Используя 96-многоугольники,

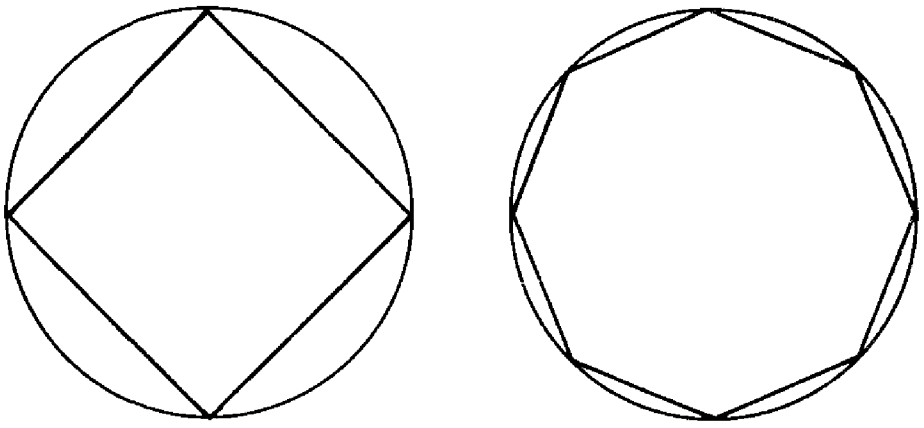


Рис. 1.2.

вписанный и описанный, Архимед сумел доказать неравенства $223/71 < \pi < 22/7$.

Существует рекурсивная формула для подобных оценок. Пусть p_n – периметр правильного вписанного многоугольника с 2^n сторонами. Тогда p_2 – периметр вписанного квадрата и $p_2 = 2\sqrt{2}$. В общем случае

$$p_{n+1} = 2^n \sqrt{2(1 - \sqrt{1 - (p_n/2^n)^2}}\ .$$

Вычислите значения p_n для $n = 3, 4, \ldots, 60$. Попытайтесь объяснить свои результаты. (Эту задачу предложил Аллан Клайн.)

Глава 2

Машинная арифметика и ошибки вычислений

2.1. Введение

Люди занимаются вычислениями уже тысячи лет. Теорема Пифагора, одна из первых вех математики, представляет собой вычислительную формулу. В Древней Греции Архимед и другие математики находили приближенные значения числа π. Сотни лет назад математические таблицы уже использовались в военном деле и навигации. И тем не менее область Численного Анализа как таковая оформилась лишь примерно сорок лет назад, вскоре после окончания второй мировой войны. Как же в течение многих веков человечество избегало вычислительных катастроф?

Хотя Численный Анализ как самостоятельная область существует сранительно недавно, это не относится к лежащим в его основе идеям и задачам. Но лишь с изобретением электронного компьютера в 40-х годах автоматизированные вычисления большого масштаба стали важным инструментом в науке и технологии. Мы должны учитывать два обстоятельства, связанные с этим изобретением.

1) Машинная арифметика отличается от арифметики «карандаша и бумаги». В ручных вычислениях промежуточные результаты непосредственно доступны для обозрения, и точность вычислений можно изменить в соответствии с требованиями момента. В машинной арифметике каждое число имеет фиксированное количество разрядов, которого в известных случаях может не хватать для получения приемлемой точности.

2) Ручные вычисления обычно не бывают длинными, тогда как машинный процесс вычислений может состоять из миллионов шагов. Ничтожно малые ошибки, которыми в коротком вычислении можно было бы пренебречь, накапливаясь в протяженном процессе, могут приводить к разрушительным последствиям. Кроме того, методы, вполне удовлетворительные для малых задач, могут быть безнадежно неэффективными для больших задач того же типа (см. обсуждение формул Крамера в § 1 гл. 3).

В этой главе мы сосредоточимся, главным образом, на первом из названных обстоятельств, а именно на свойствах машинной арифметики. Ее отличия от арифметики «реальной» могут казаться несущественными, однако не таковы последствия этих отличий. Проиллюстрируем сказанное некоторыми простыми примерами.

Пример 2.1. Ряд Тейлора для e^x: неустойчивый алгоритм. Известно, что ряд Тейлора для функции e^x

$$e^x \approx 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

сходится для всех конечных значений аргумента x. Ниже приведен текст Фортран-программы суммирования этого ряда.

```
      WRITE (*, *) ' Задать X'
      READ (*, *) X
      SUM = 1.0
      TERM = 1.0
      I = 1
1     TERM = TERM * (X/I)
      IF (SUM + TERM.EQ.SUM) THEN
         WRITE (*, *) 'E(X) = ', SUM, EXP(X)
         STOP
C     ПРЕКРАТИТЬ СУММИРОВАНИЕ ПРИ ОТСУТСТВИИ
C     ИЗМЕНЕНИЙ В ЧАСТИЧНОЙ СУММЕ РЯДА.
      ELSE
         SUM = SUM + TERM
         I = I + 1
         GO TO 1
      ENDIF
      END
```

Обратите внимание на проверку, заканчивающую суммирование. Она учитывает то обстоятельство, что машинная арифметика является лишь приближенной. Выражение SUM + TERM будет иметь то же значение, что и SUM, если число TERM достаточно мало. (Мы обсудим эти вопросы более подробно в § 3.) Если провести вычисления по этой

Таблица 2.1

x	$E(X)$	e^x
1	2.718282	2.718282
5	148.4132	148.4132
10	22026.47	22026.46
15	3269017.	3269017.
20	4.8516531×10^8	4.8516520×10^8
−1	.3678794	.3678795
−5	6.7377836×10^{-3}	6.7379470×10^{-3}
−10	$-1.6408609 \times 10^{-4}$	4.5399930×10^{-5}
−15	$-2.2377001 \times 10^{-2}$	3.0590232×10^{-7}
−20	1.202966	2.0611537×10^{-9}

программе на машине VAX для различных значений x, то получим числа, представленные в табл. 2.1. Для $x > 0$ эти числа согласуются с тем, чего мы могли бы ожидать, но не так будет при $x < 0$. В некоторых случаях неверны даже знаки результатов. □

Пример 2.2. Разностное отношение для производной.
Производная функции f в точке x определяется формулой

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h},$$

поэтому при «малом» h

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \equiv \Delta_h f(x).$$

В табл. 2.2 показаны результаты вычислений по этой формуле, проведенных на той же машине VAX, для функции $f(x) = e^x$ в точке $x = 1$. Точное значение производной в этой точке (с восемью разрядами после десятичной точки.– *Перев.*) равно 2.71828175.

Таблица 2.2

h	$\Delta_h f(1)$	e	Ошибка
10^{0}	4.67077446	2.71828183	1.95×10^{0}
10^{-1}	2.85884380	2.71828183	1.41×10^{-1}
10^{-2}	2.73191929	2.71828183	1.36×10^{-2}
10^{-3}	2.71987939	2.71828183	1.60×10^{-3}
10^{-4}	2.72035623	2.71828183	2.07×10^{-3}
10^{-5}	2.71797204	2.71828183	3.10×10^{-4}
10^{-6}	2.62260461	2.71828183	9.57×10^{-2}
10^{-7}	4.76837206	2.71828183	2.05×10^{0}
10^{-8}	0.00000000	2.71828183	2.72×10^{0}

Мы видим, что при уменьшении h приближение к производной, даваемое разностным отношением, вначале улучшается, но затем становится все хуже и хуже. Результат, указанный в последней строке, означает, что мы достигли предела точности машинной арифметики. □

Пример 2.3. Два линейных уравнения.
Решим систему двух уравнений

$$0.780x + 0.563y = 0.217,$$
$$0.457x + 0.330y = 0.127.$$

Используя стандартный метод (описываемый в гл. 3) и работая в трех-

разрядной десятичной арифметике, найдем решение

$$x = 1.71,$$
$$y = -1.98.$$

Если подставить эти числа в исходные уравнения, то получим

$$0.780 \times (1.71) + .563 \times (-1.98) - 0.217 = 0.00206,$$
$$0.457 \times (1.71) + .330 \times (-1.98) - 0.127 = 0.00107.$$

Разности между правыми и левыми частями, называемые невязками, достаточно малы. В то же время точное решение системы имеет вид

$$x = 1.000,$$
$$y = -1.000.$$

Хотя этот пример сконструирован искусственно, он все же ясно показывает, что вычисленное решение может очень отличаться от точного (т. е. вычисленные значения совсем не похожи на точные) и в то же время вести себя примерно так же, как точное решение (вычисленные значения «почти решают» уравнения). □

Машинная арифметика есть лишь приближение к тому, что мы обычно понимаем под арифметикой. Различия между ними и являются причиной странных эффектов, иллюстрируемых рассмотренными примерами. Более полные объяснения будут даны позже.

Хорошие численные методы должны учитывать эти эффекты машинной арифметики. Но даже такие методы не всегда в состоянии вычислить решение задачи с приемлемой точностью. Метод, использованный для решения системы в примере 2.3, считается высококачественным, и тем не менее вычисленное решение оказалось неточным. В этом нет вины алгоритма. Данная конкретная задача *плохо обусловлена, некорректно поставлена*, или *сверхчувствительна*; это означает, что ничтожно малые изменения коэффициентов задачи способны привести к большим изменениям ее решения. Не следует ожидать от алгоритма, что он будет давать хорошие результаты, если задача плохо обусловлена.

Что касается примера 2.1, то ниже мы увидим, что существуют эффективные способы вычисления функции e^x для всех значений x, однако применение рядов Тейлора не относится к их числу. В этом случае виноват алгоритм: он *неустойчив*. Следует избегать использования неустойчивых алгоритмов.

Все эти соображения подводят к вопросу о том, что же нужно понимать под «решением» численной задачи. Может оказаться неразумным (как это было в примере 2.3) надеяться на то, что вычисленное решение будет близко к точному решению задачи. Однако для устойчивого алгоритма мы часто сможем гарантировать, что вычисленное решение *точно решает* возмущенную задачу, т. е. задачу, получающуюся из исходной малым изменением коэффициентов. Если задача хорошо обусловлена, отсюда будет следовать, что вычисленное и точное

решения похожи (вычисленное решение есть хорошее приближение к точному решению). Для плохо обусловленной задачи это, по всей вероятности, будет не так, хотя нередко (опять-таки как в примере 2.3) вычисленное решение будет вести себя примерно тем же образом, что и точное решение.

2.2. Представление чисел

В машинных вычислениях участвуют числа двух типов: целые числа и числа «с плавающей точкой» (вещественные числа). Первые компьютеры допускали только целочисленную арифметику. Для представления дробей использовалась воображаемая точка в фиксированной позиции внутри целого числа (например, между четвертой и пятой позициями, считая слева). Это называлось арифметикой «с фиксированной точкой». В 1954 году фирма IBM начала производство компьютера 704, в котором все алгоритмы для чисел с плавающей точкой были реализованы как машинные команды, что чрезвычайно упрощало использование нецелочисленной арифметики. Арифметика с фиксированной точкой более не является стандартным режимом компьютера, исключение составляют лишь некоторые специализированные устройства, например графические терминалы. Стандартна скорее *аппаратно реализованная* арифметика с плавающей точкой, по крайней мере как один из возможных режимов.

В большинстве компьютеров внутреннее представление чисел – двоичное, т. е. в виде последовательностей нулей и единиц. Этого требуют соображения технологии, однако для человека, который учился считать на десяти пальцах, такое представление чисел неестественно. Чтобы яснее показать особенности машинной арифметики, отделив их от деталей аппаратной реализации, мы будем в большей части нашего обсуждения опираться на десятичную, а не на двоичную арифметику.

Ниже мы обрисуем важнейшие свойства машинной арифметики с той степенью подробности, какая нужна для лучшего понимания процесса машинных вычислений. Рассказ будет иллюстрироваться примерами, основанными главным образом на десятичной арифметике, однако будут приведены также существенные детали относительно двоичной арифметики. При этом мы исходим из обсуждаемого ниже стандарта IEEE для двоичной арифметики с плавающей точкой (1985 г.).

Для машинных вычислений можно использовать и другие типы арифметики. Например, было построено несколько компьютеров, работающих в арифметике с основанием 3. Да и люди, случалось, пользовались при счете иными основаниями, скажем основанием 20. Вспомним слова Авраама Линкольна "four score and seven years ago", где в роли основания выступает число 20 (1 score). Слово "score" ведет свое происхождение со времен возникновения фондовой биржи, когда для записи финансовых сделок делали зарубки на деревянных палочках. (В этом объяснении авторы исходят из другого значения слова score –

зарубка, насечка или, в качестве глагола, делать зарубки.– *Перев.*) Следы системы счета с основанием 20 можно различить в таких французских словах, как "quatre-vingt" (восемьдесят), что буквально означает «четырежды двадцать». Или взять хотя бы способ, каким мы считаем по-английски, применяя специальные слова типа "twelve" или "sixteen" вместо словосочетаний "ten-two" или "ten-six", более согласующихся с последующими числительными, такими как "twenty-two" или "fifty-six".

2.2.1. Машинное представление целых чисел

Машинные целые числа представляются конечным количеством разрядов. Для иллюстрации будем считать, что у нас целые числа имеют шесть десятичных знаков. Каждое целое число характеризуется также знаком + или −. Все это означает, что количество машинных целых чисел конечно. В нашем примере наименьшим будет число −999999, наибольшим – число 999999. Целые числа вне этой области для данного компьютера не существуют.

Если работать с целыми числами, далекими от границ числовой области компьютера, то машинная арифметика дает правильные результаты, например $5 + 7 = 12$, $8 - 27 = -19$ и $27 \times 3 = 81$. Деление целых чисел приводит снова к целому числу: в качестве результата операции принимается частное, а остаток отбрасывается. Это значит, что $1/3 = 0$, $4/2 = 2$, $7/(-3) = -2$, и т. д.

Если результат операции над целыми числами слишком велик или слишком мал для данного компьютера, то последствия трудно предсказать. На одних компьютерах будет выдано сообщение об ошибке, а выполнение текущего вычисления прекращено, на других результат будет заменен по циклическому правилу, а вычисление продолжено без какого-либо указания об ошибке. Так, для нашего примера было бы $999999 + 1 = -999999$. Таким образом, нельзя полагаться на результаты вычислений, выходящих за пределы числовой области компьютера.

Эти замечания в равной степени приложимы к операциям с целыми числами, выполняемым в двоичной арифметике, хотя значения наибольшего и наименьшего числа будут иными. В компьютерах ряда популярных моделей для хранения целого числа отводится 32 двоичных разряда, причем один разряд – для знака числа. В этом случае наибольшим целым числом является $2^{31} - 1 = 2\,147\,483\,647$, а наименьшим – число -2^{31}.

2.2.2. Машинное представление вещественных чисел: арифметика с плавающей точкой

Десятичное число с плавающей точкой – это число, представленное в виде $a \times 10^b$; число a называется *мантиссой*, а b – *показателем*. Обычно число b целое, а у числа a слева от десятичной точки находится только

один знак. У обоих чисел a и b количество разрядов конечно, поэтому имеется лишь конечное множество чисел с плавающей точкой и, в частности, существуют наибольшее и наименьшее числа с плавающей точкой.

Для последующих иллюстраций введем арифметику с плавающей точкой, в которой a имеет четыре, а b – два десятичных разряда. Вот несколько примеров чисел с плавающей точкой:

$$4.678 \times 10^{00} = 4.678, \quad -3.355 \times 10^{03} = -3355.,$$
$$9.876 \times 10^{-12} = 0.000000000009876.$$

В этой системе наименьшее и наибольшее числа суть -9.999×10^{99} и 9.999×10^{99}. Между нулем и наименьшим положительным числом имеется разрыв $0.001 \times 10^{-99} = 10^{-102}$.

Представление чисел в этой системе не всегда однозначно, например

$$1.000 \times 10^{00} = 0.100 \times 10^{01}.$$

Чтобы избежать неоднозначности, обычно требуют, чтобы первая цифра мантиссы a была ненулевой, т. е. второе представление в нашем примере становится незаконным. Переход к законному представлению называется *нормализацией*. Исключением из правила является представление нуля: 0.000×10^{00}. Нормализованная система чисел с плавающей точкой содержит меньшее количество чисел. К примеру, в нашей системе наименьшим числом с плавающей точкой будет теперь $1.000 \times 10^{-99} = 10^{-99}$.

Мы уже говорили о том, что существует наибольшее число с плавающей точкой. Если результат операции превышает это значение, то происходит *переполнение*, и для большинства компьютеров данное вычисление на этом заканчивается. Если результатом операции является число, слишком близкое к нулю (в нашем примере меньшее, чем 10^{-99}), то происходит *исчезновение порядка* (говорят также о возникновении машинного нуля.–Перев.). Это событие обычно не имеет таких катастрофических последствий, как переполнение, и многие компьютеры заменяют результат нулем без какого-либо указания на то, что случилось нечто из ряда вон выходящее. Тем не менее есть вычисления, для которых факт появления машинного нуля важен; см., например, задачу 2.6.

Результат операции над числами с плавающей точкой обычно не бывает точным. Для примера рассмотрим в нашей системе вычисление

$$1.234 \times 10^{00} + 5.678 \times 10^{-04} = 1.234 + 0.0005678 = 1.2345678.$$

Его ответ не может быть представлен точно как число с плавающей точкой, но должен быть редуцирован до четырех разрядов. *Усечение* – отбрасывание последних цифр результата – в данном случае дает 1.234×10^{00}. Предпочтительнее, однако, *округление* до ближайшего числа с плавающей точкой, которое привело бы к более точному результату 1.235×10^{00}. Различие между точным ответом и ответом,

полученным в машинной арифметике, называется *ошибкой округления*, независимо от того, использовалось ли действительно округление или усечение.

В компьютере с двоичной арифметикой числа с плавающей точкой обычно представляются в памяти 32 двоичными разрядами, или битами. Стандарт IEEE отводит 24 бита для мантиссы и 8 битов для показателя; сюда входят также биты для хранения знаков мантиссы и показателя. Предполагается, что самый левый бит мантиссы любого числа равен 1, что позволяет не хранить этот бит. По техническим причинам показатель хранится как целое число в интервале [0,255]; чтобы получить фактическое значение показателя, нужно вычесть 127 из хранимого числа. Значение 255 резервируется для представления бесконечности, а также указания незаконных результатов, например квадратных корней из отрицательных чисел. Подобные незаконные результаты называются «не-числами». Из сказанного следует, что наибольшее число с плавающей точкой равно

$$+1.1\ldots1_2 \times 2^{127} \approx 10^{38}.$$

Наименьшее число с плавающей точкой равно приблизительно 10^{-38}. Мантисса из 24 битов соответствует примерно 7 десятичным разрядам. Стандарт IEEE рекомендует также, чтобы компьютеры выполняли арифметику с большей разрядностью, несмотря на то, что результаты операций записываются в память с 32 битами. В машинах, поддерживающих этот стандарт, арифметика с плавающей точкой нередко реализована с внутренней разрядностью 80 битов.

Все компиляторы с Фортрана допускают числа с плавающей точкой удвоенной точности. Для их представления используется вдвое большее количество битов, чем для обычных чисел с плавающей точкой. Если вычисления удвоенной точности не поддержаны аппаратно, то они гораздо медленнее вычислений с одинарной точностью. Большинство компьютеров, соответствующих стандарту IEEE, обладают аппаратно реализованным режимом удвоенной точности, по крайней мере как опцией. Фортран допускает также арифметику комплексных чисел с плавающей точкой одинарной и удвоенной точности, при этом комплексное число представляется парой вещественных чисел с плавающей точкой, обычных или удвоенной точности. Комплексная арифметика редко поддерживается аппаратно; как правило, она реализуется программным путем.

Проектирование процессоров, обеспечивающих действительно надежную арифметику с плавающей точкой, оказалось нелегким делом. Имеется масса примеров обычных вычислений, дающих неправильные результаты, последствия которых могут быть весьма серьезными. Профессор Кахан (Калифорнийский университет в г. Беркли) сыграл важную роль в выявлении недочетов арифметики большинства существующих компьютеров. Он также руководил разработкой четко сформулированного стандарта *IEEE Floating Point Standard*. Этот документ

позволил производителям вычислительной техники наладить выпуск эффективных и надежных чипов, реализующих арифметику с плавающей точкой. Стандарт IEEE точно определяет, как должно выполняться округление и что следует делать, если вычисления приводят к переполнению, исчезновению порядка или к необходимости извлечения квадратного корня из отрицательного числа. В распространенных моделях персональных компьютеров и рабочих станций используются несколько микрокомпьютеров, поддерживающих этот стандарт. Среди них – Intel 8087/80287/80387 и процессоры серий Motorola 6888X.

2.3. Машинные константы

При сложении машинных чисел различной величины результат может оказаться точно равен одному из слагаемых. Например, в нашей четырехразрядной арифметике с плавающей точкой

$$1.000 \times 10^{00} + 1.000 \times 10^{-04} = 1.000 + 0.0001000 = 1.0001 \rightarrow 1.000 \times 10^{00}.$$

Вполне можно было бы, не изменяя результата, заменить меньшее число нулем. Наименьшее число с плавающей точкой, которое при сложении с числом 1.0 дает результат, больший чем 1.0, называется *машинным эпсилоном* и обозначается $\varepsilon_{маш}$. Конкретное значение $\varepsilon_{маш}$ зависит от того, какая арифметика используется – с округлением или усечением. На нашей четырехразрядной машине $\varepsilon_{маш} = .001 = 1.000 \times 10^{-03}$ в случае усечения и $\varepsilon_{маш} = .0005$ при округлениях. В любом случае сумма числа с плавающей точкой и 1.0 имеет только три верных десятичных знака.

Машинный эпсилон определяет относительную погрешность арифметики компьютера (в оригинале – relative precision, т. е. относительная плотность; советские специалисты также не менее, если не более часто говорят об относительной точности, а не погрешности арифметики. – *Перев.*). Если x и y – два положительных числа с плавающей точкой и $x > y$, то их сумму можно записать в виде

$$x + y = x\left(1 + \frac{y}{x}\right).$$

Очевидно, что при $y/x < \varepsilon_{маш}$ сумма с плавающей точкой чисел x и y совпадает с x. Более тщательное исследование показывает, что относительная погрешность сложения чисел с плавающей точкой ограничена величиной $\varepsilon_{маш}$.

Для 32-битовой арифметики с плавающей точкой, удовлетворяющей стандарту IEEE, $\varepsilon_{маш} = 2^{-22} \approx 1.2 \times 10^{-7}$ при использовании округлений. Поэтому бессмысленно рассчитывать более чем на семь верных десятичных знаков в любом результате вычисления с плавающей точкой или на то, что мы сумеем разрешить относительные различия, меньшие этого уровня. Несколько подпрограмм из этой книги имеют входной параметр ε, задающий желаемую точность. Неразумно присваивать ему значение, меньшее $\varepsilon_{маш}$.

Запись десятичного числа, скажем 0.3, в память компьютера сопряжена с ошибкой, поскольку 0.3 не имеет точного представления с плавающей точкой (имеется в виду – двоичного. – *Перев.*). Если программы ввода, входящие в состав компилятора с Фортрана, работают правильно, то ошибка представления локализована в последнем бите. Иногда мы будем выражать это записью вида

$$x_{\text{хран}} = x(1 + \delta_x), \quad \text{или} \quad x_{\text{хран}} - x = x\,\delta_x,$$
$$|\delta_x| \le \varepsilon_{\text{маш}}.$$

Это означает, что относительная погрешность хранимого приближения с плавающей точкой для числа x может достигать величины $\varepsilon_{\text{маш}}$.

Можно воспользоваться сказанным, чтобы разобраться в явлении *катастрофической потери верных цифр* (в оригинале – catastrophic cancellation. – *Перев.*), под которым понимают рост ошибки вследствие вычитания почти равных чисел. Рассмотрим наш модельный компьютер; для него $\varepsilon_{\text{маш}} = 0.0005$. Пусть

$$x_{\text{хран}} = 1.001, \quad y_{\text{хран}} = 1.002$$

и

$$y_{\text{хран}} - x_{\text{хран}} = 0.001,$$

причем последнее равенство верно даже в арифметике с плавающей точкой. Однако в общем случае числа $x_{\text{хран}}$ и $y_{\text{хран}}$ нельзя считать точными. В типичной ситуации они являются хранимыми версиями некоторых других чисел x и y и имеют относительные ошибки, не превосходящие $\varepsilon_{\text{маш}}$. Относительную ошибку разности $y_{\text{хран}} - x_{\text{хран}}$ можно тогда записать в виде

$$\frac{(y_{\text{хран}} - x_{\text{хран}}) - (y - x)}{y - x} = \frac{\delta_y - \delta_x}{0.001}.$$

Абсолютное значение правой части может достигать величины

$$\frac{2\,\varepsilon_{\text{маш}}}{0.001} \approx 2000\,\varepsilon_{\text{маш}} = 1.0.$$

Таким образом, все цифры разности могут быть неверны.

Пример 2.4. Дальнейший анализ рядов Тейлора для функции e^x.

Вернемся к примеру 2.1. Вычисляя значение e^{-15} с помощью ряда Тейлора, имеем

$$3.06 \times 10^{-7} \approx e^{-15} = 1 - 15 + \ldots - 312540.3 + 334864.6 - \ldots.$$

Дальнейшее будет много проще понять, если выписать члены этой суммы со всеми знаками, как они вычислены на машине VAX (табл. 2.3). Заметим, что некоторые из этих членов и, следовательно, промежуточные суммы много больше, чем финальный результат ($\approx 10^{-7}$). Когда один из таких больших членов вычитается из накопленной к этому

Таблица 2.3

n	n-й член ряда Тейлора
0	1.000000
1	−15.00000
2	112.5000
3	−562.5000
⋮	
13	−312540.3
14	334864.6
15	−334864.6
16	313935.5
17	−277001.9
18	230834.9
19	−182238.1
20	136678.6
21	−97627.55
⋮	
51	−0.000000061660813
52	0.000000017786773
53	−0.000000050339924
54	0.000000013983313
55	−0.000000038136312
56	0.000000010215083
57	−0.000000026881800

моменту суммы, разность становится малой и приобретает большую относительную ошибку. В конечном счете разности вообще не имеют верных знаков. Это еще один пример описанного выше явления катастрофической потери верных цифр. Возможным выходом из положения является использование компьютера с меньшей константой $\varepsilon_{маш}$. В табл. 2.4 приведены результаты, вычисленные той же программой, что и в примере 2.1, но на машине CDC Cyber 855, для которой $\varepsilon_{маш} \approx 10^{-14}$. Отметим, что ответы лучше прежних, но при $x < -10$ трудности сохраняются. В примере 2.8 мы предложим другое решение этой задачи. □

Ниже мы суммируем важнейшие свойства вычислений с плавающей точкой. Конкретные параметры приведены для двоичной 32-битовой

Таблица 2.4

x	$E(X)$	e^x
1	2.718282	2.718282
5	148.4132	148.4132
10	22026.47	22026.46
15	3269017.	3269017.
20	4.8516520×10^{8}	4.8516520×10^{8}
−1	.3678794	.3678795
−5	6.7379470×10^{-3}	6.7379470×10^{-3}
−10	4.5399952×10^{-5}	4.5399930×10^{-5}
−15	3.0508183×10^{-7}	3.0590232×10^{-7}
−20	3.865358×10^{-7}	2.0611537×10^{-9}

арифметики с плавающей точкой, удовлетворяющей стандарту IEEE. Ближе познакомиться с числами с плавающей точкой вам поможет задача 2.4.

(1) Множество чисел с плавающей точкой конечно (количество таких чисел равно приблизительно 2^{31}).

(2) Существует наибольшее число с плавающей точкой OFL = = overflow level $\approx 10^{38}$.

(3) Существует наименьшее число с плавающей точкой UFL = = underflow level $\approx 10^{-38}$.

(4) Числа с плавающей точкой между 0 и OFL распределены неравномерно. Между каждыми двумя соседними степенями двойки находится 2^{22} чисел с плавающей точкой, например 2^{22} чисел между 2^{-128} и 2^{-127} и столько же чисел между 2^{126} и 2^{127}. Таким образом, числа с плавающей точкой гуще расположены вблизи нуля.

(5) Арифметические операции над числами с плавающей точкой не всегда приводят к точно представимым результатам, поэтому результаты приходится усекать или округлять до ближайшего числа с плавающей точкой.

(6) Константа $\varepsilon_{маш}$ есть наименьшее число с плавающей точкой, для которого $1.0 + \varepsilon_{маш} > 1.0$ в арифметике с плавающей точкой. Для 32-битового компьютера $\varepsilon_{маш} \approx 10^{-7}$. Это число характеризует относительную точность машинной арифметики.

(7) Константы OFL и UFL определяются главным образом количеством битов показателя, тогда как $\varepsilon_{маш}$ – количеством битов мантиссы. Тем самым эти константы характеризуют разные части представления с плавающей точкой. Имеют место неравенства

$$0 < \text{UFL} < \varepsilon_{маш} < \text{OFL}.$$

Некоторые подпрограммы из этой книги должны знать значения

машинных констант типа $\varepsilon_{маш}$ или OFL. Их значения программы получают с помощью пакета машинных констант (Machine Constants Package), первоначально разработанного сотрудниками AT&T Bell Telephone Laboratories [Fox, 1978]. Пакет содержит три фортранные функции I1MACH(K), R1MACH(K) и D1MACH(K), имеющие единственный целочисленный параметр K; их значениями являются соответственно целое число, число с плавающей точкой и число удвоенной точности. По определению R1MACH(1) = UFL, R1MACH(2) = OFL и R1MACH(4) = $\varepsilon_{маш}$. Прочие значения определяются аналогичным образом. Преимущество подобной организации состоит в том, что она локализует все машинно-зависимые данные стандартизованным способом, не позволяя им по-разному появляться в разных программах. Ее недостаток – необходимость дополнительных подпрограмм.

Некоторые пользователи вставляют в свои программы конкретные константы, например значения PI = 3.14159 или SQ2 = 1.414. Это не препятствует переносимости, однако константы могут оказаться недостаточно точными. Лучше всюду, где это возможно, довериться компьютеру; в наших примерах можно положить

$$\text{PI} = \text{ATAN}(1.0) * 4, \qquad \text{SQ2} = \text{SQRT}(2.0).$$

Стандартные функции Фортрана, такие как SQRT или ATAN, составлены тщательно, и выдаваемые ими результаты обычно отличаются от «точных» значений разве что в последнем бите.

Можно заставить компьютер не только проводить вычисления с плавающей точкой, но и следить за совершаемыми ошибками округлений с тем, чтобы получить затем оценки их накопления. Эта техника называется *интервальным анализом*; она подробно описана в книге [Moore, 1979].

2.4. Ошибки в научных вычислениях

Если результаты вычислений с плавающей точкой отличаются от тех, каких мы ожидали, налицо ошибка:

$$\text{ошибка} = \text{точное значение} - \text{приближенное значение}.$$

Мы часто будем также пользоваться термином «*относительная ошибка*»:

$$\text{относительная ошибка} = \frac{\text{ошибка}}{\text{точное значение}}.$$

Относительная ошибка определена, если знаменатель дроби не равен нулю, и нередко является более полезной мерой достигнутой точности, поскольку меньше зависит от способа масштабирования данных. Так, если мы решим измерять веса в граммах, а не килограммах, то относительная ошибка не изменится, тогда как ошибка умножится на 1000.

Ошибки могут возникать по ряду причин:

(1) Неправильная работа машинных устройств. Эта причина в настоящее время встречается чрезвычайно редко (смотря где.– *Перев.*), но в раннюю эпоху машинных вычислений была делом обычным: среднее время между отказами аппаратуры составляло тогда несколько минут.

(2) Ошибки программиста. Например, запрограммирована неверная формула, и т. п.

(3) Ошибки эксперимента. Это бывает, если данные получены с помощью средств ограниченной точности, например измерительных инструментов.

(4) Игнорирование существенных особенностей задачи. Если в качестве приближения для e^x взять сумму первых пяти членов ряда Тейлора, то, независимо от точности наших вычислений и используемого компьютера, неизбежна некоторая «ошибка усечения». См. в связи с этим пример 2.5.

Пример 2.5. Неустранимая ошибка для разностного отношения.

Вернемся к примеру 2.2. Если разложить $f(x+h)$ в ряд Тейлора относительно точки x, то разностное отношение можно записать в виде

$$\Delta_h f(x) \equiv \frac{f(x+h)-f(x)}{h} = \frac{f(x)+hf'(x)+h^2f''(\xi)/2-f(x)}{h} = \\ = f'(x)+hf''(\xi)/2, \quad x<\xi<x+h.$$

Таким образом, неустранимая ошибка этого приближения равна $hf''(\xi)/2$.

В численном анализе ошибка нередко зависит от некоторого параметра, каким в данном примере было h. Во многих случаях достаточно указать характер зависимости ошибки от параметра, не выписывая выражение для ошибки более детально. В подобных случаях используется введенное Бахманом и Ландау обозначение $O(h)$, так называемое «O большое»; оно означает лишь, что неустранимая ошибка стремится к нулю не медленнее, чем само h. Более точно, мы будем писать

$$p(t) = O(q(t)) \quad \text{при} \quad t \to t_0,$$

имея в виду, что отношение $p(t)/q(t)$ ограничено для t, достаточно близких к t_0. Позднее мы встретимся с другими примерами применения этого полезного обозначения. □

(5) Ошибки вычислений, или ошибки округления. Они проистекают из комбинации двух факторов:

(а) обусловленности, или чувствительности данной задачи;

(б) устойчивости алгоритма.

Иногда говорят, что неустранимая ошибка происходит от членов, которые мы *отбрасываем*, а ошибка вычислений – от членов, которые *оставляем*. См. иллюстрации в последующих примерах.

Пример 2.6. Плохо обусловленная задача: корни полинома четвертой степени.

Нужно вычислить все четыре корня полинома

$$x^4 - 4x^3 + 8x^2 - 16x + 15.99999999 = (x-2)^4 - 10^{-8} = 0.$$

Корни удовлетворяют уравнению $(x-2)^2 = \pm 10^{-4}$ и, следовательно,

$$x - 2 = \pm\sqrt{\pm 10^{-4}} = \pm 10^{-2} \quad \text{или} \quad x - 2 = \pm 10^{-2} i.$$

Итак, корнями будут числа $x_1 = 2.01$, $x_2 = 1.99$, $x_3 = 2 + .01i$, $x_4 = 2 - .01i$. Если мы работаем на компьютере, для которого $\varepsilon_{\text{маш}} > 10^{-10}$, то свободный член полинома будет округлен до 16.0. С точки зрения компьютера, теперь решается уравнение

$$(x-2)^4 = 0.$$

У этой новой задачи четыре корня, равных 2.0, что отличается от корней исходного полинома на 0.5%. В этой задаче малые изменения входных данных (например, изменение одного коэффициента на 10^{-8}) приводят к гораздо бóльшим изменениям решения *независимо от того, каким методом вычисляется решение.* Подобные задачи называют *плохо обусловленными.* Эту сверхчувствительность решения мы не сможем уменьшить каким-либо вычислительным приемом; она связана с самой задачей, а не с методом. □

Пример 2.7. Ряд Тейлора для e^x: более удачный алгоритм.

Вернемся к примеру 2.1 при $x < 0$. Так как аппроксимация с помощью ряда дает при положительных x хорошие результаты, то можно ожидать, что для отрицательных значений аргумента можно использовать формулу

$$e^{-x} = \frac{1}{e^x} = \frac{1}{1 + x + \dfrac{x^2}{2!} + \ldots}.$$

Программа из примера 2.1 была модифицирована так, чтобы для $x < 0$ значение функции e^x вычислялось по этой формуле. В табл. 2.5 показаны результаты, полученные на машине VAX. Теперь они почти точны. В данном случае причина затруднений была не в задаче, а в выборе алгоритма: прямое суммирование ряда оказалось для отрицательных x *неустойчивым алгоритмом.* □

Пример 2.8. Ошибки округления для разностного отношения.

Вернемся к примеру 2.2. Мы можем проанализировать влияние округлений на вычисление $f'(x)$. Предположим, что ни в x, ни в $x + h$ ошибок нет и что при вычислении f единственная ошибка происходит в момент записи вычисленного значения в память. Тогда можно показать, что ошибка в разностном отношении $\Delta_h f'(x)$ ограничена величиной

Таблица 2.5

x	$E(X)$	e^x
1	2.718282	2.718282
5	148.4132	148.4132
10	22026.47	22026.46
15	3269017.	3269017.
20	4.8516531×10^{8}	4.8516520×10^{8}
−1	.3678794	.3678795
−5	6.7379461×10^{-3}	6.7379470×10^{-3}
−10	4.5399924×10^{-5}	4.5399930×10^{-5}
−15	3.0590232×10^{-7}	3.0590232×10^{-7}
−20	2.0611530×10^{-9}	2.0611537×10^{-9}

$2|f(x)|\varepsilon_{\text{маш}}/h$. Тем самым ошибка округления в разностном отношении растет при уменьшении h. Отсюда и из примера 2.5 заключаем, что полную ошибку (неустранимая ошибка + ошибка округления) можно оценить так:

$$|\text{Ошибка}| \le E_{\text{полн}} = \frac{h}{2}|f''(\xi)| + \frac{2}{h}|f(x)|\varepsilon_{\text{маш}} = O(h) + O(1/h).$$

Это выражение демонстрирует то же поведение, что и числа в примере 2.2: убывание поначалу и рост впоследствии. Если продифференцировать функцию $E_{\text{полн}}$ по h и приравнять производную нулю, то найдем значение h, дающее минимум оценки:

$$h = 2\sqrt{\frac{|f(x)|\varepsilon_{\text{маш}}}{|f''(\xi)|}}.$$

Если $|f| \approx |f''|$, то правило

$$h \approx \sqrt{\varepsilon_{\text{маш}}}$$

дает хорошее приближение к значению h, минимизирующему $E_{\text{полн}}$. □

Чтобы получить более точное приближение для производной, можно попытаться уменьшить либо неустранимую ошибку, либо ошибку округления. Если пойти вторым путем, переходя, быть может, на машину с меньшей константой $\varepsilon_{\text{маш}}$, то мы сможем провести вычисления при меньшем значении h, прежде чем округления начнут доминировать; тем самым неустранимая ошибка будет уменьшена, а вместе с нею и полная ошибка $E_{\text{полн}}$. Для иллюстрации сказанного повторим вычисления примера 2.2, но на этот раз, как и в примере 2.4, на машине Cyber. Результаты представлены в табл. 2.6. Для машин VAX и Cyber приближенными значениями величины $\sqrt{\varepsilon_{\text{маш}}}$ будут соответственно

Таблица 2.6

h	$\Delta_h f(1)$	e	Ошибка
10^{0}	4.67077427	2.71828183	1.95×10^{0}
10^{-1}	2.85884195	2.71828183	1.41×10^{-1}
10^{-2}	2.73191866	2.71828183	1.36×10^{-2}
10^{-3}	2.71964142	2.71828183	1.36×10^{-3}
10^{-4}	2.71841775	2.71828183	1.36×10^{-4}
10^{-5}	2.71829542	2.71828183	1.36×10^{-5}
10^{-6}	2.71828318	2.71828183	1.36×10^{-6}
10^{-7}	2.71828199	2.71828183	1.62×10^{-7}
10^{-8}	2.71828213	2.71828183	3.04×10^{-7}
10^{-9}	2.71826650	2.71828183	1.51×10^{-5}

4×10^{-4} и 8×10^{-8}. Отметим, что и в примере 2.2, и здесь указанные значения h дают хорошие приближения для $f'(1)$.

Альтернативный способ получения лучшей аппроксимации состоит в том, чтобы каким-то образом снизить неустранимую ошибку при тех же значениях h. Очевидным решением является выбор более точной формулы. Например, если бы мы аппроксимировали производную центральным разностным отношением

$$\delta_h f(x) \equiv \frac{f(x+h) - f(x-h)}{2h},$$

то такие же операции с рядом Тейлора, как в примере 2.5, привели бы к равенству

$$\delta_h f(x) = f'(x) + \frac{h^2}{6} f'''(x) + \frac{h^4}{120} f^{(5)}(x) + \ldots,$$

и неустранимая ошибка была бы тогда величиной порядка $O(h^2)$. В качестве примера вычислим на машине Cyber приближенное значение $e'(1)$, пользуясь на этот раз центральными разностными отношениями (см. табл. 2.7).

Пример 2.1 иллюстрирует то обстоятельство, что плохо продуманный алгоритм может дать плохой ответ для идеально обусловленной задачи; затруднение было устранено сменой алгоритма. Пример 2.6 показывает, что для некоторых задач «хорошие» ответы не могут быть найдены никаким алгоритмом, потому что задача чувствительна к малым ошибкам во входных данных и в арифметике. Напомним, что речь идет о результатах машинных вычислений, поскольку в теоретическом анализе никаких ошибок округления нет. Важно различать эти две причины неудовлетворительного счета, так как неустойчивые алго-

Таблица 2.7 Аппроксимация посредством центральных разностных отношений

h	$D_h f(1)$	e	Ошибка
10^{0}	3.194528049	2.718281828	4.76×10^{-1}
10^{-1}	2.722814564	2.718281828	4.53×10^{-3}
10^{-2}	2.718327133	2.718281828	4.53×10^{-5}
10^{-3}	2.718282285	2.718281828	4.53×10^{-7}
10^{-4}	2.718281833	2.718281828	4.62×10^{-9}
10^{-5}	2.718281829	2.718281828	8.58×10^{-10}
10^{-6}	2.718281820	2.718281828	8.38×10^{-9}
10^{-7}	2.718281849	2.718281828	2.00×10^{-8}
10^{-8}	2.718282133	2.718281828	3.04×10^{-7}
10^{-9}	2.718266501	2.718281828	1.53×10^{-5}

ритмы и плохо обусловленные задачи встречаются практически во всех областях вычислительной математики. Как только вы осведомлены об их симптомах, диагностировать названные причины становится легко. Что касается примера 2.8, то отметим: вычисление производных не обязательно является плохо обусловленной задачей, однако большинство алгоритмов, использующих разностные отношения, при малых h неустойчивы.

* 2.5. Экстраполяция

Применение центральных разностных отношений иллюстрирует один из возможных подходов к уменьшению неустранимой ошибки. Другим широко используемым приемом является *экстраполяция*. Она приложима к широкому кругу задач, включающему вычисление интегралов и решение дифференциальных уравнений. Ключевой момент в применении экстраполяции – это получение разложения неустранимой ошибки в ряд. Идеальный пример дает неустранимая ошибка для центрального разностного отношения. Найденный для нее ряд непосредственно показывает, что если вычислять разностное отношение, используя h и $h/10$, то во втором случае ошибка должна составлять приблизительно 1/100 ошибки первого случая. Именно это и демонстрирует табл. 2.7, по крайней мере до того момента, когда начинает преобладать ошибка округления. Поскольку в ряд Тейлора для неустранимой ошибки входят члены более высокого порядка, из этой таблицы можно извлечь значительно больше информации. Примем для $\delta_h f(x)$ сокращенное обозначение δ_h. Тогда

$$\delta_h = f'(x) + \frac{h^2}{6} f'''(x) + \frac{h^4}{120} f^{(5)}(x) + \ldots,$$

$$\delta_{h/10} = f'(x) + \frac{1}{100}\frac{h^2}{6}f'''(x) + \frac{1}{10\,000}\frac{h^4}{120}f^{(5)}(x) + \ldots.$$

Комбинируя эти приближения, получаем

$$\frac{100\,\delta_{h/10} - \delta_h}{99} = f'(x) - \frac{h^4}{100 \cdot 120}f^{(5)}(x) + \ldots.$$

Выражение в левой части представляет собой приближение к $f'(x)$. Его неустранимая ошибка начинается с более высокой степени h, а именно с $O(h^4)$, и, по меньшей мере для малых h, это приближение должно быть лучше исходных. Итак, взяв два приближения для различных значений h и скомбинировав их, мы получили третье приближение, более точное, чем два первоначальных. На практике левую часть формулы обычно вычисляют, представляя ее в виде

$$\delta_{h/10} + \frac{\delta_{h/10} - \delta_h}{99}.$$

Это выражение менее чувствительно к ошибкам округления. Вообще, в практических вычислениях, как правило, имеет смысл такая организация формул, когда требуемая величина представлена как малая поправка к имеющемуся хорошему приближению.

Для иллюстрации обратимся к табл. 2.7 центральных разностных отношений. Рассмотрим аналогичную табл. 2.8, где столбец $E'(1)$ содержит экстраполированные приближения. Каждый элемент этого столбца получен из двух элементов столбца первоначальных разностных отношений по приведенной выше формуле. Например, первый элемент в столбце $E'(1)$ вычислен по первым двум элементам столбца $\delta_h f(1)$ таким образом: 2.722815 + (2.722815 − 3.194528)/99 = 2.718050. Столбец ошибок показывает, что погрешность экстраполированных приближений очень быстро уменьшается: примерно в 10 000 раз при каждом переходе,– пока не начинает сказываться влияние округлений. Это

Таблица 2.8

h	E′ (1)	e	Ошибка
10^{-1}	2.71804978127	2.71828182846	2.32×10^{-4}
10^{-2}	2.71828180580	2.71828182846	2.27×10^{-8}
10^{-3}	2.71828182845	2.71828182846	9.08×10^{-12}
10^{-4}	2.71828182855	2.71828182846	9.44×10^{-11}
10^{-5}	2.71828182928	2.71828182846	8.20×10^{-10}
10^{-6}	2.71828181999	2.71828182846	8.47×10^{-9}
10^{-7}	2.71828184879	2.71828182846	2.03×10^{-8}
10^{-8}	2.71828213559	2.71828182846	3.07×10^{-7}

согласуется с предсказанным для неустранимой ошибки поведением как $O(h^4)$. Наилучшее из экстраполированных приближений с погрешностью 9.08×10^{-12} на несколько порядков точнее любого неэкстраполированного приближения.

Эти соображения можно обобщить. Например, экстраполированные приближения сами имеют неустранимую ошибку, представимую рядом по степеням h, начинающимся с члена с h^4. Следовательно, несколько таких приближений можно скомбинировать, чтобы получить приближение еще более высокого качества. Литература на тему экстраполяции очень обширна; см., например, статью [Joyce, 1971]. Работая с последовательностями, всегда важно учитывать возможность применения этой техники. Отдельные элементы последовательности могут приближаться к пределу медленно, однако, комбинируя их надлежащим образом, часто можно получить новую последовательность, сходящуюся к тому же пределу гораздо быстрее.

2.6. Историческая справка: Эккерт и Мочли

В 1971 году газета «Нью-Йорк таймс» назвала «огромной несправедливостью» то, что имена Эккерта и Мочли «едва ли будут когда-нибудь столь же знакомы обывателю, как имена братьев Райт или Томаса Эдисона, не говоря уже о Битлах». Джон Уильям Мочли (1907–1980) и Джон Преспер Эккерт (1919) – кто же эти неизвестные люди?

В 1940 году профессор физики Мочли (Урсинус-колледж, штат Пенсильвания) занялся вопросом о том, как можно было бы более производительно выполнять сложные вычисления, связанные с уравнениями метеорологии. В те времена существовали механические калькуляторы, но они были слишком медленными для приложений, которые он имел в виду. На научной конференции в декабре того же года Мочли познакомился с Джоном Атанасовым, а следующим летом посетил его в государственном колледже Айовы. Атанасов вместе со своим аспирантом Клиффордом Берри сооружал электронный калькулятор на вакуумных трубках, который потенциально должен был быть на несколько порядков быстрее всего, что тогда существовало. Мочли вернулся из своей поездки, крайне взволнованный увиденным. Примерно в то же время он прослушал летний курс в школе инженеров-электроников имени Мура при Университете штата Пенсильвания, чтобы расширить свои познания в области электронных устройств. В школе были поражены его способностями, и Мочли получил предложение остаться в ней в качестве преподавателя. Школа имени Мура имела контракт с Лабораторией баллистических исследований (BRL) на разработку таблиц стрельбы, играющих существенную роль при пристрелке новых орудий. Естественно, что Мочли пришлось втянуться в вычислительные проблемы.

В это время Эккерт, аспирант в области электротехники, работал над

усовершенствованием дифференциальных анализаторов, применявшихся тогда для таких баллистических расчетов. Использовав в своей машине несколько сотен вакуумных трубок, он сумел в десять раз увеличить скорость и точность, однако было ясно, что дальнейшие улучшения на этом пути невозможны. Независимо друг от друга Эккерт и Мочли спроектировали электронный сумматор, и в сочетании с опытом эксплуатации электронного дифференциального анализатора это ясно показало им поразительный потенциал электроники для вычислений. Они обратились в Отдел планирования армии США с предложением о создании универсальной цифровой машины. В 1943 году Мочли в качестве научного консультанта и Эккерт как главный инженер получили полномочия для проектирования и построения вычислительного устройства, основанного на их идеях. Это и был ENIAC (Electronic Numerical Integrator And Computer), первый электронный цифровой компьютер. Его сооружение было закончено в 1946 году в обстановке большой рекламной шумихи. ENIAC не успел поработать в военное время, однако введенный в строй, он находился в BRL в постоянной эксплуатации до 1955 года. Было подсчитано, что за десять лет службы ENIAC проделал больше вычислений, чем все человечество за свою историю до 1945 года.

ENIAC был огромным достижением, но имел в то же время массу недостатков: он был чересчур велик (8 футов в высоту, 3 – в ширину и 100 – в длину; форма его была подковообразной), обладал мизерной памятью (20 десятиразрядных чисел) и программировался лишь посредством ручной установки наружных переключателей. Зимой 1943/44 г. Эккерт и Мочли начали обдумывать идею нового компьютера, который должен был называться EDVAC. Предполагалось, что у него будет память в 1000 слов, что в нем будут использоваться ртутные линии задержки для хранения данных и, самое главное, программа будет храниться в памяти компьютера. В сентябре 1944 года к проекту в качестве консультанта присоединился Джон фон Нейман; в дальнейшем он помогал в конструировании машины. Состояние дела фон Нейман описал в июне 1945 г. в «Первой версии сообщения об EDVAC'е». К несчастью для Эккерта и Мочли, эта «версия» получила широкое распространение вскоре после того, как была написана. Поскольку в сообщении фигурировало только имя фон Неймана, – а фон Нейман был знаменитый математик, – то и было решено, что все идеи сообщения принадлежат ему одному, хотя наиболее важная из них (идея хранимой программы) появилась до того, как он включился в работу над проектом.

Сообщение, кроме того, уничтожило всякие шансы на получение патента. В 1946 г., когда EDVAC еще не был построен, Эккерт и Мочли покинули школу имени Мура из-за разногласий с проводимой ею патентной политикой. Руководство университета считало, что все патенты должны принадлежать школе, а Эккерт и Мочли полагали, что их должны получать изобретатели. После своего ухода Эккерт и Мочли

подали заявку на патент для своей конструкции компьютера. Эта заявка была отклонена; юристы решили, что все идеи были опубликованы в «Первой версии...», тем самым они стали общественным достоянием и не могут быть защищены патентом. Вдобавок не был урегулирован вопрос о том, какие из идей должны быть приписаны Атанасову.

Оказавшись в частном секторе, Эккерт и Мочли организовали компанию с целью разработки коммерчески выгодного компьютера. Мочли стал ее президентом и отвечал за логическое проектирование; Эккерт, предпочитавший оставаться за сценой, был вице-президентом и главным инженером. (За период 1948–1966 гг. Эккерт получил от Патентной Службы США 85 патентов). Бюро переписей и Национальное бюро стандартов должны были первыми приобрести эту новую машину, названную UNIVAC. Как и современные компании, занятые производством высокотехнологичной продукции, детище Эккерта и Мочли постоянно нуждалось в дополнительных капиталах. Хотя UNIVAC оказался успешной разработкой, финансовые трудности заставили владельцев компании продать ее в 1950 г. фирме Remington Rand. И Эккерт, и Мочли продолжали работать в новой компании почти до 1960 г. Remington Rand слилась в 1955 г. со Sperry, отчего образовалась фирма Sperry Rand. В 1986 г. Sperry Rand была поглощена корпорацией Burroughs; расширившаяся корпорация получила название Unisys и выпустила на рынок семейство компьютеров под названием Sperry.

Постепенно компьютеры захватили воображение публики. Отчасти это произошло благодаря телевидению. В 1952 г. телекомпания Си-Би-Эс в рамках программы освещения президентских выборов заключила контракт о том, чтобы на машине UNIVAC был выполнен прогноз их исхода. Были подготовлены соответствующие программы; их пускали на пятом экземпляре UNIVAC'а, еще не покинувшем заводских помещений. Телестудия в Нью-Йорке демонстрировала макет панели управления, лампочки на нем были подсоединены к огням новогодней елки. Результаты выборов стали известны еще только в нескольких округах, а UNIVAC уже предсказал Эйзенхауэру триумфальную победу над Стивенсоном. Этого не ожидали, а потому было решено, что в программах есть ошибка. (В оригинале используется слово bug, что означает «насекомое», а также «технический дефект»; по поводу происхождения этого второго значения авторы дают следующее примечание: «Этот термин возник в 1947 г. в Гарварде в ходе разработки компьютера Mark. Предложил проект и руководил его осуществлением Ховард Эйкен, человек сильного характера (младший преподавательский состав и студенты называли его «командир»), постоянно отдававший надежности предпочтение перед скоростью. И тем не менее электромеханический компьютер Mark II допускал необъяснимые сбои в работе. Случайно один из разработчиков открыл аппаратный шкаф и обнаружил там застрявшего между контактов двух реле мертвого мотылька. Грейс Хоппер, которая впоследствии сыграла важную роль в создании языка Кобол, извлекла мотылька и приклеила его к своей

летописи этой эпопеи, присоединив запись о насекомом, причинившем столько хлопот».—*Перев.*) Было проделано несколько манипуляций с алгоритмами, чтобы получить прогноз победы Эйзенхауэра с небольшим преимуществом. Однако, по мере того как поступали результаты из новых округов, становилось все более очевидным, что первоначальное предсказание компьютера было правильным. Тем же вечером, около девяти часов, Уолтер Кронкайт публично признал перед телезрителями факт подделки исходного прогноза.

2.7. Задачи

2.1. Ряд Тейлора для функции ошибок имеет вид

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{n!(2n+1)}.$$

Этот ряд сходится для всех x. Напишите программу для вычисления $\operatorname{erf}(x)$ посредством этого ряда. Возьмите ровно столько членов ряда, чтобы первый отброшенный член не менял накопленной суммы, будучи добавлен к ней в арифметике с плавающей точкой. Так как ряд знакопеременный, то ошибка от его обрывания будет в этом случае меньше единственной ошибки округления. Исследуйте влияние ошибок округления, сравнивая вычисленную сумму со значением, взятым из таблицы для $\operatorname{erf}(x)$, или значением, полученным по надежной подпрограмме (см. задачу 1.1). Попробуйте $x = 0.5$, 1.0, 5.0 и 10.0. Объясните ваши результаты. *Указание*: Внутренний цикл вашей программы может быть таким:

```
10  OLDS = S
    EN = EN + 1.0
    T = -XSQ * T * (2.0 * EN - 1.0)/(EN * (2.0 * EN + 1.0))
    S = S + T
    IF(OLDS .NE. S) GO TO 10
```

Какие значения должны быть присвоены переменным T, S, EN, XSQ перед входом в этот цикл? Не забудьте множитель $2/\sqrt{\pi}$.

2.2. Прочтите следующие две Фортран-программы:

```
    EPS = 1.0
10  EPS = EPS/2.0
    WRITE(*, *) EPS
    EPSP1 = EPS + 1.0
    IF(EPSP1 .GT. 1.0) GO TO 10
    STOP
    END
    EPS = 1.0
10  EPS = EPS/2.0
    WRITE(*, *) EPS
```

```
      IF (EPS .GT. 0.0) GO TO 10
      STOP
      END
```

Запустите обе программы на вашей машине и объясните различие результатов. Если вы имеете доступ к персональному компьютеру IBM с математическим сопроцессором, то запустите программы и там и объясните результаты.

2.3. Составьте программу, которая вычисляла бы сумму ряда

$$\phi(x) = \sum_{k=1}^{\infty} \frac{1}{k(k+x)}$$

для $x = 0.0,\ 0.1,\ \ldots,\ 1.0$ с ошибкой, не превышающей 0.5×10^{-6}. *Замечание.* Эта задача требует как человеческого анализа, так и машинной вычислительной мощи, и одно без другого вряд ли приведет к успеху. Прежде всего не тратьте годовой бюджет машинного времени, пытаясь просуммировать ряд «грубой силой». *Указание.* Используйте соотношение

$$\frac{1}{k(k+1)} = \frac{1}{k} - \frac{1}{k+1}$$

для доказательства того, что $\phi(1) = 1$. Затем представьте разность $\phi(x) - \phi(1)$ рядом, который сходится быстрее исходного ряда, определяющего $\phi(x)$. Вам придется повторить этот прием, прежде чем вы получите достаточно быстро сходящийся ряд для вычисления функции $\phi(x)$: разложить дробь $1/[k(k+1)(k+2)]$. Ссылка: [Hamming, 1969], с. 48–50.

2.4. Рассмотрим воображаемую систему с плавающей точкой, состоящую из следующих чисел:

$$S = \{\pm b_1.b_2b_3 \times 2^{\pm y}\},$$

где каждое число b_2, b_3 и y принимает одно из значений 0 или 1, а b_1 всегда равно 1, за исключением случая $b_1 = b_2 = b_3 = y = 0$.

(а) Изобразите фрагмент вещественной оси с нанесенными на нее элементами множества S.

(б) Покажите, что S содержит двадцать пять элементов. Каковы здесь значения констант OFL, UFL и $\varepsilon_{\text{маш}}$?

2.6. Попробуем вычислить приближенные значения производных для следующих функций:

$$f_1(x) = \sin x \text{ в точке } x = 1,$$
$$f_2(x) = 10\,000 \sin x \text{ в точке } x = 1,$$
$$f_3(x) = \operatorname{tg} x \text{ в точке } x = 1.59.$$

Первую производную будем аппроксимировать центральным разностным отношением, а для второй производной используем формулу

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}.$$

(а) Найдите приближения к первой производной по указанному правилу для всех трех функций. Обычно, чтобы получить наиболее точную оценку производной, полагают $h \approx \sqrt[3]{\varepsilon_{\text{маш}}}$. Испробуйте формулу центрального разностного отношения для значений $h = 10^k \sqrt[3]{\varepsilon_{\text{маш}}}$, где $k = -2, -1, 0, 1, 2$.

(б) Теперь вычислите приближения для вторых производных. В данном случае традиционным выбором для h является $h \approx \sqrt[4]{\varepsilon_{\text{маш}}}$. Испробуйте для всех трех функций значения $h = 10^k \sqrt[4]{\varepsilon_{\text{маш}}}$, где $k = -2, -1, 0, 1, 2$.

В каждом случае сравните вычисленные приближения с истинными значениями производных. Можете ли вы объяснить, как на выбор наилучшего значения h влияет поведение функции $f(x)$ вблизи точки x?

(в) Используйте технику экстраполяции (см. § 5), чтобы улучшить приближения, полученные в пунктах (а) и (б).

2.6. Задача состоит в том, чтобы написать программу вычисления значений функции

$$X(p, k, N) = \binom{N}{k} p^k (1-p)^{N-k}, \quad 0 \le p \le 1, \quad 0 \le k \le N \le 2000,$$

часто встречающейся в статистических исследованиях. Символ $\binom{N}{k}$ обозначает биномиальный коэффициент

$$\binom{N}{k} = \frac{1}{k!(N-k)!} = \frac{N(N-1)\ldots(N-k+1)}{k(k-1)\ldots 2 \cdot 1}.$$

Изучите следующий фрагмент Фортран-программы для вычисления функции $X(p, k, N)$:

```
      Q = 1.0 - P
      X = 1.0
      DO 1 J = 1,K
1        X = X*(N + 1 - J)
      DO 2 J = 1,K
2        X = X/J
      DO 3 J = 1,K
3        X = X*P
      DO 4 J = K + 1,N
4        X = X*Q
```

(а) Используя данную программу, попробуйте вычислить значение $X(0.1, 200, 2000) \approx 0.03$. Прокомментируйте трудности, с которыми вы столкнулись.

(б) Один из возможных подходов к преодолению этих трудностей указан в книге [Sterbenz, 1974]. Пусть B – высокая степень двойки, выбранная так, чтобы произведение $2000 \cdot B$ не достигало уровня переполнения, а частное $B/2000$ – уровня исчезновения порядка, например $B = 2^{100}$. Если X приобретает значение, большее, чем B, то мы делим X на B. После такого деления правильным представлением вычисляемой величины будет $X * B$. Пусть счетчик I указывает, сколько раз происходило деление на B; тогда имеет место равенство

$$X(p, k, N) = X \cdot (B)^I, \qquad I \geq 0.$$

Аналогично, введем величину $S = 1/B$. Если X становится меньше, чем S, то мы умножаем X на B и вычитаем единицу из значения счетчика. Пусть по окончании вычислений $I < 0$. Возможно, что мы сумеем $|I|$ раз разделить X на B без исчезновения порядка и получить таким образом разумный результат. В противном случае нуль – это лучший ответ, который можно получить. Измените в этом духе свою программу и проверьте ее на задаче пункта (а).

2.7. В задаче 1.2 была получена следующая формула приближенного оценивания числа π:

$$p_{n+1} = 2^n \sqrt{2(1 - \sqrt{1 - (p_n/2^n)^2})}\ ,$$
$$p_2 = 2\sqrt{2}.$$

Она уязвима в двух отношениях; допускает исчезновение порядка и катастрофическую потерю верных знаков.

(а) Объясните недостатки формулы.

(б) Формулу можно улучшить так, чтобы устранить из нее вычитание. Прежде всего запишем p_{n+1} в виде

$$p_{n+1} = 2^n \sqrt{r_{n+1}},$$

где

$$r_{n+1} = 2(1 - \sqrt{1 - (p_n/2^n)^2}),$$
$$r_3 = 2/(2 + \sqrt{2}).$$

Покажите, что

$$r_{n+1} = \frac{r_n}{2 + \sqrt{4 - r_n}}.$$

Полученную итерационную формулу используйте для вычисления величин p_n и r_n для значений $n = 3, 4, \ldots, 60$. (Эта модификация предложена У. Каханом).

(в) В конечном счете, разность $4 - r_n$ будет округляться до значения

4. Таким образом, формула, полученная в пункте (б), при больших значениях n также подвержена влиянию ошибок округления. Однако есть ли теперь основания для беспокойства?

2.8. К методу Архимеда для приближенного оценивания числа π можно прийти иным путем. Пусть $P(h)$ обозначает периметр вписанного правильного n-угольника; здесь $h = 1/n$.

(а) Используя геометрические соображения, покажите, что

$$P(h) = (1/h)\sin(\pi h).$$

Составьте программу, которая бы вычисляла эту функцию для $n = 3$, 6, 12, 24, 48, 96. Для каждого значения n выведите на печать полученное приближение и его ошибку.

(б) Разлагая функцию $\sin x$ в ряд, проверьте равенство

$$P(h) = \pi + \sum_{j=1}^{\infty} \frac{(-1)^j \pi^{2j+1}}{(2j+1)!\,2j} h^{2j}.$$

Другими словами, $P(h) = \pi + O(h^2)$. Измените свою программу так, чтобы можно было проверить, действительно ли погрешность ведет себя как $O(h^2)$. Для этого выводите на печать отношение ошибки данного шага к ошибке предыдущего. К какому пределу сходятся эти отношения? Почему?

(в) Поскольку мы знаем, что ошибка в $P(h)$ представляет собой $O(h^2)$, для улучшения оценки можно применить экстраполяцию. Покажите, что новая последовательность

$$P_{(1)}(h) \equiv \frac{4P(h) - P(2h)}{3}$$

должна сходиться к π с ошибкой, ведущей себя как $O(h^2)$. Преобразуйте формулу так, чтобы новое приближение представлялось в виде поправки к прежнему приближению (см. § 5), и измените свою программу, чтобы вычислить новые оценки. Будут ли они более точны, чем $P(h)$? *Указание.* Если вы постройте таблицу со значениями $P(h)$ в первом столбце и значениями $P_{(1)}(h)$ во втором, то это поможет вам как в самом процессе вычислений, так и в осмыслении его результатов. Будет полезно также проводить эти вычисления с удвоенной точностью.

(г) Покажите, что последовательности

$$P_{(2)}(h) = \frac{2^4 P_{(1)}(h) - P_{(1)}(2h)}{2^4 - 1}, \qquad P_{(3)}(h) = \frac{2^6 P_{(2)}(h) - P_{(2)}(2h)}{2^6 - 1}$$

сходятся к числу π с ошибкой $O(h^6)$ и $O(h^8)$ соответственно. Преобразуйте формулы, как в пункте (в), вычислите новые приближения и добавьте их к своей таблице в качестве третьего и четвертого столбцов.

(д) Выведите формулы для пятого и шестого столбцов и вычислите соответствующие приближения. Какова наилучшая оценка вашей таблицы и насколько она точнее оценки Архимеда (см. задачу 1.2)?

Проявляется ли каким-нибудь образом влияние ошибок округления?

2.9. Трансляторы с Фортрана не обязательно должны содержать функцию, вычисляющую котангенс. Вы можете написать такую функцию сами, исходя из определения котангенса COT(X) = = COS(X)/SIN(X). Проверку результата на точность можно организовать многими способами. Простой тест состоит в сравнении значения COT(X) со значением выражения (COT(X/2) - 1.0/COT(X/2))/2.0, математически эквивалентного котангенсу. Составьте программу, которая бы вычисляла значения обеих функций для 2000 точек интервала (6π, $25\pi/4$). Выведите на печать среднюю ошибку и максимальную ошибку. Насколько удовлетворительны результаты? Как узнать, откуда проистекают наблюдаемые ошибки: из вычисления COT(X) или из вычитания во втором выражении? Для проверки вычислите последнее с удвоенной точностью.

2.10. Эта задача касается некоторых деталей арифметики с плавающей точкой.

(а) Пусть I и J – целые числа. Объясните, почему фортранное выражение (I + J) * (I – J) следует предпочесть выражению I * I – J * J. Применимо ли то же объяснение к случаю, когда I и J – вещественные числа?

(б) Составьте программу, которая оценивала бы для вашего компьютера константу OFL для одинарной и удвоенной точности, а также определяла наибольшее целое число. Что случится, если превышено какое-либо из этих значений?

2.11. Примеры из § 4 могут внушить вам мысль, что неустойчивость алгоритма всегда связана с потерей верных знаков вследствие вычитания. Однако это не обязательно, как показывает следующий пример (принадлежащий У. Кахану). Пусть N – число битов в слове с плавающей точкой вашего компьютера/компилятора. (Для машин DEC Vax, IBM PC, Sun или Apple Macintosh $N = 32$). Рассмотрим следующую Фортран-функцию:

```
      REAL FUNCTION F(X, N)
      REAL X
      F = ABS(X)
      DO 10 I = 1,N
         F = SQRT(F)
   10 CONTINUE
      DO 20 I = 1,N
         F = F * F
   20 CONTINUE
      RETURN
      END
```

Если бы округлений не было, то мы должны были бы получить F = ABS(X), однако результат не таков. Поэкспериментируйте с этой функцией для нескольких значений X, причем возьмите как $|X| < 1$, так и $|X| > 1$. Объясните ваши результаты. Установите режим работы транслятора, при котором в случае исчезновения порядка число заменяется нулем.

Глава 3
Системы линейных уравнений

3.1. Введение

Одна из задач, наиболее часто встречающихся в научных вычислениях,– решение системы линейных уравнений; при этом обычно число уравнений равно числу неизвестных. Такую систему можно записать в виде

$$Ax = b,$$

где A – заданная квадратная матрица порядка n, b – заданный вектор-столбец с n компонентами и x – неизвестный вектор-столбец с n компонентами.

К линейным уравнениям непосредственно приводят многие приложения; такова, например, рассматриваемая в конце главы задача расчета моста. Линейные уравнения могут возникать и опосредованно, как шаг в решении более сложной проблемы. Системы нелинейных уравнений, образцом которых могут служить системы, сопутствующие проектированию интегральных схем, решают, используя последовательность линейных приближений, что порождает последовательность линейных систем; эта тема обсуждается в гл. 7. При решении краевых задач для дифференциальных уравнений часто ограничиваются поиском решения только в конечном множестве точек, что во многих важных случаях ведет к системам линейных уравнений. Пример такого приложения дан в § 9.2.

Задачи аппроксимации данных также могут порождать системы линейных уравнений. Рассмотрим в этой связи пример 3.1.

Пример 3.1. Аппроксимация данных с помощью линейных уравнений. Пусть нужно аппроксимировать точки

$$(1,0), \quad (2,-1), \quad (3,2)$$

квадратичным полиномом. Искомый полином

$$p(x) = a + bx + cx^2$$

должен удовлетворять условиям

$$p(1) = 0, \quad p(2) = -1, \quad p(3) = 2.$$

Запишем их более подробно:

$$\begin{aligned} a + b + c &= 0, \\ a + 2b + 4c &= -1, \\ a + 3b + 9c &= 2. \end{aligned}$$

Мы получили систему трех линейных уравнений с тремя неизвестными. В матрично-векторной записи она имеет вид

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ 2 \end{bmatrix}$$

Эта система имеет решение

$$a = 5, \quad b = -7, \quad c = 2.$$

Легко убедиться, что полином $p(x) = 5 - 7x + 2x^2$ действительно удовлетворяет требованиям задачи. Этот вопрос более подробно обсуждается в гл. 4.

Для решения систем линейных уравнений предложено множество способов. Знаменитый метод, называемый формулами Крамера, выражает каждую компоненту решения отношением двух определителей. Если вы попытаетесь, пользуясь формулами Крамера, решить систему из 30 уравнений, то вам потребуется вычислить 31 определитель порядка 30. Если делать это «в лоб», то для решения линейной системы понадобится $31 \times 30! \times 29$ умножений и приблизительно такое же число сложений. На очень быстром компьютере с быстродействием 1 миллиард умножений в секунду, что находится на грани достижений современной технологии, одна только мультипликативная часть вычисления определителя займет 7 556 414 967 271 268 000 лет. Разумеется, это много больше того времени, какое разумно тратить на задачу подобного типа. (Конечно, в этом примере мы не вполне справедливы к формулам Крамера. Определители вычислялись наихудшим возможным способом. Существуют гораздо более хорошие способы вычисления определителей; однако при хорошем выборе метода линейную систему можно решить примерно за то же время, которого требует вычисление одного определителя. Все же у формул Крамера есть по крайней мере одно привлекательное свойство: в них компоненты решения вычисляются независимо друг от друга. По этой причине они могут оказаться вполне практичным способом решения некоторых специальных типов систем на параллельных компьютерах.)

Другой подход, математически привлекательный, но уязвимый в вычислительном отношении, заключается в том, что решение системы $Ax = b$ записывается в виде $x = A^{-1}b$, где A^{-1} – обратная матрица. Однако практически в любом конкретном приложении нет необходимости вычислять матрицу A^{-1} в явном виде, и мы не советуем делать это. В качестве крайнего, но поучительного примера рассмотрим систему, состоящую ровно из одного уравнения

$$7x = 21.$$

Наилучший способ решения этой «системы» – деление

$$x = 21/7 = 3.$$

Использование «обратной матрицы» привело бы к вычислению

$$x = (7^{-1})(21) = (.142857)(21) = 2.99997.$$

Второй способ требует больше арифметики – деление и умножение вместо одного деления – и дает менее точный результат. Однако лишние действия – это главная причина, по которой мы рекомендуем избегать обращения. Все сказанное справедливо и для систем со многими уравнениями. Это же остается верным и в распространенной ситуации, когда имеется несколько систем уравнений с одной и той же матрицей A, но различными правыми частями b. Поэтому мы уделим главное внимание прямому решению систем уравнений, а не вычислению обратной матрицы.

Методы, обсуждаемые в следующем параграфе, имеют пять преимуществ по сравнению с вычислением обратной матрицы: (1) они втрое дешевле; (2) в общем случае они дают более точные ответы; (3) они являются более гибкими в следующем смысле: матрица системы приводится к такой форме, что все произведения вида

$$Ab,\ A^{-1}b,\ A^{T}b,\ A^{-T}b$$

легко вычисляются для любого вектора b; (4) вообще говоря, они лучше сохраняют «структуру» матрицы A (даже если в A много нулевых элементов, в A^{-1} нулей может не быть вообще); (5) они более информативны в том отношении, что позволяют дать оценку точности вычисленного решения. Вместо формирования обратной матрицы в этих методах A разлагается в произведение матриц более простого вида.

Важно различать два типа матриц:

1. *Хранимая матрица*, т.е. матрица, все n^2 элементов которой хранятся в оперативной памяти машины. Это ограничивает значение порядка n несколькими сотнями для машин средней мощности и примерно тысячей на больших машинах.

2. *Разреженная матрица*, т.е. матрица, большинство элементов которой – нули, а ненулевые элементы могут или храниться в какой-либо специальной структуре данных, или регенерироваться по мере необходимости. Матрицы этого типа часто возникают в конечно-разностных и конечно-элементных методах решения дифференциальных уравнений с частными производными. Порядок n зачастую достигает нескольких десятков тысяч, а иногда и того больше. Приведем пример разреженной матрицы, элементы которой легко восстанавливаются:

$$\begin{pmatrix} 4 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 4 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 4 \end{pmatrix}$$

Эти два типа матриц в известной степени пересекаются. Хранимая матрица может иметь много нулевых элементов и, следовательно, быть в то же время разреженной; однако если для нулевых элементов отводится место в оперативной памяти, то разреженность матрицы не важна. Очень большая неразреженная матрица может храниться во внешней памяти, на диске или ленте, и вследствие этого требовать более изощренных приемов обработки. *Ленточная матрица* – это матрица, все ненулевые элементы которой находятся вблизи главной диагонали, точнее, $a_{ij} = 0$ для всех таких i, j, что $|i - j| > m$, где $m \ll n$. *Шириной ленты* называется число $2m + 1$; ненулевые элементы размещаются на $2m + 1$ диагоналях. Матрица, показанная выше, имеет ширину ленты 3 и называется *трехдиагональной матрицей*; для нее условие ленточности выполняется при $m = 1$.

Некоторые численные методы, применяемые для хранимых матриц, весьма отличаются от методов, приспособленных для разреженных матриц. Методы для хранимых матриц более универсальны, и им будет уделено основное внимание. Эти методы можно модифицировать таким образом, чтобы обрабатывать матрицы, находящиеся во внешней памяти, ленточные матрицы и другие типы больших или умеренно разреженных матриц.

Пусть x^* – вычисленное решение линейной системы $Ax = b$. Существуют две общеупотребительные меры погрешности в x^*: *вектор ошибки*

$$e = x - x^*$$

и *невязка*

$$r = b - Ax^* = A(x - x^*) = Ae.$$

Невязка – это количественная мера несоответствия между правыми и левыми частями уравнений системы при подстановке в них вычисленного решения. Теория матриц говорит, что при невырожденной матрице A из равенства нулю ошибки следует равенство нулю невязки и наоборот. Но, как будет видно из дальнейшего обсуждения, эти два вектора не обязаны быть «малы» одновременно. Методы, описываемые в этой главе, дают приближенные решения, хорошие в том смысле, что отвечающие им невязки «малы». Однако нельзя гарантировать наперед, что ошибка также будет мала. Таким образом, вычисленное решение «почти удовлетворяет» уравнениям, но оно может совсем не походить на подлинное решение. Для многих приложений это вполне удовлетворительно. Если матрица A почти вырождена (т. е. если малым изменением уравнений систему можно сделать вырожденной), то, даже при игнорировании ошибок округления, ничтожные возмущения коэффициентов системы могут приводить к большим изменениям решения. Поэтому для таких систем уравнений нереалистично рассчитывать на то, что x^* будет хорошим приближенным решением. Если известно, что матрица далека от вырожденности, то и невязка, и ошибка будут достаточно

малы. При решении системы $Ax = b$ оказывается возможным одновременно оценить, насколько матрица A близка к вырожденной, что позволяет при желании оценить погрешность вектора x^*.

3.2. Линейные системы с хранимыми матрицами

В этом параграфе мы обсудим решение линейной системы алгебраических уравнений

$$Ax = b$$

с хранимой $n \times n$-матрицей A и векторами b и x порядка n. Будем предполагать, что A – невырожденная матрица. Если A вырождена, то с точки зрения теории это обнаружится в процессе вычислений, хотя на практике установить вырожденность матрицы может быть нелегко. (Кстати, *вырожденной* называется матрица A, не имеющая обратной. Эквивалентные определения: детерминант матрицы A равен нулю; строки матрицы линейно зависимы; существует ненулевой вектор z, такой, что $Az = 0$. Из последнего определения видно, что если A вырождена и $Ax = b$ для некоторого вектора x, то $A(x + \alpha z) = b$ для любого числа α; таким образом, решение линейной системы не единственно. Следствием приведенных определений является также то, что в случае вырожденной матрицы A система $Ax = b$ имеет решения не для всякой правой части b.)

Почти всегда применяемый алгоритм, являющийся одним из старейших численных методов,– это метод последовательного исключения неизвестных, называемый обычно именем Гаусса. В конце 40-х годов, вскоре после изобретения электронных вычислительных машин, этот метод одним из первых был проанализирован на предмет его поведения в арифметике конечной разрядности. Результаты, полученные Джоном фон Нейманом и другими авторами, оценивались как пессимистические, отчасти вследствие технической сложности, но также потому, что их подход подчеркивал некоторые худшие аспекты метода. Гауссово исключение утратило популярность. Но примерно около 1960-го года, главным образом благодаря деятельности Дж. Уилкинсона, обнаружилось, что метод представляет собой почти идеальный алгоритм. Он дает решения системы, не худшие, чем мог бы дать любой другой мыслимый алгоритм. С этих пор гауссово исключение заняло центральное положение в численном анализе.

Современные исследования, относящиеся к гауссову исключению, вскрыли важность двух идей: необходимости выбора главного элемента и правильной интерпретации ошибок округления. Гауссово исключение и другие аспекты матричных вычислений подробно рассматривают в книге [Golub, Van Loan, 1983]. Читатель, желающий получить больше информации, чем сообщается в данной главе, должен обратиться к этой книге.

В основе гауссова исключения – идея приведения системы уравнений общего вида к системе более простой формы, которую легче решить.

В данном случае «простая» означает «треугольная». Проиллюстрируем это примером 3.2.

Пример 3.2. Решение треугольной системы уравнений. Рассмотрим систему уравнений вида

$$\begin{aligned} 10x_1 - 7x_2 + 0x_3 &= 7, \\ 2.5x_2 + 5x_3 &= 2.5, \\ 6.2x_3 &= 6.2. \end{aligned}$$

В матричной форме она запишется так:

$$\begin{bmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & 0 & 6.2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 2.5 \\ 6.2 \end{bmatrix}$$

Все ненулевые элементы матрицы находятся в ее правом верхнем «треугольнике». Эту систему легко решить. Из последнего уравнения

$$6.2x_3 = 6.2$$

находим $x_3 = 1$. Это значение подставляется во второе уравнение

$$2.5x_2 + (5)(1) = 2.5.$$

Следовательно, $x_2 = -1$. Наконец, значения x_3 и x_2 подставляются в первое уравнение:

$$10x_1 + (-7)(-1) + (0)(1) = 7.$$

Отсюда $x_1 = 0$. Правильность найденного решения легко проверить подстановкой в исходные уравнения. □

Изложенная методика решения треугольных систем уравнений называется *обратной подстановкой* (или *обратным ходом*).

В общем случае решение верхнетреугольной системы

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ & a_{22} & \cdots & a_{2n} \\ & & \ddots & \vdots \\ & & & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

можно получить с помощью формул

$$x_i = \begin{cases} b_n/a_{nn}, & \text{если } i = n; \\ (b_i - \sum_{j=i+1}^{n} a_{ij}x_j)/a_{ii}, & \text{в противном случае.} \end{cases}$$

Это эффективный алгоритм, требующий приблизительно $n^2/2$ умножений (заметим, что ненулевых элементов в матрице как раз примерно $n^2/2$). Кроме того, алгоритм можно применить к любой системе, в которой все элементы a_{ii} не равны нулю. Если некоторый элемент $a_{ii} = 0$, то матрица вырождена. (Почему?)

Обратная подстановка составляет вторую половину гауссова исключения. В первой половине, называемой *прямым ходом*, невырожденная матрица общего вида приводится к верхнетреугольной форме. Этот алгорим иллюстрируется примером 3.3.

Пример 3.3. Гауссово исключение.
Рассмотрим следующий пример порядка 3:

$$\begin{bmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 4 \\ 6 \end{bmatrix}.$$

Это запись системы из трех уравнений

$$\begin{aligned} 10x_1 - 7x_2 &= 7, \\ -3x_1 + 2x_2 + 6x_3 &= 4, \\ 5x_1 - x_2 + 5x_3 &= 6. \end{aligned}$$

На первом шаге с помощью первого уравнения x_1 исключается из других уравнений. Это достигается прибавлением первого уравнения, умноженного на 0.3, ко второму уравнению и прибавлением первого уравнения, умноженного на -0.5, к третьему уравнению (величины 0.3 и -0.5 называются *множителями*):

$$\begin{bmatrix} 10 & -7 & 0 \\ 0 & -0.1 & 6 \\ 0 & 2.5 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 6.1 \\ 2.5 \end{bmatrix}.$$

На втором шаге можно было бы использовать второе уравнение, чтобы исключить x_2 из третьего уравнения. Однако коэффициент при x_2 во втором уравнении – малое число -0.1. Поэтому последние два уравнения переставляются. В данном примере это делать не обязательно, поскольку здесь нет ошибок округления, но в общем случае такие перестановки очень важны. Получаем

$$\begin{bmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & -0.1 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 2.5 \\ 6.1 \end{bmatrix}.$$

Теперь с помощью второго уравнения можно исключить x_2 из третьего уравнения. Для этого второе уравнение, умноженное на 0.04, прибавля-

ется к третьему уравнению. (Каков был бы множитель, если бы уравнения не переставлялись?) Получается система

$$\begin{bmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & 0 & 6.2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 2.5 \\ 6.2 \end{bmatrix}.$$

Это та самая верхнетреугольная система, которая рассматривалась выше. □

В общем случае прямой ход состоит из $n-1$ шагов. На k-м шаге кратные k-го уравнения вычитаются из оставшихся уравнений, чтобы исключить k-е неизвестное. Если коэффициент при x_k «мал», то рекомендуется переставить уравнения перед исключением. Обратная подстановка заключается в решении последнего уравнения относительно x_n, затем предпоследнего уравнения относительно x_{n-1}, и т. д., пока из первого уравнения не будет вычислено x_1.

На k-м шаге гауссова исключения элементы матрицы A пересчитываются в соответствии с формулой

$$a_{ij} \leftarrow a_{ij} - (a_{ik}/a_{kk})a_{kj}.$$

Эти вычисления проводятся для $i > k$ и $j > k$. Отношение (a_{ik}/a_{kk}) называется *множителем*.

Мы уже отмечали, что обратная подстановка требует приблизительно $n^2/2$ умножений. Аналогичный подсчет можно провести для прямого хода. Он показывает, что для этого этапа необходимо выполнить примерно $n^3/3$ умножений. Для всех реалистических значений n прямой ход доминирует в общей стоимости вычислительного процесса. По аналогии с примером 2.5 главы 2 мы будем часто говорить, что гауссово исключение требует $O(n^3)$ операций.

Весь алгоритм можно компактно записать в матричных обозначениях. Он соответствует разложению матрицы A в произведение более простых матриц: $A = PLU$, где L и U – треугольные матрицы, а P – матрица перестановки, хранящая информацию о переупорядочении строк. Для разобранного выше примера

$$\begin{bmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ -0.3 & -0.04 & 1 \end{bmatrix} \times$$

$$\times \begin{bmatrix} 10.0 & -7.0 & 0.0 \\ 0 & 2.5 & 5.0 \\ 0 & 0 & 6.2 \end{bmatrix}.$$

Матрица P – это просто перестановка строк единичной матрицы; она показывает, что при прямом ходе строки 2 и 3 были переставлены. Матрица U – это финальная верхнетреугольная матрица, получаемая по

завершении прямого хода. Множители, использованные в прямом ходе, запоминаются в матрице L.

Представление $A = PLU$ называется *LU-разложением* или *треугольным разложением* матрицы A. Нужно подчеркнуть, что никакого нового алгоритма здесь введено не было. Треугольное разложение – это попросту гауссово исключение, выраженное в матричных обозначениях.

Треугольное разложение можно сформировать, не зная правую часть b. После этого решение линейной системы $Ax = b$ сводится к последовательному решению более простых линейных систем: вначале решается система $Pz = b$, что равносильно переупорядочению элементов вектора b, затем система $Ly = z$, к которой применяется «прямая подстановка», наконец, вектор x определяется из треугольной системы $Ux = y$; здесь используется обратная подстановка. Математически это можно выразить так:

$$x = U^{-1}y = U^{-1}L^{-1}z = U^{-1}L^{-1}P^{-1}b = (PLU)^{-1}b = A^{-1}b,$$

что эквивалентно решению исходной системы. Применяя эти соображения к разобранному выше примеру, получим

$$Pz = b: \quad z = \begin{bmatrix} 7 \\ 6 \\ 4 \end{bmatrix}$$

$$Ly = z: \quad y = \begin{bmatrix} 7.0 \\ 2.5 \\ 6.2 \end{bmatrix}$$

$$Ux = y: \quad x = \begin{bmatrix} 0.0 \\ -1.0 \\ 1.0 \end{bmatrix}$$

Вычислен тот же, что и раньше, вектор x.

Диагональные элементы матрицы U называются *главными элементами*; k-й главный элемент есть коэффициент при k-м неизвестном в k-м уравнении на k-м шаге исключения. В нашем примере главными элементами являются 10, 2.5 и 6.2.

Подпрограмма SGEFS, описываемая в этой главе, использует то обстоятельство, что прямой ход и обратную подстановку можно выполнять раздельно. Матрица, заданная пользователем, не сохраняется; на место ее верхнего треугольника записывается матрица U. В нижнем треугольнике хранится поддиагональная часть матрицы L. Как продемонстрировано выше, этой информации вместе с дополнительным массивом, описывающим последовательность главных элементов, достаточно, чтобы получить решение для любой правой части b.

И вычисление множителей, и обратная подстановка требуют деления на главные элементы. Поэтому алгоритм неприменим, если какой-либо из главных элементов равен нулю. Интуиция должна подсказать нам, что продолжать вычисления, в ходе которых некоторые из главных

элементов оказались близки к нулю,– не слишком удачная идея. Чтобы убедиться в этом, рассмотрим пример 3.4.

Пример 3.4. Близкие к нулю главные элементы.
Изменим пример 3.3 так, чтобы получилась система

$$\begin{bmatrix} 10 & -7 & 0 \\ -3 & 2.099 & 6 \\ 5 & -1 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 3.901 \\ 6 \end{bmatrix}.$$

Значение 2.000 элемента (2,2) матрицы заменено на 2.099, и правая часть также изменена, чтобы вектор $(0, -1, 1)^T$ был по-прежнему точным ответом. Предположим, что решение вычисляется на гипотетической машине, которая имеет десятичную 5-разрядную арифметику с плавающей точкой.

Первый шаг исключения дает

$$\begin{bmatrix} 10 & -7 & 0 \\ 0 & -1.0 \times 10^{-3} & 6 \\ 0 & 2.5 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 6.001 \\ 2.5 \end{bmatrix}.$$

Элемент (2,2) теперь очень мал в сравнении с прочими элементами матрицы. Тем не менее закончим исключение, не прибегая к перестановке. Следующий шаг требует прибавления второго уравнения, умноженного на 2.5×10^3, к третьему. Для правых частей это означает, что нужно умножить 6.001 на 2.5×10^3. Результат равен 1.50025×10^4, и он не может быть представлен точно в нашей гипотетической машинной арифметике. Он должен быть или усечен до 1.5002×10^4, или округлен до 1.5003×10^4. Полученное число прибавляется затем к 2.5. Предположим, что используется арифметика с усечением. Тогда последнее уравнение принимает вид

$$1.5005 \times 10^4 x_3 = 1.5004 \times 10^4,$$

откуда

$$x_3 = \frac{1.5004 \times 10^4}{1.5005 \times 10^4} = 0.99993.$$

Поскольку точный ответ – это $x_3 = 1$, кажется, что мы не сделали серьезных ошибок. К сожалению, однако, x_2 приходится определять из уравнения

$$-1.0 \times 10^{-3} x_2 + (6)(0.99993) = 6.001,$$

что дает

$$x_2 = \frac{1.5 \times 10^{-3}}{-1.0 \times 10^{-3}} = -1.5.$$

Наконец, x_1 определяется из первого уравнения

$$10 x_1 + (-7)(-1.5) = 7,$$

и это приводит к

$$x_1 = -0.35.$$

Вместо (0, −1, 1) мы получили (−0.35, −1.50, 0.99993). □

Где же была допущена ошибка? Здесь не было накопления ошибок округления, вызываемого выполнением тысяч арифметических операций. Матрица не близка к вырожденной. Трудности возникли в связи с выбором малого главного элемента на втором шаге исключения. Как результат этого, множитель равен 2.5×10^3, и последнее уравнение имеет коэффициенты, превосходящие в 10^3 раз по величине коэффициенты исходной задачи. Ошибки округления, малые по отношению к этим большим коэффициентам, неприемлемы по отношению к элементам исходной матрицы и действительного решения.

Предоставляем читателю проверить, что если второе и третье уравнения переставить, то больших множителей не возникнет и конечный результат будет удовлетворительным. Это оказывается верным и в общем случае: если множители не превосходят 1 по абсолютной величине, то гауссово исключение является устойчивым алгоритмом, и вычисляемое им решение, по всей вероятности, не уступает в точности решению, полученному для той же задачи любым другим алгоритмом.

Обеспечить, чтобы множители по модулю были не больше 1, можно с помощью процесса, известного как *частичный выбор главного элемента:* на k-м шаге прямого хода в качестве главного берется наибольший (по абсолютной величине) элемент в неприведенной части k-го столбца. Строка, содержащая этот элемент, переставляется с k-й строкой, чтобы перевести главный элемент в позицию (k, k). Такие же перестановки производятся с элементами правой части b. Неизвестные в векторе x не переупорядочиваются, поскольку столбцы в A не переставлялись.

Ошибки округления, возникающие в ходе вычислений, почти всегда имеют следствием некоторое отклонение вычисленного решения, которое мы будем обозначать через x^*, от теоретического решения $x = A^{-1}b$. В действительности два вектора *должны* отличаться, поскольку компоненты вектора x обычно не допускают точного представления в виде чисел с плавающей точкой.

Следующий пример иллюстрирует то обстоятельство, что невязка и вектор ошибки – две общепринятые меры погрешности вычислений – не обязаны быть малы одновременно.

Пример 3.5. Вектор ошибки и невязка.

Исследуем более подробно пример 2.3 главы 2. Рассмотрим задачу

$$\begin{bmatrix} 0.780 & 0.563 \\ 0.457 & 0.330 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.217 \\ 0.127 \end{bmatrix}.$$

Что случится, если применить гауссово исключение с частичным выбором главного элемента на гипотетическом компьютере, работающем в трехразрядной десятичной арифметике с округлением? Прежде всего

будет вычислен множитель

$$\frac{0.457}{0.780} = 0.586 \quad (\text{с тремя разрядами}).$$

Далее из второй строки вычитается первая, умноженная на 0.586; в результате получается система

$$\begin{bmatrix} 0.780 & 0.563 \\ 0 & 0.0000820 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.254 \\ -0.000162 \end{bmatrix}.$$

Наконец, выполняется обратная подстановка

$$x_2 = \frac{-0.00162}{0.0000820} = -1.98 \quad (\text{с тремя разрядами}),$$

$$x_1 = \frac{[0.217 - 0.563\,x_2]}{0.780} = 1.71 \quad (\text{с тремя разрядами}).$$

Итак, вычислено решение

$$x^* = \begin{bmatrix} 1.71 \\ -1.98 \end{bmatrix}.$$

Чтобы оценить его погрешность, не зная точного решения, вычислим (точно) невязку

$$r = b - Ax^* = \begin{bmatrix} 0.217 - [(0.780)(1.71) + (0.563)(-1.98)] \\ 0.127 - [(0.457)(1.71) + (0.330)(-1.98)] \end{bmatrix} = \begin{bmatrix} -0.00206 \\ -0.00107 \end{bmatrix}.$$

Компоненты *невязки* меньше, чем 10^{-2}. Едва ли можно было ожидать лучшего результата на трехразрядной машине. Однако легко видеть, что точным решением системы является вектор

$$x = \begin{bmatrix} 1.000 \\ -1.000 \end{bmatrix}.$$

Таким образом, *вектор ошибки* почти столь же велик, как и само решение. □

Были ли малые невязки счастливой случайностью? Прежде всего читатель уже начал понимать, что приведенный пример в высшей степени искусствен. Матрица невероятно близка к вырожденной и не является типичной для большинства задач, которые встречаются на практике. Тем не менее проследим причины малости невязки.

Если выполнить гауссово исключение с частичным выбором на машине с шестью или более разрядами, то прямой ход приведет к системе такого вида:

$$\begin{bmatrix} 0.780000 & 0.563000 \\ 0 & 0.000140 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.212000 \\ -0.000140 \end{bmatrix}.$$

Теперь обратная подстановка дает

$$x_2 = \frac{-0.000140}{0.000140} = -1.00000,$$

$$x_1 = \frac{0.217 - 0.563\,x_2}{0.780} = 1.00000,$$

что совпадает с точным ответом. На нашей трехзначной машине x_2 было получено как частное двух величин, каждая из которых имеет тот же порядок, что и ошибки округления. Следовательно, x_2 могло оказаться почти произвольным числом. И действительно, если бы мы использовали машину с девятью двоичными разрядами, то получили бы совершенно другое значение. Затем это вполне произвольное значение x_2 подставляется в первое уравнение, чтобы определить x_1. У нас есть все основания ожидать, что невязка первого уравнения будет мала: x_1 вычисляется таким образом, чтобы гарантировать это. Теперь мы подходим к тонкому, но очень важному месту. Именно можно также ожидать, что и невязка второго уравнения будет мала *как раз потому, что матрица столь близка к вырожденной.* Каждое из двух уравнений почти кратно другому, так что любая пара (x_1, x_2), которая почти удовлетворяет первому уравнению, будет в то же время почти удовлетворять второму. Если бы матрица была в точности вырожденной, то второе уравнение нам не потребовалось бы вовсе – любое решение первого автоматически удовлетворяло бы второму.

Хотя приведенный пример сконструирован искусственно и нетипичен, вывод, который мы из него сделали, таковым не является. Вероятно, это наиболее важный факт, установленный за последние 40 лет специалистами по матричным вычислениям: *гауссово исключение с частичным выбором главного элемента гарантированно дает малые невязки.*

Теперь, сформулировав это утверждение в столь сильной форме, сделаем пару уточняющих замечаний. Говоря «гарантированно», мы подразумеваем, что можно доказать точную теорему, предполагающую некоторые технические детали относительно машинной арифметики и устанавливающую определенные неравенства, которым должны удовлетворять компоненты невязки. Если арифметическое устройство работает каким-то иным образом или в данной программе имеется дефект, то, разумеется, «гарантия» отсутствует. Далее, под «малостью» мы понимаем порядок ошибок округления *по отношению* к величинам следующих трех типов: элементам исходной матрицы коэффициентов, элементам матриц на промежуточных шагах процесса исключения и компонентам вычисленного решения. Если какие-либо из этих величин «большие», то невязка не обязательно будет мала в абсолютном смысле. Мы можем выразить сказанное приближенной формулой

$$\text{Величина невязки} \approx \text{величина решения} \times \text{величина } A \times \varepsilon_{\text{маш}}.$$

Однако даже если невязка мала, это, вообще говоря, *не означает*, что мал и вектор ошибки. Связь между величиной невязки и величиной ошибки отчасти определяется характеристикой, называемой *числом обусловленности* матрицы A и обозначаемой $\mathrm{cond}(A)$. Действует приближенное равенство

$$\text{величина ошибки в решении} \approx \text{величина решения} \times \mathrm{cond}(A) \times \varepsilon_{\text{маш}}.$$

Обусловленность – это внутреннее свойство матрицы, не связанное с тем, как именно решается система уравнений. Матрицы с большими числами обусловленности дают большие ошибки при решении систем. Логарифм числа обусловленности допускает полезную интерпретацию: он приближенно равен числу значащих цифр, теряемых в решении системы $Ax = b$. Так, если $\mathrm{cond}(A) = 10^5$, а машинный эпсилон равен 10^{-8}, то лучшее, на что можно рассчитывать, – это приблизительно три верных разряда в компонентах вычисленного решения.

Что такое число обусловленности и почему оно играет столь важную роль? Мы покажем в § 6, что число $\mathrm{cond}(A)$ измеряет, насколько близка к вырожденной матрица A и, что еще важнее, насколько чувствительно решение системы $Ax = b$ к изменениям в A и b. Коэффициенты матрицы и правой части системы редко бывают известны точно. Некоторые системы возникают из эксперимента, и тогда коэффициенты подвержены ошибкам наблюдения. Коэффициенты других систем записываются формулами, что влечет за собой ошибки округления при их вычислении. Даже если систему можно точно ввести в память машины, в ходе ее решения почти неизбежно будут сделаны ошибки округления. Таким образом, знание того, как меняется решение при изменениях в A и b, оказывается полезным. Однако – и это еще более важно – можно показать, что ошибки округления в гауссовом исключении оказывают то же влияние на ответ, что и ошибки в исходных коэффициентах. Мы подробно обсудим эти вопросы в § 6, но приведенные выше приближенные формулы для величины невязки и ошибки – это главное, что вы должны запомнить, чтобы эффективно пользоваться описываемой в данной главе подпрограммой.

Изложенные соображения не вполне применимы, если решение плохо масштабировано, т. е. если компоненты вектора x сильно различаются по величине. Примером плохо масштабированного вектора может служить

$$x = (10^5,\ 10^{-6})^T.$$

Если x – приближенное решение системы линейных уравнений, то его погрешность обычно пропорциональна наибольшей компоненте вектора, в данном случае числу 10^5. Если значение x_1 имеет четыре верных десятичных разряда, то ошибка в компоненте x_2 равна приблизительно $10^5 \times 10^{-4} = 10^1$, и неразумно рассчитывать на то, что x_2 имеет хоть один верный знак. Подобную ситуацию иногда удается исправить в момент формирования системы путем изменения единиц измерения.

В некоторых подпрограммах для решения линейных систем предпринимается попытка автоматического масштабирования. К сожалению, не известны методы, гарантированно находящие удачное масштабирование; подчас масштабирование может даже увеличить погрешность вычисленного решения. По этой причине подпрограмма из данной главы не пытается масштабировать уравнения.

3.2.1. Векторные нормы

Обсуждение гауссова исключения упрощается и становится более точным, если ввести представление о нормах векторов и матриц. Это позволит сравнивать величины тех и других. Для сравнения чисел мы пользуемся функцией абсолютного значения; например

$$|7| > |5| \text{ и } |-3| > |-1.5|.$$

Норма вектора x – это число, измеряющее общую величину элементов вектора. Она обозначается символом $\|x\|$. Задать норму можно многими способами. Теоретически всякая функция, удовлетворяющая приводимым ниже четырем условиям, приемлема в качестве нормы. Большинство теорем, в формулировках которых участвуют нормы, верны независимо от того, какие именно это нормы. Поэтому их формулировки можно читать, заменяя для большего удобства символ нормы символом длины вектора. Однако нормы используются также при вычислениях, и здесь выбор конкретной нормы имеет важное значение. Наиболее употребительная векторная норма – это евклидова длина, или 2-норма

$$\|x\|_2 = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2}.$$

Она имеет то достоинство, что соответствует нашему интуитивному представлению о расстоянии. Эта норма будет широко использоваться в гл. 6. Однако применение 2-нормы к исследованию линейных уравнений ведет к избыточным вычислениям. Поэтому в данной главе норма вектора x с n компонентами определяется формулой

$$\|x\|_1 = \sum_{i=1}^{n} |x_i| .$$

Она называется 1-нормой. Иногда о ней говорят, так же как о «манхэттенском расстоянии», поскольку в городе величина $\|x - y\|_1$ может интерпретироваться как число кварталов, какое вы должны прошагать, чтобы попасть из пункта x в пункт y. Евклидова длина соответствует расстоянию по прямой линии. Третьей часто используемой нормой является max-норма, или ∞-норма

$$\|x\|_\infty = \max_i |x_i|.$$

Для различения норм мы будем пользоваться индексами:

$$\|x\|_2, \quad \|x\|_1, \quad \|x\|_\infty .$$

Отсутствие индекса означает, что речь идет об 1-норме.

Возвращаясь к определению нормы, заметим, что нормой называется любая функция, удовлетворяющая условиям

$$\|x\| > 0, \quad \text{если} \quad x \neq 0, \quad \|0\| = 0,$$
$$\|cx\| = |c| \cdot \|x\| \quad \text{для любого числа } c,$$
$$\|x + y\| \leqslant \|x\| + \|y\| .$$

Таким образом, норма обладает многими аналитическими свойствами евклидовой длины. Некоторые геометрические свойства последней утрачиваются, но они не столь важны.

3.3. Подпрограмма SGEFS

Почти в любой машинной библиотеке имеются подпрограммы решения систем линейных уравнений, основанные на том или ином варианте гауссова исключения с частичным выбором главного элемента. Детали реализации метода в разных подпрограммах сильно различаются. Эти детали могут существенно влиять на время работы конкретной подпрограммы, но если она правильно составлена, они не могут заметно отразиться на ее точности.

В настоящем параграфе мы опишем одну из таких подпрограмм – подпрограмму SGEFS, надстроенную над программами библиотеки Linpack, разработанной в Аргоннской национальной лаборатории. Linpack включает в себя программы для решения систем линейных уравнений и родственных матричных вычислений. Алгоритмы основаны на гауссовом исключении, подчас адаптированном для систем специального вида, например для ленточных или симметричных систем. Каждая программа существует в четырех вариантах для разных типов арифметики: вещественной одинарной точности, вещественной удвоенной точности; комплексной одинарной точности и комплексной удвоенной точности. Linpack имеет ядро, состоящее из базисных подпрограмм нижнего уровня (BLAS – Basic Linear Algebra Subroutines); они выполняют рутинные операции типа вычисления нормы вектора. Если Linpack устанавливается на каком-либо специальном компьютере, например на векторной или параллельной машине, то можно использовать версию BLAS, предназначенную именно для этого компьютера, и тогда программы будут выполняться с почти оптимальной эффективностью. Не имей библиотека Linpack такой модульной структуры, ее пришлось бы существенно перерабатывать для каждой марки компьютеров, что, разумеется, было бы сопряжено с большими затратами. Комплекс BLAS оказался настолько полезным, что в настоящее время готовится расширенный набор базисных подпрограмм, способный производить более мощные матричные операции.

Подпрограмма SGEFS позволяет выполнять разложение матрицы и обратную подстановку. Если нужно решить несколько систем уравнений, различающихся только правыми частями, то разложение достаточно проделать лишь один раз. SGEFS вычисляет также оценку количества верных разрядов в найденном решении, которая получается из оценки для числа обусловленности матрицы. Если матрица с вычислительной точки зрения вырождена, то выдается соответствующий код ошибки.

Число верных разрядов в X является значением выходного параметра IND. Он вычисляется по значению параметра RCOND, дающего оценку для числа, обратного к числу обусловленности. Величина 1/RCOND есть нижняя оценка фактической обусловленности, но вычисляется она таким образом, что почти всегда не более чем в n раз отличается от истинного значения; обычно различие обеих величин гораздо меньше этого. Другими словами, для почти всех матриц оценка RCOND удовлетворяет неравенствам

$$\mathrm{cond}(A)/n \leqslant 1/\mathrm{RCOND} \leqslant \mathrm{cond}(A).$$

Но и в случаях, когда $1/\mathrm{RCOND} < \mathrm{cond}(A)/n$, величина RCOND характеризует чувствительность решений для большинства правых частей.

Ошибки округления обычно препятствуют тому, чтобы SGEFS или любая другая программа гауссова исключения могла определить, вырождена или нет заданная матрица. Если в прямом ходе встретится главный элемент, в точности равный нулю, то SGEFS установит соответствующее значение кода ошибки и не будет пытаться произвести обратную подстановку. Однако появление нулевого главного элемента не обязательно означает, что матрица вырождена, а вырожденная матрица не всегда порождает нули в главных позициях. В действительности наиболее часто встречающейся причиной возникновения нулевых главных элементов являются те или иные ошибки в вызывающей программе!

Нужно осознать то обстоятельство, что при наличии частичного выбора главного элемента *любая* матрица допускает треугольное разложение. Выполнение подпрограммы SGEFS фактически убыстряется при возникновении нулевого главного элемента, поскольку оно означает, что соответствующий столбец матрицы уже находится в треугольной форме. Единственная трудность с нулевыми главными элементами состоит в том, что они делают обратную подстановку невозможной.

Чтобы прокомментировать некоторые детали подпрограммы SGEFS, мы должны напомнить способ хранения матриц в фортранных системах. Если программа содержит оператор

```
DIMENSION A(3, 5),
```

то в памяти будет зарезервировано $3 \times 5 = 15$ мест для элементов матрицы A. Они будут храниться в следующем порядке:

```
A(1, 1)  A(2, 1)  A(3, 1)  A(1, 2)  A(2, 2)  A(3, 2)  A(1, 3) ...
```

Другими словами, элементы каждого столбца хранятся подряд. Элементы одной строки отделены друг от друга числом позиций, равным первому индексу в операторе размерности. Это соглашение включено в спецификации Американского национального института стандартов для Фортрана.

Большинство диалектов Фортрана (хотя и не все) допускают *переменные размерности* массивов, являющихся параметрами подпрограмм. В основную программу можно включить описание

```
DIMENSION A(50,70),
```

но в действительности работать с $N \times N$-матрицей, где N меняется от задачи к задаче. Подпрограммы типа SGEFS требуют задания как действительного рабочего порядка N, так и величины 50, содержащейся в операторе размерности, поскольку эта величина задает приращение памяти между последовательными элементами строки. Соответствующий параметр подпрограммы SGEFS обозначается LDA, что представляет собой сокращение от "Leading Dimension of A".

Подпрограмму SGEFS можно использовать для вычисления определителей. Эта возможность вытекает из трех основных свойств определителей. Вычитание кратного одной строки из другой не меняет определителя. Перестановка двух строк изменяет знак определителя. Определитель *треугольной* матрицы равен попросту произведению ее диагональных элементов. Таким образом, с точностью до знака определитель есть произведение диагональных элементов матрицы на выходе подпрограммы.

Неприятной особенностью вычисления определителей является то, что промежуточные произведения, а зачастую и сам определитель, имеют тенденцию быть очень большими или очень малыми числами и, следовательно, легко могут повести к переполнению или исчезновению порядка. Одна из простых предупредительных мер – вычисление логарифма абсолютной величины определителя как суммы логарифмов его сомножителей. В библиотеке Linpack имеются подпрограммы для вычисления определителя и обращения матрицы; рекомендуем обратиться к ним, если определитель действительно нужен.

Полную документацию подпрограммы SGEFS можно найти в § 11. Приведенная ниже основная программа иллюстрирует использование подпрограммы SGEFS. Заметьте, что заявленная размерность LDA массива A равна 10, в то время как для действительного порядка матрицы N = 3. Наша практика показала, что неправильное задание параметра LDA является частым источником ошибок. Чтобы избежать хлопот с форматированием данных, мы задаем значения элементов матрицы и правой части посредством операторов присваивания.

Это тот же пример, который рассматривается в § 2. На выходе подпрограммы будет получена такая информация:

```
МАТРИЦА КОЭФФИЦИЕНТОВ =
          10.000000   - 7.000000   0.000000
         - 3.000000     2.000000   6.000000
           5.000000   - 1.000000   5.000000
ПРАВАЯ ЧАСТЬ =
           7.000000     4.000000   6.000000
ЧИСЛО ВЕРНЫХ РАЗРЯДОВ =
                                    6
РЕШЕНИЕ =
           0.000000   - 1.000000    1.000000
```

Вот текст программы:

```
C   ДЕМОНСТРАЦИОННАЯ ПРОГРАММА ДЛЯ
    ПОДПРОГРАММЫ SGEFS
        PARAMETER (LDA = 10)
        REAL A(LDA, LDA), B(LDA), WORK(LDA), RCOND
        INTEGER IWORK (LDA), I, J, N, ITASK, IND
C
C   СФОРМИРОВАТЬ ЗАДАЧУ
C
        N = 3
        ITASK = 1
        A(1, 1) = 10.0
        A(2, 1) = - 3.0
        A(3, 1) = 5.0
        A(1, 2) = - 7.0
        A(2, 2) = 2.0
        A(3, 2) = - 1.0
        A(1, 3) = 0.0
        A(2, 3) = 6.0
        A(3, 3) = 5.0
        B(1) = 7.0
        B(2) = 4.0
        B(3) = 6.0
C
C   ВЫДАТЬ НА ПЕЧАТЬ ИНФОРМАЦИЮ О ЗАДАЧЕ
C
        WRITE (*, *) 'МАТРИЦА КОЭФФИЦИЕНТОВ ='
        DO 10 I = 1, N
          WRITE (*, 800) (A (I, J), J = 1, N)
10  CONTINUE
        WRITE (*, *) 'ПРАВАЯ ЧАСТЬ ='
        WRITE (*, 800) (B (J), J = 1, N)
C
C   РЕШИТЬ ЛИНЕЙНУЮ СИСТЕМУ
C
```

```
      CALL SGEFS (A, LDA, N, B, ITASK, IND, WORK, IWORK,
     RCOND)
C
C   ВЫВЕСТИ РЕЗУЛЬТАТЫ НА ПЕЧАТЬ
C
      IF (IND .EQ. – 10) THEN
        WRITE (*, *) 'КОД ОШИБКИ = ', IND
      ELSE IF (IND .LT. 0) THEN
        WRITE (*, *) 'КОД ОШИБКИ =', IND
        STOP
      ELSE
        WRITE (*, *) 'ЧИСЛО ВЕРНЫХ РЯЗРЯДОВ =', IND
      END IF
C
      WRITE (*, *) 'РЕШЕНИЕ ='
      WRITE (*, 800) (B (J), J = 1, N)
C
      STOP
800 FORMAT (4X, 3F12.6)
    END
```

3.4. Историческая справка: Дж. Х. Уилкинсон

Деятельность Джеймса Х. Уилкинсона полностью изменила наше понимание машинных алгоритмов. До 50-х годов качество вычислительного алгоритма определялось в соответствии с точностью получаемых им решений; иначе говоря, для оценки алгоритма мы должны были бы задаться вопросом: «Насколько велика разность

$$x_{\text{вычисленное}} - x_{\text{точное}}?»$$

Этот вопрос кажется вполне разумным; более того, кажется, что это единственное, что разумно спрашивать об алгоритме.

До изобретения электронных компьютеров алгоритмы редко оценивались на основании ошибок округления. При работе с карандашом и бумагой точность вычислений можно было приспособить к конкретным обстоятельствам, потенциальные трудности учитывались почти инстинктивно, а общее количество операций было настолько невелико, что ошибки округления не представляли собой серьезной проблемы. Однако в автоматизированных вычислениях арифметические операции производят с фиксированным конечным числом разрядов, а количество их измеряется тысячами и миллионами. Разработчики первых компьютеров понимали это, и одновременно с вводом в действие первых ЭВМ в 40-х годах начался анализ алгоритмов на предмет их поведения в арифметике с округлениями.

Первой исследованной задачей было решение системы $Ax = b$ с помощью гауссова исключения. Среди других Алан Тьюринг в Великобри-

тании и Джон фон Нейман в Соединенных Штатах обнаружили, что в общем случае вычисляемые решения систем уравнений могут быть совсем не похожи на точные решения. Иллюстрацией может служить система 2×2 из примера 3.5. Результаты, полученные Тьюрингом и фон Нейманом, носили сложный характер, поскольку нужно было учитывать технические детали машинной арифметики; они были отчасти загадочны из-за присутствия в них странного множителя. Этим «странным множителем» было число обусловленности, роль которого в то время не вполне осознавалась. Результаты, по-видимому, означали, что для многих систем линейных уравнений ошибки округления будут доминировать в вычислениях и найденное решение не будет иметь никакой ценности.

В годы второй мировой войны Джеймс Уилкинсон занимался расчетами, связанными с баллистикой. До конца 40-х годов в Великобритании не было компьютеров, и вычисления производились на настольных арифмометрах. Чаще всего это было женским занятием, поскольку мужчины были призваны на военную службу. Неоднократно Уилкинсону приходилось подобным образом решать системы линейных уравнений, число переменных в которых иногда достигало 12. Это требовало значительно более 1000 ручных операций, поскольку каждое вычисление нуждалось в проверке. В качестве заключительной проверки полученного решения Уилкинсон вычислял невязку $b - Ax$, и она всегда оказывалась приемлемо малой.

Имея подобный опыт, Уилкинсон с подозрением отнесся к пессимистическим анализам ошибок округления в гауссовом исключении. Не то чтобы теоретические результаты были неверными. Как показывает наш пример, вычисленное решение действительно может не иметь ничего общего с точным решением. Так происходит всякий раз, когда матрица почти вырождена. Однако неправильна сама постановка вопроса. Несправедливо ожидать от алгоритма, что он сможет найти решение с хорошей точностью, если решение плохо определено исходными данными задачи, иными словами, если задача близка к некоторой другой задаче, имеющей много решений.

Уилкинсон получил математическое образование в Кембридже, в Тринити-колледже, и после войны мог вернуться туда. Однако в 1946 году он поступил на службу в Национальную физическую лабораторию. Половину своего времени он должен был посвящать вычислениям на настольном калькуляторе; в течение другой половины он помогал Алану Тьюрингу в конструировании машины ACE (Automatic Computing Engine), одного из первых британских компьютеров. Тьюринг работал уже над версией V своего проекта, выросшего из деятельности по расшифровке немецких кодов, которой он занимался в годы войны. Блестяще одаренный человек, Тьюринг быстро перешел от версии V к версиям VI, VII и VIII. Однако в его характере была эксцентричность, и с ним подчас нелегко было ладить. Он ушел из Лаборатории на работу в Кембридж до того, как машина была построена. Разработка

компьютера в Лаборатории продолжалась, в разное время под разным руководством, однако обстоятельства заставили сократить размах проекта. Упрощенная версия V и была реализована как компьютер Pilot ACE.

Уилкинсон оставался в составе разработчиков и свой первый вычислительный опыт приобрел как раз на этой машине. Чтобы проверить арифметику компьютера, он написал программу вычисления корней полинома и стал отлаживать ее на примерах, ответы в которых ему были известны. Одним из его первых тестов было вычисление корней уравнения

$$p(x) = (x - 1)(x - 2) \ldots (x - 20) = x^{20} - 210x^{19} + \ldots = 0.$$

К его удивлению, вычисленные корни были нисколько не похожи на правильные значения. Поскольку машина была экспериментальной, он предположил, что причина в программе либо в аппаратуре, однако недельные поиски не дали результата.

Единственное, что оставалось,- это исследовать саму задачу. Уилкинсон обнаружил, что корни данного полинома невероятно чувствительны к изменению его коэффициентов. Другими словами, при всей кажущейся неточности вычисленных корней они все же являются корнями полинома, близкого к исходному. Это показывает, что трудность заключена в самой задаче, и позволяет более объективно отнестись к методу.

Позже эту точку зрения Уилкинсон применил к решению системы $Ax = b$. Он рассуждал таким образом: решение, вычисленное гауссовым исключением, может быть весьма неточным, однако оно является *точным* решением *некоторой близкой системы*. Чего же еще можно желать? Почти любая задача одним только актом записи ее в конечно-разрядной арифметике компьютера обращается в некоторую близкую *машинно-хранимую* задачу. Поэтому в общем случае ни от одного алгоритма нельзя ожидать, что он будет в состоянии точно решить систему линейных уравнений. Однако хороший алгоритм должен всегда давать решение слабо возмущенной задачи. Это же можно выразить иначе: вычисленное решение должно «вести себя» примерно так же, как точное решение. В нашем случае это означает, что невязки будут малы, и вычисленное решение будет «почти решать» уравнения.

Эта точка зрения и установилась в численном анализе. За методикой, использовавшейся Уилкинсоном, закрепилось название «обратного анализа ошибок». Хорошим теперь считается тот алгоритм, который хорош в смысле обратного анализа.

3.5. Столбцово-ориентированные алгоритмы

Многие часто встречающиеся матричные операции естественней всего описываются на языке строк. Например, в гауссовом исключении кратное одной строки вычитается из другой. Реализованные на Фортране, такие операции обычно приводят к внутренним циклам, меняющим

второй индекс массивов. Вследствие этого, а также потому, что в Фортране матрицы хранятся по столбцам, возможны два потенциально неблагоприятных воздействия на эффективность программы: (1) вычисления индексов могут оказаться более дорогостоящими; (2) операционным системам, управляющим обменом между *оперативной* и *внешней* памятью в ходе вычислений, возможно, придется проделать чрезмерную работу. По этим причинам в подпрограмме SGEFS гауссово исключение реализовано несколько необычным образом: так, что все внутренние циклы меняют первый индекс. Такая реализация для некоторых типов операционных систем может оказаться значительно более эффективной. Это особенно верно в отношении некоторых суперкомпьютеров (например, машин Cray), имеющих специальные аппаратно реализованные команды для быстрого перемещения данных, находящихся в последовательно расположенных позициях памяти. Это верно также, если матрица столь велика, что не помещается в оперативной памяти компьютера и ее приходится переносить порциями во внешнюю память и обратно.

Проиллюстрируем сказанное на примере более простого алгоритма – умножения матрицы на вектор. Рассмотрим следующую задачу порядка 3:

$$\text{Вычислить } b = Ax = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} 10 \\ 11 \\ 12 \end{bmatrix} .$$

При традиционном подходе к этому вычислению вектор b строится так:

$$b = \begin{bmatrix} 1 \times 10 + 2 \times 11 + 3 \times 12 \\ 4 \times 10 + 5 \times 11 + 6 \times 12 \\ 7 \times 10 + 8 \times 11 + 9 \times 12 \end{bmatrix} = \begin{bmatrix} 68 \\ 167 \\ 266 \end{bmatrix} .$$

В этом случае матрица используется по строкам, т.е. обращения к элементам матрицы A происходят в следующем порядке: 1, 2, 3, 4, 5, 6, 7, 8, 9. Чтобы использовать A по столбцам (т.е. в порядке 1, 4, 7, 2, 5, 8, 3, 6, 9), переупорядочим вычисления:

$$b = 10 \begin{bmatrix} 1 \\ 4 \\ 7 \end{bmatrix} + 11 \begin{bmatrix} 2 \\ 5 \\ 8 \end{bmatrix} + 12 \begin{bmatrix} 3 \\ 6 \\ 9 \end{bmatrix} .$$

Получится то же самое решение. Более того, и ошибки округления будут теми же, что и прежде, так что даже на компьютере оба ответа совпадут.

В более общем случае, если мы пытаемся вычислить произведение $b = Ax$ для произвольных A и b, то традиционные формулы имеют вид

$$b_i = \sum_{j=1}^{n} a_{ij} x_j .$$

Чтобы получить столбцово-ориентированный метод, обозначим j-й

столбец матрицы A через a_j, так что

$$A = (a_1 \ a_2 \ \dots \ a_n).$$

Эквивалентной модифицированной формулой будет

$$b = \sum_{j=1}^{n} x_j a_j .$$

Преобразование строчно-ориентированной машинной программы в столбцово-ориентированную во многих случаях проделать несложно. Приведем строчно-ориентированную программу для нашего примера:

```
      DO 20 I = 1, N
         B (I) = 0.0
         DO 10 j = 1, N
            B (I) = B (I) + A (I, J) * X (J)
10       CONTINUE
20    CONTINUE
```

Соответствующая столбцово-ориентированная программа имеет вид

```
      DO 5 I = 1, N
         B (I) = 0.0
5     CONTINUE
      DO 20 J = 1, N
         DO 10 I = 1, N
            B (I) = B (I) + A (I, J) * X (J)
10       CONTINUE
20    CONTINUE
```

Единственное существенное изменение – это переупорядочение циклов DO; арифметика программ одинакова.

* 3.6. Дополнительные сведения о числах обусловленности

Чтобы уяснить себе смысл числа обусловленности, мы должны уточнить представление о «почти вырожденности». Если A – вырожденная матрица, то для некоторых b решение x не существует, тогда как для других b оно будет неединственным. Если в сказанном что-либо непонятно, нужно перечесть определение вырожденной матрицы, приведенное в начале § 2. Если матрица A почти вырождена, то можно ожидать, что малые изменения в A и b вызовут очень большие изменения в x. С другой стороны, если A – единичная матрица, то b и x – один и тот же вектор. Следовательно, если матрица A близка к единичной, то малые изменения в A и b должны повлечь за собой соответственно малые изменения в x.

На первый взгляд может показаться, что есть некоторая связь между величиной главных элементов в гауссовом исключении с частичным выбором и близостью к вырожденности, поскольку если бы арифметику

можно было выполнять точно, то все главные элементы были бы отличны от нуля тогда и только тогда, когда матрица не вырождена. До некоторой степени верно также, что если главные элементы малы, то матрица близка к вырожденной. Однако при наличии ошибок округления обратное уже неверно (это неверно и в отсутствие ошибок округления.– *Перев.*) – матрица может быть близка к вырожденной, даже если ни один из главных элементов не мал.

Чтобы получить более точную и надежную меру близости к вырожденности, нам потребуется использовать введенное в разделе 2.1 понятие векторной нормы. Умножение вектора x на матрицу A приводит к новому вектору Ax, норма которого может очень сильно отличаться от нормы вектора x. Это изменение нормы прямо связано с той чувствительностью, которую мы хотим измерять. Область возможных изменений можно задать двумя числами:

$$M = \max_x \frac{\|Ax\|}{\|x\|} \quad (\Rightarrow \|Ax\| \leqslant M\|x\|),$$

$$m = \min_x \frac{\|Ax\|}{\|x\|} \quad (\Rightarrow m\|x\| \leqslant \|Ax\|).$$

Максимум и минимум берутся по всем ненулевым векторам. Заметим, что если матрица A вырождена, то $m = 0$. Отношение M/m называется *числом обусловленности* матрицы A:

$$\operatorname{cond}(A) = \frac{\max\limits_x \dfrac{\|Ax\|}{\|x\|}}{\min\limits_x \dfrac{\|Ax\|}{\|x\|}}.$$

Рассмотрим систему уравнений

$$Ax = b$$

и другую систему, полученную изменением правой части:

$$A(x + \Delta x) = b + \Delta b.$$

Будем считать Δb ошибкой в b, а Δx – соответствующей ошибкой в x, хотя нет необходимости предполагать, что ошибки малы. Поскольку $A(\Delta x) = \Delta b$, определения M и m немедленно ведут к неравенствам

$$\|Ax\| = \|b\| \leqslant M\|x\|$$

и

$$\|A\Delta x\| = \|\Delta b\| \geqslant m\|\Delta x\|.$$

Следовательно, при $M \neq 0$

$$\frac{\|\Delta x\|}{\|x\|} \leqslant \operatorname{cond}(A)\frac{\|\Delta b\|}{\|b\|}.$$

Величина $\|\Delta b\|/\|b\|$ есть *относительное* изменение правой части, а величина $\|\Delta x\|/\|x\|$ – *относительная* ошибка, вызванная этим изменением. Использование относительных изменений имеет то преимущество, что они безразмерны, т. е. нечувствительны к общим масштабирующим множителям.

Полученное неравенство показывает, что число обусловленности выполняет роль коэффициента увеличения относительной ошибки. Изменения правой части могут повлечь за собой изменения в решении, бóльшие в $\mathrm{cond}(A)$ раз. Оказывается, что то же самое справедливо в отношении изменений в коэффициентах матрицы.

Число обусловленности является также мерой близости к вырожденности. Хотя мы не станем развивать математический аппарат, необходимый для точной формулировки этого утверждения, можно рассматривать число обусловленности как величину, обратную к относительному расстоянию от данной матрицы до множества вырожденных матриц. Таким образом, если число $\mathrm{cond}(A)$ велико, то матрица A близка к вырожденной.

Некоторые из основных свойств числа обусловленности выводятся просто. Ясно, что $M \geqslant m$ и потому

$$\mathrm{cond}(A) \geqslant 1.$$

Если P – матрица перестановки, то компоненты вектора Px отличаются от компонент вектора x лишь порядком. Отсюда следует, что $\|Px\| = \|x\|$ для всех x и, значит,

$$\mathrm{cond}(P) = 1.$$

В частности, $\mathrm{cond}(I) = 1$. Если A умножается на скаляр c, то и M, и m умножаются на модуль этого скаляра, так что

$$\mathrm{cond}(cA) = \mathrm{cond}(A).$$

Если D – диагональная матрица, то

$$\mathrm{cond}(D) = \frac{\max|d_{ii}|}{\min|d_{ii}|}.$$

Так, для матрицы

$$D = \begin{pmatrix} 1 & & & & \\ & 2 & & & \\ & & 3 & & \\ & & & 4 & \\ & & & & 5 \end{pmatrix}$$

Имеем $M = 5$ и $m = 1$, поэтому $\mathrm{cond}(D) = 5/1 = 5$.

Последние два свойства в известной мере объясняют, почему $\mathrm{cond}(A)$ является лучшей мерой близости к вырожденности, чем определитель матрицы A. В качестве крайнего примера рассмотрим диагональную матрицу порядка 100 с числом 0.1 на главной диагонали. Тогда $\det(A) = 10^{-100}$, что обычно считается малым числом. Но $\mathrm{cond}(A) = 1$,

и компоненты вектора Ax отличаются от соответствующих компонент вектора x лишь множителем 0.1. Для линейных систем уравнений такая матрица A ведет себя скорее как единичная, а не как вырожденная.

Следующий пример иллюстрирует понятие числа обусловленности.

Пример 3.6. Число обусловленности.

Рассмотрим линейную систему $Ax = b$, где

$$A = \begin{pmatrix} 9.7 & 6.6 \\ 4.1 & 2.8 \end{pmatrix}, \quad b = \begin{pmatrix} 9.7 \\ 4.1 \end{pmatrix}, \quad x = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

Ясно, что вектор $x = (1, 0)^T$ является ее решением. При этом

$$\|b\| = 13.8, \quad \|x\| = 1.$$

Для данного примера можно вычислить точное значение числа обусловленности, хотя мы не станем вдаваться в объяснения по этому поводу (обычно же удается получить лишь оценку для $\mathrm{cond}(A)$).

$$\mathrm{cond}\,(A) = 2249.4.$$

Напомним смысл этого значения: относительное изменение решения превышает относительное изменение правой части, грубо говоря, на четыре порядка. Для проверки заменим правую часть на

$$b' = \begin{pmatrix} 9.70 \\ 4.11 \end{pmatrix}.$$

Новым решением будет вектор

$$x' = \begin{pmatrix} 0.34 \\ 0.97 \end{pmatrix}.$$

Положим $\Delta b = b - b'$ и $\Delta x = x - x'$. Тогда

$$\|\Delta b\| = 0.01, \quad \|\Delta x\| = 1.63.$$

Малое возмущение b совершенно исказило x. Относительные изменения равны

$$\frac{\|\Delta b\|}{\|b\|} = 0.00072464, \quad \frac{\|\Delta x\|}{\|x\|} = 1.63.$$

Их отношение есть $1.63/0.00072464 = 2249.4$, т.е. совпадает с $\mathrm{cond}(A)$. Конечно, обычно такое совпадение не имеет места. Мы знаем наверное лишь то, что

$$\frac{\|\Delta x\|}{\|x\|} \leqslant \mathrm{cond}\,(A)\,\frac{\|\Delta b\|}{\|b\|},$$

но для данного примера b и Δb специально были подобраны так, чтобы проиллюстрировать наихудшее возможное поведение. Однако при случайном выборе b и Δb картина была бы схожей. □

Важно понимать, что в нашем примере речь шла о *точных* решениях двух слабо различающихся систем уравнений и что метод, которым были получены эти решения, здесь не важен. Пример конструировался с очень большим числом обусловленности так, чтобы эффект изменений в b был отчетливо выражен; однако подобного поведения можно ожидать от любой задачи с большим числом обусловленности. Такие задачи называют плохо обусловленными.

Предположим, что мы хотим решить систему, в которой $a_{1,1} = 0.1$, а все остальные элементы в A и b – целые числа, и $\mathrm{cond}\,(A) = 10^5$. Предположим далее, что у нас двоичный компьютер с 24 битами для дробной части числа и что мы каким-то образом умеем вычислять точное решение для системы, уже записанной в память машины. Тогда единственная ошибка будет связана с двоичным представлением числа 0.1, и тем не менее можно ожидать, что

$$\frac{\|\Delta x\|}{\|x\|} \approx \mathrm{cond}\,(A) \times 2^{-24} \approx 6 \times 10^{-3}.$$

Другими словами, простой акт записи матрицы коэффициентов в машине может вызвать изменения в третьей значащей цифре компонент правильного решения. Мы можем подытожить сказанное следующим практическим правилом: *при решении системы линейных уравнений относительная погрешность решения пропорциональна относительным погрешностям матрицы коэффициентов системы и ее правой части, причем константа пропорциональности равна числу обусловленности.*

Число обусловленности играет также фундаментальную роль в анализе ошибок округления, совершенных в ходе гауссова исключения. Предположим, что все элементы A и b точно представляются числами с плавающей точкой, и пусть x^* – вектор из чисел с плавающей точкой, который получен на выходе подпрограммы решения линейных уравнений типа подпрограммы SGEFS. Предположим также, что точной вырожденности матрицы зарегистрировано не было и что не было ни машинных нулей, ни переполнений. Тогда можно установить следующие неравенства:

$$\frac{\|b - Ax^*\|}{\|A\|\,\|x^*\|} \leqslant C\varepsilon_{\text{маш}},$$

$$\frac{\|x - x^*\|}{\|x^*\|} \leqslant C\,\mathrm{cond}\,(A)\,\varepsilon_{\text{маш}}.$$

Здесь $\varepsilon_{\text{маш}}$ – это обсуждавшийся в § 5 гл. 2 машинный эпсилон. О константе C мы скажем ниже, но обычно она не намного больше 1.

Первое неравенство говорит, что, как правило, относительная невязка будет иметь величину, сравнимую с ошибкой округления, независимо от того, насколько плохо обусловлена матрица. Это обстоятельство было проиллюстрировано примером 3.5. Второе неравенство требует, чтобы матрица A была не вырождена; в него входит точное решение x.

Это неравенство вытекает непосредственно из первого неравенства и определения числа cond (A); его смысл в том, что относительная ошибка будет мала, если мало число обусловленности, но она может быть очень велика, если матрица почти вырождена. В предельном случае, когда A вырождена, но это не было обнаружено программой, первое неравенство все же сохраняет силу, в то время как второе не имеет смысла.

Основной результат в исследовании ошибок округления в гауссовом исключении принадлежит Дж. Х. Уилкинсону. Он доказал, что вычисленное решение x^* точно удовлетворяет системе

$$(A + E)x^* = b,$$

где E – матрица, элементы которой имеют величину порядка ошибок округления в элементах матрицы A. Тем самым все ошибки округления могут быть слиты воедино и рассматриваться как единственное возмущение, внесенное в матрицу в момент ее записи в память компьютера; само же исключение осуществляется без ошибок. Поскольку запись почти любой матрицы в память сопровождается возмущениями, сравнимыми с E, то указанное свойство метода Гаусса – это лучшее, что можно сказать о каком-либо алгоритме решения линейных уравнений. В этом смысле гауссово исключение представляет собой идеальный алгоритм решения системы $Ax = b$.

Чтобы сформулировать более точные утверждения относительно константы C и матрицы возмущений E, необходимо ввести понятие матричной нормы и установить некоторые вспомогательные неравенства. Для читателей, интересующихся этими вопросами, они обсуждаются в следующем параграфе.

* 3.7. Нормы и анализ ошибок

В этом параграфе мы более тщательно исследуем алгоритм гауссова исключения, особенно с точки зрения воздействия ошибок округления на точность вычисленного решения. Как и в случае векторов, нам нужна некоторая мера величины матрицы, т. е. нам требуется *матричная норма*. Мы могли бы определить матричную норму $\|A\|$ таким же образом, как определили векторную, например формулой

$$\|A\| = \sum_{i=1}^{n} \sum_{j=1}^{n} |a_{ij}| .$$

Такая функция обладала бы всеми свойствами векторной нормы и позволяла бы нам сравнивать величины матриц. Однако, работая с линейными уравнениями, мы заинтересованы в измерении произведений типа Ax, поэтому хотелось бы, чтобы выполнялось неравенство

$$\|Ax\| \leqslant \|A\| \cdot \|x\|$$

и чтобы для *некоторого* вектора $\bar{x}$ достигалось равенство $\|A\bar{x}\| =$

$= \|A\| \cdot \|\bar{x}\|$. Чтобы добиться этого, мы возьмем другую норму. В качестве нормы мы примем определенную выше величину M:

$$\|A\| = M = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|} \quad .$$

Эта величина имеет все свойства векторной нормы, а также упомянутые выше дополнительные свойства. Обозначая через a_j столбцы матрицы A, нетрудно вывести из принятого нами конкретного определения для $\|x\|$, что

$$\|A\| = \max_j \|a_j\| \quad .$$

Если бы мы предпочли использовать евклидову длину векторов, то вычислять $\|A\|$ было бы значительно труднее.

Вернемся теперь к результату Уилкинсона о том, что вычисленное решение x^* точно удовлетворяет системе $(A + E)x^* = b$, где элементы матрицы E имеют уровень ошибок округления. Встречаются редкие ситуации, когда промежуточные матрицы, полученные в ходе гауссова исключения, содержат элементы, намного превосходящие элементы матрицы A, что приводит к сильному росту ошибок округления. Однако можно ожидать, что если число C определено равенством

$$\frac{\|E\|}{\|A\|} = C\varepsilon_{\text{mach}} \ ,$$

то оно лишь в немногих случаях будет превосходить n.

Из этого основного результата можно тотчас вывести неравенства, относящиеся к невязке и ошибке в вычисленном решении. Для невязки имеем

$$b - Ax^* = Ex^*$$

и, следовательно

$$\|b - Ax^*\| = \|Ex^*\| \leqslant \|E\| \, \|x^*\| \ .$$

В определении невязки участвует произведение Ax^*, так что уместно рассматривать относительную невязку, которая сравнивает норму вектора $b - Ax^*$ с нормами A и x^*. Из приведенных выше неравенств сразу следует, что

$$\frac{\|b - Ax^*\|}{\|A\| \, \|x^*\|} \leqslant C\varepsilon_{\text{mach}} \ .$$

Если A не вырождена, то с помощью обратной матрицы ошибку можно записать в виде

$$x - x^* = A^{-1}(b - Ax^*) \ ,$$

а тогда

$$\|x - x^*\| \leqslant \|A^{-1}\| \, \|E\| \, \|x^*\| \ .$$

Проще всего сравнивать норму ошибки с нормой вычисленного решения. Таким образом, относительная ошибка удовлетворяет неравенству

$$\frac{\|x - x^*\|}{\|x^*\|} \leqslant C\|A\| \ \|A^{-1}\| \ \varepsilon_{\text{mach}} .$$

Оказывается, что $\|A^{-1}\| = 1/m$, а потому

$$\text{cond}\,(A) = \|A\| \ \|A^{-1}\| .$$

Итак,

$$\frac{\|x - x^*\|}{\|x^*\|} \leqslant C \ \text{cond}\,(A) \ \varepsilon_{\text{mach}} .$$

Реальное вычисление cond (A) предполагает знание A^{-1}. Если a_j – столбцы матрицы A, $\tilde{a}_j$ – столбцы матрицы A^{-1}, то для векторной нормы, которую мы используем,

$$\text{cond}\,(A) = \max_j \|a_j\| \cdot \max_j \|\tilde{a}_j\| .$$

Легко вычислить $\|A\|$, однако вычисление $\|A^{-1}\|$ примерно утраивает время, нужное для гауссова исключения. К счастью, точное значение cond (A) требуется редко. Обычно бывает достаточно любой разумной оценки для него.

Описанная в § 3 подпрограмма SGEFS оценивает обусловленность матрицы, решая две вспомогательные системы линейных уравнений. При уже произведенном разложении матрицы A это требует всего лишь $O(n^3)$ арифметических операций, что недорого в сравнении со стоимостью решения исходной системы уравнений. Некоторые подробности вычисления этой оценки приведены в § 8.

* 3.8. Оценивание числа обусловленности

После того как решена система линейных уравнений, хотелось бы иметь возможность оценить, насколько точно вычисленное решение. Проведенный выше анализ показывает, что погрешность решения можно оценить с помощью числа обусловленности матрицы коэффициентов линейной системы. Напомним определение числа обусловленности:

$$\text{cond}\,(A) = \max \frac{\|Ax\|}{\|x\|} \Big/ \min \frac{\|Ax\|}{\|x\|} .$$

Математически оно эквивалентно следующей записи:

$$\text{cond}\,(A) = \|A\| \cdot \|A^{-1}\| .$$

Для удобства мы применяем 1-норму. Если a_j – j-й столбец матрицы A, то эта норма определяется формулой:

$$\|A\| = \max_j \|a_j\| .$$

Вычислять такую норму несложно, и мы используем эту формулу в нашей оценке числа обусловленности. Однако если бы мы стремились найти точное значение величины $\|A^{-1}\|$, то нам пришлось бы вычислять матрицу A^{-1} в явном виде. Подобное вычисление стоит дороже, чем само решение системы линейных уравнений, а потому нежелательно.

Вместо этого мы попытаемся оценить число $\|A^{-1}\|$ с гораздо меньшими затратами. Наша оценка будет хорошей не для всякой матрицы A, однако почти во всех случаях она правильно передает порядок числа обусловленности. Число обусловленности требуется нам главным образом для оценивания погрешности в вычисленном решении, и полученная оценка позволяет нам указать примерное число верных цифр решения, чего для большинства приложений достаточно. Стоимость оценки составляет $O(n^2)$ арифметических операций и примерно равна стоимости двух подстановок, прямой и обратной. Это гораздо меньше, чем $O(n^3)$ операций, требуемых для разложения матрицы.

Чтобы понять принцип оценивания числа $\|A^{-1}\|$, вернемся к приведенному выше определению:

$$\|A^{-1}\| = 1/\min_x \frac{\|Ax\|}{\|x\|} = \max_x \frac{\|x\|}{\|Ax\|} = \max_y \frac{\|A^{-1}y\|}{\|y\|}.$$

Мы сделали здесь замену переменных $y = Ax$. Для получения оценки мы выберем специальным образом вектор y и решим систему $Az = y$ с помощью подстановок, тогда отношение $\|z\|/\|y\| = \|A^{-1}y\|/\|y\|$ и будет нашей оценкой для $\|A^{-1}\|$.

Если выбирать вектор y случайным образом, то имеется небольшой шанс существенно недооценить величину $\|A^{-1}\|$. Чтобы уменьшить вероятность неудачи, применяются более сложные способы выбора y. Одна из возможностей состоит в том, чтобы вначале решить систему

$$A^T y = c,$$

где c – вектор с компонентами ± 1. Знаки компонент этого вектора подбираются так, чтобы по возможности увеличить y (напомним, что мы пытаемся подобрать решение для задачи *максимизации*). Этот подход иллюстрируется примером 3.7.

Пример 3.7. Оценивание числа обусловленности.
Рассмотрим пример 2×2 из § 6:

$$A = \begin{pmatrix} 9.7 & 6.6 \\ 4.1 & 2.8 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ .4227 & 1 \end{pmatrix} \begin{pmatrix} 9.7000 & 6.6000 \\ 0 & 0.0103 \end{pmatrix}.$$

Чтобы найти вектор y, решим систему $A^T y = c$ для описанного выше специального вектора c. Это делается, исходя из представления $U^T(L^T y) = c$. Первая компонента вектора c произвольна, и мы полагаем $c_1 = 1$. Из двух возможных значений компоненты c_2 выберем то, для которого вектор $L^T y$ будет больше:

$$(L^T y)_1 = c_1 / U_{11} = 1/9.7 = .1031,$$

$$(L^T y)_2 = (c_2 - U_{21}(L^T y)_1)/U_{22} = (\pm 1 - (6.6)(.1031))/\ .0103.$$

Последнее выражение будет бо́льшим, если $c_2 = -1$. При таком выборе c_2 получаем $L^T y = (.1031, -163)^T$, откуда $y = (-163, 69)^T$.

Решая теперь систему $Az = y$, находим $z = (12690, -18640)^T$. Следовательно, наша оценка равна

$$\|A^{-1}\| \approx \frac{\|z\|}{\|y\|} = \frac{|12690| + |-18640|}{|-163| + |69|} = \frac{31330}{232} = 135.04.$$

Как и выше, $\|A\| = 13.8$, поэтому оценкой для числа обусловленности будет

$$\text{cond}\,(A) \approx 13.8 \times 135.04 = 1863.6$$

Точное значение числа обусловленности равно 2249.4. Таким образом, погрешность оценки не превышает 17%, т. е. правильный порядок сохранен. □

В подпрограмме SGEFS для оценки числа обусловленности применяется более изощренная техника, чем та, что описана выше, однако принцип ее тот же самый. Вычисляемая оценка всегда будет нижней границей для реального числа обусловленности, однако есть некоторые теоретические основания ожидать, что во всех, за исключением очень редких, случаях эта оценка будет хорошей.

Остается один деликатный момент. Поскольку оценка числа обусловленности основывается на результатах гауссова исключения, в действительности мы оцениваем число обусловленности возмущенной матрицы $A + E$, а не исходной матрицы A. Эта трудность не является серьезной. Положим $\varepsilon = \|E\|/\|A\|$; для многих задач $\varepsilon \approx \varepsilon_{\text{маш}}$. Если $\text{cond}\,(A)\,\varepsilon \leqslant 0.1$, т. е., грубо говоря, пока мы получаем в решении хотя бы один верный десятичный знак, имеют силу неравенства

$$\frac{8}{9}(1 - \varepsilon) \leqslant \frac{\text{cond}\,(A + E)}{\text{cond}\,(A)} \leqslant \frac{10}{9}(1 + \varepsilon).$$

Они означают, что число обусловленности возмущенной матрицы приблизительно такое же, как и число обусловленности заданной матрицы. Следовательно, работая в арифметике с округлениями, мы все же можем оценить число обусловленности исходной точной задачи.

3.9. Некоторые дополнительные сведения

В этом параграфе затрагивается несколько более сложных вопросов и указаны ссылки на литературу, где они обсуждаются более подробно.

*3.9.1. Пересчет решений

Предположим, что была решена некоторая конкретная система линейных уравнений $Ax = b$, а затем обнаружилось, что некоторые из

элементов матрицы неверны, или же поступили новые данные. Подобная ситуация особенно типична при обработке экспериментов, где новые данные генерируются все время, а модель постоянно пересматривается, чтобы отразить новую информацию. Разумеется, при поступлении очередной порции данных можно каждый раз заново решать систему уравнений, однако это сопряжено с избыточными вычислениями. Оказывается, что можно перестроить решение за гораздо меньшее время.

Пересчет решения основывается на *формуле Шермана–Моррисона*. Предположим, что для матрицы A вычислена обратная матрица A^{-1}. Как и прежде, мы не рекомендуем вычислять A^{-1} в явном виде; здесь эта матрица используется временно, чтобы упростить рассуждения. Пусть матрица A изменена на

$$\tilde{A} = A - uv^T,$$

где u и v – векторы размерности n. Тогда матрицу $\tilde{A}^{-1}$ можно вычислить по формуле

$$\tilde{A}^{-1} = A^{-1} + \alpha\,(A^{-1}u)\;(v^T A^{-1}),$$

где $\alpha = 1/(1 - v^T A^{-1} u)$. Это будет стоить $O(n^2)$ арифметических операций по сравнению с $O(n^3)$ операций при стандартном вычислении новой обратной матрицы.

Чтобы продемонстрировать использование этой формулы, рассмотрим следующий пример.

Пример 3.8. Формула Шермана–Моррисона.
Пусть

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{bmatrix}, \quad A^{-1} = \begin{bmatrix} -0.6667 & -1.3333 & 1.0000 \\ -0.6667 & 3.6667 & -2.0000 \\ 1.0000 & -2.0000 & 1.0000 \end{bmatrix},$$

и пусть модифицированная матрица равна

$$\tilde{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 12 \end{bmatrix}, \quad u = \begin{bmatrix} 0 \\ 0 \\ -2 \end{bmatrix}, \quad v = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

Тогда

$$A^{-1}u = \begin{bmatrix} -2 \\ 4 \\ -2 \end{bmatrix}, \quad v^T A^{-1} = (1 \quad -2 \quad 1),$$

$$\alpha = 1/(1 - (-2)) = 1/3 \approx .3333,$$

и формула Шермана–Моррисона дает

$$\tilde{A}^{-1} = \begin{bmatrix} -0.6667 & -1.3333 & 1.0000 \\ -0.6667 & 3.6667 & -2.0000 \\ 1.0000 & -2.0000 & 1.0000 \end{bmatrix} +$$

$$+(0.3333)\begin{bmatrix}-2\\4\\-2\end{bmatrix}(1\quad -2\quad 1)=\begin{bmatrix}-1.3333 & 0.0000 & 0.3333\\ 0.6667 & 1.0000 & -0.6667\\ 0.3333 & -0.6667 & 0.3333\end{bmatrix}. \square$$

Формулу Шермана–Моррисона можно применить к решению линейных уравнений в ситуации, когда нужно преобразовать решение системы $Ax = b$ в решение системы $\tilde{A}\tilde{x} = b$. Предположим, что у нас имеется подпрограмма типа SGEFS, которая в состоянии решать системы как с матрицей A, так и с матрицей A^T. Тогда из формулы Шермана–Моррисона следует

$$\tilde{A}^{-1}b = A^{-1}b + \alpha A^{-1}uv^T A^{-1}b.$$

Решение модифицированной системы можно теперь получить с помощью следующего алгоритма.

1. Решить систему $Ax = b$ относительно вектора x, т.е. найти $x = A^{-1}b$.

2. Решить систему $Ay = u$ относительно вектора y, т.е. найти $y = A^{-1}u$.

3. Решить систему $A^T z = v$ относительно вектора z, т.е. найти $z^T = v^T A^{-1}$.

4. Вычислить скаляр $\alpha = 1/(1 - v^T y)$.

5. Вычислить скаляр $\beta = z^T b$.

6. Вычислить вектор $\tilde{x} = x + \alpha\beta y$. Это и есть решение системы $\tilde{A}\tilde{x} = b$.

В этом алгоритме используются только подстановки, прямые и обратные, и вычисление скалярных произведений. Общая его стоимость составляет всего $O(n^2)$ операций. При этом нежелательная процедура вычисления обратной матрицы в явном виде обходится.

Можно указать и способ пересчета треугольного разложения $A = PLU$ при модификации матрицы A. Информацию по этому поводу, а также по смежным вопросам можно почерпнуть в статье [Gill, Golub, Murray, Saunders, 1974].

3.9.2. Разреженные системы – методы исключения

Если линейная система имеет очень большое число n уравнений и переменных, то хранить в оперативной памяти полную квадратную матрицу из n^2 элементов невозможно. Обычно такие системы получаются при дискретизации дифференциальных уравнений, обыкновенных или с частными производными, или в задачах, включающих некоторую сеточную структуру.

Пример 3.9. Простая двухточечная краевая задача.

Рассмотрим решение краевой задачи

$$u'' = f(x), \quad u(0) = 1, \quad u(1) = 2$$

на интервале [0,1]. Разобьем интервал на $n + 1$ равных частей длины

$h = 1/(n+1)$ и обозначим через $u_i = u(ih)$, $i = 1, \ldots, n$, значение решения в конце i-го подынтервала. Вместо того чтобы искать функцию $u(x)$, будем искать приближенные значения решения в n дискретных точках. Заметим, что значения $u_0 = u(0)$ и $u_{n+1} = u(1)$ определены краевыми условиями.

В качестве приближения к u'' возьмем

$$u''(x) = \frac{u(x-h) - 2u(x) + u(x+h)}{h^2}.$$

Если положить $x = ih$, то

$$u''(x) \approx \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2}.$$

Подставляя это выражение в исходное дифференциальное уравнение, приходим к линейной системе

$$u_{i-1} - 2u_i + u_{i+1} = h^2 f_i, \quad i = 1, \ldots, n,$$

где $f_i = f(ih)$. Из краевых условий имеем $u_0 = 1$ и $u_{n+1} = 2$. Подставляя эти значения в первое и последнее уравнения и записывая систему в матричной форме, получаем

$$\begin{pmatrix} -2 & 1 & & & & & & & \\ 1 & -2 & 1 & & & & & & \\ & 1 & -2 & 1 & & & & & \\ & & \cdot & \cdot & \cdot & & & & \\ & & & \cdot & \cdot & \cdot & & & \\ & & & & 1 & -2 & 1 & & \\ & & & & & 1 & -2 & 1 & \\ & & & & & & 1 & -2 & 1 \end{pmatrix} \times$$

$$\begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \cdot \\ \cdot \\ u_{n-2} \\ u_{n-1} \\ u_n \end{pmatrix} = \begin{pmatrix} h^2 f_1 - 1 \\ h^2 f_2 \\ h^2 f_3 \\ \cdot \\ \cdot \\ \cdot \\ h^2 f_{n-2} \\ h^2 f_{n-1} \\ h^2 f_n - 2 \end{pmatrix}.$$

В левой части каждого уравнения не более трех ненулевых коэффициентов. □

Зачастую (как это было в рассмотренном примере) матрицы подобных задач настолько разрежены, что оперативной памяти хватает для размещения всех ненулевых элементов вместе с информацией об их расположении. Как же нужно решать соответствующую систему линейных уравнений?

Если применение гауссова исключения возможно, то оно остается очень экономичным, точным и полезным алгоритмом. Исключение возможно, если удается разместить в памяти ненулевые элементы треугольных матриц, строящихся в ходе исключения, вместе с информацией о том, где они помещены. Пусть LU – массив, нижний треугольник которого есть матрица множителей, а верхний треугольник – верхнетреугольная матрица, получаемая прямым ходом гауссова исключения. Массив LU обычно значительно более заполнен, чем A, хотя и остается все еще разреженным. Говорят, что позиции, соответствующие нулевым элементам A и ненулевым элементам LU, *заполнились* в ходе гауссова исключения.

Для некоторых матриц размер заполнения оценить легко. Примером является ленточная матрица. Договоримся пока рассматривать ленточную матрицу как заполненную, т.е. игнорировать все нули внутри ленты. Если гауссово исключение можно проводить без выбора главных элементов, что действительно возможно и безопасно для одного класса матриц, называемых *положительно определенными* [Forsythe, Moler, 1967], то заполнения нет вообще: LU имеет ту же ширину ленты, что и A. Если же выбор главных элементов необходим (например, для невырожденной матрицы общего вида), то заполнение ограничено лентой полуторной ширины по сравнению с лентой матрицы A.

Ленточную матрицу удобно хранить в прямоугольном массиве длины n и ширины $2m + 1$; более широкий ленточный массив LU также удобно хранить и обрабатывать. Отсюда заключаем: линейные системы с ленточными матрицами легко решаются методом исключения при условии, что ленточный массив LU можно разместить в оперативной памяти. Существует ряд программ для обработки ленточных матриц, например подпрограммы пакета Linpack; они являются сравнительно простыми модификациями таких программ исключения, как SGEFS.

Для разреженных матриц более общего вида применение гауссова исключения не столь тривиально. В этом случае до начала исключения алгоритм исследует расположение нулевых и ненулевых элементов матрицы. Переупорядочивая уравнения и меняя нумерацию неизвестных, часто удается уменьшить размер заполнения, связанного с LU-разложением. Поиск идеального переупорядочения, т.е. такого, которое дает минимальное заполнение, для матриц общего вида оказывается чересчур дорогостоящим, поэтому используют *эвристические* методы, обеспечивающие хорошее, хотя и не оптимальное переупорядочение. Программное обеспечение, реализующее эти идеи, можно найти в пакете Sparspak, Йельском пакете по разреженным матрицам и в библиотеке подпрограмм Харуэлла. (Харуэлл – первоначально центр атомных ис-

следований Великобритании, в настоящее время научно-исследовательский центр широкого профиля, один из крупнейших в Западной Европе; относительно всех названных программ см. книгу [Разреженные матрицы. Библиотека программ.– М.: Изд-во МГУ, 1986].–*Перев.*) Дополнительную информацию по этому поводу см. в книге [George, Liu, 1981] (а также в книгах [Златев З., Эстербю О. Прямые методы для разреженных матриц.–М.: Мир, 1987]; [Писсанецки С. Технология разреженных матриц.–М.: Мир, 1988].–*Перев.*).

Иногда полная или даже ленточная матрица слишком велика, чтобы разместить ее в оперативной памяти. В этом случае необходимо часть матрицы хранить во внешней памяти–на дисках или магнитных лентах. Если задача столь велика, то, хотя гауссово исключение и возможно, один только объем вычислений делает его очень дорогостоящим. Время выполнения арифметических операций обычно значительно больше, чем время, нужное для операций обмена частями матрицы с внешней памятью. Поэтому в целях экономии важно так организовать ход вычислений или работу операционной системы, чтобы процессор не простаивал, ожидая завершения обмена. Это можно сделать различными способами. Однако готовых универсальных программ не существует, хотя на большинстве мощных вычислительных установок есть опыт решения таких больших систем.

Огромные усилия системных программистов были сосредоточены на том, чтобы создать у пользователя иллюзию, будто он располагает очень большой (так называемой *виртуальной*) памятью для своих данных, хотя в действительности эти данные разбиты на *страницы* или *сегменты*, которыми постоянно идет обмен с внешней памятью. Наличие виртуальной памяти освобождает программиста от забот, связанных с обменом данными. Правда, цена этой свободы может для некоторых стратегий разбиения на страницы оказаться очень высокой: если программа вынуждена ждать, пока механизм замещения отыскивает каждую очередную строку матрицы, то время ее работы может вырасти в недопустимой степени. Однако виртуальная память, как правило, сочетается с режимом мультипрограммирования, и во время поиска очередной страницы процессор обычно принимает к исполнению другую программу. Во многих операционных системах время прерывания не записывают на счет пользователя даже в том случае, когда другой программы нет. Следовательно, «стоимость» выполнения матричной программы остается приблизительно той же, был или нет обмен страницами. Однако в любом случае обмен увеличивает полное время счета.

* 3.9.3. Разреженные матрицы–итерационные методы

Существует обширный класс систем линейных уравнений, для которых элементы матрицы A задаются какой-нибудь простой формулой и, следовательно, могут вычисляться по мере необходимости. Так было

в примере 3.9. В подобных случаях элементы можно вовсе не хранить, а генерировать их, когда они понадобятся. Кроме того, часто порядки n столь велики, что было бы невозможно хранить заполненный массив LU.

Желательно решать такие линейные системы $Ax = b$ методами, которые вообще не меняют матрицу A и требуют хранения лишь нескольких векторов размерности n. (Заметим, что вектор b обычно нужно хранить, так же как и вектор x.)

Методы, отвечающие этим требованиям, существуют и называются *итерационными*. В них начинают с какого-нибудь пробного приближения $x^{(0)}$ и выполняют некоторый процесс, использующий A, b и $x^{(0)}$ и приводящий к новому вектору $x^{(1)}$. Затем процесс повторяют. На k-м шаге итерационного процесса по A, b и $x^{(k-1)}$ получают $x^{(k)}$. При соответствующих предположениях векторы $x^{(k)}$ сходятся к пределу при $k \to \infty$. Существует множество подобных итерационных процессов. Наиболее удачные из них обязаны своим успехом тесной связи с решаемой задачей. Хотя итерационный процесс математически может быть прост, структура матрицы A обычно сложна и специальным образом связана с данной конкретной задачей. Подпрограммы итерационных методов решения линейных систем можно найти далеко не во всякой программной библиотеке. Все же такие подпрограммы существуют, например подпрограммы пакета Itpack, описанного в статье [Kincaid et al., 1982].

На заре компьютерной эры итерационные методы представляли гораздо бóльший интерес, чем гауссово исключение или так называемые «прямые» методы. Первые вычислительные машины имели очень малую оперативную память (обычно несколько десятков слов), а итерационные методы очень эффективны по памяти.

Один простой итерационный метод обсуждается в § 24 книги [Forsythe, Moler, 1967]; этот метод называют *методом Гаусса–Зейделя* или *методом последовательных замещений*. Здесь основной итерационный шаг состоит в том, что i-е уравнение решается относительно i-й компоненты x_i нового вектора x, причем для всех прочих компонент вектора берутся значения, вычисленные прежде. Можно доказать, что метод сходится для некоторых типов матриц, например для любой симметричной положительно определенной матрицы A. Однако сходимость обычно медленная.

У многих итерационных процессов численного анализа сходимость настолько медленная, что очень важной задачей является *ускорение сходимости*, например сходимости векторов $x^{(k)}$ к решению. И действительно, алгоритмы, ускоряющие сходимость последовательностей, составляют важную часть вычислительной математики. Метод последовательной верхней релаксации (SOR) представляет собой подобного рода ускорение процесса Гаусса–Зейделя. Сходимость ускоряется в такой степени, что метод SOR весьма широко используется при решении конечно-разностных уравнений, моделирующих двумерные эллиптичес-

кие краевые задачи. Программы указанных методов опять-таки нечасто можно встретить в программных библиотеках.

Имеется еще семейство итерационных методов, называемых методами *сопряженных градиентов* или *сопряженных направлений*. Хорошее описание этих алгоритмов можно найти в книге [Golub, Van Loan, 1983]. Эти методы применимы к симметричным положительно определенным матрицам и не содержат предположений относительно структуры матрицы A. В точной арифметике они сходятся за конечное число шагов, но на компьютере из-за ошибок округления должны рассматриваться как итерационные методы. Существует ряд опубликованных алгоритмов метода сопряженных градиентов.

В ряде случаев исходную систему линейных уравнений удается аппроксимировать другой системой, более простой для решения. Например, линейную систему, отвечающую уравнению Лапласа в области нерегулярной формы, можно аппроксимировать системой, соответствующей уравнению Лапласа в прямоугольнике; для последней системы разработаны специальные эффективные алгоритмы. Разумно было бы использовать при решении систем преимущества, даваемые подобными аппроксимациями. Для многих итерационных методов это возможно, и таким образом достигается резкое повышение эффективности. Указанный прием называется *предобусловливанием*. Его обсуждение применительно к методу сопряженных градиентов можно найти в статье [Concus, Golub, O'Leary, 1976].

3.10. Задачи

3.1. С помощью подпрограммы SGEFS найдите решение следующей системы порядка 3:

$$\begin{bmatrix} 1.00 & 0.80 & 0.64 \\ 1.00 & 0.90 & 0.81 \\ 1.00 & 1.10 & 1.21 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \mathrm{erf}(0.80) \\ \mathrm{erf}(0.90) \\ \mathrm{erf}(1.10) \end{bmatrix}.$$

Определение функции erf см. в задаче 1.1. Выведите на печать оценку погрешности решения (параметр IND) и само решение x_1, x_2, x_3. Напечатайте также сумму $x_1 + x_2 + x_3$ и сравните ее со значением erf (1.00). Почему эти два числа так близки? Если вы не можете ответить на этот вопрос, загляните в § 1.

3.2. Матрицу, обратную к матрице A, можно определить как матрицу X, столбцы x_j которой удовлетворяют условиям

$$Ax_j = e_j,$$

где e_j есть j-й столбец единичной матрицы.

(а) Напишите подпрограмму с заголовком
SUBROUTINE INVERT (A, LDA, N, X, IND, WORK, IWORK, RCOND),
входом которой является матрица порядка N, а выходом – матрица X, приближающая обратную к A, а также оценка обусловленности и ин-

формация о главных элементах. Ваша программа должна факторизовать матрицу только один раз, при первом обращении к SGEFS, а при последующих $N - 1$ обращениях нужно решать лишь треугольные системы, по две для каждого столбца матрицы X. Содержимое X не определено, если SGEFS обнаруживает вырожденность. Проверьте вашу подпрограмму на матрицах, элементы которых могут быть точно представлены как машинные числа с плавающей точкой, а обратные матрицы известны.

(б) Имеется несколько способов оценки точности результатов:

$$\|AX - I\|,$$
$$\|XA - I\|,$$
$$\|X - A^{-1}\|.$$

Можно было бы также дважды использовать подпрограмму INVERT, первый раз для обращения A и второй раз для обращения X. Результатом будет матрица Z, которая совпала бы с A, не будь округлений. Таким образом, еще одной мерой точности является величина

$$\|Z - A\|.$$

Можете ли вы вывести неравенство, включающее C, $\mathrm{cond}(A)$ и $\varepsilon_{\text{маш}}$, которое оценивает, насколько велико может быть число $\|Z - A\|$? (См. § 6.)

3.3. Пусть

$$A = \begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \\ 0.7 & 0.8 & 0.9 \end{bmatrix}, \quad b = \begin{bmatrix} 0.1 \\ 0.3 \\ 0.5 \end{bmatrix}.$$

(а) Покажите, что система линейных уравнений $Ax = b$ имеет много решений. Опишите множество возможных решений.

(б) Предположим, что система $Ax = b$ решается с помощью подпрограммы SGEFS на гипотетической машине, которая точно выполняет арифметику. Поскольку решений много, нет оснований ожидать, что будет вычислено какое-то одно, частное решение. Что же произойдет?

(в) С помощью SGEFS вычислите решение системы на каком-либо двоичном компьютере. Так как на таком компьютере некоторые элементы матрицы A не являются точными числами с плавающей точкой, то матрица, задаваемая подпрограмме SGEFS, не будет точно вырожденной. Какое решение получено? Почему? В каком смысле оно будет «хорошим» решением? В каком смысле оно будет «плохим» решением?

3.4. При интерполировании данных кубическими сплайнами возникает, как мы увидим в следующей главе, такая трехдиагональная матрица:

$$A = \begin{bmatrix} -1 & 1 & & & & & & & \\ 1 & 4 & 1 & & & & & & \\ & 1 & 4 & 1 & & & & & \\ & & . & . & . & & & & \\ & & & . & . & . & & & \\ & & & & 1 & 4 & 1 & & \\ & & & & & 1 & 4 & & 1 \\ & & & & & & 1 & -1 & \end{bmatrix} .$$

Какие специальные свойства имеет факторизованный массив, получаемый на выходе подпрограммы SGEFS? Как упростить алгоритм гауссова исключения для этого специального случая? Как бы вы стали решать линейную систему очень высокого порядка с такой матрицей? С помощью SGEFS решите линейную систему с этой матрицей и правой частью $b = (1, \ldots, 1)^T$. Как меняется значение параметра RCOND с возрастанием числа n уравнений?

3.5. Рассмотрим линейную систему $Ax = b$, где

$$A = \begin{bmatrix} 7 & 9 & 2 \\ 6 & 4 & 1 \\ 3 & 9 & 3 \end{bmatrix}, \quad b = \begin{bmatrix} 6 \\ 4 \\ 2 \end{bmatrix} .$$

(а) Решите эту линейную систему, пользуясь SGEFS.

(б) Предположим, правая часть заменена на вектор

$$b = \begin{bmatrix} 5 \\ 2 \\ 3 \end{bmatrix} .$$

С помощью SGEFS решите модифицированную систему, не факторизуя повторно матрицу A.

(в) Пусть обнаружилось, что при формировании матрицы A допущена ошибка и элемент (3,3) в действительности должен был бы быть равен 4. Используя формулу Шермана–Моррисона и подпрограмму SGEFS, найдите решение модифицированной системы, не выполняя повторно разложения матрицы A.

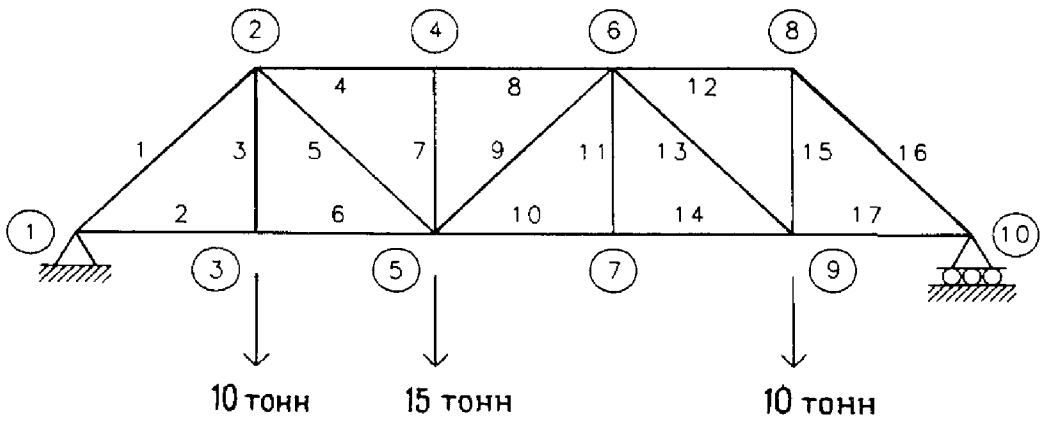


Рис. 3.1.

3.6. Требуется рассчитать усилия в плоской ферме, состоящей из 17 стержней, которая изображена на рис. 3.1. Предполагается, что стержни фермы соединены в узлах шарнирами без трения. Теорема элементарной механики утверждает, что поскольку число узлов j и число стержней m связаны соотношением $2j - 3 = m$, то ферма статически определима. Это значит, что усилия полностью определяются условиями статического равновесия в узлах. Пусть F_x обозначает горизонтальные, а F_y – вертикальные компоненты сил. Если положить $\alpha = \sin 45^\circ = \cos 45^\circ$ и допустить малые перемещения, то условиями равновесия будут:

узел 2
$$\begin{cases} \sum F_x = -\alpha f_1 + f_4 + \alpha f_5 = 0, \\ \sum F_y = -\alpha f_1 - f_3 - \alpha f_5 = 0; \end{cases}$$

узел 3
$$\begin{cases} \sum F_x = -f_2 + f_6 = 0, \\ \sum F_y = f_3 - 10 = 0; \end{cases}$$

узел 4
$$\begin{cases} \sum F_x = -f_4 + f_8 = 0, \\ \sum F_y = -f_7 = 0; \end{cases}$$

узел 5
$$\begin{cases} \sum F_x = -\alpha f_5 - f_6 + \alpha f_9 + f_{10} = 0, \\ \sum F_y = \alpha f_5 + f_7 + \alpha f_9 - 15 = 0; \end{cases}$$

узел 6
$$\begin{cases} \sum F_x = -f_8 - \alpha f_9 + f_{12} + \alpha f_{13} = 0, \\ \sum F_y = -\alpha f_9 - f_{11} - \alpha f_{13} = 0; \end{cases}$$

узел 7
$$\begin{cases} \sum F_x = -f_{10} + f_{14} = 0, \\ \sum F_y = f_{11} = 0; \end{cases}$$

узел 8
$$\begin{cases} \sum F_x = -f_{12} + \alpha f_{16} = 0, \\ \sum F_y = -f_{15} - \alpha f_{16} = 0; \end{cases}$$

узел 9
$$\begin{cases} \sum F_x = -\alpha f_{13} - f_{14} + f_{17} = 0, \\ \sum F_y = \alpha f_{13} + f_{15} - 10 = 0; \end{cases}$$

узел 10
$$\{ \sum F_x = -\alpha f_{16} - f_{17} = 0.$$

Напишите Фортран-программу, которая находила бы усилия из этой линейной системы уравнений, используя подпрограмму SGEFS. Хорошо ли обусловлена матрица линейной системы?

3.7. Компания, занимающаяся производством красок, пытается использовать имеющиеся у нее излишки зеленой краски четырех оттенков. Их смешивают с тем, чтобы получить более модную расцветку. Для получения одного галлона новой краски нужны x_1 галлонов краски 1, x_2 галлонов краски 2, и так далее. Каждая краска составлена из четырех пигментов, что приводит к следующей системе линейных уравнений:

$$\begin{bmatrix} 80 & 0 & 30 & 10 \\ 0 & 80 & 10 & 10 \\ 16 & 20 & 60 & 72 \\ 4 & 0 & 0 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 40 \\ 27 \\ 31 \\ 2 \end{bmatrix}.$$

Каждое число интерпретируется как соответствующий процент; например, краска 4 содержит 72% пигмента 3, а более модный оттенок должен содержать 27% пигмента 2. Решите эту систему с помощью подпрограммы SGEFS.

3.8. Как показывает пример 3.5, ошибка в вычисленном решении x может быть велика, несмотря на малость соответствующей невязки. Иногда бывает необходимо найти решение с большей точностью. В таких случаях можно обратиться к технике, называемой *итерационным уточнением*. Чтобы она оказалась эффективной, нужно иметь возможность часть вычислений проводить с удвоенной точностью. Рассмотрим систему линейных уравнений $Ax = b$, где

$$A = \begin{bmatrix} 21.0 & 67.0 & 88.0 & 73.0 \\ 76.0 & 63.0 & 7.0 & 20.0 \\ 0.0 & 85.0 & 56.0 & 54.0 \\ 19.3 & 43.0 & 30.2 & 29.4 \end{bmatrix} \text{ и } b = \begin{bmatrix} 141.0 \\ 109.0 \\ 218.0 \\ 93.7 \end{bmatrix}.$$

Решением этой системы является вектор $x = (1, 1, 1, 1)^T$.

(а) Используя SGEFS, решите данную линейную систему с одинарной точностью.

(б) Применяя арифметику удвоенной точности, вычислите невязку $r = b - Ax$. Не забудьте, что SGEFS не сохраняет коэффициенты исходной матрицы A.

(в) С помощью SGEFS решите линейную систему $Ae = r$ относительно вектора e. Поскольку матрица та же, что и в (а), нет необходимости факторизовать A повторно. Вектор e будет оценкой ошибки в x, поэтому улучшенное приближение к решению можно получить, полагая

$$x \leftarrow x + e.$$

Если точность все еще недостаточна, процесс можно повторить, возвращаясь к шагу (б).

3.11. Пролог: SGEFS

```
      SUBROUTINE SGEFS (A, LDA, N, V, ITASK, IND, WORK,
     IWORK, RCOND)
C***  НАЧАЛО ПРОЛОГА SGEFS
C***  ДАТА НАПИСАНИЯ 800317 (ГГММДД)
C***  ДАТА ПЕРЕСМОТРА 870916 (ГГММДД)
C***  КАТЕГОРИЯ NO. D2A1
C***  КЛЮЧЕВЫЕ СЛОВА: система линейных уравнений общего
C***  вида, линейные уравнения
C***  АВТОР: VOORHEES E. (LOS ALAMOS NATIONAL
```

```
C***  LABORATORY)
C***  НАЗНАЧЕНИЕ: SGEFS решает вещественную систему линей-
C                 ных уравнений N × N общего вида с одинар-
C                 ной точностью.
C***  ОПИСАНИЕ
C
C     Из книги D. Kahaner, C. Moler, S. Nash
C              "Numerical Methods and Software"
C              Prentice-Hall, 1988
C
C     Подпрограмма SGEFS решает с одинарной точностью систему
C     линейных уравнений N × N общего вида, при этом исполь-
C     зуются подпрограммы SGECO и SGESL из библиотеки
C     LINPACK. Таким образом, если A – вещественная матрица
C     N × N, а X и B – вещественные векторы длины N, то SGEFS
C     решает уравнение
C
C                          A * X = B.
C     Вначале матрица A факторизуется в произведение нижне- и
C     верхнетреугольной матриц L и U, для чего используется
C     частичный выбор главных элементов. С помощью треугольных
C     матриц и информации о главных элементах находится вектор
C     решения X. Вычисляется также приближенное число обуслов-
C     ленности, чтобы дать грубую оценку числа верных знаков
C     в полученном решении.
C
C     Если уравнение A * X = B нужно решать более чем для одного
C     вектора B, то повторная факторизация матрицы A не нужна,
C     и при решении второго и последующих уравнений быстрее
C     будет работать опция, предусматривающая только решение
C     треугольных систем (ITASK. EQ. 2). В этом случае по окончании
C     факторизации (ITASK = 1) пользователь не должен изменять
C     содержимое массивов A и IWORK, а также значения пара-
C     метров N и LDA. В таком случае SGEFS не изменит значение
C     параметра IND. Другие значения параметра ITASK исполь-
C     зуются при решении линейных систем с транспонированной
C     матрицей A.
C
C     Описание параметров:
C
C     A         REAL (LDA, N)
C               На входе – двумерный массив с размерами
C                 (LDA, N), содержащий матрицу коэффициен-
C               тов. На выходе – содержит верхнетреугольную
C                 матрицу U и множители, необходимые для
C                 построения матрицы L, такой, что A = L * U.
```

```
C       LDA       INTEGER
C                   Первая размерность массива A. Не должна
C                   быть меньше, чем N (в противном случае
C                   фатальная ошибка с кодом IND = – 1).
C       N         INTEGER
C                   Порядок матрицы A. Первые N элементов
C                   массива A суть элементы первого столбца
C                   одноименной матрицы. N не должно быть
C                   меньше, чем 1 (в противном случае фатальная
C                   ошибка с кодом IND = – 2).
C       V         REAL (N)
C                   На входе – одномерный массив (вектор) раз-
C                   мерности N, содержащий правую часть
C                   системы линейных уравнений A * X = B.
C                   На выходе – содержит вектор решения X.
C       ITASK     INTEGER
C                   Если ITASK = 1, то матрица A факторизуется,
C                   а затем решается линейная система.
C                   Если ITASK = 2, то решается только линейная
C                   система с использованием ранее полученной
C                   факторизации матрицы A, а также массива
C                   IWORK.
C                   Если ITASK = 3, то матрица факторизуется
C                   и решается система A'*X = B.
C                   Если ITASK = 4, то решается только система с
C                   транспонированной матрицей с использова-
C                   нием ранее полученной факторизации ма-
C                   трицы A, а также массива IWORK.
C                   Если ITASK. LT. 1 или ITASK. GT. 4, то
C                   выдается сообщение о фатальной ошибке
C                   с кодом IND = – 3.
C       IND       INTEGER
C                   GT. 0 В этом случае IND дает грубую оценку
C                   числа верных знаков в решении X.
C                   LT. 0 См. ниже значения кода ошибки и текст
C                   выдаваемых сообщений
C       WORK      REAL(N)
C                   Одномерный массив длины не меньше N.
C       IWORK     INTEGER(N)
C                   Одномерный массив длины не меньше N.
C       RCOND REAL
C                   Оценка числа 1.0/cond (A)
C
C     Сообщения об ошибках, выдаваемые на печать
C       IND = – 1   Фатальная ошибка. N больше, чем LDA.
C       IND = – 2   Фатальная ошибка. N меньше, чем 1.
```

```
C          IND = – 3   Фатальная ошибка. ITASK меньше 1 или боль-
C                      ше 4.
C          IND = – 4   Фатальная ошибка. Матрица A численно вы-
C                      рожденная. Решение не вычислено.
C          IND = – 10  Предупреждение. Вычисленное решение едва ли
C                      представляет какую-либо ценность. Оно, ско-
C                      рей всего, очень неточно. Возможно, матрица
C                      A плохо масштабирована.
C
C***  ССЫЛКИ: Подпрограмма SGEFS разработана группой C-3
C              (Los Alamos Scientific Laboratory, Los Alamos,
C              NM 87545). Подпрограммы из библиотеки
C              LINPACK, используемые в SGEFS, детально
C              описаны в книге "LINPACK Users Guide", The
C              Society for Industrial and Applied Mathematics
C              (SIAM), 1979.
C***  ВЫЗЫВАЕМЫЕ ПОДПРОГРАММЫ: R1MACH, SGECO,
C***  SGESL, XERROR
C***  КОНЕЦ ПРОЛОГА SGEFS
```

Глава 4

Интерполяция

4.1. Введение

В 1913–1923 гг., задолго до создания компьютера, сэр Эдмунд Уиттекер читал студентам старших курсов и аспирантам лекции по вычислительной математике в Эдинбургском университете. Многим из его студентов и последователей довелось сыграть важную роль в этой области. Один из них, Г. Робинсон, записывал его лекции и вместе с Уиттекером в 1924 г. опубликовал их; ныне эта книга считается первым учебником по современному численному анализу. Их объяснение понятия интерполяции сегодня столь же убедительно, как и в те времена.

«Если функция y аргумента x определена равенством $u = g(x)$, где $g(x)$ – алгебраическое выражение, содержащее только такие арифметические операции, как возведение в квадрат, деление и т. д., то, выполняя эти операции, мы можем точно найти значение y, которое соответствует любому значению x. Но если, скажем, $y = \log_{10} x$, то невозможно вычислить y, выполняя простые арифметические операции над x (во всяком случае, невозможно точно вычислить y, выполняя конечное число таких операций), и мы вынуждены прибегнуть к таблице, которая дает значения y, отвечающие нескольким выбранным значениям x. Тогда возникает вопрос, как можно найти значения функции $\log_{10} x$ для величин аргумента x, лежащих в промежутках между табулированными значениями. Ответ на этот вопрос дается теорией интерполяции, которую в ее наиболее элементарном аспекте можно назвать наукой чтения между строк математической таблицы».

Интерполяция – это часто встречающаяся операция как в повседневной жизни, так и при работе на компьютерах. Например, если стрелка спидометра нашего автомобиля находится между делениями, мы мысленно интерполируем, чтобы оценить скорость. Если у нас есть данные, полученные с большими затратами всего в нескольких точках, нам может понадобиться определить величины между этими точками. Примером являются данные переписи населения, которая проводится раз в десять лет.

Простейший случай – интерполяция функции одной переменной. Даны точки (x_i, y_i), $i = 1, \ldots, n$, а требуется найти функцию $f(x)$, которая проходит через эти точки, т. е.

$$f(x_i) = y_i, \quad i = 1, \ldots, n.$$

Говорят, что функция f *интерполирует данные*, и в этом случае она называется *интерполянтом* или *интерполирующей функцией*. Как указал Уиттекер, мы часто выполняем интерполяцию, потому что нам нужны

значения, которых нет среди (x_i, y_i), например для построения графиков или для других вычислений, в которых требуется использовать непрерывную функцию. Хотя в определении это и не требуется, интерполянт $f(x)$ обычно является функцией, которую можно вычислить в любой интересующей нас точке x. Так, $\log_{10} x$, разумеется, интерполирует таблицу логарифмов, но Уиттекер сказал бы, что это не очень удобный интерполянт, и его надо бы заменить на другой, который легче вычислять.

Со времен Уиттекера математики расширили использование термина интерполяция, и теперь он включает в себя любой процесс, определяющий функцию, которая совпадает с некоторыми «данными». Если вернуться к примеру $y = g(x) = \log_{10} x$, то можно напомнить, что $g'(x) = (\log_{10} e)/x$. Поскольку деление – простая арифметическая операция, можно расширить нашу таблицу, включив в нее не только величины x и $\log_{10} x$, но и $0.4342944819/x = d$. Тогда для каждого x_i нашими «данными» являются два числа (y_i, d_i), и нужно найти просто вычисляемую функцию $f(x)$, которая удовлетворяет условиям

$$f(x_i) = y_i \quad \text{и} \quad f'(x_i) = d_i.$$

В этом случае f называется *эрмитовым интерполянтом* [1]). Интуиция и анализ говорят нам, что поскольку эта функция отвечает не только значениям логарифма, но и его производной, она будет лучше аппроксимировать значения, находящиеся между заданными в таблице.

Вы уже знакомы даже с более общим случаем такого типа интерполяции. Разложение Тейлора для $g(x)$ представляет собой

$$g(x) = \sum_{i=0}^{n} a_i x^i + \frac{g^{(n+1)}(\xi) x^{n+1}}{n!}, \qquad a_i = \frac{g^{(i)}(0)}{i!}.$$

Конечная сумма $\sum a_i x^i$ является полиномом степени n, который интерполирует данные в том смысле, что он сам и его первые n производные согласуются со значениями функции g и ее производных в нуле, т. е. «данными» здесь являются $g^{(i)}(0)$, $i = 0, \ldots, n$. Такой полиномиальный интерполянт является превосходной аппроксимацией $g(x)$ возле $x = 0$, но отклоняется от исходной функции все больше и больше, когда x удаляется от нуля.

Интерполяцию можно также выполнять и в более чем одном измерении. Если бы нашими данными были значения температуры T газа при точно измеренных значениях давления P и объема V, то можно было бы поискать интерполянт от двух переменных $f(P, V)$ такой, что

$$T_i = f(P_i, V_i).$$

Есть и много других возможностей, но существенным всегда является

---

[1]) В оригинале используется также термин "osculatory"; "osculari" означает по-латыни «целовать».– *Прим. перев.*

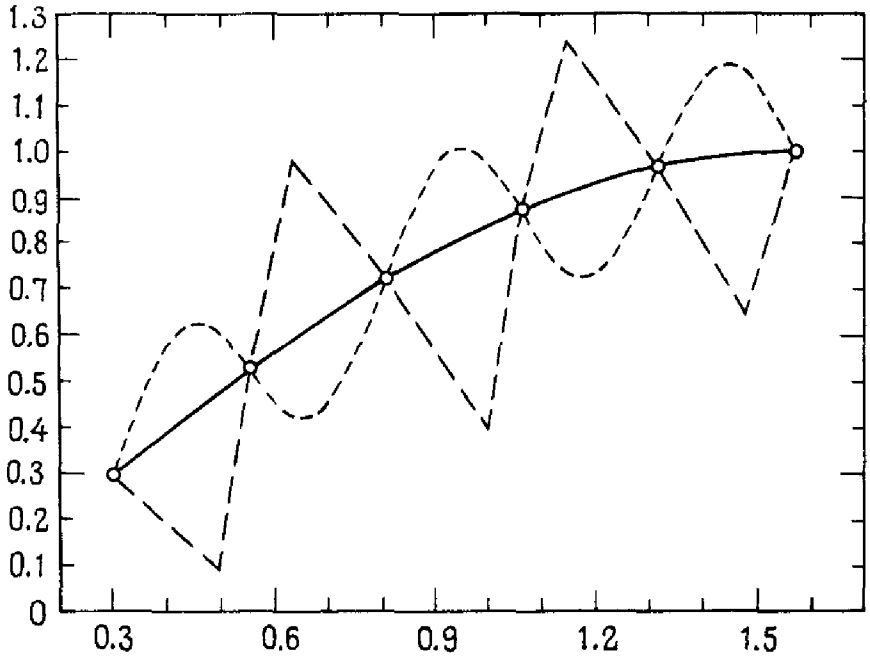


Рис. 4.1. Три различных интерполянта для одних и тех же данных.

нахождение функции, которая соответствует некоторой заданной информации.

Эта глава полностью относится к интерполяции функций одной переменной. В § 2–7 описана полиномиальная интерполяция, в § 8–14 обсуждаются кусочно-полиномиальные функции, кубические сплайны и кривые Безье. Последние не всегда являются интерполянтами, но играют важную роль в машинном проектировании и хорошо вписываются в общее обсуждение в этой главе.

На рис. 4.1 показано несколько заданных точек и три разных интерполянта. Если данные представляют некоторый физический процесс, то каждый из трех интерполянтов можно рассматривать как аппроксимацию указанного процесса. Отсюда видно следующее.

(а) Данные сами по себе не могут определить интерполянт. Для фиксированного набора данных существует бесконечно много интерполянтов.

(б) Интерполяция может быть полезной только в том случае, если данные не содержат ошибок. Экспериментальные данные, содержащие ошибки, надо аппроксимировать как-то по-другому. На рис. 4.2 показаны экспериментальные данные и функция, которая описывает эти данные лучше, чем любой интерполянт. В этой главе мы предполагаем, что (x_i, y_i) – точные значения функции, которую мы хотим аппроксимировать. Наиболее употребительным способом аппроксимации данных, содержащих ошибки, является *метод наименьших квадратов*; он рассматривается в гл. 6.

Предположим, что $x_1 < x_2 < \ldots < x_n$. Наша задача – найти интерполянт f, который дает приемлемые значения при $x \neq x_i$. Это никогда

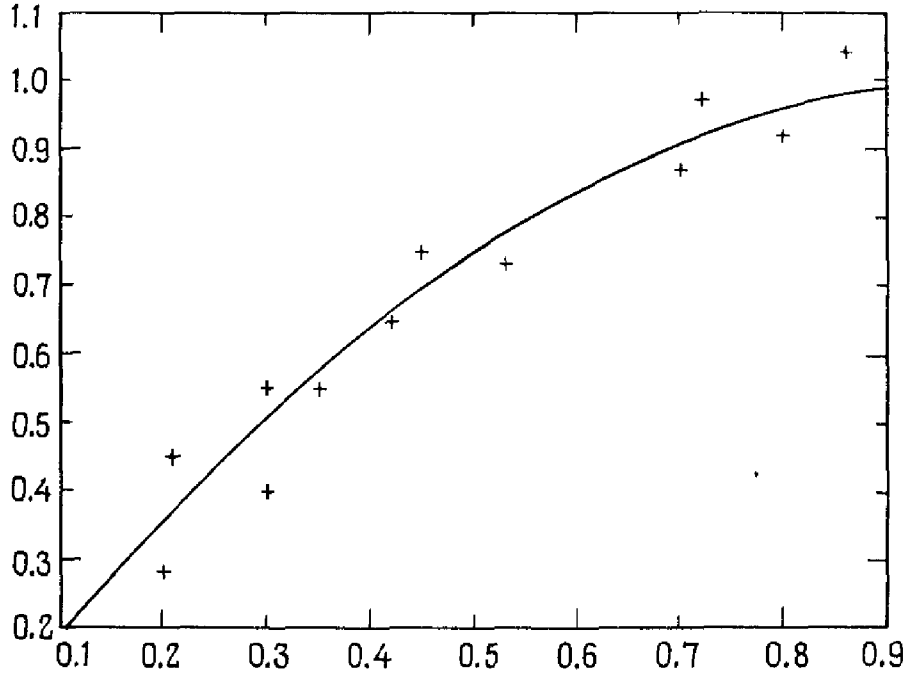


Рис. 4.2. Неточные данные аппроксимируются функцией, не являющейся интерполянтом.

нельзя сделать совершенно строго, поскольку все зависит от процесса, порождающего данные, нашего представления о приемлемости таких значений и т. д. При стандартном подходе предварительно задаются набором *базисных функций* $b_1(x)$, $b_2(x)$, ..., $b_n(x)$. Они могут быть выбраны из соображений опыта, по рекомендации или на основе математической или физической интуиции; в любом случае предполагается, что они известны. Из базисных функций строится *модель*

$$f(x) = \sum_{j=1}^{n} \alpha_j b_j(x),$$

в которой числа α_j неизвестны и определяются так, чтобы функция f была интерполянтом. Поэтому модель должна удовлетворять условиям интерполяции

$$f(x_i) = y_i, \quad \text{или} \quad \sum_{j=1}^{n} \alpha_j b_j(x_i) = y_i, \quad i = 1, \ldots, n.$$

Отсюда получаем систему n линейных уравнений для α с матрицей коэффициентов

$$B = [b_{ij}], \quad b_{ij} = b_j(x_i), \quad i, j = 1, \ldots, n.$$

Если функции b выбраны разумно, то можно решить систему относительно неизвестных α и, следовательно, найти интерполянт f. Поскольку в гл. 3 уже обсуждалось решение линейных уравнений, вас может заинтересовать, все ли сказано о задаче утверждением, что интерполяция приводит к такой системе. В данной главе рассматриваются два разных

набора базисных функций – полиномиальные и кусочно-полиномиальные. Для каждого из них матрица B оказывается настолько специальной, что это обстоятельство позволяет искать интерполянт намного эффективнее. Кроме того, само по себе знание, что мы имеем возможность вычислить неизвестные α, не дает какой-либо информации о качестве интерполянта. Поэтому главное в этой главе – разработка методов эффективного вычисления интерполянтов и представление о том, как выбрать наиболее подходящий инструмент для каждой конкретной задачи.

Прежде чем завершить этот вводный параграф, мы хотим повторить, что интерполяция – это только один из способов аппроксимации данных. Выше, в замечании (б), отмечалось, что для данных со значительными ошибками предпочтительнее подход наименьших квадратов. В других ситуациях требуются иные методы. Рассмотрим, например, такую задачу: производителю компьютеров нужна программа, которая давала бы аппроксимацию $\sin(x)$. По-видимому, можно предположить, что x принадлежит некоторому фиксированному интервалу, скажем $0 \leqslant x \leqslant \pi/2$, потому что другие значения можно получить из этих. Но в этом интервале все точки x одинаково подходят на роль аргументов. Нет причины выделять конкретный набор x_i и заставлять аппроксимирующую функцию интерполировать $\sin(x_i)$ в этих точках. Мы бы предпочли, чтобы аппроксимирующая функция никогда не давала ошибку более чем в одном или двух последних знаках для всех x из данного интервала, хотя она не обязана интерполировать данные ни в одной конкретной точке x. Такой тип аппроксимации требует более утонченной математики и здесь обсуждаться не будет. Заинтересованных читателей отсылаем к книге Дэвиса [Davis, 1963].

Наконец, читатель должен принимать во внимание, что в зависимости от задачи могут существовать две различные цели интерполяции или любого процесса аппроксимации данных.

(1) Определить неизвестные коэффициенты α_i и сделать о них определенные выводы. Методы, которые мы обсудим, решают систему уравнений для коэффициентов. Если матрица плохо обусловлена, они будут неточны.

(2) Вычислить интерполянт $f(x)$ для построения графика или в других целях. Вспомним, что, согласно гл. 3, решение системы линейных уравнений с плохо обусловленной матрицей приводит к неточным компонентам решения, но малым невязкам. Погрешность в компонентах решения означает, что коэффициенты интерполяции α_i будут иметь малую точность; малость невязок означает, что $B\alpha \approx y$. Влияние этого на величину интерполянта $\sum \alpha_j b_j(x)$ в произвольной точке x можно оценить, рассматривая его в одной из точек x_j. Но это в точности эквивалентно изучению невязок в уравнениях, а мы видели, что они будут малы. Таким образом, если главный интерес представляют значения интерполянта, то допустимы даже довольно большие числа обусловленности. Ко-

нечно, разумно всегда стремиться к возможно меньшим числам обусловленности, и, как мы увидим в § 3, всегда существует несколько разных наборов базисных функций, которые приводят к одному и тому же интерполянту. Конкретный пример см. в задаче 4.2.

4.1.1. Историческая справка: построение таблиц

Как заметил Уиттекер, чаще всего возникает такая форма интерполяции, когда мы отыскиваем данные в таблице, которая не содержит нужных нам точных значений. Методы интерполирования часто тестировались на таблицах и разрабатывались одновременно с ними. На протяжении нескольких столетий одним из наиболее важных приложений численного анализа было составление таблиц специальных функций, таких, как тригонометрические функции, например $\sin x$. «Математика вычислений» ("Mathematics of Computation"), первый современный журнал, посвященный численным методам, первоначально назывался «Математические таблицы и другие способы вычислений» ("Mathematical Tables and Other Aids to Computation"). Название было изменено в 1960 г. Эти таблицы вышли из употребления только в последние сорок лет, особенно после изобретения карманного калькулятора. Первые приложения таблицы нашли в морской навигации, где они использовались для определения широты и долготы. Ученому конца восемнадцатого века, возможно, приходилось пользоваться при вычислениях более чем сотней томов таблиц.

Создание исчерпывающего множества таблиц было монументальной задачей. После Великой французской революции французское правительство выпустило такое издание. Его необходимость была продиктована новейшей метрической системой, однако роль стимула сыграла и слава, сопутствующая такому проекту. Работа проводилась под руководством лучших математиков страны, но ее самая скучная часть была выполнена безработными парикмахерами: многие модники, когда-то носившие сложные напудренные парики, сложили свои головы на гильотине.

Чаще всего таблицы составлялись путем простого копирования: оптом копировались числа из более ранних работ. Это значит, что большинство таблиц были ненадежными, потому что не только воспроизводились первоначальные ошибки вычислений, но в каждое новое издание вносились и новые ошибки. Попытка справиться с указанной проблемой привела Чарльза Бэббиджа (1792–1871) к изобретению его «разностной машины». Он, математик с кембриджским образованием, был уверен, что механический компьютер можно построить из шестеренок и рычагов, а в качестве носителей программ для него использовать перфокарты. Некоторое время его поддерживало британское правительство, но большинство разработок финансировалось из его собственных средств и средств его спонсора, графини Ады Августы Лавлэйс,

дочери лорда Байрона. Чтобы получить бóльший доход, он и леди Лавлэйс пытались изобрести систему, которая угадывала бы победителей на скачках. Хотя его компьютер так и не был полностью построен, Бэббидж считается дедушкой современных вычислений.

Большинство учебников по численному анализу, опубликованных до середины 70-х годов нашего века, содержат несколько разделов по составлению таблиц и отысканию ошибок в них. Некоторые способы обнаружения ошибок построены очень умно на основе сопоставления разностей последовательных элементов таблицы (или разностей таких разностей, и т. д.). О них полезно почитать на тот случай, если вы когда-нибудь столкнетесь с подозрительной таблицей или захотите получить некоторые советы, помогающие отладить программы с табличным выводом результатов.

Последний бум в деле издания таблиц возник в Соединенных Штатах несколько десятилетий назад. В попытке способствовать решению проблемы занятости в годы Великой депрессии федеральное правительство сформировало специальную администрацию (Works Progress Administration, сокращенно WPA). У нее было много программ: строительство, шитье, фотография, живопись и т. д. Менее известна была программа разработки математических таблиц, начатая незадолго до второй мировой войны.

Одной из целей этой программы было дать работу безработным математикам, но, как и во времена французской программы, значительная доля работы была выполнена людьми без специального образования. Многие из работников попросту не были способны на что-то большее, чем сложение чисел. Даже отрицательные числа представляли для них загадку. Чтобы преодолеть затруднения с отрицательными числами, использовались красные и черные карандаши: красные для отрицательных чисел, черные для положительных. На стенах рабочей комнаты были развешаны огромные плакаты, на манер пропагандистских, утверждавшие: «Черный + черный = черный» или, что более спорно, «Красный × красный = черный».

Однако все эти обстоятельства не сузили целей проекта. Одной из первых была составлена таблица для e^x. Вычисления были основаны на разложении в ряд Тейлора, иногда вплоть до пятидесяти членов, и на интерполяции. Работа была разбита на простые шаги, нечто вроде программ для компьютера. Если какой-то шаг требовал более сложных операций, например выполнения умножения, то его поручали сотруднику более высокого уровня. Много вычислений делалось вручную, поскольку не хватало даже простых механических сумматоров. Много сил тратилось на проверку точности результатов: каждый расчет проводился по крайней мере дважды, некоторые элементы таблиц вычислялись двумя независимыми способами, интенсивно велась сверка. Напечатанные таблицы были, возможно, самыми точными из когда-либо составленных, они почти не содержали ошибок.

Во время войны эта программа была включена в число специальных

программ министерства обороны и использовалась в качестве медленного программируемого калькулятора. В 1945 г. она перешла к Национальному бюро стандартов, а с изобретением электронного компьютера стала во многом устаревшей. Однако составленные таблицы сохранили ценность, и многие из них вошли в книгу Милтона Абрамовица и Ирен Стиган [Abramowitz, Stegun, 1965], которую по-прежнему переиздают и используют во всем мире.

4.2. Полиномиальная интерполяция

В силу исторических и практических причин наиболее важным для интерполяции классом базисных функций является множество алгебраических полиномов. У полиномов есть очевидные преимущества: их легко вычислять (см. § 6), их легко складывать, умножать, интегрировать и дифференцировать.

Конечно, класс функций может обладать всеми указанными свойствами, но не подходить для аппроксимации функций. К счастью, у нас есть полное основание быть уверенными в том, что любую непрерывную функцию $g(x)$ можно приблизить на замкнутом интервале некоторым полиномом $p_n(x)$. Это следует из раннего результата теории аппроксимации, известного как *аппроксимационная теорема Вейерштрасса* [1]: Если g – произвольная непрерывная на конечном замкнутом интервале $[a, b]$ функция, то для любого $\varepsilon > 0$ найдется такой полином $p_n(x)$ степени $n = n(\varepsilon)$, что

$$\max_{x \in [a,b]} |g(x) - p_n(x)| < \varepsilon.$$

За подробным доказательством этого и других результатов о полиномиальной интерполяции отсылаем читателей к книге [Ralston, 1965] или [Wendroff, 1966].

Хотя некоторые доказательства теоремы Вейерштрасса являются конструктивными, обычно в результате получается полином такой высокой степени, что использовать его на практике невозможно. Более того, теорема Вейерштрасса ничего не говорит о существовании удовлетворительного интерполирующего полинома для заданного набора данных. И хотя приятно знать, что некоторый полином аппроксимирует $g(x)$ с предписанной точностью на всем отрезке $[a, b]$, нет никакой гарантии, что такой полином удастся найти с помощью практического алгоритма.

Если выбрать в качестве базисных функций неотрицательные целые

[1] Карл Теодор Вильгельм Вейерштрасс (1815–1897), наиболее значительный после Гаусса и Римана немецкий математик девятнадцатого столетия, преподавал также ботанику, географию, историю, немецкий язык, гимнастику и чистописание. Он разработал фундаментальные идеи теории функций. В числе его студентов были Фробениус, Гегенбауэр, Клейн, Ли и Минковский. Он считал геометрические доказательства дурным тоном и редко использовал чертежи для объяснений.

степени переменной x

$$b_i(x) = x^{i-1}, \quad i = 1, \ldots, n,$$

то модель примет вид

$$p_{n-1}(x) = \alpha_1 + \alpha_2 x + \ldots + \alpha_n x^{n-1}$$

с матрицей B

$$B = \begin{pmatrix} 1 & x_1 & \ldots & x_1^{n-1} \\ 1 & x_2 & \ldots & x_2^{n-1} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ 1 & x^n & \ldots & x_n^{n-1} \end{pmatrix}.$$

Пример 4.1. Линейная интерполяция.

Единственный линейный интерполянт для точек (x_1, y_1) и (x_2, y_2) задается формулой

$$p_1(x) = \alpha_1 + \alpha_2 x,$$

где α_1 и α_2 удовлетворяют уравнениям

$$\alpha_1 + \alpha_2 x_1 = y_1 \quad \text{и} \quad \alpha_1 + \alpha_2 x_2 = y_2, \quad \text{т.е. } B = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \end{pmatrix}.$$

Если $x_1 \neq x_2$, то матрица B невырожденна и можно найти α_1 и α_2. Получаем

$$p_1(x) = \frac{y_1 x_2 - y_2 x_1}{x_2 - x_1} + \frac{y_2 - y_1}{x_2 - x_1} x.$$

В качестве конкретного примера укажем линейный интерполянт для $g(x) = \sqrt{x}$, построенный по значениям этой функции в точках $x = 0$ и $x = 0.25$: $p_1(x) = 2x$. При $x = 1/9$ ошибка интерполяции составляет $|p_1(1/9) - g(1/9)| = 1/9$. □

Пример 4.2. Квадратичная интерполяция.

Найдем квадратический интерполянт по точкам $(-1, 2)$, $(1, 1)$ и $(2, 1)$. Таким интерполянтом является

$$p_2(x) = \alpha_1 + \alpha_2 x + \alpha_3 x^2.$$

Все α находятся из условия, что полином p_2 должен проходить через три указанные выше точки; это приводит к системе уравнений

$$\begin{aligned} \alpha_1 - \alpha_2 + \alpha_3 &= 2, \\ \alpha_1 + \alpha_2 + \alpha_3 &= 1, \\ \alpha_1 + 2\alpha_2 + 4\alpha_3 &= 1, \end{aligned} \qquad B = \begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \end{pmatrix} = \begin{pmatrix} 1 & -1 & 1 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{pmatrix}.$$

Можете проверить, что определитель матрицы B

$$\det(B) = (x_3 - x_2)(x_3 - x_1)(x_2 - x_1) = 6,$$

так что матрица B невырожденна. Решение системы дает интерполянт

$$f(x) = \frac{4}{3} - \frac{1}{2}x + \frac{1}{6}x^2. \quad \square$$

В этих двух примерах матрица коэффициентов B невырожденна, потому что система решается однозначно. Но в общем случае откуда нам знать, можно ли это сделать? Можно показать, что для полиномиального интерполянта матрица B имеет определитель

$$\det(B) = \prod_{1 \leqslant i < j \leqslant n} (x_j - x_i) = (x_n - x_1)(x_n - x_2)\ldots(x_n - x_{n-1})\ldots$$
$$\ldots(x_3 - x_1)(x_3 - x_2)(x_2 - x_1),$$

который не равен нулю, если все x_i различны. Заметьте, что определители матриц в примерах 4.1 и 4.2 имеют именно эту структуру. Таким образом, если никакие две абсциссы не совпадают, то система уравнений для полиномиальной интерполяции всегда имеет невырожденную матрицу коэффициентов и, следовательно, единственное решение. Поскольку математическая функция должна быть однозначной (она не может принимать два разных значения y при одном и том же x), это условие совершенно разумно.

Вывод:

Для заданных на плоскости n точек с различными абсциссами существует единственный полином степени не выше $n - 1$, который проходит через все эти точки.

4.3. Использование других базисных функций

Посмотрим, что получится, если в примере 4.1 выбрать в качестве базисных функции

$$b_1(x) = \frac{x - x_2}{x_1 - x_2}, \qquad b_2(x) = \frac{x - x_1}{x_2 - x_1}.$$

Тогда получится матрица

$$B = \begin{pmatrix} b_1(x_1) & b_2(x_1) \\ b_1(x_2) & b_2(x_2) \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$

т.е. единичная матрица $B = I$. Решением системы уравнений $B\alpha = y$ будут $\alpha_1 = y_1$ и $\alpha_2 = y_2$. Мы знаем, что существует только один многочлен первой степени, проходящий через две различные точки, а так как обе функции $b_1(x)$ и $b_2(x)$ линейные, то построенный из них интерполянт должен совпадать с $p_1(x)$ из примера 4.1. Таким образом, $p_1(x)$ можно представить либо как в примере 4.1, либо в виде

$$p_1(x) = y_1 \frac{x - x_2}{x_1 - x_2} + y_2 \frac{x - x_1}{x_2 - x_1};$$

одно выражение есть просто алгебраическое преобразование другого.

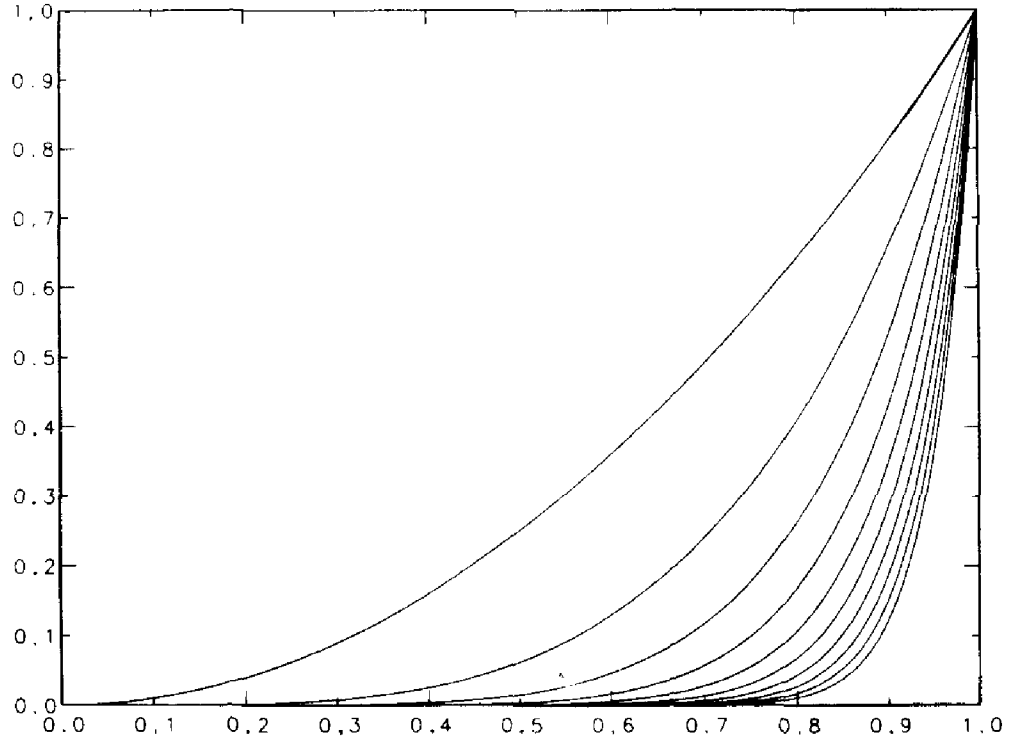


Рис. 4.3. Некоторые функции базиса из неотрицательных целых степеней на [0, 1].

Другие изменения, которые мы можем сделать, это переставить b_1 и b_2 так, что $b_1(x) = x$, $b_2(x) = 1$, или заменить b_1 и b_2 на их сумму и разность: $x + 1$ и $x - 1$. Любая замена функций b на другой набор независимых функций, которые являются линейными комбинациями исходных, не влияет на получающийся интерполянт и называется *заменой представления* или *заменой базиса*. Замена базиса может быть полезной, если она приводит к более простому виду интерполянта или дает дополнительное представление о задаче. Базис из степеней $b_j(x) = x^{j-1}$ соответствует интуиции, но часто приводит к плохо обусловленной системе уравнений. На рис. 4.3 показаны некоторые из таких степенных функций x^k, $k = 0, \ldots, 20$, на [0, 1]. На этом интервале функции x^{18}, x^{19} и x^{20} почти идентичны, следовательно, соответствующие столбцы матрицы B будут почти равными. Точное равенство столбцов означало бы вырожденность матрицы, так что можно ожидать, что матрица имеет высокое число обусловленности. Этого можно ожидать еще и потому, что элементы b_{ij} сильно различаются по величине: они могут принимать значения от 1 до числа x_i^{19}, которое может быть очень малым. Определяемый ниже базис Лагранжа [1] $b_j(x) = l_j(x)$ получается заменой базиса, которая приводит к точно такому же полиномиальному интерполянту, однако для нового базиса

[1] Жозеф Луи Лагранж (1736–1813), уроженец Турина, бóльшую часть своей зрелой жизни провел в Париже и Берлине. Он был одним из создателей вариационного исчисления (термин придуман Эйлером) и внес огромный вклад

матрица B будет единичной. Таким образом, решение системы уравнений становится тривиальной задачей.

Предположим, что у нас есть набор функций $l_1(x), \ldots, l_n(x)$, каждая из которых является полиномом степени $n-1$, а также удовлетворяет условию

$$l_j(x_i) = \begin{cases} 1, & \text{если } i = j, \\ 0 & \text{в противном случае.} \end{cases}$$

Иными словами, l_j принимает значение 1 в точке x_j и равна нулю во всех других точках x_i. (Отметим, что b_1 и b_2, указанные в начале этого раздела, обладают таким свойством.) Любая линейная комбинация функций l_j снова является полиномом степени $\leqslant (n-1)$. Рассмотрим, в частности,

$$p_{n-1}(x) = y_1 l_1(x) + y_2 l_2(x) + \ldots + y_n l_n(x).$$

Из свойств l_j следует, что

$$p_{n-1}(x_i) = y_1 l_1(x_i) + y_2 l_2(x_i) + \ldots + y_n l_n(x_i) = y_i l_i(x_i) = y_i.$$

Поэтому p_{n-1} является интерполяционным полиномом, а поскольку мы видели, что такой полином единственный, это эквивалентно решению линейной системы. Метод требует знания функций l_j, но их легко выписать:

$$l_j(x) = \frac{(x-x_1)(x-x_2)\ldots(x-x_{j-1})(x-x_{j+1})\ldots(x-x_n)}{(x_j-x_1)(x_j-x_2)\ldots(x_j-x_{j-1})(x_j-x_{j+1})\ldots(x_j-x_n)} =$$

$$= \prod_{i=1,\, i\neq j}^{n} (x-x_i) \Big/ \prod_{i=1,\, i\neq j}^{n} (x_j-x_i).$$

Пример 4.3. Лагранжева интерполяция по трем заданным точкам. Рассмотрим пример 4.2, используя лагранжево представление. Находим, что

$$p_2(x) = 2\frac{(x-1)(x-2)}{(-1-1)(-1-2)} + 1\frac{(x+1)(x-2)}{(1+1)(1-2)} + 1\frac{(x+1)(x-1)}{(2+1)(2-1)}.$$

Читателю следует проверить, что это выражение легко приводится к квадратному трехчлену из предыдущего примера. □

На рис. 4.4 показаны графики первых пяти функций базиса Лагранжа. Мы должны задать числа x_i; для иллюстрации они были выбраны как пять равномерно распределенных на [0, 1] точек. Повторяем, что лагранжевы базисы различны для разных наборов несовпадающих точек

в механику (включая проблему трех тел), теорию чисел и дифференциальные уравнения. Он пользовался столь большим уважением, что в 1793 г. сохранил должность председателя комиссии по стандартизации мер и весов (по метрической системе), тогда как Лавуазье, Лаплас, Кулон и Бриссон были изгнаны по политическим мотивам.

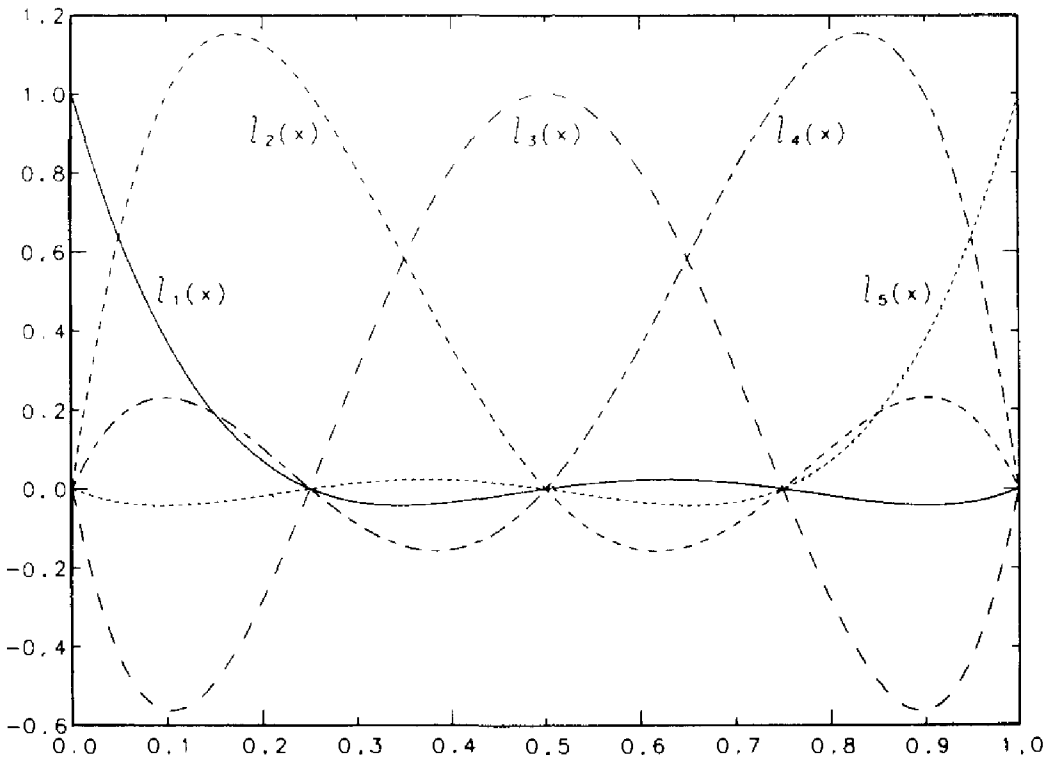


Рис. 4.4. Пять функций из базиса Лагранжа на [0, 1].

x_i, но интерполянты получаются такими же, как при выборе базиса из целых неотрицательных степеней переменной x, который не зависит от x_i. Для базиса Лагранжа число обусловленности матрицы B равно единице.

Вообще, лагранжев базис проще для записи, но более труден и менее эффективен для вычислений, чем базис из неотрицательных целых степеней. Но матрица B, связанная с базисом из степеней x, часто оказывается плохо обусловленной. Другие важные замены базиса обладают различными специальными свойствами. Например, некоторые из них предназначены для ситуаций, в которых мы получаем дополнительные данные после того, как интерполянт был построен, и хотим эффективно использовать их.

Обычно полиномиальный интерполянт по неотрицательным целым степеням хранится в виде массива его коэффициентов при этих степенях. В лагранжевом представлении тот же самый интерполянт хранится в виде пары массивов, один из которых содержит числа x_i, требуемые для вычисления $l_j(x)$, а другой – заданные значения y_j, являющиеся коэффициентами при $l_j(x)$.

Если в примере 4.1 заменить $b_2(x) = x$ на $b_2(x) = e^x$, то получим совершенно другой интерполянт: $\alpha_1 + \alpha_2 e^x$. Вы можете легко проверить, что для двух точек данных из примера 4.1 $\alpha_1 = -0.25/(\exp(0.5) - 1) = -0.3853736$, $\alpha_2 = -\alpha_1$. Это уже не прямая линия и даже не полином. В этом случае мы говорим, что изменяем модель. Часто встречаются как замена базиса, так и замена модели, и важно уметь различать их.

4.4. Хороша ли полиномиальная интерполяция?

Как можно оценить качество интерполянта? После того как вычислены коэффициенты α_j, следующий шаг – вычислить значения интерполянта в заданных точках x_i и проверить, что данные y_i воспроизводятся в пределах ошибок округления. Если это не получается, то либо матрица B плохо обусловлена, либо в нашей программе есть ошибка. Но на практике интерполянт, вероятно, будет вычисляться во многих других точках, и невозможно установить его общее поведение, зная только, что он воспроизводит входные данные. Лучший способ, не требующий большого анализа,– вычислить значения интерполянта в значительно большем числе точек и напечатать результаты или построить график.

Но все-таки в некоторых ситуациях качество интерполянта можно проанализировать. Предположим, что величина y_i представляет собой точные значения известной функции $g(x)$ в точках x_i. Пусть $p_{n-1}(x)$ – единственный полином $(n-1)$-й степени, интерполирующий функцию по этим n точкам (x_i, y_i), $i = 1, \ldots, n$. Предположим, что во всех точках x функция g имеет n непрерывных производных. Тогда можно доказать, что для любого x

$$g(x) - p_{n-1}(x) = \frac{g^{(n)}(\xi)}{n!}(x - x_1)(x - x_2)\ldots(x - x_n),$$

где ξ – некоторая неизвестная точка между x_1 и x_n. Практическую пользу из этого выражения можно извлечь только в простых случаях; иногда оно дает границы ошибок, но чаще всего подталкивает нашу интуицию и помогает подтвердить заключение, которое делается ниже.

Пример 4.4. Ошибка полиномиальной интерполяции для ln x.

Функция $g(x) = \ln(x)$ интерполируется кубическим полиномом по точкам $x = 0.4$, 0.5, 0.7 и 0.8. Мы хотим оценить ошибку интерполяции в точке $x = 0.6$. Лагранжева форма интерполянта имеет вид

$$\begin{aligned} p_3(x) = {} & \ln(0.4)\frac{(x-0.5)(x-0.7)(x-0.8)}{(0.4-0.5)(0.4-0.7)(0.4-0.8)} + \\ & + \ln(0.5)\frac{(x-0.4)(x-0.7)(x-0.8)}{(0.5-0.4)(0.5-0.7)(0.5-0.8)} + \\ & + \ln(0.7)\frac{(x-0.4)(x-0.5)(x-0.8)}{(0.7-0.4)(0.7-0.5)(0.7-0.8)} + \\ & + \ln(0.8)\frac{(x-0.4)(x-0.5)(x-0.7)}{(0.8-0.4)(0.8-0.5)(0.8-0.7)} \end{aligned}$$

и $p_3(0.6) = -0.509975$. Выражение для погрешности дает

$$\ln(0.6) - p_3(0.6) = -\frac{6}{\xi^4}\frac{1}{4!}(0.6-0.4)(0.6-0.5)(0.6-0.7)(0.6-0.8),$$

$$0.4 < \xi < 0.8.$$

Таким образом, ошибка будет меньше чем

$$\frac{6}{0.4^4}\frac{1}{24}0.0004 \approx 0.0039.$$

Фактическая величина погрешности есть $|p_3(0.6) - \ln(0.6)| = 0.000851$. Однако в некоторых задачах даже границы ошибок не дают никакой информации. Такой случай представляет собой пример 4.1, где $y_i = \sqrt{x_i}$. □

Теперь посмотрим, что получится, если интерполировать известную функцию $g(x)$ все в большем и большем числе точек на фиксированном интервале. Мы надеемся, что оценка погрешности интерполяции в других точках x улучшится. Выражение для погрешности состоит из трех разных частей; факториал и произведение разностей с увеличением n уменьшают ошибку, но порядок производной при этом растет. Для большинства функций величины производных увеличиваются быстрее, чем $n!$ В результате полиномиальные интерполянты редко сходятся к обычной непрерывной функции. Склонный к математике студент может проверить, не противоречит ли это теореме Вейерштрасса. Практический эффект выражается в том, что полиномиальный интерполянт высокой степени может вести себя очень плохо в точках x, отличных от x_i. Поэтому почти всегда используются интерполянты степени не выше 4 или 5 [1].

Полиномиальная интерполяция высокой степени – плохая идея.

Пример 4.5. Функция Рунге.

Подробный анализ опасностей, возникающих при полиномиальной интерполяции, был впервые опубликован Рунге в 1901 г. Он пытался интерполировать полиномами простую функцию

$$R(x) = \frac{1}{1 + 25x^2}$$

в точках равномерной сетки на интервале $[-1, 1]$. Он обнаружил, что при стремлении степени n интерполирующего полинома p_n к бесконечности $p_n(x)$ расходится в интервале $0.726\ldots \leqslant |x| < 1$. Это явление графически показано на рис. 4.5. Заметим, что в данном случае полиномиальная интерполяция хорошо работает в средней части указанного интервала. □

Есть несколько способов понять, почему полиномиальная интерполяция не годится для функции Рунге. Наиболее прямой из них – заметить, что производные функции $R(x)$, которые фигурируют в выражении

[1] Полиномиальная интерполяция высокой степени работает хорошо, если точками данных являются значения функции $f(x)$ вроде $\sin x$ или e^x, обладающей специальным свойством: для каждого фиксированного Z справедлива $|f^{(p)}(Z)/p!| \leqslant M$, $p = 0, 1, \ldots$.

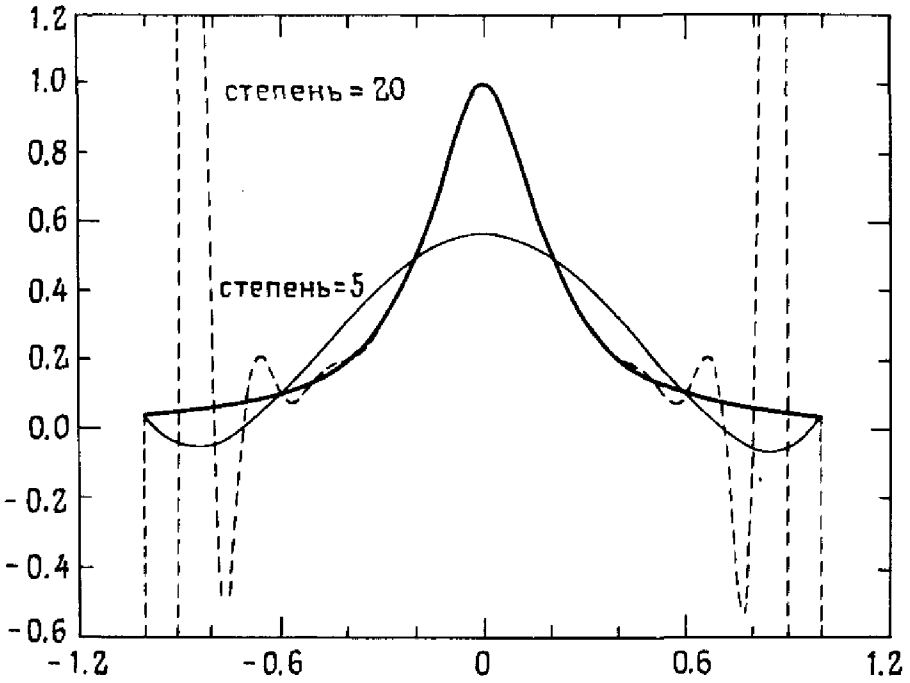


Рис. 4.5. Полиномиальная интерполяция функции Рунге.

для погрешности интерполяции, быстро растут с ростом n. Другой путь – рассмотреть R как функцию комплексной переменной. Эта функция имеет особые точки, удовлетворяющие уравнению $1 + 25x^2 = 0$, т. е. точки $\pm i/5$. Эти особые точки притаились совсем рядом с интервалом интерполяции $[-1, 1]$, достаточно близко, чтобы воздействовать на интерполянт.

Если абсциссы данных расположить неравномерно, сконцентрировав их ближе к концам интервала, то затруднение с функцией Рунге исчезнет. В результате полиномиальные интерполянты будут сходиться к $R(x)$ при стремлении n к бесконечности для любого x из $[-1, 1]$. К сожалению, в общем случае этот прием не срабатывает. Теорема Фабера говорит, что не существует правила для выбора точек, которое работало бы для всех непрерывных функций. В то же время для любой конкретной функции g можно индивидуально подобрать расположение таких точек. Можно также найти общее правило их выбора для всех функций, имеющих по крайней мере две непрерывных производных. Дальнейшие подробности см в учебнике [Davis, 1963].

4.5. Историческая справка: Рунге

Карл Давид Рунге (1856–1927) родился в Германии, в Бремене. Он был четвертым, младшим сыном в процветающей купеческой семье. Его родители прожили почти 20 лет до его рождения на Кубе, в Гаване, а затем уехали в Бремен. Они предпочитали английский язык и привили сыну английские взгляды на жизнь, придавая, в частности, особое значение спорту, воспитывая в мальчике уверенность в своих силах и честность. Молодой Рунге поражал своей словно из камня высеченной фигурой. Высокий, с большой, прекрасно вылепленной головой, он был

к тому же исключительным конькобежцем. Всю свою профессиональную жизнь он провел в Германии – в Мюнхене, Берлине, Ганновере и, наконец, в Геттингене. Близким и горячим другом Рунге в течение всей его жизни был Макс Планк. Но все же Рунге считал себя в интеллектуальном отношении последователем Вейерштрасса. Его ранняя работа по теории функций была выполнена под руководством Леопольда Кронекера, но вскоре Рунге погрузился в задачи спектроскопии и астрофизики, которыми в основном и занимался в дальнейшем. Почти все его значительные работы относились к этим областям, но он никогда не прекращал считать себя математиком. Его интересы постепенно сфокусировались на вопросах точности, обработки и преобразования данных.

Прикладная математика, как ее понимал и применял Рунге, отличалась от науки его современников. Его совсем не интересовала строгая математическая трактовка моделей, выведенных из физического опыта, и очень мало интересовали математические методы, которые в то время использовались в технике. Он главным образом хотел заниматься теорией и практикой численных методов, делая при этом особое ударение на практике. Некоторыми из его методов пользуются еще и сегодня, особенно методом Рунге – Кутты для решения дифференциальных уравнений. Тем не менее математики не признавали Рунге членом своего цеха; не признавали его своим и физики. В результате он долго не получал достойной университетской должности. В 1904 г. при интенсивном содействии со стороны Планка и Феликса Клейна Рунге получил место профессора в Геттингене; это была первая (и последняя) профессорская должность по прикладной математике в Германии. В некотором смысле он был создателем этой дисциплины и в те времена оставался в ней единственным практикующим преподавателем. Несмотря на свои либеральные политические взгляды во время первой мировой войны, Рунге получал все большее признание среди коллег по профессии, и в 1920 г. Питер Дебай рекомендовал его как своего преемника на заведование кафедрой, которую сам оставлял, потому что Рунге был «единственным человеком в Геттингене, способным руководить физическим заведением».

Рунге отошел от дел в 1925 г., даже не оставив какого-либо талантливого студента, пожелавшего изучить его форму прикладной математики, которую сегодня мы назвали бы численным анализом. Он пребывал в добром здравии вплоть до своей смерти в 1927 г. Рунге оставил двух сыновей, четырех эмансипированных дочерей и остался в семейной памяти дедушкой, который делал стойку на руках на праздновании своего семидесятилетия.

4.6. Вычисление полиномов

Полиномы столь распространены в математике, что часто приходится сталкиваться с проблемой быстрого их вычисления по заданным коэффициентам. Полином

$$p(x) = a_1 x^n + a_2 x^{n-1} + \ldots + a_{n+1}$$

можно вычислить с помощью Фортран-программы

```
      P = A(N + 1)
      DO 10 I = 1, N
          P = P + A(I) * X ** (N - I + 1)
10    CONTINUE
```

что потребует n умножений, n возведений в степень и n сложений. В простом способе, называемом *схемой Горнера*, $p(x)$ записывают в виде

$$p(x) = a_{n+1} + x(a_n + x(a_{n-1} + x(\ldots(a_2 + a_1 x)\ldots))).$$

На Фортране это выглядит так:

```
      P = A(1)
      DO 10 I = 1, N
          P = P*X + A(I + 1)
10    CONTINUE.
```

Схема Горнера требует только n умножений и n сложений и, возможно, известна вам как «синтетическое деление». Метод носит имя Горнера, потому что он привел это правило в статье (написанной по другому поводу) в 1819 г. На самом деле указанная расстановка скобок была опубликована Исааком Ньютоном на 100 лет раньше. С тех пор были найдены некоторые обобщения схемы Горнера; см. об этом, например, в книге [Knuth, 1969]. Правило Горнера – это наиболее подходящий метод для вычисления значений полиномов в произвольных точках. Однако если $p(x)$ нужно вычислить для последовательности точек равномерной сетки, например чтобы построить график, то более эффективны другие методы (см. задачу 4.10).

4.7. Кусочно-линейная интерполяция

Полиномиальная интерполяция является глобальной. Это значит, что одна полиномиальная функция должна проходить через все заданные точки. При добавлении данных приходится увеличивать степень полинома, что, как мы видели, приводит к затруднениям. С середины 60-х годов популярность приобрел альтернативный подход: использование *кусочно-полиномиальных* функций. Мы вводим их в данном параграфе, а впоследствии обобщим этот подход, чтобы охватить кубические сплайны и эрмитовы кубические интерполянты, наиболее полезные из всех. Ценным учебником и справочником по сплайнам является книга [de Boor, 1978] [1]. Сплайны часто используются не только в интерполяции, которая является темой настоящей главы, но и при

[1] Карл Вильгельм Райнхолд де Боор, блестящий математик, эмигрировал из Восточной Германии в США. В 1960-1964 гг. работал математиком-исследователем в фирме Дженерал Моторз, разрабатывал математическое описание форм автомобильных кузовов. Он сформулировал концепцию B-сплайнов и построил много алгоритмов, широко используемых теперь во всем мире в системах автоматизированного проектирования. Сейчас он преподает в Университете штата Висконсин в Мэдисоне.

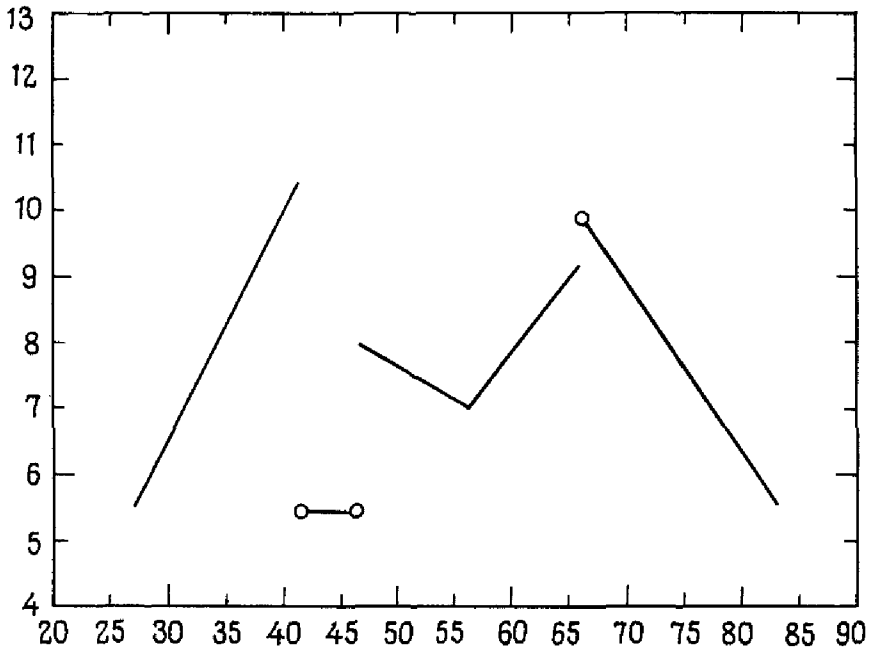


Рис. 4.6. Линейная кусочно-полиномиальная функция.

решении дифференциальных уравнений. Об этом хорошо написано в книге [Prenter, 1975].

При работе с кусочно-полиномиальными функциями абсциссы данных называются *узлами*, *сочленениями* или *точками излома*. Между этими названиями есть различия технического характера, но все три термина часто используются как взаимозаменяемые. Линейная кусочно-полиномиальная функция $L(x)$ – это функция, определенная при всех x, обладающая тем свойством, что $L(x)$ является прямой линией между x_i и x_{i+1}. Определение допускает, что в промежутках между разными парами соседних узлов $L(x)$ может совпадать с разными прямыми. На рис. 4.6 изображена одна линейная кусочно-полиномиальная функция. Заметим, что любая линейная комбинация таких функций также будет линейной кусочно-полиномиальной функцией.

Линейным кусочно-полиномиальным интерполянтом для набора данных (x_i, y_i) является линейная кусочно-полиномиальная функция, обладающая свойством

$$L(x_i) = y_i, \qquad i = 1, \ldots, n.$$

На рис. 4.7 показан линейный кусочно-полиномиальный интерполянт. Это тот самый рисунок «от точки к точке», который мы рисовали в детстве. Заметим, что определение ничего не говорит об $L(x)$ при $x < x_1$ или $x > x_n$, поэтому для одних и тех же данных существует много линейных кусочно-полиномиальных интерполянтов с различными свойствами на этих внешних интервалах. Тем не менее на $[x_1, x_n]$ указанный интерполянт будет единственным. Мы ограничимся рассмотрением именно этого интервала.

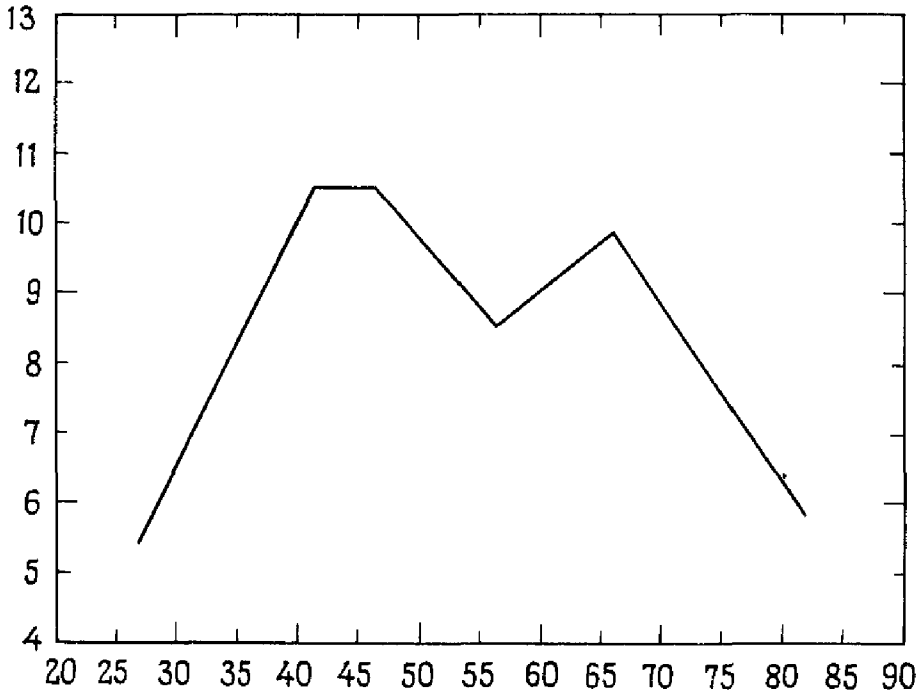


Рис. 4.7. Линейный кусочно-полиномиальный интерполянт.

Кусочно-полиномиальные функции могут показаться необычными, но это самые заурядные функции. Например, нетрудно написать правило для вычисления $L(x)$. На основании примера 4.1

$$L(x) = y_i \frac{x - x_{i+1}}{x_i - x_{i+1}} + y_{i+1} \frac{x - x_i}{x_{i+1} - x_i},$$

если $x_i \leqslant x \leqslant x_{i+1}$, $\quad i = 1, 2, \ldots, n-1$.

Линейный кусочно-полиномиальный интерполянт обладает таким приятным свойством, что если y_i – значения известной непрерывной функции $g(x)$ и если между x_1 и x_n появляются дополнительные точки, то интерполянт улучшается, т. е. приближается к исходной функции. Более того, если данными y_i являются значения функции $g(x)$, имеющей непрерывную вторую производную, то можно доказать, что

$$|L(x) - g(x)| < \frac{1}{8} h^2 \max |g''(x)| = O(x^2),$$

где h – наибольшее из расстояний между смежными узлами. Важность этого результата состоит в том, что выражение для оценки погрешности содержит лишь вторую производную и не зависит от числа узлов. Если удвоить число равномерно расположенных узлов, то погрешность для нового интерполянта составит около 1/4 погрешности старого. Таким образом, выбрав достаточно много узлов, ошибку интерполяции можно сделать сколь угодно малой. Конечно, на практике интерполируемая функция редко бывает известна, а добавление дополнительных точек является роскошью. Однако подобные утверждения о сходимости дают нам уверенность в этом методе, особенно по сравнению с «несходящейся» полиномиальной интерполяцией.

Поскольку функция $L(x)$ между узлами линейна, ее можно там продифференцировать. Получим

$$L'(x) = \frac{y_i}{x_i - x_{i+1}} + \frac{y_{i+1}}{x_{i+1} - x_i} = \frac{y_{i+1} - y_i}{x_{i+1} - x_i},$$

если $x_i < x < x_{i+1}$, $\quad i = 1, 2, \ldots, n-1$.

Это не что иное, как приближение производной разностным отношением, вероятно, хорошо вам известно. Его можно применять везде, кроме узлов, и оно дает хорошие оценки производной. Можно доказать, что

$$|L'(x) - g'(x)| = O(h), \qquad x \neq x_i.$$

Таким образом, производная интерполянта дает оценку производной функции, которая порождает заданные точки. В этом существенное отличие от полиномиальной интерполяции.

Пример 4.6. Погрешность кусочно-линейной интерполяции функции $R(x)$.

Сколько надо взять узлов равномерной сетки, чтобы построить линейный кусочно-полиномиальный интерполянт для функции Рунге, дающий ошибку менее 10^{-5}? Имеем

$$g'' = -50(1 + 25x^2)^{-2}\left[\frac{-100x^2}{(1 + 25x^2)} + 1\right]$$

и прямым вычислением показываем, что $|g''| \leqslant 50$. Поэтому, используя приведенное выше выражение для оценки погрешности, получаем, что h надо выбрать так, чтобы

$$\frac{1}{8} \cdot 50h^2 < 10^{-5},$$

или $h < 0.0013$. Так как $h = 2/(n-1)$, потребуется около 1540 узлов. □

4.8. Кусочно-кубические функции

Кусочно-линейная интерполяция решает одну проблему, возникающую при полиномиальной интерполяции,– она обладает сходимостью, но порождает при этом другую проблему – недостаток гладкости: график $L(x)$ имеет изломы. Чтобы найти более гладкий интерполянт, рассмотрим кусочно-полиномиальные функции более высокой степени. Мы увидим, что, в отличие от случая полиномиальных интерполянтов, это дает результаты, обладающие практической ценностью. В задаче 4.6 рассматриваются кусочно квадратические функции, однако более полезны кусочно-кубические функции. Кусочно-кубической называется функция, определенная при всех x и совпадающая с полиномом третьей степени между соседними узлами. Кусочно-кубическим интерполянтом $C(x)$ является кусочно-кубическая функция, которая интерполирует

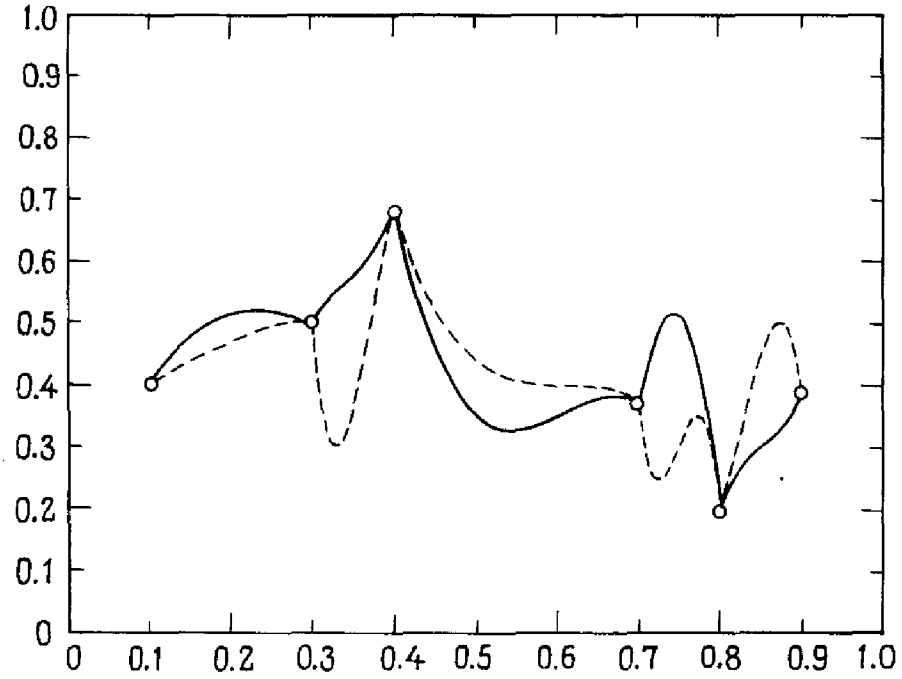


Рис. 4.8. Два кусочно-кубических интерполянта.

наши данные. На рис. 4.8 показаны два кусочно-кубических интерполянта. В данном случае видно, что даже внутри $[x_1, x_n]$ интерполянт не является единственным. Требования, чтобы кусочно-кубическая функция проходила через заданные точки, недостаточно для единственности, но если наложить условие некоторой гладкости, то можно получить единственный интерполянт.

Между узлами $C(x)$ представляет собой кубическую функцию. Ее можно продифференцировать сколько угодно раз. Если где-то дифференцируемости нет, так только в узлах. $C(x)$ автоматически непрерывна всюду, поскольку она является интерполянтом (см. рис. 4.8). Построить более гладкий интерполянт – это значит построить интерполянт с бóльшим числом непрерывных на $[x_1, x_n]$ производных. *Эрмитовым кубическим интерполянтом* [1] называется кусочно-кубический интерполянт с непрерывной производной. *Кубическим сплайном* называется кусочно-кубический интерполянт с двумя непрерывными производными. Оба типа интерполянтов важны для приложений. В инженерном и научном обиходе термин «сплайн» был первоначально синонимом кубического сплайна. Сегодня известны и применяются сплайны как низких, так и более высоких степеней. Однако наиболее популярны по-прежнему кубические сплайны, и мы уделим им все свое внимание.

А что если потребовать наличия трех непрерывных производных? Поскольку третья производная кубической функции постоянна, то легко показать, что любая кусочно-кубическая функция с тремя непрерывными в каждом узле производными должна быть в точности одной и той же кубической функцией на всех интервалах. Поскольку один полином третьей степени нельзя провести более чем через четыре точки, указан-

ное требование, вообще говоря, в задачах интерполяции предъявлять нельзя.

На каждом интервале $[x_i, x_{i+1}]$ функция $C(x)$ является кубической и задается четырьмя коэффициентами. Для программы, основанной на таком представлении, потребуется массив для хранения x_i и четыре массива a, b, c и d для коэффициентов кубической функции на каждом интервале. Это называется кусочно-полиномиальным представлением. Для наших целей мы предпочитаем использовать другое представление, интуитивно более ясное. Определим $2n$ базисных функций $c_i(x)$ и $\hat{c}_i(x)$, $i = 1, \ldots, n$. Пусть каждая из них является кусочно-кубической с непрерывной на $[x_1, x_n]$ производной. Тогда и любая их линейная комбинация обладает теми же свойствами. Определение этих функций должно гарантировать, что

$$c_i(x_i) = 1, \quad c_i(x_j) = 0 \quad \text{при} \quad j \neq i$$
$$\text{и} \quad \hat{c}_i(x_j) = 0 \quad \text{для всех} \quad i, j.$$

В этом случае функция

$$C(x) = \sum_{i=1}^{n} (y_i c_i(x) + d_i \hat{c}_i(x))$$

является эрмитовым кубическим интерполянтом при любом выборе d_i. Потребуем еще, чтобы

$$c_j'(x_i) = 0 \text{ для всех } i, j \text{ и } \hat{c}_i'(x_i) = 1,$$
$$\hat{c}_i'(x_j) = 0 \text{ при } j \neq i.$$

Тогда

$$C'(x_k) = \sum_{i=1}^{n} (y_i c_i'(x_k) + d_i \hat{c}_i'(x_k)) = d_k \hat{c}_k'(x_k) = d_k,$$

На рис. 4.10 показаны типичные c_i и $\hat{c}_i$. Глядя на их графики, легко представить себе различные эрмитовы кубические интерполянты $C(x)$. Все они представляют собой кусочно-кубические функции, интерполирующие по заданным точкам и имеющие по одной непрерывной производной. Значения производных в узлах интерполяции задаются числами d_i. Такая форма интерполянта особенно полезна, если, кроме самих значений в точках x_i, известны еще и величины углов наклона касательных к интерполируемой функции. В этом случае в качестве d_i естественно выбрать заданные угловые коэффициенты. На рис. 4.9 показаны две разные функции $C(x)$, одна с $d_i = 1$, $\forall i$, а другая с $d_i = 0$. Возможно, это и не имеет практического значения, но иллюстрирует эффект от введения $\hat{c}_i$.

Пример 4.7. Эрмитова кубическая интерполяция функции $R(x)$, $d_i = R'(x_i)$.

Постройте эрмитов кубический интерполянт для функции Рунге $R(x)$. Используйте $R'(x_i)$ при задании d_i. Мы опускаем все детали и приводим ниже график такого интерполянта. □

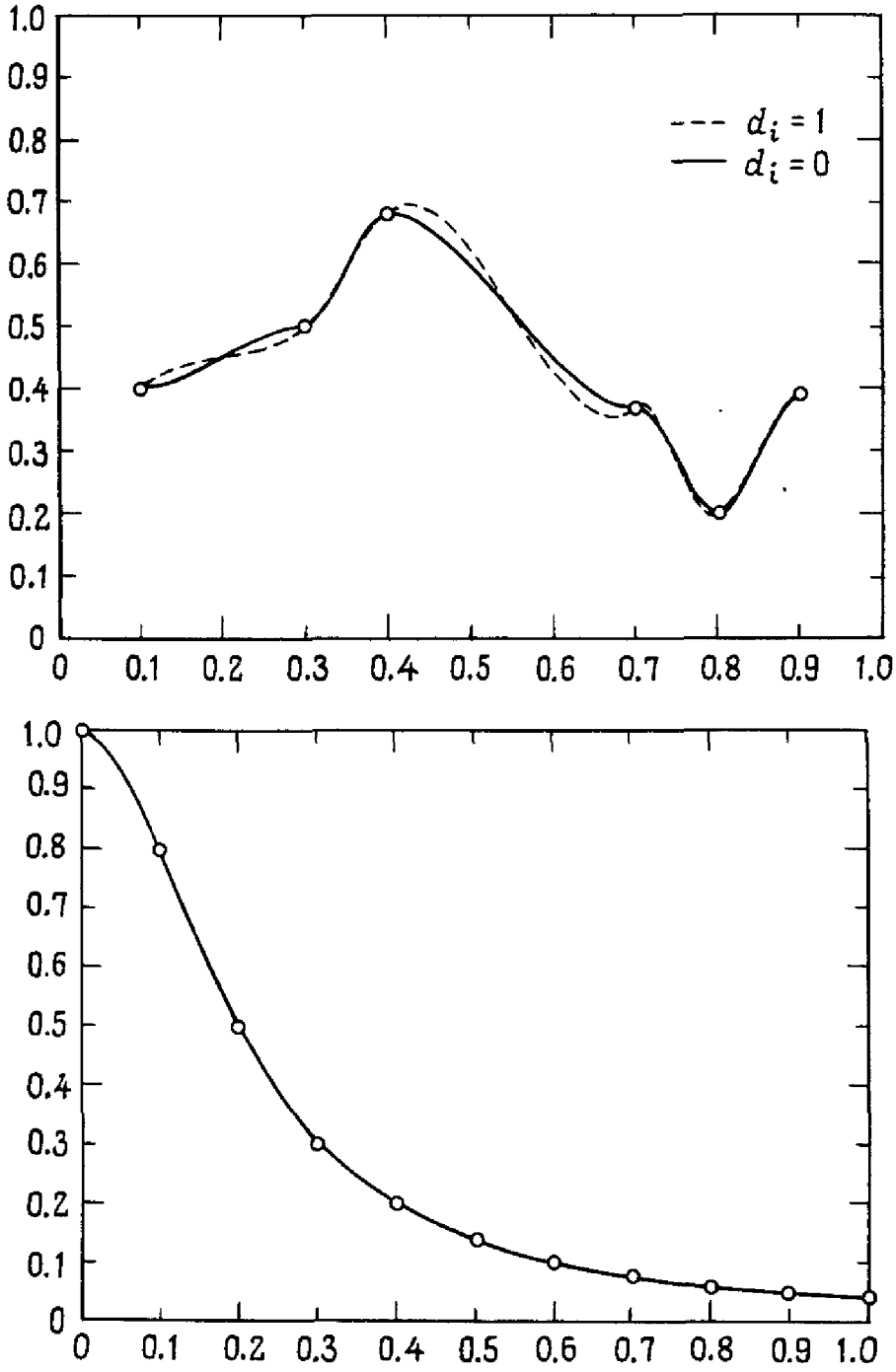


Рис. 4.9. Два эрмитовых кубических интерполянта с разными производными.

Если числа d_i неизвестны, то можно использовать факт равенства $d_i = C'(x_i)$ как подсказку и попробовать оценить их, исходя из данных значений функции. Например, существует единственный квадратный трехчлен, проходящий через (x_{i-1}, y_{i-1}), (x_i, y_i) и (x_{i+1}, y_{i+1}). Его производная в точках x_i равна

$$\delta_i \equiv \frac{\Delta_i h_{i-1} + \Delta_{i-1} h_i}{h_{i-1} + h_i}, \text{ где } \Delta_i = \frac{y_{i+1} - y_i}{h_i}.$$

С помощью δ_i можно построить приближения для $d_2, \ldots, d_{n-1}$. Для

приближения к d_1 и d_n можно использовать Δ_1 и Δ_{n-1} или взять более точные выражения, такие, как производная (в точке x_1) указанного выше квадратного трехчлена. Есть и много других способов оценки производных.

Пример 4.8. Эрмитова кубическая интерполяция функции $R(x)$, $d_i \approx R'(x_i)$.

Если построить эрмитов кубический интерполянт для функции Рунге $R(x)$, но при этом использовать приведенные выше приближения к d_i, то график интерполянта будет невозможно отличить от графика, изображенного в примере 4.7. Конечно, такой хороший результат получается не для всех наборов данных. □

Представление с помощью функций c_i, $\hat{c}_i$ требует трех массивов: по одному для хранения x_i, y_i и d_i. Оно называется эрмитовым кубическим представлением. При этом требуется только три пятых объема памяти, необходимого для кусочно-полиномиального представления, о котором говорилось раньше, однако, поскольку n редко превышает несколько сотен, объем памяти не является важной характеристикой. Основное преимущество эрмитова кубического представления – это очевидный смысл коэффициентов. Но когда интерполянт надо вычислять многократно (к примеру, при построении графика), кусочно-полиномиальное представление может оказаться предпочтительнее. В обоих случаях программа, вычисляющая значение интерполянта в некоторой заданной точке x, должна сначала отнести x к одному из интервалов. Если при этом используется кусочно-полиномиальное представление, то остается только вычислить значение полинома третьей степени, коэффициенты которого для найденного интервала известны. Если используется эрмитово кубическое представление, то программа сначала решает, какие из функций c_i и $\hat{c}_i$ отличны от нуля на этом интервале (обычно по две функции каждого типа), а затем вычисляет их в точке x. Поэтому вычисления для эрмитова кубического представления происходят медленнее, чем для кусочно-полиномиального представления. Пакеты для работы с кусочно-кубическими функциями часто содержат программы перевода из одного представления в другое.

В программах кусочно-кубической интерполяции обычно используются пары подпрограмм. Первая вычисляет какие-либо неизвестные необходимые параметры – например d_i. Эта подпрограмма обычно вызывается для каждого набора данных только один раз. Вторая подпрограмма собственно и есть вычислитель и может вызываться столько раз, сколько потребуется для определения $C(x)$ в точке или в последовательности точек. Программа-вычислитель часто выдает также значения первой и/или второй производной в заданной точке (точках).

Резюме

Эрмитов кубический интерполянт не является единственным. Существует n-параметрическое семейство кусочно-кубических функций,

которые интерполируют n данных значений и имеют по одной непрерывной производной. Приведенная ниже подпрограмма PCHEZ выбирает такого представителя этого семейства, который «выглядит наиболее приятно». Мотивировку и обсуждение этой идеи можно найти в § 12.

4.9. PCHIP, пакет программ кусочно-кубической эрмитовой интерполяции

PCHIP представляет собой пакет подпрограмм, созданный Фритчем и Карлсоном [Fritsch, Carlson, 1980] как гибкий инструмент для работы с эрмитовыми кубическими интерполянтами и сплайнами. В этом параграфе описаны два простых в использовании драйвера для работы с PCHIP: драйвер PCHEZ строит наиболее приятный на вид эрмитов кубический интерполянт или кубический сплайн; драйвер PCHEV вычисляет значения интерполянта, построенного PCHEZ, и его производной.

В этих программах преследовалась цель обеспечить наибольшую простоту использования, поэтому они не являются особенно гибкими. Но пакет PCHIP содержит много других программ, которые можно приспособить для конкретных приложений. В нем также есть программы интегрирования кусочно-кубических функций. В примере, приведенном ниже, вы найдете вызов программы PCHQA, но мы отложим его обсуждение до § 9 гл. 5. Дополнительные подробности см. в указанной работе.

В качестве иллюстрации рассмотрим задачу построения интерполянта для функции Рунге, подобную рассмотренной в примере 4.7. Величины абсцисс и ординат хранятся в массивах X и F, а массив для промежуточного вывода D содержит производные эрмитова кубического интерполянта или сплайна. Как только найдены производные, можно одним обращением к PCHEV вычислить значения интерполянта (и его производных) в точках, записанных в массиве XE; искомые значения интерполянта получатся в массиве FE.

```
      REAL X(21), F(21), D(21), WK(42), FE(101), XE(101), FD(101)
      LOGICAL SPLINE
C
C  Арифметические оператор-функции для вычисления функции
C  Рунге и ее производной.
C
      R(U) = 1.0/(1.0 + 25.0*U*U)
      RP(U) = - 50.0*U*R(U)**2
C
C  Вычисление функции Рунге в 21 точке на [-1, 1].
C
      DO 11 I = 1,21
         X(I) = - 1.0 + (I - 1)/10.0
         F(I) = R(X(I))
```

```
 1    CONTINUE
      N = 21
      NWK = 42
      SPLINE = .FALSE.
C
C     Так как SPLINE присвоено значение .FALSE.,
C     строится кубический эрмитов интерполянт.
C
      CALL PCHEZ (N, X, F, D, SPLINE, WK, NWK, IERR)
      IF (IERR.LT.0)THEN
         WRITE ( *, *) 'AN ERROR CALLING PCHEZ, IERR = ', IERR
         STOP
      ENDIF
C
      NE = 101
C
C     Вычисление значений интерполянта и его производной
C     в 101 точке от  − 1 до 0.
C
      DO 2 I = 1, NE
            XE(I) = − 1.0 + (I − 1.0)/(NE - 1.0)
 2    CONTINUE
      CALL PCHEV (N, X, F, D, NE, XE, FE, FD, IERR)
      IF(IERR.NE.O)THEN
      WRITE(*, *) 'AN ERROR CALLING PCHEV, IERR = ', IERR
      STOP
      ENDIF
C
      DO 3 I = 1, NE
            ERROR = FE(I) − R(XE(I))
            ERRORD = FD(I) − RP(XE(I))
            WRITE(*, *) XE(I), FE(I), ERROR, ERRORD
   3  CONTINUE
C
C     Вычисление интеграла по отрезку [0, 1].
C
      A = 0.0
      B = 1.0
      Q = PCHQA(N, X, F, D, A, B, IERR)
      WRITE(*, *) 'INTEGRAL FROM O TO 1 AND IERR ARE ', Q,
      IERR
C
      STOP
      END
```

*4.10. Детали кубической эрмитовой интерполяции

Пусть $h_i = x_{i+1} - x_i$. Определим на каждом из интервалов $[x_i, x_{i+1}]$, $i = 1, \ldots, n-1$, четыре кубические функции

$$c_0^i(x) = \frac{2}{h_i^3}(x - x_{i+1})^2\left(x - x_i + \frac{h_i}{2}\right), \quad \hat{c}_0^i(x) = \frac{1}{h_i^2}(x - x_i)(x - x_{i+1})^2,$$

$$c_1^i(x) = -\frac{2}{h_i^3}(x - x_i)^2\left(x - x_{i+1} - \frac{h_i}{2}\right), \quad \hat{c}_1^i(x) = \frac{1}{h_i^2}(x - x_i)^2(x - x_{i+1}).$$

Теперь определим $c_1(x)$ и $\hat{c}_1(x)$ как

$$c_1(x) = \begin{cases} c_0^1 \text{ на } [x_1, x_2], \\ \emptyset \text{ на } [x_2, x_n]; \end{cases} \quad \hat{c}_1(x) = \begin{cases} \hat{c}_0^1 \text{ на } [x_1, x_2]; \\ \emptyset \text{ на } [x_2, x_n]. \end{cases}$$

Положим для $i = 2, 3, \ldots, n-1$

$$c_i(x) = \begin{cases} \emptyset \text{ на } [x_1, x_{i-1}]; \\ c_1^{i-1} \text{ на } [x_{i-1}, x_i]; \\ c_0^i \text{ на } [x_i, x_{i+1}]; \\ \emptyset \text{ на } [x_{i+1}, x_n]; \end{cases} \quad \hat{c}_i(x) = \begin{cases} \emptyset \text{ на } [x_1, x_{i-1}]; \\ \hat{c}_1^{i-1} \text{ на } [x_{i-1}, x_i]; \\ \hat{c}_0^i \text{ на } [x_i, x^{i+1}]; \\ \emptyset \text{ на } [x_{i+1}, x_n]; \end{cases}$$

а для $i = n$

$$c_n(x) = \begin{cases} \emptyset \text{ на } [x_1, x_{n-1}]; \\ c_1^{n-1} \text{ на } [x_{n-1}, x_n]; \end{cases} \quad \hat{c}_n(x) = \begin{cases} \emptyset \text{ на } [x_1; x_{n-1}]; \\ \hat{c}_1^{n-1} \text{ на } [x_{n-1}, x_n]. \end{cases}$$

Наконец, определим

$$C(x) = \sum_{i=1}^{n} y_i c_i(x) + d_i \hat{c}_i(x).$$

На рис. 4.10. показаны типичные c_i и $\hat{c}_i$.

Попробуем обрисовать свойства этих функций на примере $\hat{c}_7(x)$. По данному выше определению $\hat{c}_7(x)$ тождественно равна нулю при $x \leqslant x_6$ и $x \geqslant x_8$. Для $x_6 \leqslant x \leqslant x_7$ имеем $\hat{c}_7(x) = \hat{c}_1^6(x) = (x - x_6)^2(x - x_7)/h_6^2$. Для $x_7 \leqslant x \leqslant x_8$ имеем $\hat{c}_7(x) = \hat{c}_0^7(x) = (x - x_7)(x - x_8)^2/h_7^2$. Из этих формул видно, что функция $\hat{c}_7(x)$ определена при всех x и является кусочно-полиномиальной. Она обращается в нуль в каждом узле. В точках x_6 и x_8 у нее нули второго порядка, следовательно, в этих точках и производная обращается в нуль. В узле x_7 производную можно вычислить по одной из двух формул, в зависимости от того, приближаемся ли мы к x_7 слева или справа. Производная слева равна

$$\hat{c}_7'(x_7^-) = \lim_{x \to x_7^-} \hat{c}_7'(x) = \lim_{x \to x_7^-} [\hat{c}_1^6(x)]' =$$

$$= \lim_{x \to x_7^-} \frac{1}{h_6^2}[(x - x_6)^2 + 2(x - x_6)(x - x_7)] = 1,$$

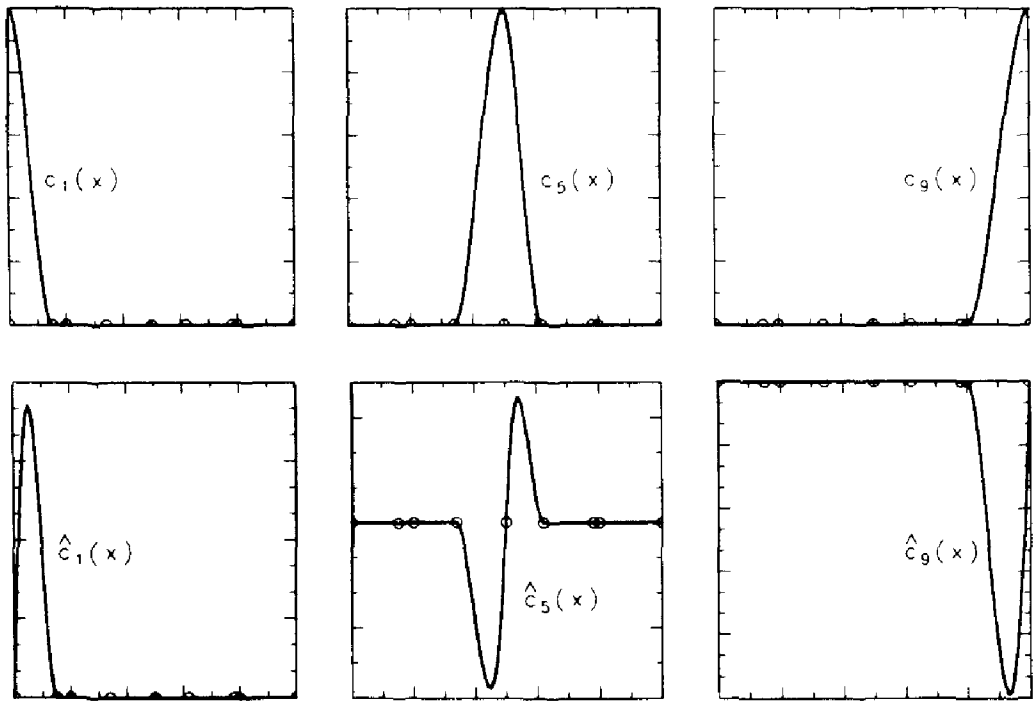


Рис. 4.10. Типичные эрмитовы кубические базисные функции.

производная справа равна

$$\hat{c}_7'(x_7^+) = \lim_{x \to x_7^+} \hat{c}_7'(x) = \lim_{x \to x_7^+} [\hat{c}_0^7(x)]' =$$

$$\lim_{x \to x_7^+} \frac{1}{h_7^2}[2(x - x_7)(x - x_8) + (x - x_8)^2] = 1.$$

Поскольку односторонние производные с обеих сторон равны, то $\hat{c}_7'(x_7) = 1$.

Если проявить терпение, то можно проверить, что функции c_i и $\hat{c}_i$ обладают всеми свойствами, указанными в § 8. В частности, каждая из них имеет непрерывную производную. Непрерывную производную имеет и функция $C(x)$, следовательно, она является эрмитовым кубическим интерполянтом. Функцию $C(x)$ легко вычислить, если известны величины d_i. Для нахождения значения интерполянта в произвольной фиксированной точке x достаточно заметить, что функции c_i и $\hat{c}_i$ отличны от нуля не более чем на двух интервалах. Поэтому большинство членов в сумме тождественно равны нулю; надо учитывать не более четырех слагаемых. Итак, вычисление значения C включает в себя, во-первых, локализацию x в некотором интервале, скажем, между x_i и x_{i+1}, во-вторых, вычисление членов c_i, c_{i+1}, $\hat{c}_i$, $\hat{c}_{i+1}$, в-третьих, умножение на требуемые y и d и суммирование.

4.11. Кубические сплайны

Другой подход к выбору d_i заключается в том, чтобы заставить функцию $C(x)$ удовлетворять некоторым дополнительным условиям: это позволит снизить число свободных параметров. Разумно потребовать, чтобы функция C была более гладкой, т. е. чтобы отдельные составляющие ее полиномы при стыковке в узлах давали непрерывную вторую производную. Мы увидим, что свободных параметров, которые можно подчинить этому условию, более чем достаточно. Указанное требование приводит к интерполяции кубическим сплайном. По построению эрмитова кубического интерполянта известно, что $C(x)$ имеет непрерывную на $[x_1, x_n]$ производную независимо от d_i. Для многих приложений этого достаточно, но для некоторых нет. Более того, порождаемая произвольностью выбора d_i неопределенность также может привести к затруднениям. Каждая задача требует решения, какие значения следует задать для d_i. Посмотрим, можно ли выбрать d_i так, чтобы функция $C(x)$ имела две непрерывные производные. Вместо того чтобы выводить формулы сплайн-интерполяции, надо сначала решить, реалистичны ли наши надежды. Один из способов – сосчитать число ограничений, которые мы хотим наложить на $C(x)$, и посмотреть, достаточно ли у нас для этого свободных параметров. Это еще ничего не доказывает, но число параметров даст нам определенное представление о задаче. Есть $n-1$ частичных интервалов: $x_1 < \ldots < x_n$. На каждом из них $C(x)$ является полиномом третьей степени, определяемым четырьмя коэффициентами независимо от способа представления. Всего у нас $4(n-1)$ параметров. Из-за условия интерполяции на каждом частичном интервале $[x_i, x_{i+1}]$ действуют два ограничения на $C(x)$: вспомним, что $C(x_i) = y_i$ и $C(x_{i+1}) = y_{i+1}$. Требование непрерывности C'' добавляет $n-2$ ограничения (по одному в каждом внутреннем узле). Разность между числом параметров и числом ограничений равна $4(n-1) - 2(n-1) - (n-2) = n$. Таким образом, эрмитов кубический интерполянт должен иметь n свободных параметров, а это в точности равно числу имеющихся в нашем распоряжении чисел d_i. Условие непрерывности C'' добавляет $n-2$ ограничения. Итак, мы приходим к заключению, что постановка вопроса о кубической сплайн-интерполяции допустима и даже оставляет нам еще два свободных параметра.

Проведенные рассуждения с подсчетом параметров не показывают, как найти сплайн-интерполянт, а только делают правдоподобным его существование. Техника вывода формул для интерполянта достаточно сложна и обычно включает в себя два этапа.

(1) Написать общее выражение для $C''(x)$. Оно будет содержать неизвестные d_i.

(2) Записать аналитически требование непрерывности $C''(x)$ в $n-2$ внутренних узлах. Это приведет к системе линейных уравнений для d_i.

Если бы эту систему можно было решить, то мы бы нашли сплайн.

Но, как мы и предвидели, система получается недоопределенной, т. е. для нахождения n чисел d_i у нас только $n-2$ уравнения. Есть несколько способов выделить единственный сплайн. Чаще всего применяющийся в пакетах программ подход основан на ограничениях, накладываемых на поведение сплайна в концевых точках x_1 и x_n. Ниже приведены некоторые возможности. Заметим, что подпрограмма PCHEZ, которая обсуждалась в § 9, автоматически выбирает пункт (д), если входной параметр SPLINE имеет значение .TRUE..

(а) Задать производную функции $C(x)$ в точках x_1 и x_n, т. е. положить d_1 и d_n равными известным величинам или нулю. Тогда мы получим так называемый полный кубический сплайн-интерполянт. Во многих задачах производные в концевых точках известны a priori из физических соображений, поэтому такой подход часто оказывается полезным.

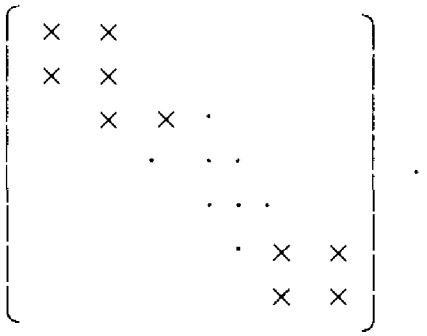
(б) Оценить производные по данным и использовать полученные оценки в качестве d_1 и d_n. Это надо делать с осторожностью. Известны случаи, когда построенный сплайн терял всякий практический смысл из-за слишком грубых оценок величин производных на концах. Возможен следующий подход: вывести формулу для полинома третьей степени, который проходил бы через четыре первые заданные точки, а затем продифференцировать его в точке x_1 и принять это значение за d_1. Аналогично для x_n.

(в) Потребовать, чтобы вторая производная сплайна совпадала в точке x_1 со второй производной полинома третьей степени из (б). Аналогично для x_n.

(г) Приравнять $C''(x_1) = 0 = C''(x_n)$. Такой сплайн называется *естественным сплайн-интерполянтом*. Физически это означает, что график сплайна представляет собой прямую линию вне заданного интервала $[x_1, x_n]$.

(д) Наложить условие *запрета стыка*. По построению функция C'' непрерывна во всех узлах (термин «сплайн» как раз это и подразумевает), однако функция C''' не обязана быть непрерывной. Можно потребовать, чтобы $C'''(x)$ была непрерывной в точке x_2. Это эквивалентно условию, что кубические функции на первом и втором частичном интервалах совпадают. Это также эквивалентно удалению узла x_2 с сохранением требования, чтобы единая на $[x_1, x_3]$ кубическая функция обеспечивала интерполяцию не только на концах этого интервала, но и в точке x_2. Иными словами, x_2 есть точка интерполяции, но не узел. Аналогично для x_{n-1}.

Если использовать одну из перечисленных возможностей, то система уравнений для d_i будет иметь матрицу следующего вида:



Это симметричная трехдиагональная невырожденная матрица, и можно показать, что она хорошо обусловлена при любом разумном выборе узлов $x_1 < x_2 < \ldots < x_n$. Систему можно решить при помощи программы SGEFS из гл. 3. Однако можно показать, что из-за специальных свойств матрицы при решении системы методом Гаусса выбор главных элементов не нужен. Вследствие трехдиагональности не требуется также хранить полную $n \times n$-матрицу. В программах сплайн-интерполяции всегда используют эти преимущества и не применяют процедуры решения полных линейных систем. Вместо этого подключают программы для решения систем специального вида.

Симметричные трехдиагональные системы, которые можно решить без выбора главных элементов, часто встречаются в различных областях вычислительной математики. Поразмышляем несколько мгновений о процессе исключения для таких систем. На каждом шаге требуется выполнить незначительную работу: один элемент исключить и один элемент модифицировать. Для обычных полных систем метод исключения требует выполнения $O(n^3)$ арифметических операций; для рассматриваемых специальных систем число операций снижается до $O(n)$. В результате можно быстро решать даже очень большие системы.

Некоторые программы сплайн-интерполяции предоставляют пользователю и другие возможности выбора.

(а) Заставить функцию $C(x)$ быть периодической, т. е. положить

$$C(x_1) = C(x_n), \quad C'(x_1) = C'(x_n) \text{ и } C''(x_1) = C''(x_n).$$

(б) Задать значение интеграла функции $C(x)$, которое часто выбирают равным единице:

$$1 = \int_{x_1}^{x_n} C(x)\,dx.$$

Такие условия могут разрушить симметричную трехдиагональную структуру матрицы в системе для определения d_i, и поэтому подобные возможности встречаются только в более детализированных пакетах программ. Для большинства задач нет наилучшего способа выбора дополнительных условий. С другой стороны, многие наборы данных приводят к интерполянтам, которые независимо от сделанного выбора выглядят почти одинаково.

Как только неизвестные коэффициенты сплайна d_i найдены,

сплайн становится полностью определенным и его можно вычислять в любой точке x. Величины d_i – это именно те коэффициенты, которые позволяют функции $C(x)$ иметь две непрерывные производные. Часто возникает путаница в ситуации, когда данные y_i являются значениями известной гладкой функции, например функции Рунге. Казалось бы, можно ожидать, что вычисленные значения d_i совпадут с $g'(x_i)$. Но для такого совпадения нет никаких оснований. Единственный случай, когда такое ожидание оправдано, это если сама функция $g(x)$ является сплайном. Между тем эрмитов кубический интерполянт из примера 4.7 определенно не является сплайном и не имеет двух непрерывных производных.

Пример 4.9. Сплайн-интерполянт для R (x).

Сплайн-интерполянт для функции Рунге (с равномерной сеткой узлов и с краевыми условиями типа (д)) очень похож на приведенный в примере 4.7. Для этих данных можно использовать практически любой разумный кусочно-кубический интерполянт. Но вспомним, что полиномиальная интерполяция здесь непригодна (см. рис. 4.5). □

4.12. Практические различия между сплайнами и кубическими эрмитовыми интерполянтами

Если у нас есть конкретный набор данных для интерполяции, то для выбора одного из этих двух похожих способов кусочно-полиномиальной интерполяции можно предложить несколько общих принципов. Конечно, если требуется гладкость второй производной, то выбирают сплайны, но часто этого требования в явном виде нет. Для «хороших» наборов данных качественное различие двух интерполянтов невелико. Но иногда можно заметить, что кубический эрмитов интерполянт менее гладкий. Чтобы получить сплайн, надо решить систему уравнений для его коэффициентов. Как отмечалось выше, решить ее можно быстро, но систему надо еще составить, и следует учитывать необходимые для этого время и усилия. Эрмитов кубический интерполянт также определяется своими коэффициентами, но они находятся без решения линейной системы. Все же обычно выбор между сплайнами и эрмитовой кубической интерполяцией не зависит от затрат на вычисление d_i, потому что в практических задачах итоговая кусочно-кубическая функция вычисляется во многих точках и это составляет наиболее значительную часть общей работы. Естественно, время вычисления значений функции для сплайна и эрмитова кубического интерполянта совершенно одинаково. Если известны производные интерполянта или модельной функции, то их можно легко включить в построение эрмитова кубического интерполянта, но не сплайна, кроме случая, когда эти производные заданы на концах интервала. Если производные неизвестны, то можно найти и использовать различные оценки для них; здесь тоже потребуется поработать.

Как мы уже видели, сплайн не определяется однозначно условием интерполяции; нужна дополнительная информация. Слишком часто программы для сплайн-интерполяции не позволяют пользователю задать эту информацию, и можно подумать, что в ней нет необходимости. Но, скорее всего, в этом случае в программе используется какое-либо специальное предположение; простейший вариант – это естественный сплайн, но возможны и другие. Например, программа может по входным данным оценивать первые производные на обоих концах интервала и использовать эти оценки в качестве d_1 и d_n. Пользователь должен быть внимательным, когда от него не требуют задания дополнительных условий; в этом случае следует изучить программную документацию и понять, какие условия используются по умолчанию.

Теоретическую оценку погрешности для сплайн-интерполяции или эрмитовой кубической интерполяции можно получить, сравнивая интерполянт $C(x)$ с какой-либо известной функцией, порождающей данные, как мы это делали в случае полиномиальной и кусочно-линейной интерполяции. Если данными y_i являются значения функции $g(x)$, которая имеет непрерывную четвертую производную, то

$$|C(x) - g(x)| < \text{const} \cdot h^4 \max |g^{(4)}(x)| = O(h^4),$$

где $C(x)$ может быть либо кубическим сплайном, либо эрмитовым кубическим интерполянтом, построенным по тому же множеству узлов. Величина постоянной оставлена неопределенной, поскольку она зависит от того, является ли C сплайном или эрмитовым кубическим интерполянтом, и от того, как задаются два дополнительных условия в первом случае или выбираются d_i во втором. Приведенное выше неравенство имеет место для всех разумных значений не указанных точно величин. (Напомним, что h – это наибольшее расстояние между соседними узлами.) Существенно, что интерполяция улучшается при увеличении числа узлов и уменьшении расстояний между ними, и что скорость улучшения пропорциональна четвертой степени расстояния между узлами, т.е. намного выше, чем для кусочно-линейной интерполяции. Это оправдывает наш интерес к кусочно-полиномиальным интерполянтам, но еще не значит, что между узлами $C(x)$ всегда будет выглядеть именно так, как нам хочется. В большинстве задач имеется только один набор данных и их число невозможно увеличить для изучения сходимости. Строится только один интерполянт, и либо он оказывается удовлетворительным, либо нет.

Часто имеет смысл использовать производные сплайна или эрмитова кубического интерполянта в качестве аппроксимаций соответствующих производных исходной функции. Можно показать, что если данные генерируются достаточно гладкой функцией, то

$$|C'(x) - g'(x)| = O(h^3), \quad |C''(x) - g''(x)| = O(h^2) \text{ и}$$
$$|C'''(x) - g'''(x)| = O(h).$$

Если $C(x)$ является сплайном, первые две формулы имеют место для

любого x, а последняя – только если x отлично от узла. Если $C(x)$ является эрмитовым кубическим интерполянтом, то вторая и третья формулы справедливы для отличных от узлов точек x. Конечно, эти теоретические выводы, как и другие, следует воспринимать с долей критики. Например, в § 9 мы видели, как вычисляется сплайн-интерполянт по 21 равномерно распределенным точкам для функции Рунге $R(x)(h = 0.1)$. В этом примере наибольшая ошибка интерполянта в значениях составляет около 0.003 для значений функции и около 0.08 для производной. Рекомендуем всегда, когда это возможно, строить график интерполянта, чтобы убедиться, что он обладает нужными физическими свойствами. А если вы собираетесь дифференцировать его, то и подавно надо исследовать график. (Дальнейшие подробности см. в книге [Schultz, 1973].)

Алгоритмы сплайн-интерполяции завоевали огромный рынок сбыта из-за того, что большинство пользователей получали удовлетворительные результаты интерполяции. Этих программ, однако, было продано слишком уж много. Есть много примеров наборов данных, для которых сплайн-интерполяция (с какими угодно дополнительными условиями) дает плохую «подгонку». Это чаще всего происходит в случаях, когда данные претерпевают быстрые изменения на малом интервале. На рис. 4.11 изображен такой набор данных и показан естественный сплайн-интерполянт для них. В этом конкретном случае было известно, что данные описывают некоторый возрастающий на указанном интервале физический процесс; сплайн же таким свойством не обладает. Показанный на этом рисунке практический результат не противоречит приведенному в предыдущем абзаце свойству сходимости.

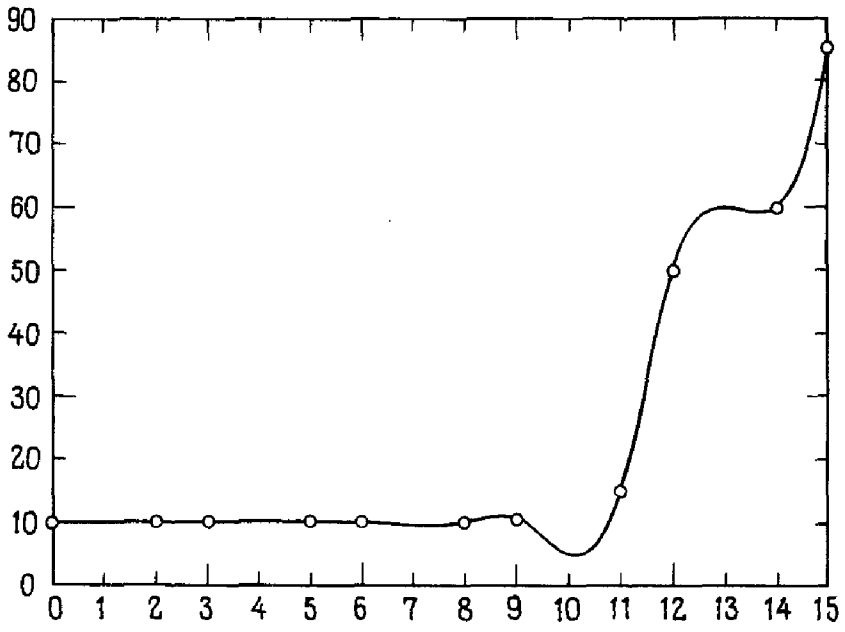


Рис. 4.11. Сплайн-интерполянт для монотонных данных.

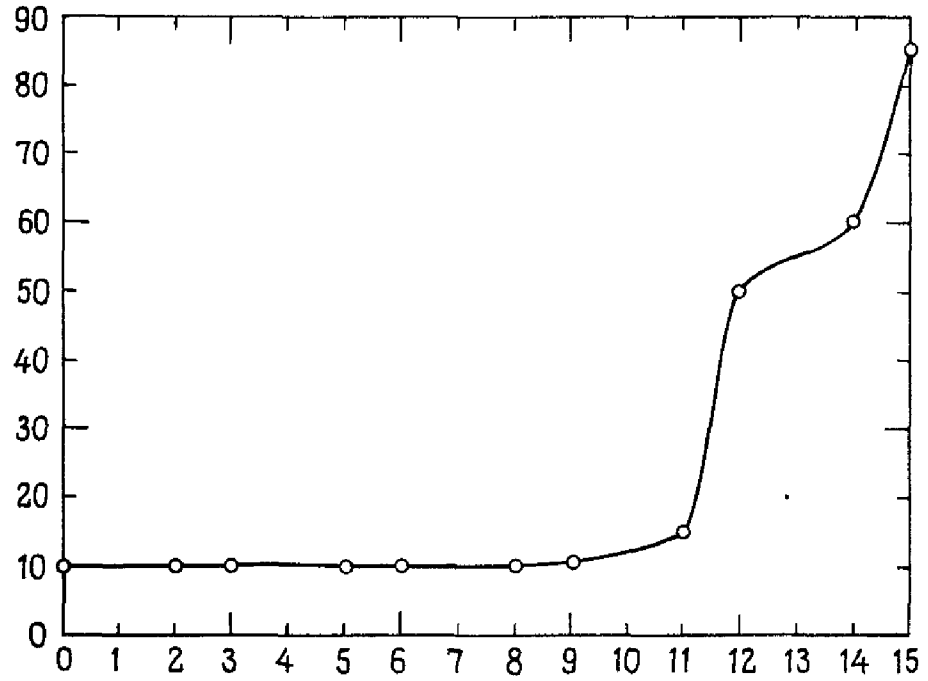


Рис. 4.12. Приятный на вид интерполянт для монотонных данных.

Существует несколько способов обойти эту проблему. Обсудим вкратце два популярных метода.

(1) Рассмотрим эрмитов кубический интерполянт для данных из предыдущего абзаца. Как вы теперь уже знаете, существует много возможных интерполянтов с различными d_i. Можно попробовать выбрать среди них тот, который всюду возрастает. Вообще в статьях Фритча, Карлсона, Бродли и других рассматривалась задача о выборе эрмитова кубического интерполянта, который был бы «наиболее приятен на вид» в некотором интуитивном смысле. Такой алгоритм по затратам сравним с вычислением сплайна, но получающийся при этом интерполянт часто оказывается намного более полезным. На рис. 4.12 изображен один из таких интерполянтов для тех же данных. Построение гарантирует, что если данные монотонны, то таким же будет и интерполянт. Если данные немонотонны (как значения функции Рунге), то эти алгоритмы предоставляют различные возможности для работы в области, где данные изменяют характер монотонности. В этом случае точность вычисления d_i может не быть оптимальной. Поэтому такой метод может оказаться мало пригодным для немонотонных данных, если интерполянт затем будут дифференцировать.

(2) Слово «сплайн» впервые стали употреблять чертежники. Сплайном они называли гибкий стержень, который можно закрепить в специально выбранных точках (называемых узлами) вдоль его длины. Чертежник закреплял сплайн в узлах так, чтобы он проходил через определенные позиции, а затем переводил форму

полученной кривой на чертеж. Наше определение сплайна является математическим описанием этого физического устройства. Между 1966 и 1974 гг. Клейн и другие заново осмыслили это определение с целью устранения некоторых проблем, описанных выше. Они руководствовались физическим понятием эластичной нити, которая проходит через кольца, закрепленные в точках интерполяции, и которую можно натягивать за концы для устранения лишних провисаний. Если потянуть достаточно сильно, то в результате получится ломаная линия, т. е. кусочно-линейный интерполянт. Такой подход приводит к объекту, который они назвали «сплайном с натяжением». Похоже, что он хорошо подходит для интерполяции по неудобным данным, а кроме того, имеет то преимущество, что пользователь задает один интуитивно ясный параметр σ – «натяжение». Получающийся интерполянт уже не является кусочно-полиномиальным, он содержит $\exp(\sigma x)$ и $\exp(-\sigma x)$. Это не является недостатком, так как и кусочно-полиномиальные функции достаточно произвольны. Заинтересованный читатель может обратиться к статье [Cline, 1974].

При работе по обеим методикам пользователю важно знать, что никакую интерполяцию (при помощи сплайна, эрмитовой кубической функции, сплайна с натяжением и т. д.) не нужно рассматривать как «черный ящик». Повторяем наш совет исследовать график интерполянта во всех случаях, где только можно.

4.13. Кривые Безье

Важной задачей в машинном проектировании (МП) является функциональное описание кривой (или поверхности), заданной графически. Она часто решается в интерактивном режиме: дизайнер сидит за графическим терминалом и перебирает различные значения параметров до тех пор, пока не получит «нужную» функцию. В 1962 г. Безье и де Кастельо независимо разработали метод решения этой задачи в процессе своей работы на системах МП во французских автомобильных фирмах Рено и Ситроен. Программное обеспечение фирмы Рено было вскоре описано в нескольких публикациях Безье, поэтому лежащая в основе метода теория носит его имя. Такие методы теперь стали математическим фундаментом многих систем МП и основным инструментом машинной графики. Хорошее введение в них дается в книге [Bartels et al, 1987].

Начнем с изучения простого множества функций, называемых *базисными функциями Бернштейна*[1]. Предположим, что все точки данных

[1] Сергей Натанович Бернштейн (1880–1968), сын преподавателя физиологии из Одессы, получил образование в Париже, но вернулся в Россию, где и преподавал на Украине, а затем в Ленинграде и Москве. В его кандидатской и докторской диссертациях решены девятнадцатая и двадцатая из знаменитых проблем Гильберта. В основном он работал в областях дифференциальных уравнений в част-

находятся в интервале $a \leqslant x \leqslant b$. Тогда i-я базисная функция Бернштейна является полиномом степени n вида

$$B_i^n(x) = \binom{n}{i} \frac{(b-x)^{n-i}(x-a)^i}{(b-a)^n}, \quad i = 0, \ldots, n.$$

Здесь через $\binom{n}{i}$ обозначается биномиальный коэффициент:

$$\binom{n}{i} \equiv \frac{n!}{i!(n-i)!}.$$

Например, для $n = 3$ четыре кубические функции базиса Бернштейна имеют вид

$$B_0^3(x) = \frac{(b-x)^3}{(b-a)^3}, \qquad B_1^3(x) = 3\frac{(b-x)^2(x-a)}{(b-a)^3},$$
$$B_2^3(x) = 3\frac{(b-x)(x-a)^2}{(b-a)^3}, \qquad B_3^3(x) = \frac{(x-a)^3}{(b-a)^3}.$$

На самом деле для вычисления значения бернштейновской базисной функции в точке x определение не применяют. Вместо этого используют рекуррентные формулы

$$B_0^0(x) = 1, \; B_{-1}^n(x) = 0,$$
$$B_i^n(x) = \frac{(b-x)B_i^{n-1}(x) + (x-a)B_{i-1}^{n-1}(x)}{(b-a)}, \quad 0 \leqslant i \leqslant n,$$
$$B_{n+1}^n(x) = 0,$$

которые менее чувствительны к ошибкам округления. Приведем программу для вычисления $B_i^n(x)$, когда $a = 0$, $b = 1$ (позже мы увидим, что это наиболее важный случай).

```
      SUBROUTINE BP (N, B, X)
C
C     Вычисляет значения (в точке X) N + 1 функций базиса
C     Бернштейна степени N на [0, 1]
C
C     Ввод: N (целое). GE. 0
C     X (действительное) — абсцисса при вычислении полиномов
C
```

ных производных, теории аппроксимации (здесь ему принадлежит термин «конструктивная теория функций») и теории вероятностей. Имя Бернштейна также связано с аппроксимационной теоремой Вейерштрасса, так как ее можно доказать, с умом используя классическую вероятностную теорему, которая называется слабым законом больших чисел. Ключевым пунктом в этом коротком доказательстве является использование биномиального распределения, которое оказывается замаскированной базисной функцией Бернштейна.

```
C     Вывод: B(0:N) (действительный массив)
C     B(I) = N!/(I!*(N - I)!)*(1 - X)**(N - I)*X**I I = 0, 1, ..., N
      INTEGER N, C, R
      REAL B(0:N), X
C
      IF (N.EQ.0) THEN
         B(0) = 1.0
      ELSE IF (N.GT.0) THEN
         DO 2 C = 1, N
         IF(C.EQ.1) THEN
               B(1) = X
         ELSE
               B(C) = X*B(C - 1)
         END IF
C
         DO 1 R = C - 1, 1, - 1
            B(R) = X*B(R - 1) + (1.0 - X)*B(R)
    1    CONTINUE
         IF (C.EQ.1)THEN
C              B(0) = 1.0 - X
         ELSE
               B(0) = (1.0 - X) *B(0)
         END IF
    2  CONTINUE
       END IF
       RETURN
       END
```

Поскольку каждая функция B_i^n представляет собой полином степени n, то и их линейная комбинация

$$P_n(x) = \sum_{i=0}^{n} p_i B_i^n(x)$$

является таким же полиномом, и можно попробовать найти такие p_i, чтобы полином P_n производил интерполяцию по $n + 1$ точкам: $P_n(x_i) = y_i$. (В данном параграфе предполагаем, что имеется не n, а $n + 1$ точка данных, и для удобства мы нумеруем их от 0 до n.) Это можно сделать, решая систему из $n + 1$ уравнений для p_i, как описано в § 1. Используя $B_i^n(x)$ вместо степенного базиса или базиса Лагранжа, получаем замену базиса в смысле § 3.

Как соотносятся коэффициенты p_i и y_i? Если пользоваться представлением по целым неотрицательным степеням переменной x, то коэффициенты при x_i не имеют непосредственного отношения к данным. Иначе говоря, трудно угадать поведение полинома высокой степени, такого, как $3 - x + 2x^2 + x^3 + 4x^4 - 1.3x^5$, глядя на его коэффициенты. В лагранжевом представлении коэффициенты *в точности* являются ордина-

тами точек данных, но даже это обстоятельство не помогает угадать поведение полинома, поскольку он может сильно колебаться между интерполируемыми значениями. Первый и последний коэффициенты $P_n(x)$, т.е. p_0 и p_n, также равны заданным величинам: из-за того, что $B_i^n(a) = 0$ при $i > 0$ и $B_0^n(a) = 1$, коэффициент p_0 должен быть равным y_0; аналогично $p_n = y_n$. Замечательным свойством данного представления является то, что остальные коэффициенты p_i *почти* столь же просто соотносятся с данными y_i, и с их помощью можно угадывать форму кривой. Связь между p_i и формой кривой столь сильна, что из-за нее такие кривые стали использовать в качестве основного инструмента при проектировании. Сказанное означает, что если каким-то путем, хотя бы угадыванием, мы получили p_i, то еще до вычисления полинома $P_n(x)$ можно интуитивно предсказать, на что он будет похож. При полиномиальной интерполяции знание, что интерполянт проходит через заданные точки, не позволяет хорошо предсказать поведение этой функции. Теперь же мы можем строить интерполянт, угадывая коэффициенты p_i и глядя затем на $P_n(x)$. Если полином получается неудовлетворительным, то вносим небольшие поправки в p_i и повторяем процедуру. Достоинство такого подхода в том, что его можно применять, даже когда интерполяция не является конечной целью. Во многих случаях у дизайнера вовсе нет никаких данных для интерполяции, а есть только общее представление о форме желаемой кривой.

Как же угадать коэффициенты p_i, которые дадут приемлемую кривую? Основная идея – определить p_i графически (см. рис. 4.13). С этой целью поставим в соответствие каждому p_i точку (t_i, p_i) на плоскости, которая называется *управляющей точкой*, и положим по определению

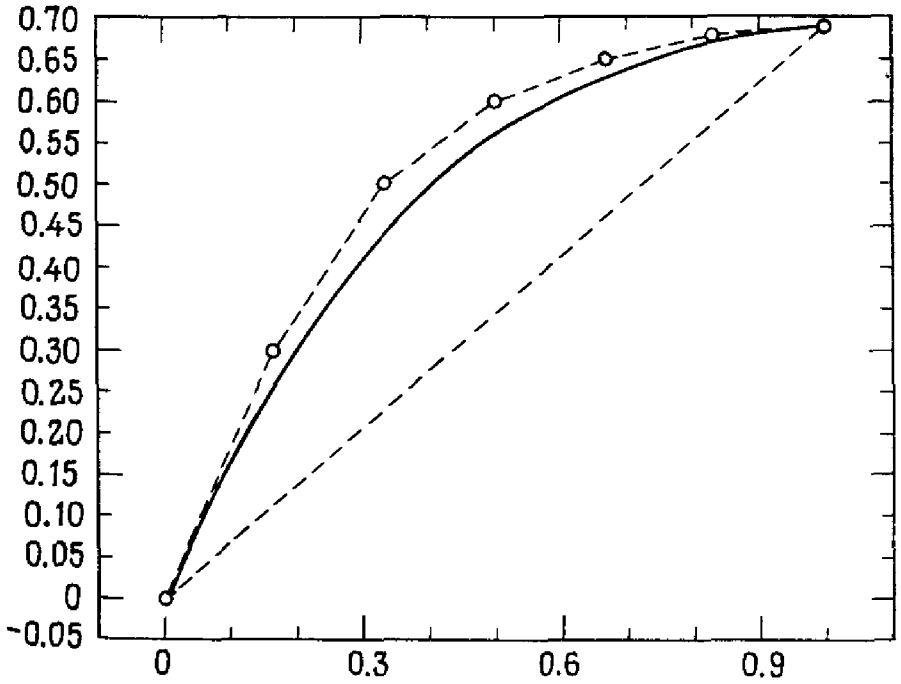


Рис. 4.13. Управляющие точки, ломаная Безье и $P_n(x)$.

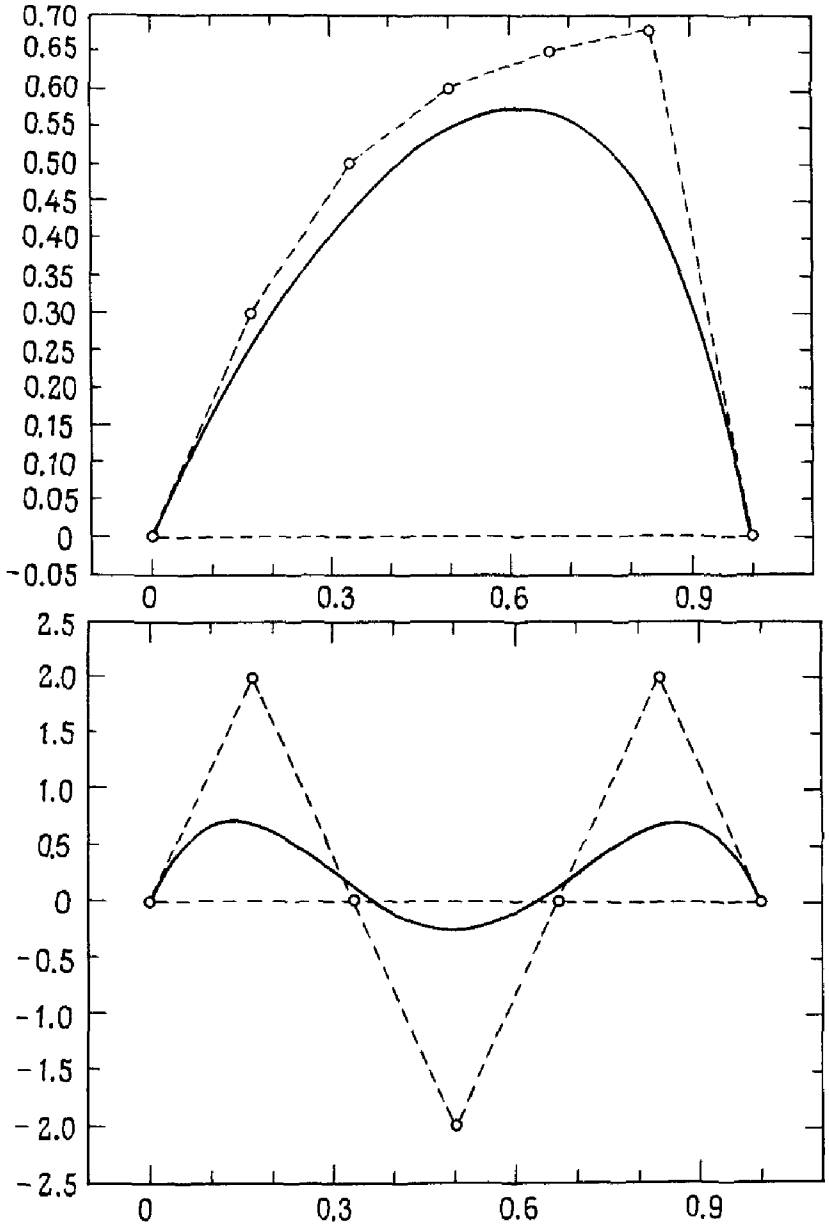


Рис. 4.13. (продолжение) Управляющие точки, ломаная Безье и $P_n(x)$.

$t_i = x_0 + i(x_n - x_0)/n = a + i(b - a)/n$, чтобы равномерно распределить точки t_i по интересующему нас интервалу. Заметим, что $t_0 = x_0$ и $t_n = x_n$, но остальные точки t_i обычно не совпадают с x_i. Для каждого пробного набора управляющих точек можно построить *ломаную Безье:* начиная с (t_0, p_0), соединим последовательные управляющие точки отрезками прямых линий и, наконец, соединим (t_n, p_n) с (t_0, p_0). Можно показать, что эта ломаная дает грубое представление о форме $P_n(x)$. Известно также, что функция $P_n(x)$ обладает свойством уменьшать вариации. Это значит, что если функции p_i монотонны, то и $P_n(x)$ – монотонная функция, если они выпуклы (вогнуты), то и функция $P_n(x)$ такая же. Кроме того, $P_n(x)$ полностью лежит внутри выпуклой оболочки, натянутой на

управляющие точки. (Множество S выпукло, если вместе с любыми двумя точками из S оно содержит и соединяющий их отрезок прямой. Выпуклая оболочка – это наименьшее выпуклое множество, содержащее все управляющие точки.) Их выпуклая оболочка часто совпадает с областью, ограниченной ломаной Безье. Поэтому сдвиг управляющей точки вверх или вниз оказывает прямое и интуитивно ясное воздействие на функцию P_n.

Обычно число и положение управляющих точек заранее неизвестны, исходят лишь из наименьшего числа, необходимого для получения удовлетворительной кривой. Эта процедура не является интерполяцией, потому что нельзя точно сказать заранее, через какие точки пройдет $P_n(x)$, за исключением концевых точек. Но на практике можно заставить $P_n(x)$ пройти через какую угодно точку, регулируя положение управляющих точек. Если дизайнер хочет, чтобы кривая проходила через определенную точку, он локализует ее графически, и некоторое незначительное расширение описанной теории заставит кривую обеспечить интерполяцию в этой точке.

На рис. 4.13 изображены ломаные Безье, построенные по нескольким наборам управляющих точек, и получающиеся в результате полиномы $P_n(x)$.

Одна трудность, связанная с этим подходом, состоит в том, что невозможно сдвигать управляющие точки вправо или влево. Как только выбрано n, точки t_i фиксируются: $t_i = a + i(b-a)/n$. Это не соответствует интуиции дизайнера, который видит естественный эффект от сдвигов вверх и вниз и хочет иметь аналогичную возможность сдвигать управляющие точки в любом направлении. Чтобы обеспечить такую возможность, сделаем некоторое обобщение. Пусть $\mathbf{p}_i$ – точка на плоскости; определим векторную *кривую Безье*

$$\mathbf{P}_n(x) = \sum_{i=0}^{n} \mathbf{p}_i B_i^n(x).$$

Когда x изменяется от a до b, кривая $\mathbf{P}_n(x)$ описывает траекторию на плоскости, начинающуюся в $\mathbf{p}_0$ и кончающуюся в $\mathbf{p}_n$. В этом варианте $\mathbf{p}_i$ могут быть произвольными точками. Нет необходимости предполагать что-либо относительно t_i и даже что $\mathbf{p}_i$ лежит слева от $\mathbf{p}_{i+1}$. Следовательно, при помощи $\mathbf{P}_n(x)$ можно описать намного более сложные кривые. Например, если $\mathbf{p}_0 = \mathbf{p}_n$, то $\mathbf{P}_n(x)$ будет замкнутой кривой. Для векторной кривой Безье числа a и b произвольны и обычно выбираются равными нулю и единице. Другими словами, x – это просто параметр, изменяющийся вдоль кривой. В литературе прилагательное «векторная» часто опускают. На рис. 4.14. изображена такая кривая Безье.

Пример 4.10. Сравнение $\mathbf{P}_n$ и $\mathbf{P}_n$.

Три квадратические базисные функции Бернштейна на интервале [0, 1] равны $(1-x)^2$, $2x(1-x)$ и x^2. Рассмотрим три величины $p_0 = 1$, $p_1 = 0$, $p_2 = 1$. В случае, когда строится $P_n(x)$, им ставятся в соответствие

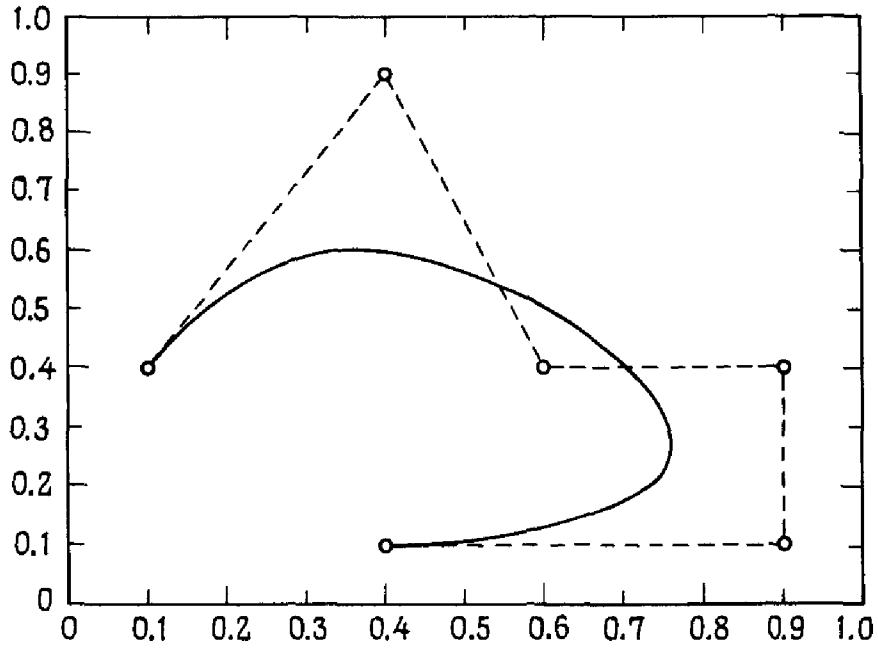


Рис. 4.14. Кривая Безье.

управляющие точки (0, 1), (1/2, 0), (1, 1), которые дают

$$P_2(x) = (1-x)^2 + 0 \cdot 2x(1-x) + x^2 = 1 - 2x + 2x^2.$$

Заметим, что при $x = 3/4$ получаем $P_2(x) = 20/32$. С другой стороны, для $\mathbf{P}_n(x)$ ограничений на управляющие точки нет; возьмем для иллюстрации $\mathbf{p}_0 = (0, 1)$, $\mathbf{p}_1 = (3/4, 0)$ и $\mathbf{p}_2 = (1, 1)$. Тогда получится кривая Безье

$$\mathbf{P}_2(x) = (0 \cdot (1-x)^2 + 3/4 \cdot 2x(1-x) + 1 \cdot x^2,\ (1-x)^2 + x^2) =$$
$$= (1 - x/2 - x^2/2,\ 1 - 2x + 2x^2).$$

При $x = 3/4$ получаем $\mathbf{P}_2(x) = (27/32, 20/32)$, т.е. абсцисса отлична от

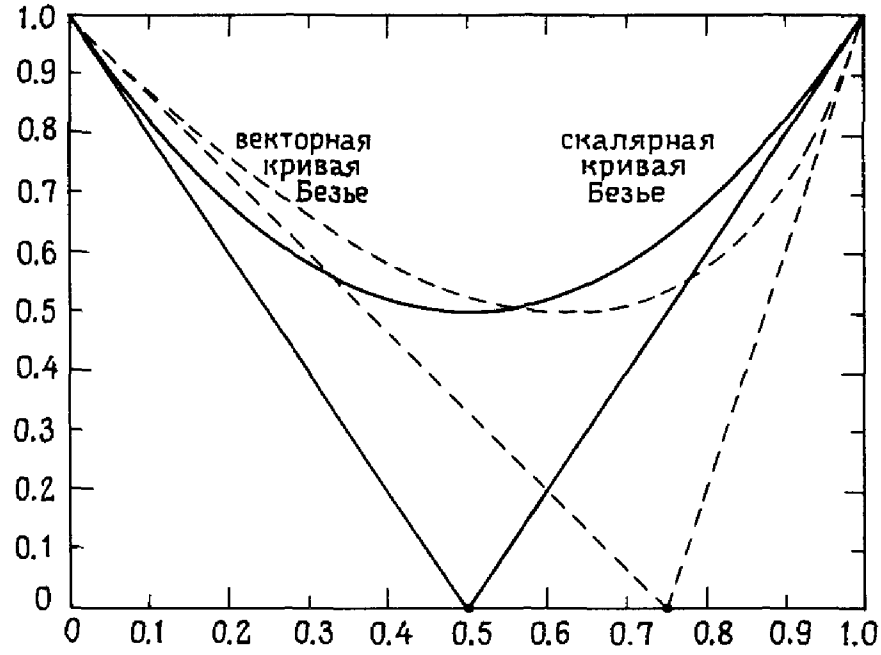


Рис. 4.15. Сравнение $P_n(x)$ и $P(x)$.

3/4. При такой форме задания кривой Безье x – это только параметр, а $\mathbf{P}_2(x)$ описывает некоторую кривую между первой и последней управляющими точками. Когда абсцисса точки на кривой $\mathbf{P}_2$ имеет значение 3/4, ее ордината приблизительно равна 0.536. На рис. 4.15 показаны две кривые P_2 и $\mathbf{P}_2$. На каждой кривой указана точка, соответствующая $x = 3/4$. Наконец, построены выпуклые оболочки каждого из двух наборов управляющих точек. Чтобы проверить, правильно ли вы все поняли, постройте $\mathbf{P}_2(x)$ по управляющим точкам (0, 1), (1/2, 0), (1, 1) и убедитесь, что кривая $\mathbf{P}_2(x)$ совпадает с $P_2(x)$. □

*4.14. B-сплайны

Из § 11 мы знаем, что сплайн-интерполянт можно записать в виде

$$C(x) = \sum y_i c_i(x) + d_i \hat{c}_i(x),$$

где значения d_i находятся из системы линейных уравнений. Но заметим, что ни $c_i(x)$, ни $\hat{c}_i(x)$ сами по себе сплайнами не являются, так как не являются дважды дифференцируемыми. Для многих приложений полезно уметь записывать интерполянт в виде линейной комбинации функций, которые сами являются сплайнами:

$$C(x) = \sum a_i N_i^3(x).$$

Наиболее часто используемый и гибкий способ получается тогда, когда функции $N_i^3(x)$ представляют собой *B-сплайны*. Верхний индекс 3 указывает на то, что *B*-сплайн является кубическим; это общепринятое обозначение, поскольку существуют и другие типы *B*-сплайнов. График *B*-сплайна похож на график одной из функций $c_i(x)$ на рис. 4.10 тем, что на большей части рассматриваемого интервала он равен нулю и что при равномерной сетке узлов каждый *B*-сплайн выглядит как результат сдвига любого другого *B*-сплайна. На рис. 4.16 показаны типичные *B*-сплайны. Заметим, что *B*-сплайн всегда неотрицателен и отличен от нуля не более чем на четырех частичных интервалах. Следовательно, для любого фиксированного x написанная выше сумма для $C(x)$ содержит не более четырех ненулевых членов. В приложениях это приводит к ленточным матрицам. Например, с помощью *B*-сплайнов можно построить сплайн-интерполянт $C(x)$ с коэффициентами d_i, которые находятся из ленточной системы, подобной той, что упоминалась в § 11.

Кубический *B*-сплайн – это сплайн, т. е. кусочно-полиномиальная функция с двумя непрерывными производными. Для некоторых приложений важно овладеть деталями построения *B*-сплайнов. Но в данном учебнике мы обсудим их лишь поверхностно, отослав заинтересованного читателя к любой из указанных выше работ.

Программы, оперирующие *B*-сплайнами, обычно требуют в качестве

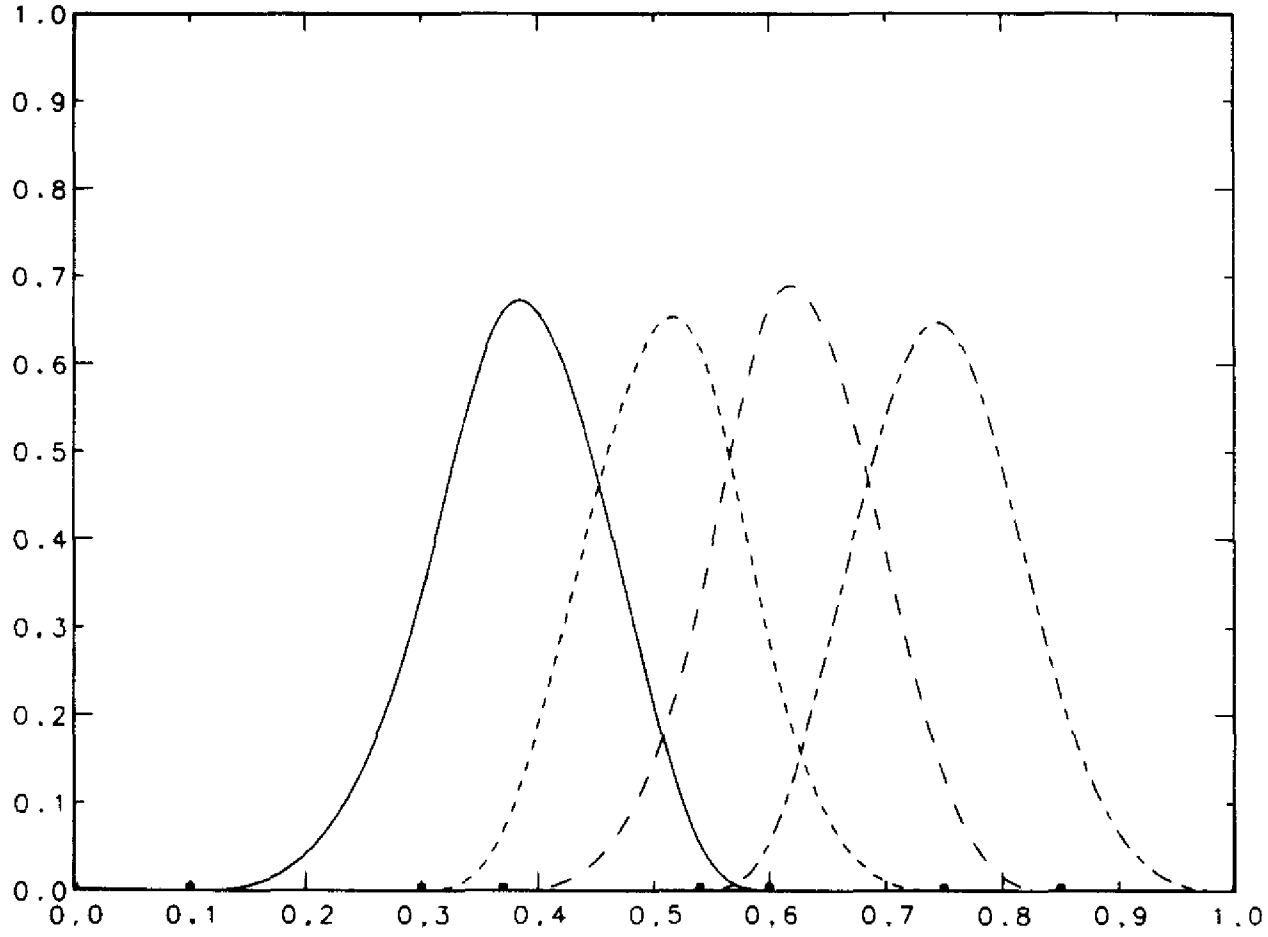


Рис. 4.16. Типичные кубические B-сплайны с различными узлами.

входного параметра массив узлов. Большинство пользователей это приводит в замешательство. Путаница происходит из-за того, что определение B-сплайна допускает совпадение некоторых узлов. Хотя это может показаться неестественным для задач интерполяции, указанное обстоятельство позволяет работать с функциями, у которых нет двух непрерывных производных. Если $C(x)$ – линейная комбинация B-сплайнов с двумя совпадающими узлами, скажем в точке η, то C – это функция, имеющая две непрерывные производные всюду, кроме точки η, где она имеет только одну непрерывную производную. Если в точку η попадают три узла, то в ней функция $C(x)$ непрерывна, а если четыре узла, то $C(x)$ претерпевает в точке η разрыв первого рода. Это обобщение позволяет использовать программы интерполяции B-сплайнами для решения важной задачи построения интерполянта, составленного из двух гладких частей, которые стыкуются в некоторой точке с нарушением гладкости. Например, такая задача возникает при проектировании кузова автомобиля, линия которого имеет излом. На рис. 4.17 показано несколько B-сплайнов в случаях, когда некоторые узлы совпадают.

Правила задания узлов B-сплайнов несложны. Чтобы построить кубический сплайн-интерполянт по n заданным точкам с помощью

B-сплайнов, нужно $n + 6$ узлов. В выборе расположения узлов допускается некоторая гибкость, но обычно выбирают по четыре узла в каждой из граничных точек x_1 и x_n и по одному узлу в каждой из внутренних заданных точек. Если интерполянт в некоторой точке должен иметь меньше двух производных, то в ней добавляют еще один или более дополнительный узел. Самих *B*-сплайнов будет на четыре меньше числа узлов (т. е. по крайней мере $n + 2$), а на выходе программы интерполяции получится массив стольких же коэффициентов интерполянта a_i. Интерполянт пройдет через заданные точки, будет иметь нужное число производных и удовлетворять еще двум дополнительным условиям. Если совпадающих внутренних узлов нет, то полученный таким способом сплайн будет иметь две непрерывные производные и совпадать с интерполянтом, построенным в эрмитовом кубическом представлении при тех же дополнительных условиях. Коэффициенты при *B*-сплайнах находятся из системы линейных уравнений, подобной рассмотренной в § 11.

Одним из наиболее интересных фактов, относящихся к *B*-сплайнам, является то, что существует замечательная связь между *B*-сплайнами и кривыми Безье, а коэффициенты при *B*-сплайнах имеют физическое значение, аналогичное коэффициентам кривой Безье. В частности, по аналогии с § 13, *B*-сплайновые кривые Безье определяются как

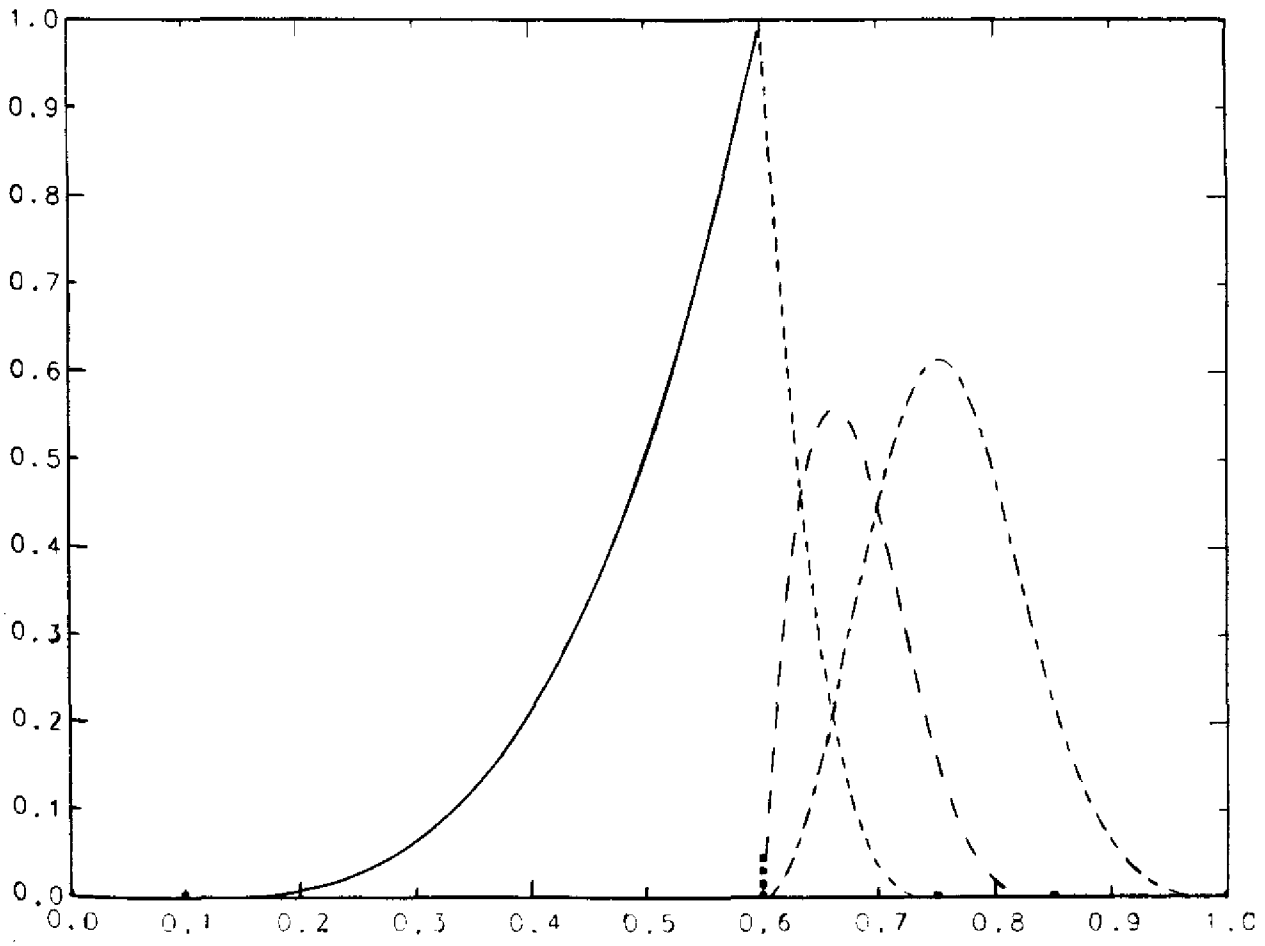


Рис. 4.17. Типичные кубические *B*-сплайны с совпадающими узлами.

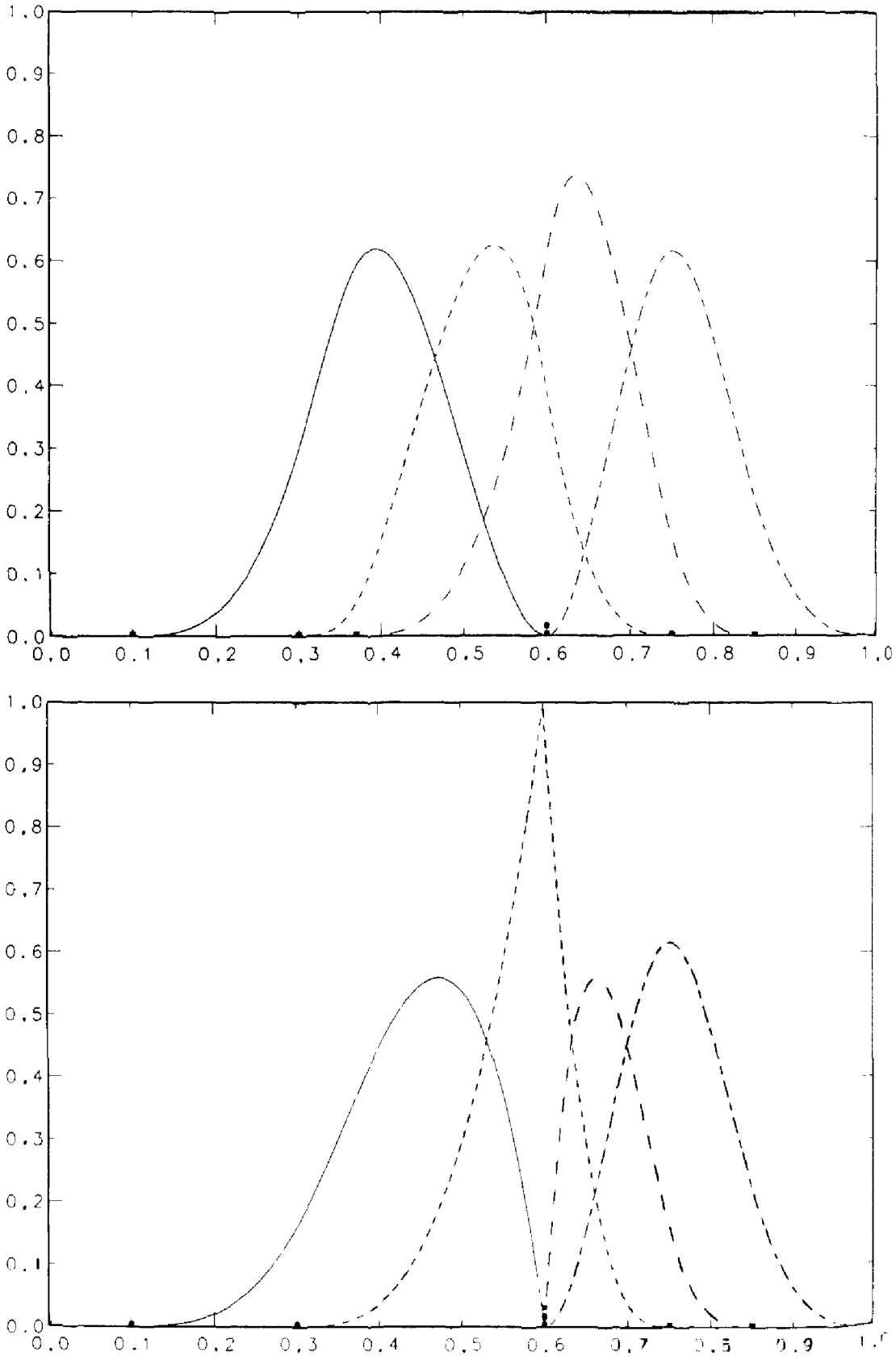


Рис. 4.17. (продолжение) Типичные кубические *B*-сплайны с совпадающими узлами.

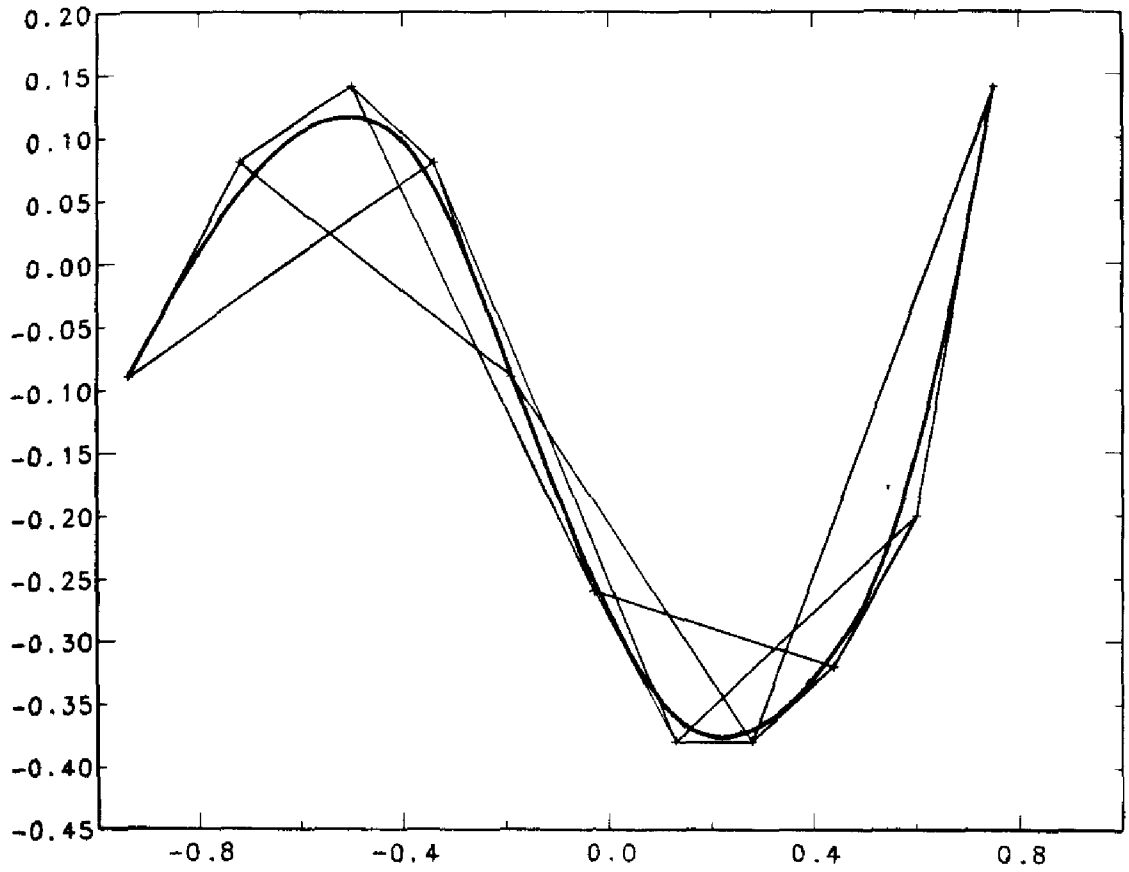


Рис. 4.18. *B*-сплайновая кривая Безье.

$$\mathbf{B}(x) = \sum_{i=0}^{n+1} \mathbf{p}_i N_i^3(x),$$

где $\mathbf{p}_i$ – произвольные точки на плоскости, а узлы *B*-сплайнов расположены равномерно в интервале [0, 1]. Нанесем на график управляющие точки $\mathbf{p}_i$, $i = 0, \ldots, n+1$, и построим выпуклую оболочку точек $\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$, затем выпуклую оболочку точек $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$, $\mathbf{p}_4$ и т. д., используя каждый раз четыре точки, получающиеся при добавлении одной новой точки справа и отбрасывании одной слева. Можно показать, что кривая $\mathbf{B}(x)$ лежит в объединении таких ломаных. На рис. 4.18 показаны некоторые управляющие точки, область, составленная из этих ломаных, и *B*-сплайновая кривая Безье $\mathbf{B}(x)$. Таким образом, изменение коэффициентов при *B*-сплайнах оказывает точно такое же влияние, как изменение управляющих точек кривой Безье (см. книгу [Barnhill, Riesenfeld, 1979].)

4.15. Задачи

4.1. Постройте $n = 11$ точек данных, взяв

$$\left.\begin{aligned} x_i &= \frac{i-1}{10} \\ y_i &= \operatorname{erf}(x_i) \end{aligned}\right\} \quad i = 1, \ldots, 11.$$

Значения функции $\mathrm{erf}(x)$ найдите в таблицах или получите при помощи подпрограммы на вашем компьютере.

(а) Попробуйте интерполировать данные при помощи полинома десятой степени. Заполните матрицу, как описано в § 2, и используйте SGEFS для оценивания ее числа обусловленности. Даже если число обусловленности велико и коэффициенты неточны, все же возможно, что значения интерполирующего полинома не будут сильно искажены ошибками округления. Почему? Найдите интерполирующий полином и примените метод Горнера для вычисления его значений между заданными точками. Сделайте то же самое, используя представление Лагранжа. Сравните результаты в обоих случаях со значениями $\mathrm{erf}(x)$. Какова наибольшая ошибка?

(б) Примените для интерполяции программу PCHEZ, используя как кубический эрмитов интерполянт, так и кубический сплайн-интерполянт. Вычислите с помощью PCHEV значения этих интерполянтов в тех же точках, что и в (а), и сравните результаты.

4.2. В табл. 4.1 приведены данные американского Бюро переписи населения, характеризующие численность населения Соединенных Штатов.

Таблица 4.1

Год	Население
1900	75,994,575
1910	91,972,266
1920	105,710,620
1930	122,775,046
1940	131,669,275
1950	150,697,361
1960	179,323,175
1970	203,235,298

(а) Существует единственный полином седьмой степени, интерполирующий эти данные. Рассмотрим четыре представления этого полинома:

$$\sum_{j=0}^{7} a_j t^j, \quad \sum_{j=0}^{7} b_j (t-1900)^j,$$

$$\sum_{j=0}^{7} c_j (t-1935)^j, \quad \sum_{j=0}^{7} d_j ((t-1935)/35)^j.$$

В точной арифметике не имеет значения, каким представлением пользоваться, но с вычислительной точки зрения одни предпочтительнее

других. Для каждого из этих полиномов запишите матрицу системы для коэффициентов размера 8 × 8 и оцените ее число обусловленности при помощи SGEFS. Если матрица не слишком плохо обусловлена, примените SGEFS для поиска коэффициентов. Насколько хорошо построенные полиномы воспроизводят исходные данные? Объясните, почему графики последних трех полиномов похожи, даже если их коэффициенты содержат мало или вовсе не содержат значащих цифр (IND = – 10).

(б) Интерполируйте данные найденным в пункте (а) полиномиальным интерполянтом, отвечающим наилучшей обусловленности матрицы, а также кубическим эрмитовым интерполянтом при помощи PCHEZ. Постройте графики полинома и кусочно-полиномиальной функции с шагом в один год в промежутке между 1900 г. и 1980 г. Они должны дать весьма хорошее соответствие вплоть до 1970 г., но между 1970 г. и 1980 г. поведут себя уже совершенно по-разному. Численность населения в 1980 г. составила 226 547 082. Какой из интерполянтов дает более точное предсказание?

4.3. В табл. 4.2 приведены данные, которые были использованы для рис. 4.11. Интерполируйте их кубическим сплайном при помощи

Таблица 4.2

x	y
0	10.0
2	10.0
3	10.0
5	10.0
6	10.0
8	10.0
9	10.5
11	15.0
12	50.0
14	60.0
15	85.0

PCHEZ. Является ли интерполянт монотонным? Сделайте то же самое, задав параметр SPLINE = .FALSE..

4.4. Параметрическая интерполяция. При применении компьютеров в книгопечатании возникает задача поиска интерполянта по точкам, которые лежат на плоской кривой, например на напечатанной заглавной букве *S*. Ее форму нельзя описать какой-либо функцией от x, потому что функция не была бы однозначной. Один из способов – занумеровать

точки $\mathbf{P}_1, \ldots, \mathbf{P}_n$ вдоль кривой. Пусть d_i – расстояния (по отрезкам прямых) между $\mathbf{P}_i$ и $\mathbf{P}_{i+1}$, $i = 1, \ldots, n-1$, и пусть $t_i = \sum_{j=1}^{i-1} d_j$, $i = 1, \ldots, n$. Это значит, что $t_1 = 0$, $t_2 = d_1$, $t_3 = d_1 + d_2$ и т. д. Зададим точки покоординатно: $\mathbf{P}_i = (x_i, y_i)$ и рассмотрим два набора данных $\{(t_i, x_i)\}$ и $\{(t_i, y_i)\}$, $i = 1, \ldots, n$. Можно независимо интерполировать их функциями $f(t)$ и $g(t)$ соответственно. Тогда $\mathbf{P}(t) \equiv (f(t), g(t))$, $0 \leqslant t \leqslant t_n$, при фиксированном t задает точку на плоскости, а при изменении t от 0 до t_n интерполирует данные и в случае удачи воспроизводит форму буквы. Можно считать t параметром, который изменяется вдоль кривой.

Используя миллиметровую бумагу, нарисуйте букву S и измерьте координаты восьми точек $\mathbf{P}_i$ на ней. Интерполируйте эти данные с помощью PCHEZ

(а) с параметром SPLINE = .FALSE.;

(б) с параметром SPLINE = .TRUE.

Изобразите полученные результаты и посмотрите, насколько верно они описывают букву. Если в качестве модельной взять замкнутую кривую, например букву O, то можно получить более хорошие результаты, требуя периодичности интерполянта.

4.5. Предположим, что даны n точек $(x_i, f(x_i))$, $i = 1, \ldots, n$, и желательно найти точки, в которых $f'(x) = 0$.

(а) Почему идея интерполировать полиномом $p(x)$ степени $n-1$, а затем решить уравнение $p'(x) = 0$, не является хорошей?

(б) Опишите эффективный алгоритм решения этой задачи с использованием кубических интерполирующих сплайнов.

(в) Напишите программу, реализующую этот алгоритм и использующую PCHEZ. Опробуйте эту программу на данных из задачи 4.3.

4.6. Эта задача показывает, почему кусочно-квадратические функции применяются не слишком часто. Кусочно-квадратическая функция $Q(x)$ – это функция, определенная при всех x, которая является квадратической между соседними узлами. Если даны узлы $x_1 < x_2 < \ldots < x_n$, то между x_i и x_{i+1} функция $Q(x)$ определяется тремя параметрами и может быть записана в виде

$$Q(x) = a_i + b_i(x - x_i) + c_i(x - x_i)(x - x_{i+1}),$$
$$x_i \leqslant x \leqslant x_{i+1},\ i = 1, \ldots, n-1.$$

Требуя, чтобы функция $Q(x)$ обеспечивала интерполяцию по всем узлам, т. е. $Q(x_i) = y_i$, определим a_i и b_i.

(а) Покажите, что $Q(x)$ будет обеспечивать интерполяцию, если на каждом частичном интервале ее можно представить в виде.

$$Q(x) = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i) + c_i(x - x_i)(x - x_{i+1}),$$

и сделайте вывод о непрерывности $Q(x)$ при всех x.

(б) Покажите, что такой интерполянт Q имеет к тому же одну непрерывную производную, если

$$c_i = \frac{1}{x_i - x_{i+1}}\left[c_{i-1}(x_i - x_{i-1}) + \frac{y_i - y_{i-1}}{x_i - x_{i-1}} - \frac{y_{i+1} - y_i}{x_{i+1} - x_i}\right],$$
$$i = 2, \ldots, n-1.$$

Если задать c_1, то это однозначно определит все остальные c_i и, следовательно, $Q(x)$.

(в) Используя эту конструкцию, рассмотрим интерполяцию по двум наборам данных, которые отличаются только первым значением: y_1 и $\tilde{y}_1$. Обозначим соответствующие интерполянты $Q(x)$ и $\tilde{Q}(x)$. Покажите, что если в них обоих использовать одну и ту же величину c_1 и если точки x_i расположены равномерно, то

$$\tilde{c}_1 = c_1, \quad \tilde{c}_i = c_i + (-1)^i\left(\frac{y_1 - \tilde{y}_1}{h^2}\right), \quad i = 2, \ldots, n-1,$$

и таким образом

$$\tilde{Q}(x) = Q(x) + (-1)^i\left(\frac{y_1 - \tilde{y}_1}{h^2}\right)(x - x_i)(x - x_{i+1}), \quad x_i \leqslant x \leqslant x_{i+1},$$
$$i = 1, 2, \ldots, n-1.$$

(г) Сделайте вывод, что малое изменение в первом заданном значении распространяет свое влияние на всю область интерполяции без уменьшения. Хотя такого влияния и можно избежать, именно из-за него кусочно-квадратические функции используются намного реже, чем кусочно-кубические (см. [de Boor, 1978], с. 75).

4.7. Обратная интерполяция. В обычной постановке задачи интерполяции даны величины x_i, y_i, а требуется найти функцию $y = f(x)$. Однако если величины y_i монотонны, то можно с тем же успехом искать функцию $x = g(y)$. Если такая функция определена, то легко оценить значения x, отвечающие «специальным» значениям y, например значение x, соответствующее $y = 0$. Это называется *обратной интерполяцией*.

Для использования в статистических расчетах требуется заполнить таблицу процентилей функции ошибок $\mathrm{erf}(x)$, т. е. таких чисел x_i, что

$$\frac{i}{10} = \mathrm{erf}(x_i), \quad 0 < i < 10.$$

Найдите шестнадцать значений $\mathrm{erf}(x)$ для $0 \leqslant x \leqslant 1.5$ в таблицах или посчитайте их при помощи подпрограммы для $\mathrm{erf}(x)$. Глядя на эти числа, оцените девять процентилей. Используйте PCHEZ для построения кубического сплайн-интерполянта в форме $x = g(y)$. Нарисуйте график этой функции. Затем примените PCHEV для вычисления $g(y)$ при $y = 0{,}0.1$, 0.2, ..., 0.9. Сравните эти результаты с вашими предыдущими оценками.

4.8. Кривые Безье

(а) При помощи рекуррентных формул из § 13 вычислите четыре кубические функции базиса Бернштейна и нарисуйте их графики на [0, 1].

(б) Напишите программу для построения кривой Безье $\mathbf{P}_n(x)$ по заданным на плоскости точкам. Если у вас есть доступ к графическому терминалу, попробуйте написать программу таким образом, чтобы можно было вводить управляющие точки графически. Воспользуйтесь этой программой при аппроксимации заглавной буквы S в задаче 4.4.

4.9. Предлагаемая в данной и в следующей задачах методика будет особенно интересна всем, кто хочет быстро рисовать окружности, например в интенсивно используемых пакетах графических программ или на спецпроцессорах. Задача состоит в построении двух массивов $X(I)$, $Y(I)$, $I = 1, \ldots, N$, содержащих координаты точек единичной окружности в первом квадранте. Вот как выглядит сегмент программы, которая это делает:

```
      N = 90
      R = PI/180.0
      DO 1 I = 0,90
            RAD = R*I
            X(I) = COS(RAD)
            Y(I) = SIN(RAD)
    1 CONTINUE
```

Но здесь требуется вычислять синус и косинус для каждой точки на окружности. Вместо этого рассмотрите четыре управляющие точки Безье: $\mathbf{p}_0 = (1, 0)$, $\mathbf{p}_1 = (1, 0.552)$, $\mathbf{p}_2 = (0.552, 1)$, $\mathbf{p}_3 = (0, 1)$.

(а) Покажите, что соответствующей векторной кривой Безье будет

$$\mathbf{P}(t) = (1 - 1.344\,t^2 + 0.344\,t^3,\ 1.656t - 0.312t^2 - 0.344t^3).$$

(б) При изменении t от 0 до 1 $\mathbf{P}(t)$ описывает на плоскости кривую, аппроксимирующую четверть окружности. Применяя правило Горнера, вычислите $P(t)$ в 100 точках на [0, 1] и сравните эти значения с точными координатами точек на окружности. Какова наибольшая погрешность? Поскольку параметр t не является величиной, пропорциональной углу, надо немного подумать о том, как измерять погрешность. Быстрее ли такой метод, чем приведенная выше программа? Почему?

4.10. Эта задача предназначена для того, чтобы показать, что если надо вычислить значения полинома на последовательности точек равномерной сетки, т. е. методы, которые сделают это гораздо быстрее схемы Горнера. По таблице N значений функции в равноотстоящих точках можно составить $N - 1$ ее первых разностей $\Delta f_i = f_{i+1} - f_i$. Отсюда можно получить $N - 2$ вторых разностей $\Delta^2 f_i = \Delta f_{i+1} - \Delta f_i$ и т. д. Их часто сводят в таблицы, аналогичные табл. 4.3.

Таблица 4.3

x	$f(x)$	$\Delta f \cdot 10^3$	$\Delta^2 f \cdot 10^6$	$\Delta^3 f \cdot 10^7$	$\Delta^4 f \cdot 10^{11}$
20	0.229314955248				
		.701747247			
22	0.230016702495		.602297		
		.702349544		-.1944	
24	0.230719052039		.600353		.4
		.702949897		-.1940	
26	0.231422001936		.598413		.3
		.703548310		-.1937	
28	0.232125550246		.596476		
		.704144786			
30	0.232829695032				

(а) Напишите программу для проверки последних четырех столбцов табл. 4.3.

(б) Таблицу разностей можно использовать для отыскания ошибок в таблицах. В таком случае входные данные соответствуют столбцам x и $f(x)$. В данной же задаче, где $f(x)$ – полином, вместо этого задана верхняя диагональ таблицы, а нам надо заполнить остальные ее позиции. Получить элементы следующей диагонали можно простым сложением:

$$f_1 = f_0 + \Delta f_0,\ \Delta f_1 = \Delta f_0 + \Delta^2 f_0, \ldots .$$

Покажите, что если $x_0 = 0$, то для $f(t) = a + bt + ct^2 + dt^3$ и $f_i = f(x_0 + i \cdot \delta)$ имеют место равенства

$$\begin{aligned} \Delta f_0 &= b\delta + c\delta^2 + d\delta^3, \\ \Delta^2 f_0 &= 2c\delta^2 + 6d\delta^3, \\ \Delta^3 f_0 &= 6d\delta^3, \\ \Delta^i f_0 &= 0,\ i > 3. \end{aligned}$$

(Вообще, для значений f_i полинома степени p в точках равномерной сетки разности порядка выше p-го равны нулю.) Используйте эти формулы при вычислении элементов на верхней диагонали таблицы для полинома третьей степени из предыдущей задачи. Возьмите $\delta = 0.01$. После заполнения этой диагонали покажите, что $\mathbf{P}(t)$ из задачи 4.9 можно вычислять в точках равномерной сетки, выполняя только сложения, в цикле вида

$$\begin{aligned} f &\leftarrow f + \Delta f, \\ \Delta f &\leftarrow \Delta f + \Delta^2 f, \\ \Delta^2 f &\leftarrow \Delta^2 f + \Delta^3 f. \end{aligned}$$

Сравните этот способ со схемой Горнера.

4.11. Следующая схема сочетает в себе интерполяцию и дух аппроксимации Безье. Рассмотрим множество из $n + 5$ точек на плоскости, обозначим их через $\mathbf{p}_i$, $i = -2, \ldots, n + 2$, $p_i = (x_i, y_i)$. Определим новые точки $\mathbf{p}_{i+1/2}$:

$$\mathbf{p}_{i+1/2} = (1/2 + w)(\mathbf{p}_i + \mathbf{p}_{i+1}) - w(\mathbf{p}_{i-1} + \mathbf{p}_{i+2}),\ i = -1, 0, \ldots, n,\ w > 0.$$

Добавим эти новые точки к исходному множеству, отбросим первую и последнюю точки $\mathbf{p}_{-2}$ и $\mathbf{p}_{n+2}$, перенумеруем точки от -2 до $2n + 2$ и повторим все снова. Было показано, что если $0 < w < 0.25$, то множество точек приближается к некоторой гладкой кривой. Поскольку исходные данные всякий раз сохраняются (за исключением первой и последней точек), эта кривая интерполирует данные.

(а) Примените этот алгоритм для двадцати одного значения функции Рунге. Постройте достаточное число новых точек, чтобы можно было изобразить график получающейся гладкой кривой.

(б) Повторите пункт (а) для данных из задачи 4.3. Число w — это что-то вроде «натяжения»; при $w = 0$ получаются отрезки прямых между точками, $w > 0$ «ослабляет» кривую, хотя она по-прежнему интерполирует исходные данные. Незначительная трудность, связанная с этим алгоритмом, состоит в том, что он не порождает новых точек возле концов набора данных. Для функции Рунге эту трудность можно обойти, добавляя точки к исходным данным слева и справа. Другая возможность — повторять первую и последнюю точки данных, чтобы каждая из них появлялась среди $\mathbf{p}_i$ дважды.

(в) Повторите пункт (а) для данных из задачи 4.4. (Статья [Dyn, 1987] содержит больше математических подробностей об этом алгоритме.)

4.12. Эта задача является иллюстрацией двумерной интерполяции на решетке. Предположим, что задан набор величин $g_{ij} \approx g(x_i, y_j)$, $1 \leqslant i, j \leqslant N$, и мы хотим найти такую функцию двух переменных $f(x, y)$, что $f(x_i, y_j) = g_{ij}$. Здесь предлагается простая методика решения этой задачи. Найдите интерполянты для каждой строки массива данных, т. е. найдите для каждого фиксированного i функцию $f_i(y)$, удовлетворяющую условиям $f_i(y_j) = g_{ij}$, $j = 1, \ldots, N$. Для вычисления значения интерполянта от двух переменных в точке (r, s) сначала вычислите значения каждой из интерполянтов от одной переменной в точке s, т. е. посчитайте $f_i(s)$, $i = 1, \ldots, N$. Затем найдите интерполянт, проходящий через $(x_i, f_i(s))$, $i = 1, \ldots, N$, и вычислите его значение в точке r. Примените эту методику для данных $g_{ij} = \exp(x_i^2 \cdot y_j)$, $x_i = i/10$, $y_j = j/10$, $i, j = 0, \ldots, 10$. Вычислите интерполянт в серединах всех квадратов решетки и распечатайте величины погрешности интерполяции.

4.13. Полиномы Чебышёва. Рассмотрим полиномиальную интерполяцию функции, заданной на фиксированном интервале, скажем на $-1 \leqslant x \leqslant 1$, с фиксированным числом n точек, например интерполяцию

функции Рунге $R(x)$. Погрешность интерполяции представляется произведением, в один из сомножителей которого включается n-я производная функции $R^{(n)}$, а другой содержит произведение скобок $(x-x_1)(x-x_2)\ldots(x-x_n)$. Последний сомножитель не зависит от R, но определяется выбором точек. Предположим, что мы хотим a priori ограничить погрешность, которая возникает при вычислении интерполянта в произвольной точке из $[-1, 1]$. Эта погрешность не превзойдет

$$\max_{-1\leqslant\xi\leqslant 1} |R^{(n)}(\xi)| \cdot \max_{-1\leqslant x\leqslant 1} |(x-x_1)\ldots(x-x_n)|/n!.$$

Единственное, чем мы можем здесь управлять, это выбор точек, поэтому рассмотрим всевозможные точки x_i с целью уменьшить модуль произведения скобок.

(а) Пусть $n=21$ и точки x_i равномерно расположены на $[-1, 1]$. Постройте график полинома $t_n(x)=(x-x_1)(x-x_2)\ldots(x-x_n)$ и найдите максимум его абсолютной величины. Поэкспериментируйте с разными величинами x_i и посмотрите, насколько можно уменьшить наибольшее значение $|t_n(x)|$.

(б) Рассмотрите следующие две последовательности полиномов:

$$T_0(x)=1,\ T_1(x)=x,\ T_{n+1}(x)=2xT_n(x)-T_{n-1}(x),\ n>0,$$
$$\text{и } \tilde{T}_0(x)=1,\ \tilde{T}_n(x)=T_n(x)/(2^{n-1}),\ n>0.$$

Полиномы T_n называются *полиномами Чебышёва*. Вычислите $\tilde{T}_0,\ldots,\tilde{T}_6(x)$ и постройте их графики.

(в) Используя тригонометрические тождества, покажите, что

$$\cos(n+1)\theta = 2\cos\theta\cos n\theta - \cos(n-1)\theta.$$

Полагая формально $x=\cos\theta$ и $\cos n\theta = T_n(x)$, убедитесь, что это то же самое, что рекурсия в пункте (б). Таким образом, можно дать другое определение $T_n(x)$: $T_n(x)=\cos(n\arccos x)$. Пользуясь последним выражением, вычислите $\tilde{T}_0,\ldots,\tilde{T}_6(x)$ и постройте их графики; убедитесь, что получаются те же значения, что в пункте (б).

(г) Покажите, что полиномы $T_n(x)$ имеют нули первого порядка в n точках $x_k=\cos[(2k-1)\pi/(2n)]$, $k=1,\ldots,n$. Постройте график $t_n(x)$, используя эти точки. Почему максимум модуля этого полинома должен быть равен 2^{1-n}? Известно, что никакой другой выбор точек не приведет к меньшему значению максимума абсолютной величины t_n.

(д) Возьмите в качестве точек интерполяции функции Рунге нули полинома $T_{21}(x)$ и постройте график получающегося интерполянта. Есть ли улучшение по сравнению со случаем равноотстоящих точек?

4.14. *B*-сплайны. Пусть $t_0,\ldots,t_{14}$ — пятнадцать узлов на $[0, 1]$, по четыре на каждом конце и семь остальных в точках $i/8$, $i=1,\ldots,7$.

Определим $N_i^0(x)$ как

$$N_i^0(x) = \begin{cases} 1, & t_i \leqslant x < t_{i+1}, \\ 0 \text{ в противном случае,} \end{cases} \qquad i = 0, 1, \ldots, 13.$$

(Если $t_i = t_{i+1}$, то полагаем $N_i^0(t_i) = 1$.) Определим $N_i^1(x)$:

$$N_i^1(x) = (x - t_i)\frac{N_i^0(x)}{t_{i+1} - t_i} + (t_{i+2} - x)\frac{N_{i+1}^0(x)}{t_{i+2} - t_{i+1}}, \quad i = 0, 1, \ldots, 12.$$

Определим $N_i^2(x)$:

$$N_i^2(x) = (x - t_i)\frac{N_i^1(x)}{t_{i+2} - t_i} + (t_{i+3} - x)\frac{N_{i+1}^1(x)}{t_{i+3} - t_{i+1}}, \quad i = 0, 1, \ldots, 11.$$

Наконец, определим $N_i^3(x)$:

$$N_i^3(x) = (x - t_i)\frac{N_i^2(x)}{t_{i+3} - t_i} + (t_{i+4} - x)\frac{N_{i+1}^2(x)}{t_{i+4} - t_{i+1}}, \quad i = 0, 1, \ldots, 10.$$

(а) Вычислите на основе данных выше определений одиннадцать кубических B-сплайнов и постройте их графики. Помните о соглашении, что если где-либо в приведенных определениях возникает $0/0$, то эта дробь полагается равной нулю.

(б) Выберите на плоскости управляющие точки $\mathbf{p}_i$ так, чтобы сумма

$$\sum_{i=0}^{10} \mathbf{p}_i N_i^3(x)$$

аппроксимировала e^x. Нарисуйте соответствующую ломаную Безье.

4.16. Прологи: PCHEZ и PCHEV

```
      SUBROUTINE PCHEZ (N, X, F, D, SPLINE, WK, LWK, IERR)
C*** НАЧАЛО ПРОЛОГА
C*** ДАТА НАПИСАНИЯ 870821 (ГГММДД)
C*** ДАТА ПЕРЕСМОТРА 870908 (ГГММДД)
C*** КАТЕГОРИЯ NO. E1B
C*** КЛЮЧЕВЫЕ СЛОВА: кубическая эрмитова монотонная интер-
C     поляция, сплайн-интерполяция, простая в использовании
C     кусочно-кубическая интерполяция
C*** АВТОР: KAHANER D.K. (NBS)
C           SCIENTIFIC COMPUTING DIVISION
C           NATIONAL BUREAU OF STANDARDS
C           GAITHERSBURG, MARYLAND 20899
C           (301) 975-3808
```

```
C*** НАЗНАЧЕНИЕ: простая в использовании кубическая эрмитова
C                интерполяция или сплайн-интерполяция.
C*** ОПИСАНИЕ
C
C     PCHEZ: простая в использовании кусочно-кубическая интерполя-
C     ция
C
C     Из книги D. Kahaner, C. Moler, S. Nash
C                "Numerical Methods and
C                 Software" Prentice-Hall, 1988
C
C     Вычисляет производные сплайна (две непрерывные производные)
C     или эрмитова кубического интерполянта (одну непрерывную
C     производную). Сплайн-интерполяция более гладкая, но может
C     выглядеть «не очень хорошо», если данные содержат и «крутые»,
C     и «ровные» участки. Эрмитова кубическая интерполяция может
C     дать «приятный на вид» и монотонный интерполянт для монотон-
C     ных данных. Это удобная в использовании управляющая прог-
C     рамма для программ Ф. Н. Фритча (см. ссылку (4) ниже). В прог-
C     рамме PCHEZ краевые условия задаются по умолчанию. Прог-
C     раммы PCHIC, PCHIM и PCHSP предоставляют возможность
C     выбора разных краевых условий.
C
C     Пользуйтесь PCHEV для вычисления значений построенной функ-
C     ции и ее производной.
C
C     Обращение к подпрограмме:
C       CALL PCHEZ (N, X, F, D, SPLINE, WK, LWK, IERR)
C
C       INTEGER N, IERR, LWK
C       REAL X(N), F(N), D(N), WK(LWK)
C       LOGICAL SPLINE
C
C     Описание параметров:
C
C       N (на входе) – число точек данных. (Возврат по ошибке, если
C         N.LT.2.)
C         Если N = 2, то просто выполняется линейная интерполяция.
C
C       X (на входе) – вещественный массив значений независимой пе-
C         ременной. Элементы массива X должны быть упорядочены по
C         строгому возрастанию:
C         X(I – 1). LT. X(I), I = 2(1)N.
C
C         (В противном случае – возврат по ошибке.)
C
```

```
C     F (на входе) – вещественный массив значений зависимой пере-
C     менной для проведения интерполяции по ним. Величина F(I)
C     соответствует X(I).
C
C     D (на выходе) – вещественный массив значений производной
C     в заданных точках.
C
C     SPLINE (на входе) – логическая переменная для определения,
C          будет ли интерполянт сплайном с двумя непрерывными
C          производными (задать SPLINE = .TRUE.) или эрмитовым
C          кубическим интерполянтом с одной непрерывной произ-
C          водной (задать SPLINE = .FALSE.). Замечание. Если
C          SPLINE = .TRUE., то интерполирующий сплайн будет по
C          умолчанию удовлетворять краевому условию «запрета
C          стыка» с непрерывной производной третьего порядка в
C          X(2) и X(N – 1) (см. ссылку (3)).
C             Если SPLINE = .FALSE., то эрмитов кубический ин-
C          терполянт будет монотонным для монотонных входных
C          данных. Краевые условия задаются вычислением произ-
C          водных от квадратных трехчленов, локализованных на
C          концах интервала интерполяции, если только это не нару-
C          шает монотонность.
C
C     WK – вещественный рабочий массив, который надо описать
C          в вызывающей программе как массив длины по крайней
C          мере 2 * N, если параметр SPLINE имеет значение .TRUE.;
C          в противном случае не используется.
C
C     LWK (на входе) – длина рабочего массива WK. (Если LWK.LT.
C          2 * N и параметр SPLINE имеет значение .TRUE., то
C          возврат по ошибке; в противном случае не проверяется.)
C
C     IERR (на выходе) – признак ошибки. Нормальный возврат:
C          IERR = 0 (нет ошибок).
C          Предупреждение:
C             IERR.GT.O (может возникнуть только при SPLINE =
C             = .FALSE.) означает, что было обнаружено IERR пе-
C             реключений характера монотонности. Если SPLINE =
C             = .FALSE., то PCHEZ гарантирует, что интерполянт
C             будет монотонным для монотонных данных. Это пре-
C             дупреждение обращает внимание пользователя на то,
C             что входные данные были немонотонны.
C          Устранимые ошибки:
C             IERR = – 1, если N.LT.2.
C             IERR = – 3, если массив X не упорядочен по строгому
C             возрастанию.
```

```
C            IERR = -7, если LWK меньше, чем 2*N, и параметр
C            SPLINE имеет значение .TRUE. (В любом из этих
C            случаев массив D не изменяется.) Замечание. Условия
C            неверного обращения проверяются в перечисленном
C            порядке, и при обнаружении ошибки следующие аргу-
C            менты не принимаются во внимание.
C
C*** ССЫЛКИ: 1. F.N. FRITSCH, R.E. CARLSON, MONOTONE
C                PIECEWISE CUBIC INTERPOLATION, SIAM
C                J.NUMER. ANAL. 17. 2 (APRIL 1980), 238-246.
C            2. F.N. FRITSCH, J.BUTLAND, A METHOD FOR
C                CONSTRUCTING LOCAL MONOTONE PIECE-
C                WISE CUBIC INTERPOLANTS, LINL PREPRINT
C                UCRL-87559 (APRIL 1982).
C            3. CARL DE BOOR, A PRACTICAL GUIDE TO
C                SPLINES, SPRINGER-VERLAG (NEW YORK,
C                1978) (особенно гл. IV. с. 49-62).
C            4. F.N. FRITSCH, PIECEWISE CUBIC HERMITE
C                INTERPOLATION PACKAGE, FINAL SPECIFI-
C                CATIONS, LAWRENCE LIVERMORE NATIO-
C                NAL LABORATORY, COMPUTER DOCUMEN-
C                TATION UCID-30194, AUGUST 1982.
C*** ВЫЗЫВАЕМЫЕ ПОДПРОГРАММЫ: PCHIM, PCHSP
C*** КОНЕЦ ПРОЛОГА PCHEZ
C
      SUBROUTINE PCHEV (N, X, F, D, NVAL, XVAL, FVAL, DVAL,
     IERR)
C*** НАЧАЛО ПРОЛОГА PCHEV
C*** ДАТА НАПИСАНИЯ 870828 (ГГММДД)
C*** ДАТА ПЕРЕСМОТРА 870828 (ГГММДД)
C*** КАТЕГОРИЯ E3, H1
C*** КЛЮЧЕВЫЕ СЛОВА: дифференцирование сплайна или эрмитова
C      кубического интерполянта, вычисление значений эрмитова ку-
C      бического интерполянта, простой в обращении вычислитель
C      значений сплайна или эрмитова кубического интерполянта
C*** АВТОР: KAHANER D.K. (NBS)
C           SCIENTIFIC COMPUTING DIVISION
C           NATIONAL BUREU OF STANDARDS
C           ROOM A161, TECHNOLOGY BUILDING
C           GAITHERSBURG, MARYLAND 20899
C           (301) 975-3808
C*** НАЗНАЧЕНИЕ: вычисляет значения сплайна или кусочно-куби-
C                ческого эрмитова интерполянта и их первой
C                производной в точках, заданных в массиве.
C*** ОПИСАНИЕ
C
```

```
C     PCHEV: Простой в использовании вычислитель производной
C            сплайна или кусочно-кубического эрмитова интерполян-
C            та.
C
C     Из книги D. Kahaner, C. Moler, S. Nash
C              "Numerical Methods and Software"
C              Prentice-Hall, 1988
C
C     Подпрограмма вычисляет по аргументам N, X, F, D значения
C     сплайна или кубического эрмитова интерполянта и значения их
C     первой производной в точках массива XVAL.
C
C     Простая в использовании управляющая программа для программ
C     Ф. Н. Фритча (см. ссылку (2) ниже; эти программы предоставляют
C     и другие возможности).
C
C
C     Обращение к подпрограмме:
C       CALL PCHEV (N, X, F, D, NVAL, XVAL, FVAL, DVAL, IERR)
C
C       INTEGER N, NVAL, IERR
C       REAL X(N), F(N), D(N), XVAL(NVAL), FVAL(NVAL),
C       DVAL(NVAL)
C
C     Описание параметров:
C
C       N (на входе) – число точек данных. (Возврат по ошибке при
C         N.LT.2.)
C
C       X (на входе) – вещественный массив значений независимой пе-
C         ременной. Элементы массива X должны быть упорядочены
C         по строгому возрастанию:
C         X(I – 1). LT.X(I), I = 2(1)N. (Иначе возврат по ошибке.)
C
C       F (на входе) – вещественный массив значений функции. F(I) –
C         значение, соответствующее X(I).
C
C       D (на входе) – вещественный массив значений производной.
C       D(I) – значение, соответствующее X(I).
C
C       NVAL (на входе) – число точек, в которых надо вычислить
C         значения функций.
C
C       XVAL (на входе) – вещественный массив точек, в которых надо
C         вычислить значения функций.
C         Замечания:
```

```
C        1. Вычисления выполняются наиболее эффективно, если эле-
C           менты массива XVAL расположены в возрастающем по
C           X порядке, т. е.
C           XVAL(J). GE.X(I) влечет за собой
C           XVAL (K). GE.X(I) для всех K.GE.J.
C        2. Если какие-то элементы массива XVAL находятся вне
C           интервала [X(1), X(N)], то значения функций в них экстра-
C           полируются по полиному третьей степени на ближайшем
C           конце интервала; при этом выдается предупреждение.
C
C     FVAL (на выходе) – вещественный массив значений кубической
C        эрмитовой функции, определенных по N, X, F, D в точках
C        XVAL.
C
C     DVAL (на входе) – вещественный массив значений первой произ-
C        водной той же функции в точках XVAL.
C
C     IERR (на входе) – признак ошибки.
C        Нормальный возврат:
C           IERR = 0 (нет ошибок).
C        Предупреждение:
C           IERR.GT.O означает, что была применена экстраполяция
C           в IERR точках.
C        Устранимые ошибки:
C           IERR = – 1, если N.LT.2.
C
C           IERR = – 3, если массив X не упорядочен по строгому
C           возрастанию.
C           IERR = – 4, если NVAL.LT.1.
C           (Ни в одном из этих случаев содержимое выходных масси-
C           вов не изменяется.)
C        Замечание. Условия неверного обращения проверяются в пе-
C           речисленном порядке, и при обнаружении ошибки сле-
C           дующие аргументы не принимаются во внимание.
C           IERR = – 5, если ошибка возникла в программе низшего
C           уровня CHFDV. NB: это никогда не должно случаться.
C           Если это все же произойдет, НЕМЕДЛЕННО сообщите об
C           этом автору.
C
C     ------------------------------------------------------------
C     ССЫЛКИ: 1. F. N. FRITSCH, R. E. CARLSON, MONOTONE
C                PIECEWISE CUBIC INTERPOLATION, SIAM J.
C                NUMER. ANAL. 17, 2 (APRIL 1980), 238–246.
C             2. F. N. FRITSCH, PIECEWISE CUBIC HERMITE
C                INTERPOLATION PACKAGE, FINAL SPECIFI-
C                CATIONS, LAWRENCE LIVERMORE NATIO-
```

```
C                NAL LABORATORY, COMPUTER DOCUMEN-
C                TATION UCID-30194, AUGUST 1982.
C     ВЫЗЫВАЕМЫЕ ПОДПРОГРАММЫ: PCHFD
C     КОНЕЦ ПРОЛОГА PCHEV
```

Глава 5

Численные квадратуры

5.1. Введение

В этой главе нас будет интересовать решение следующей задачи: вычислить интеграл

$$I = \int_a^b f(x)\,dx.$$

Это одна из двух фундаментальных задач математического анализа. Она тесно связана с задачей решения дифференциальных уравнений (см. гл. 8) и использует аппарат интерполирования (гл. 4).

В качестве примера ситуации, которая приводит к указанной задаче, рассмотрим анализ ошибок измерений в научных экспериментах. Предположим, что в горной местности прокладывают дорогу, и топограф производит необходимые измерения. Пусть результаты измерений получены с помощью приборов, скажем, с точностью до одного фута. Какова вероятность того, что конкретный результат измерения превзойдет истинную величину меньше чем на 2 фута? Если ошибки измерений имеют стандартное нормальное распределение, т. е. соответствуют «колоколообразной» кривой, то эта вероятность составит

$$\frac{1}{\sqrt{2\pi}} \int_0^2 e^{-x^2/2}\,dx.$$

Здесь мы сталкиваемся с вычислением функции ошибок erf, которая уже встречалась нам в предыдущих главах. Простого выражения, позволяющего получить значение этого интеграла, не существует, но его можно вычислить приближенно с помощью численных методов.

Говоря о вычислении интегралов, употребляют термин «*квадратура*»; он происходит от латинского quadratura, означающего вычисление площади, или квадрирование. Геометрическая задача древности, задача о квадратуре круга, заключалась в построении квадрата, площадь которого равнялась бы площади данного круга. Решение должно было быть получено за конечное число шагов с помощью линейки и циркуля. Почти через две тысячи лет после ее постановки удалось доказать, что задача решения не имеет, а ее название стало обозначением любой сложной или трудоемкой деятельности; «не легче ты найдешь целебную микстуру, чем разгадаешь круга квадратуру» (1652). Впоследствии термин «квадратура» стал употребляться как синоним, когда речь шла

о нахождении площадей и объемов. В этой и следующих главах мы будем использовать именно этот термин. Термин же «*интегрирование*» мы сохраним для задачи решения дифференциальных уравнений и задачи вычисления неопределенного интеграла. Обе задачи – вычисления квадратур и интегрирования дифференциальных уравнений – имеют много общего. Однако интеграл – одно-единственное число, представляющее собой площадь или объем и остающееся неизменным, а решение дифференциального уравнения – функция, отражающая, как правило, эволюционный процесс, такой, например, как движение из начальной точки.

Обычно аппаратом, используемым для построения квадратур, является аппарат интерполирования. Вместо того чтобы вычислять $\int_a^b f(x)\,dx$ непосредственно, сперва вычисляют значения функции $f(x)$ в заданных точках $x_i \in [a, b]$. Пусть $p(x)$ – интерполяционный полином, проходящий через точки $(x_i, f(x_i))$. Тогда, если $p(x) \approx f(x)$, то и $\int_a^b p(x)\,dx \approx \int_a^b f(x)\,dx$. Поскольку интегрирование полинома не составляет труда, а аппарат интерполирования вполне доступен (см. гл. 4), такой подход представляется численно реализуемым и эффективным.

Методы вычисления квадратур различаются способами выбора точек x_i, числом выбранных точек и принципами, на основе которых по выбранным точкам строятся интерполяционные полиномы. Например, в некоторых методах отрезок $[a, b]$ разбивается на части и интегралы вычисляются отдельно по каждой из частей. Простейшие правила используют равноотстоящие точки, но гораздо более эффективными оказываются способы, основанные на более тонкой стратегии. Встречаются задачи, когда значения функции известны лишь в определенных фиксированных точках; в таких случаях невозможно распорядиться выбором x_i (см. § 8).

От большинства квадратурных программ требуется, чтобы они не только численно находили значение интеграла I, но и оценивали погрешность результата. Оценки погрешности представляют интерес не для одного лишь пользователя; их анализ помогает организовать и само вычисление интеграла. Оценки погрешности могут указать на то, что на одной из частей отрезка $[a, b]$ функция $f(x)$ быстро меняется, и тогда бóльшую долю работы программа должна затратить на вычисление интеграла именно на этой части. Подобные вопросы обсуждаются в § 6.

Приведенные выше соображения относятся главным образом к тем случаям, когда отрезок интегрирования конечен, а функция $f(x)$ непрерывна на нем. Если функция не является непрерывной и, прежде всего, если она обладает особенностью (как $f(x) = \ln x$ при $x = 0$), то невозможно хорошо приблизить $f(x)$ полиномом. Если же отрезок интегрирования бесконечен, то и интеграл по этому отрезку от любого полинома, не

являющегося тождественным нулем, будет бесконечным. В таких случаях для решения квадратурных задач должна применяться особая стратегия. Это предмет обсуждения в § 9.

До сих пор мы говорили о квадратурных задачах только в одномерном случае. Однако часто приходится встречаться с интегралами двух, трех и большего числа измерений; задача их вычисления оказывается более трудной. Упомянутые выше приемы можно приспособить к вычислению двойных и тройных интегралов, но получающиеся в результате правила интегрирования отличаются большой сложностью. Даже вывод их становится весьма трудоемким делом. Подходы такого рода обсуждаются в § 10. Для многомерных задач обычно целесообразно использовать методы Монте-Карло, основанные на случайном выборе точек x_i. Скорость сходимости этих методов невелика, но зато вычислительные затраты при их применении не зависят существенно от размерности задачи (см. § 11).

В основных курсах математического анализа вычисление интегралов – одна из наиболее важных тем. Упор прежде всего делается на способы получения замкнутых выражений для неопределенных интегралов. Компьютерный аналог этого направления – *символические вычисления*, т. е. вывод компьютером формул, которые зависят от заданной функции f и выражают интеграл через концы отрезка интегрирования a, b. В этой области достигнуты значительные успехи. Система Macsyma [Moses, 1971] – превосходное средство для решения тех задач, в которых подынтегральная функция задана явно, и, следовательно, особенно желательно получить решение в виде замкнутого выражения, если оно существует. Macsyma, например, обнаружит, что[1)]

$$\int_0^b \frac{dt}{t^4+1} = \frac{\sqrt{2}}{4}\left(\frac{1}{2}\ln\left(\frac{b^2+\sqrt{2}b+1}{b^2-\sqrt{2}b+1}\right) + \operatorname{arctg}\left(\frac{\sqrt{2}b}{1-b^2}\right)\right).$$

Однако при решении многих инженерных задач главным требованием является требование эффективности, и поэтому может оказаться, что вычисление правой части для конкретного b обойдется более дорого, чем приближение исходного интеграла с помощью удачно выбранного метода. В этой главе мы будем иметь дело только с приближенными численными методами.

5.2. Одномерные квадратурные правила и формулы

Здесь мы получим простейшие квадратурные правила, основанные на интерполировании по небольшому числу точек. Сначала приведем тер-

[1)] Равенство справедливо при $|b| < 1$. – *Прим. перев.*

минологию, которую в дальнейшем будем использовать. Назовем *n-точечной квадратурной формулой* равенство

$$I = \int_a^b f(x)\,dx = \sum_{i=1}^{n} w_i f(x_i) + R_n.$$

При этом w_i и x_i называются *весами* и *узлами* квадратурной формулы, а R_n – *остатком* или *погрешностью*. Веса и узлы зависят от a, b и n, но не зависят от f. Сумму в правой части, которую можно рассматривать как приближение к I, часто называют *квадратурным правилом*. Чтобы приближенно вычислить интеграл, мы вычисляем *только* квадратурное правило, так как остаток обычно содержит выражения, значения которых нам недоступны, например производные подынтегральной функции $f(x)$. Важное свойство наших формул – их линейность. Это означает, что правило, дающее приближение к интегралу от $f + g$, можно получить сложением правил, дающих приближения к f и g.

Квадратурное правило представляет собой обобщение суммы Римана. Вспомним, что для того, чтобы получить сумму Римана, отрезок $[a, b]$ нужно разбить на n подотрезков длины Δ_i, вычислить $f(x)$ в некоторой точке каждого из подотрезков и составить сумму $\Sigma\Delta_i f(x_i)$. Может создаться впечатление, что квадратурное правило превратится в сумму Римана, если приравнять w_i к Δ_i. Однако впечатление это обманчиво, поскольку мы не требовали, чтобы w_i были положительными и даже чтобы x_i принадлежали отрезку $[a, b]$. Большей частью эти различия представляют интерес только для специалистов, и все те правила, которые мы здесь будем рассматривать, действительно являются суммами Римана. Но с другой стороны, среди известных и применяющихся на практике правил есть и такие, которые суммами Римана не являются; например, некоторые w_i могут быть в них отрицательными (см. задачу 5.6).

В течение многих лет были изучены тысячи квадратурных правил. В разд. 2.1 мы рассмотрим три простейших: правила прямоугольников, трапеций и Симпсона. Вы могли уже встречаться с ними в учебниках по основам математического анализа; вывод их прост, но они редко применяются непосредственно. Чаще всего строятся их обобщения или же они используются в сочетании с какой-либо тонкой стратегией, например стратегией применения составных квадратурных правил или стратегией адаптации; об этом речь пойдет в § 4, 6. Однако эти правила далеки от идеала. В разд. 2.1 мы также рассмотрим более эффективные правила Гаусса, а самые эффективные правила Гаусса-Кронрода отложим до § 5. Поскольку вывод правил Гаусса и Гаусса-Кронрода более сложен и не представляет большого практического интереса, мы будем опускать детали в расчете на то, что заинтересованные читатели обратятся к указанной в ссылках литературе.

Когда перед глазами столько правил, трудно определить, какое же из них выбрать для решения данной практической задачи. Основная мысль

наших рассуждений состоит в том, что в большинстве случаев ученый-практик может использовать помещенные здесь программы общего назначения. Едва ли вы станете создавать собственные программы на основе какого-либо из этих правил в то время, когда достаточно применить такую программу, как Q1DA (см. § 7). Но для того чтобы понять, почему эта программа так хорошо работает, вам нужно оценить высокое качество правил, на основе которых она создана. Конечно, в тех особых случаях, когда вам потребуется собственный алгоритм, придется приложить усилия, чтобы определить, какое из правил будет наиболее подходящим. Наконец, программы общего назначения мало пригодны в случаях бесконечных отрезков интегрирования, поэтому в § 9 приведены некоторые правила, предназначенные для этих случаев.

5.2.1. Элементарные формулы прямоугольников, трапеций, Симпсона и Гаусса

Здесь мы рассмотрим элементарные формулы. Они не применяются непосредственно, но составляют основу многих практических методов. Кроме того, они допускают наглядную геометрическую интерпретацию, поэтому при чтении этого раздела воспользуйтесь рис. 5.1.

Формулы прямоугольников и трапеций выводятся следующим образом. Площадь под кривой $f(x)$, где x принадлежит отрезку $[a, b]$, приближается площадью под отрезком прямой, соединяющей точки $(a, f(a))$ и $(b, f(b))$. Эта прямая задается уравнением

$$l(x) = f(a)\frac{x-b}{a-b} + f(b)\frac{x-a}{b-a}.$$

а площадь трапеции с вершинами $(a, 0)$, $(b, 0)$, $(b, f(b))$, $(a, f(a))$ равна

$$\frac{b-a}{2}f(a) + \frac{b-a}{2}f(b).$$

Это и есть квадратурное правило, полученное из простых геометрических соображений. Теперь выведем его снова с помощью метода, который поможет нам получить также выражение для погрешности.

Разложим $f(x)$ в ряд Тейлора в средней точке $m = (a+b)/2$:

$$f(x) = f(m) + f^{(1)}(m)(x-m) + f^{(2)}(m)(x-m)^2/2 + \\ + f^{(3)}(m)(x-m)^3/6 + f^{(4)}(m)(x-m)^4/24 + \dots .$$

Интегрирование этого разложения по отрезку $[a, b]$ дает

$$I = f(m)(b-a) + f^{(2)}(m)\frac{(b-a)^3}{24} + f^{(4)}(m)\frac{(b-a)^5}{1920} + \dots .$$

Это *одноточечная формула прямоугольников*. Теперь подставим в разложение Тейлора $x = a$ и $x = b$, а затем сложим оба ряда. Производные нечетных порядков сократятся. Разрешим полученное соотношение относительно $f(m)$:

$$f(m) = \frac{1}{2}(f(a) + f(b)) - f^{(2)}(m)(b-a)^2/8 - f^{(4)}(m)(b-a)^4/384 + \dots .$$

Подставив этот ряд в формулу прямоугольников и приведя подобные члены, получим

$$I = \frac{b-a}{2}(f(a) + f(b)) - f^{(2)}(m)\frac{(b-a)^3}{12} - f^{(4)}(m)\frac{(b-a)^5}{480} + \dots .$$

Это *двухточечная формула трапеций* с узлами $x_1 = a$, $x_2 = b$ и весами $w_1 = w_2 = (b-a)/2$, совпадающая с правилом, выведенным выше геометрическим способом.

Трехточечная формула Симпсона также может быть выведена из геометрических соображений подобно тому, как было выведено правило трапеций. Из § 3 гл. 4 мы знаем, что квадратичная функция $Q(x)$, проходящая через три точки $(a, f(a))$, $(m, f(m))$ и $(b, f(b))$, имеет вид

$$Q(x) = f(a)\frac{(x-m)(x-b)}{(a-m)(a-b)} + f(m)\frac{(x-a)(x-b)}{(m-a)(m-b)} + f(b)\frac{(x-a)(x-m)}{(b-a)(b-m)}.$$

Интегрирование по отрезку $[a, b]$ дает

$$\int_a^b Q(x)\,dx = f(a)w_1 + f(m)w_2 + f(b)w_3,$$

где

$$w_1 = \int_a^b \frac{(x-m)(x-b)}{(a-m)(a-b)}\,dx,$$

и т. д. Вы можете самостоятельно проверить, что

$$w_1 = \frac{b-a}{6},\quad w_2 = \frac{4(b-a)}{6},\quad w_3 = \frac{b-a}{6},$$

а это и есть правило Симпсона. Отметим также, что после того как функция Q проинтегрирована, формула для $Q(x)$ нам больше не требуется. Правило Симпсона впервые было получено Кавальери в 1639 г., а позже Джеймсом Грегори в 1668 г. и Томасом Симпсоном в 1743 г. Его также называют правилом парабол.

Рис. 5.1 иллюстрирует правила прямоугольников, трапеций и Симпсона.

Так же как и в случае правила трапеций, алгебраический подход поможет нам получить формулу Симпсона, содержащую член погреш-

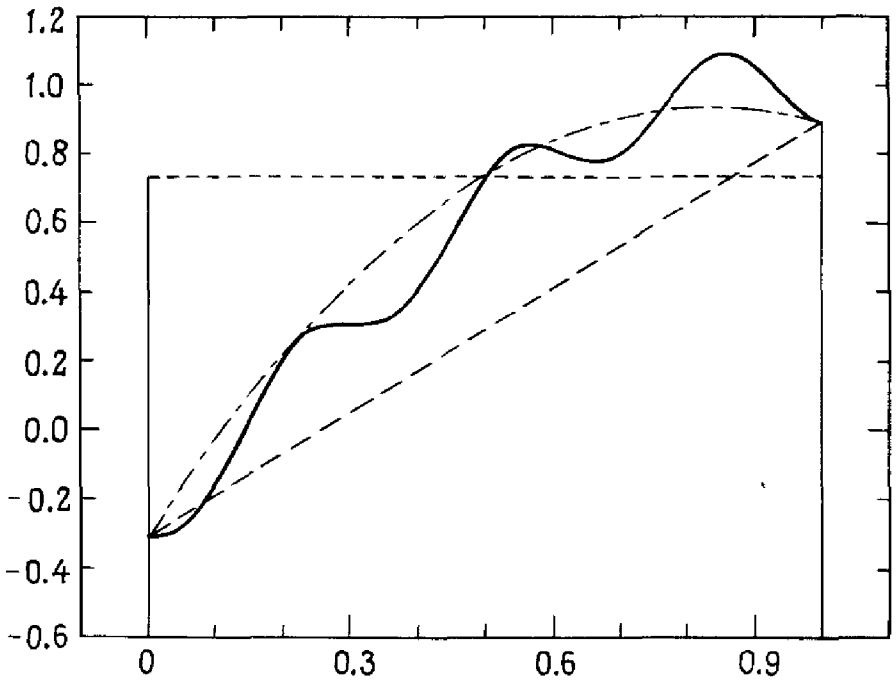


Рис. 5.1. Правила прямоугольников, трапеций и Симпсона.

ности. Сложив дважды формулу прямоугольников с формулой трапеций и разделив результат на 3, получим

$$I = \frac{b-a}{6}(f(a) + 4f(m) + f(b)) - f^{(4)}(m)\frac{(b-a)^5}{2880} + \dots .$$

Пример 5.1. Вычисление $\operatorname{erf}(1)$ **с помощью различных правил.** Найдем приближенные значения для

$$I = \operatorname{erf}(1) = \frac{2}{\sqrt{\pi}}\int_0^1 \exp(-t^2)\,dt = 0.84270079294971484\dots$$

с помощью правил прямоугольников, трапеций и Симпсона.

Одноточечное правило прямоугольников, двухточечное правило трапеций и трехточечное правило Симпсона дают следующие приближения:

$$I \approx \frac{2}{\sqrt{\pi}}e^{-\frac{1}{4}} = 0.8788\dots,$$

$$I \approx \frac{2}{\sqrt{\pi}}\left(\frac{1}{2}\right)(1 + e^{-1}) = 0.7717433\dots,$$

$$I \approx \frac{2}{\sqrt{\pi}}\left(\frac{1}{6}\right)(1 + 4e^{-\frac{1}{4}} + e^{-1}) = 0.84310283\dots \ \square$$

Отметим, что применение правила трапеций к линейному полиному

$f(x) = \alpha x + \beta$ дает

$$I = \frac{b-a}{2}(f(a) + f(b)).$$

Остаток в этом случае равен нулю. Этот факт не покажется неожиданным, если вспомнить, каким путем правило трапеций было получено в начале этого раздела. Аналогично, погрешность трехточечного правила Симпсона равна нулю, если f - кубический полином.

Говорят, что n-точечное квадратурное правило имеет *алгебраическую степень точности* d, если его остаток R_n равен нулю для любого полинома степени d, но не равен нулю для некоторого полинома степени $d+1$. Алгебраическая степень точности правил трапеций и прямоугольников равна 1. Алгебраическая степень точности правила Симпсона равна 3.

Квадратурные правила редко используются для интегрирования полиномов, но разумно предположить, что чем больше алгебраическая степень точности квадратурного правила, тем оно лучше, по крайней мере если подынтегральная функция близка к полиному. Поэтому правилам высокой алгебраической степени точности отдается предпочтение, хотя на практике часто более важными являются особенности стратегии применения правила, а не оно само.

Наш первый подход, позволивший вывести правило Симпсона, иллюстрирует общий прием, с помощью которого выводятся многие квадратурные правила: функция $f(x)$ заменяется интерполяционным полиномом, интерполяционный полином интегрируется аналитически, и в результате получается квадратурное правило. При его дальнейшем использовании явное выражение для интерполяционного полинома не требуется.

Если, например, мы проинтегрируем лагранжев интерполяционный полином $p_{n-1}(x)$, проходящий через точки $(x_i, f(x_i))$, $i = 1, \ldots, n$, то придем к правилу (см. § 3 гл. 4)

$$\int_a^b p_{n-1}(x)\,dx = \sum_{i=1}^{n} f(x_i)\,w_i, \text{ где } w_i = \int_a^b l_i(x)\,dx.$$

Правила такого типа называются *интерполяционными*. Наиболее известные квадратурные правила получаются, если выбрать точки x_i равноотстоящими на отрезке интегрирования; такие правила называются *правилами Ньютона-Котеса*. Правила трапеций и Симпсона представляют собой правила Ньютона-Котеса невысоких алгебраических степеней точности. Правила же Ньютона-Котеса, алгебраическая степень точности которых высока, обладают, к сожалению, тем свойством, что среди их весов имеются отрицательные; см. задачу 5.6. Обычно это вызывает нежелательный эффект: интегрирование положительной функции может привести к искажению результата из-за катастрофической потери значащих разрядов, о чем рассказывалось в § 4 гл. 2. Сильно

удивляться этому не приходится; мы видели в гл. 4, что интерполяционный полином высокой степени может приближать функцию весьма плохо, а поэтому и интеграл от него не должен давать хорошее приближение к интересующему нас интегралу. Однако, с другой стороны, интерполяционные квадратуры оказываются полезными в тех случаях, когда интерполяционный полином имеет невысокую степень. Описанный прием вывода интерполяционных квадратурных правил мы применим в § 8 для вывода квадратурной формулы на основе эрмитова кубического интерполянта, которая будет использована для интегрирования таблично заданных функций.

Наметим теперь вывод двухточечного правила Гаусса. Любое двухточечное правило имеет вид

$$I \approx w_1 f(x_1) + w_2 f(x_2).$$

Можем ли мы среди всех таких правил найти правило наивысшей алгебраической степени точности? Один из кандидатов - правило трапеций, но нам удастся найти правило гораздо лучше. Поскольку мы можем распорядиться выбором w_1, w_2, x_1 и x_2, то подчиним наш выбор такому требованию: правило должно быть точным для любого кубического полинома (имеющего четыре коэффициента). Если бы это требование было выполнено, то правило было бы точным и для любого из следующих кубических полиномов: $f(x) = 1 = (x - m)^0$, $f(x) = (x - m)^1$, $f(x) = (x - m)^2$ и $f(x) = (x - m)^3$, где $m = (a + b)/2$. И наоборот, в силу линейности правила из того, что оно точно для этих четырех полиномов, следует, что оно точно и для любого полинома третьей степени. Итак, мы требуем, чтобы приближенное равенство $\approx$ стало точным, если $f(x)$ - одна из указанных четырех функций. Таким образом,

$$f(x) = (x - m)^0 \Rightarrow b - a = w_1 + w_2,$$
$$f(x) = (x - m)^1 \Rightarrow 0 = w_1(x_1 - m) + w_2(x_2 - m),$$
$$f(x) = (x - m)^2 \Rightarrow \frac{(b-a)^3}{12} = w_1(x_1 - m)^2 + w_2(x_2 - m)^2,$$
$$f(x) = (x - m)^3 \Rightarrow 0 = w_1(x_1 - m)^3 + w_2(x_2 - m)^3.$$

Мы получили систему четырех уравнений с четырьмя неизвестными w_1, w_2, x_1, x_2. Уравнения нелинейны, и методы гл. 3 здесь неприменимы. В общем случае система нелинейных уравнений может как вовсе не иметь решений, так и иметь несколько решений. Если же система имеет единственное решение, то нет никакой гарантии, что x_i окажутся вещественными, ведь в уравнения входят их степени. И даже если решение вещественно, x_i могут и не принадлежать отрезку $[a, b]$ или w_i могут быть отрицательными. К счастью, в случае данной системы ничего столь ужасного не происходит. Можно показать, что система имеет единственное решение

$$x_1 = \frac{a+b}{2} + \frac{b-a}{2\sqrt{3}}, \quad x_2 = \frac{a+b}{2} - \frac{b-a}{2\sqrt{3}}, \quad w_1 = w_2 = \frac{b-a}{2},$$

которое дает нам *двухточечное квадратурное правило Гаусса*

$$I \approx G_2 = \frac{b-a}{2}\left(f\left(\frac{a+b}{2} - \frac{b-a}{2\sqrt{3}}\right) + f\left(\frac{a+b}{2} + \frac{b-a}{2\sqrt{3}}\right)\right).$$

Алгебраическая степень точности этого правила равна 3.

Пример 5.2. Вычисление $\operatorname{erf}(1)$ **с помощью правила Гаусса** Имеем

$$\operatorname{erf}(1) = 0.8427007929\ldots \approx$$

$$\approx G_2 = \frac{1}{\sqrt{\pi}}\left(\exp\left(-\frac{1}{4}\left(1 - \frac{1}{\sqrt{3}}\right)^2\right) + \exp\left(-\frac{1}{4}\left(1 + \frac{1}{\sqrt{3}}\right)^2\right)\right) =$$

$$= 0.84244189\ldots .$$

Заметим, что это приближение точнее того, которое получено с помощью трехточечного правила Симпсона. □

Как уже было сказано в § 2, двухточечные формулы применяются не часто. Но идею, на основе которой получена двухточечная формула Гаусса, можно распространить и на произвольное число узлов. А именно, мы хотим найти формулу

$$\int_a^b f(x)\,dx = \sum_{i=1}^{n} w_i f(x_i) + R_n,$$

в которой ни x_i, ни w_i заранее не предписаны, а выбираются таким образом, чтобы правило было точным для полиномов наиболее высокой степени. Для любого n существует единственная квадратурная формула Гаусса. Ее алгебраическая степень точности равна $2n - 1$, причем n-точечной формулы большей алгебраической степени точности не существует. В этом смысле формула Гаусса является наилучшей. Можно показать, что если $f(x)$ имеет $2n$ непрерывных производных на $[a, b]$, то

$$R_n \approx \frac{(b-a)^{2n+1}(n!)^4}{(2n+1)((2n)!)^3} f^{2n}(\xi).$$

Относительно ξ известно лишь, что $a < \xi < b$. Правила Гаусса принадлежат к наиболее часто применяемым на практике. Ниже мы перечислим некоторые их свойства.

1. За небольшим исключением, веса и узлы правил Гаусса иррациональны. При $n = 2$, как мы видели, $w_1 = w_2 = (b-a)/2$. Кроме того, если n нечетно, то один из узлов совпадает с серединой отрезка интегрирования $(a + b)/2$. Обычно правила Гаусса реализуются в виде программ, которые содержат узлы и веса, заранее вычисленные для набора значений n. Таблицы узлов и весов правил Гаусса имеются в книге [Abramowitz, Stegun, 1965]. Три правила Гаусса приведены

в качестве примера в табл. 5.1[1]. При ручном счете использовать правила Гаусса затруднительно, но при вычислениях на компьютере причины, заставляющие отказываться от громоздких весов и узлов, устраняются.

2. Правила Гаусса относятся к правилам открытого типа. Это означает, что ни один из узлов не совпадает ни с одним из концов отрезка интегрирования a или b. Подтверждение этому мы видели на примере правила прямоугольников, являющегося одноточечным правилом Гаусса. «Открытость» полезна тем, что трудности, которые могут возникнуть при вычислении значений подынтегральных функций, связаны обычно именно с концевыми точками. Например, о поведении подынтегральных функций в интегралах

$$I_1 = \int_0^1 \exp\left(-\frac{1}{x^2}\right) dx, \quad I_2 = \int_0^1 \frac{\sin x}{x}\, dx$$

нельзя сказать ничего плохого, однако программисту придется прибегнуть к особым ухищрениям, если он станет использовать квадратурные правила, которые требуют вычисления $f(0)$. Строго говоря, подынтегральные функции в этих интегралах обладают особенностью, так как они не определены при $x = 0$. Но в обоих случаях $\lim\limits_{x \to 0} f(x)$ существует и равен 0 и 1 соответственно, так что подынтегральная функция может быть естественным образом доопределена в левом конце отрезка интегрирования. Такие особенности называются устранимыми.

3. Множества узлов двух правил Гаусса почти не пересекаются. Иными словами, узлы n-точечного правила отличаются от узлов m-точечного правила. Исключение состоит в том, что если n нечетно, то один из узлов совпадает с серединой отрезка интегрирования.

4. Вывод правил Гаусса позволяет предположить, что если $f(x)$ близка к полиному, то правило Гаусса наиболее пригодно для вычисления интеграла от такой функции. Действительно, можно показать, что из всех правил, использующих n значений подынтегральной функции, n-точечное правило Гаусса даст наиболее точное приближение, по крайней мере если подынтегральная функция гладкая. Более того, правила Гаусса хорошо работают даже в тех случаях, когда у подынтег-

[1] Значения, приведенные в табл. 5.1 для узла при $n = 2$ и первого узла при $n = 4$, отличаются в последнем знаке (соответственно 5 вместо 6 и 2 вместо 3) от значений, получаемых правильным округлением 30-значных приближений тех же узлов, которые можно найти в книге [Strond A. H., Secrest D. Gaussian Quadrature Formulas. – Prentice-Hall, 1966.]. – *Прим. перев.*

ральной функции или у одной из ее производных имеется неустранимая особенность. Вторая производная функции $f(x) = x^2 \ln x$ не ограничена на (0, 1]. Приведенное выше выражение для погрешности правила Гаусса G_2 не дает нам никакой информации о величине погрешности, поскольку включает производную $f^{(4)}$, которая при $x = 0$ не существует. Однако не следует думать, что погрешность в этом случае велика. Несложные вычисления показывают, что $G_2 = -0.1085\ldots$, а погрешность равна $0.0023\ldots$. Двухточечное правило трапеций и трехточечное правило Симпсона дают погрешности $0.1111\ldots$ и $0.0044\ldots$.

5. Правила Гаусса являются интерполяционными. Это означает, что если мы, взяв n узлов x_i n-точечной формулы Гаусса и n соответствующих значений $f(x_i)$, построим тот единственный полином степени $n-1$, который интерполирует эти данные, то интегрирование по отрезку $[a, b]$ построенного полинома даст нам тот же результат, что и применение к f формулы Гаусса. Здесь мы имеем тот случай, когда интерполяционные квадратуры работают хорошо; это связано с тем, что узлы Гаусса заполняют отрезок $[a, b]$ неравномерно, сгущаясь к его концам. Можно доказать, что веса квадратур Гаусса всегда положительны и что при увеличении числа узлов точность приближения почти всегда возрастает.

6. Существует тесная связь между квадратурами Гаусса и *ортогональными полиномами*, предметом особого интереса физиков. Узлы n-точечного правила Гаусса являются нулями полинома Лежандра степени n. Из-за этого квадратуры, которые мы назвали квадратурами Гаусса, часто называют также квадратурами Гаусса–Лежандра.

Квадратуры Гаусса допускают немало обобщений, о которых читатель пока представления не получил. Два наиболее важных из них будут рассмотрены в § 4 и в разд. 9.4.

5.3. Переход от одного отрезка к другому

Программы, в которых реализованы квадратурные правила, обычно позволяют пользователю произвольно задавать концы отрезка интегрирования a, b. Однако в таблицах, подобных табл. 5.1, правила приведены для какого-либо определенного отрезка интегрирования, чаще всего для [0, 1] или [−1,1]. Пусть таблица содержит узлы и веса формулы

$$\int_{\alpha}^{\beta} f(x)\,dx = \sum_{i=1}^{n} w_i f(x_i) + R_n$$

(в табл. 5.1 $\alpha = -1$, $\beta = 1$). Если интегрирование производится по отрезку $[\alpha, \beta]$, то значения узлов и весов могут быть взяты непосредственно из таблицы. С другой стороны, предположим, что нужно найти

Таблица 5.1 Некоторые квадратуры Гаусса для интеграла $\int_{-1}^{1} f(x)dx$

Узлы	Веса
$n = 2$	
±0.577350269189625	1.000000000000000
$n = 4$	
±0.861136311594052	0.347854845137454
±0.339981043584856	0.652145154862546
$n = 8$	
±0.960289856497536	0.101228536290376
±0.796666477413627	0.222381034453374
±0.525532409916329	0.313706645877887
±0.183434642495650	0.362683783378362

приближенное значение интеграла

$$I = \int_a^b g(t)\,dt$$

по какому-либо другому отрезку. Тогда мы снова можем воспользоваться табличными значениями узлов и весов, но предварительно должны с помощью замены переменной осуществить переход от переменной x, принадлежащей отрезку $[\alpha, \beta]$, к переменной t, принадлежащей отрезку $[a, b]$. Если α, β, a и b конечны, то положим $t = ((b-a)x + a\beta - b\alpha)/(\beta - \alpha)$. Тогда

$$I = \left(\frac{b-a}{\beta-\alpha}\right)\int_\alpha^\beta g\left(\frac{(b-a)x + a\beta - b\alpha}{\beta-\alpha}\right)dx = \left(\frac{b-a}{\beta-\alpha}\right)\sum_{i=1}^{n} w_i\, g\left(\frac{(b-a)x_i + a\beta - b\alpha}{\beta-\alpha}\right) + \mathscr{R}_n.$$

При переходе от переменной t к переменной x отрезок $[a, b]$ переводится в отрезок $[\alpha, \beta]$. Существует много преобразований, которые позволяют осуществить этот переход, но приведенное выше преобразование – самое простое в силу своей линейности. Если $P(t)$ – полином степени d относительно переменной t, то и $P(((b-a)x + a\beta - b\alpha)/(\beta - \alpha))$ – полином той же степени относительно переменной x. Поэтому такая замена переменной не меняет алгебраической степени точности формулы. Если R_n равно нулю, когда $f(x)$ – полином степени d, то и $\mathscr{R}_n$

равно нулю, когда $g(x)$ – полином степени d. В следующих параграфах мы познакомимся с некоторыми примерами нелинейных преобразований.

Пример 5.3. Переход к другому отрезку.
Найдем приближенное значение интеграла

$$I = \int_1^2 x^x\,dx$$

по четырем узлам, используя табл. 5.1. Положив $\alpha = -1$, $\beta = 1$, $a = 1$, $b = 2$ и $g(t) = t^t$, получим

$$I \approx \frac{1}{2}\sum_{i=1}^{4} w_i g\left(\frac{x_i+3}{2}\right) = \frac{1}{2}\sum_{i=1}^{4} w_i \left(\frac{x_i+3}{2}\right)^{(x_i+3)/2} = 2.05044\ldots,$$

где w_i, x_i взяты из табл. 5.1 ($n = 4$). □

5.4. Составные квадратурные формулы и оценки погрешности

Как было отмечено в § 2, одно-, двух- и трехточечные правила редко применяются непосредственно. Обычно их используют в определенных комбинациях; этот подход мы сейчас и проиллюстрируем. Пусть точки $a = t_0 < t_1 < t_2 < \ldots < t_{s-1} < t_s = b$ разбивают отрезок $[a, b]$ на s подотрезков, которые мы назовем *элементарными отрезками*. Поскольку

$$I = \int_a^b f(x)\,dx = \sum_{i=0}^{n-1} \int_{t_i}^{t_{i+1}} f(x)\,dx,$$

мы можем получить приближенное значение I, вычислив с помощью квадратурного правила интегралы по каждому из элементарных отрезков. Если на каждом из них применяется одно и то же правило, а точки t_i расположены на равном расстоянии друг от друга, то получающиеся в результате квадратурные правила называются *составными*. Любое правило можно преобразовать в составное, но наиболее известными составными правилами являются два, которые получены на основе формул трапеций и Симпсона.

Построим составное правило трапеций. Пусть $h = (b-a)/s$ – длина элементарного отрезка. Тогда $t_i = a + ih$. Применив двухточечное правило трапеций на каждом из отрезков $[a + ih, a + (i+1)h]$, получим после сложения $(s+1)$-точечное правило

$$\int_a^b f(x)\,dx = h\left(\frac{f(a)}{2} + f(a+h) + f(a+2h) + \ldots + f(b-h) + \frac{f(b)}{2}\right) - \frac{h^3}{12}\left(f^{(2)}(m_1) + f^{(2)}(m_2) + \ldots + f^{(2)}(m_s)\right) + \ldots,$$

или

$$I = T_{s+1} + R_{s+1},$$

где $m_i = (t_{i-1} + t_i)/2$. На практике вычисляют только T_{s+1}. Что касается остатка, то его в лучшем случае удается оценить, причем при получении оценок чаще всего не требуется находить производные подынтегральной функции. Если функция $f^{(2)}$ достаточно гладкая, то сумму вторых производных можно аппроксимировать выражением $s \cdot f^{(2)}(\xi)$, откуда

$$R_{s+1} \approx -\frac{h^3}{12}(sf^{(2)}(\xi)) + \ldots = -\frac{b-a}{12}h^2 f^{(2)}(\xi).$$

Ниже мы коротко остановимся на способе оценки этой величины, не требующем вычисления $f^{(2)}$.

Составная формула Симпсона строится аналогично. Положив $h = (b-a)/2s$, $t_i = a + 2ih$, получим

$$I = \frac{h}{3}(f(a) + 4f(a+h) + 2f(a+2h) + 4f(a+3h) + \ldots + 4f(b-h) + f(b)) - (b-a)\frac{h^4}{180}f^{(4)}(\xi) + \ldots.$$

Это $(2s+1)$-точечное правило.

В литературе название «правило трапеций» может употребляться и по отношению к двухточечному правилу, и по отношению к составному правилу. Какое из правил имеется в виду, бывает ясно из контекста. Аналогичное замечание относится и к правилу Симпсона. Наконец отметим, что алгебраическая степень точности составного правила совпадает с алгебраической степенью точности правила, являющегося его основой.

Есть несколько способов получения оценок погрешностей составных правил. Квадратурное правило будет малоприменимым до тех пор, пока каким-либо способом не удастся получить оценки его остатка R_n. Для простых интегралов, подобных тому, через который выражается функция $\operatorname{erf}(x)$, первые несколько производных подынтегральной функции найти легко. Так, для погрешности приближенного вычисления $\operatorname{erf}(1)$ с помощью составного правила трапеций находим

$$|R_{s+1}| = \frac{1}{12s^2}\frac{2}{\sqrt{\pi}}|e^{-\xi^2}(4\xi^2 - 2)| < \frac{1}{12s^2}\frac{2}{\sqrt{\pi}}2 = \frac{1}{3s^2\sqrt{\pi}}.$$

Если применить 100-точечное правило, погрешность не превысит 10^{-4}. Конечно, истинная погрешность может быть и значительно меньше 10^{-4} (см. пример 5.4).

Если задача сложна, то может оказаться, что находить производные подынтегральной функции высоких порядков неудобно или затруднительно. Приведенный выше пример может вызвать у читателя неверное

представление о применимости содержащих производные выражений для остатков. В любой реальной задаче остаток оценивается на основе числовых (а не аналитических) данных. Типичный алгоритм оценки погрешности описан ниже, еще один алгоритм рассматривается в § 5. Оценки погрешности, полученные аналитически, оказываются слишком сложными или слишком пессимистическими. Во многих прикладных задачах подынтегральная функция даже не задается формулой, а ее значения находят с помощью длинной цепочки вычислений, проанализировать которую невозможно.

Распространенный метод получения оценок погрешности состоит в комбинации двух или большего числа квадратурных правил. В простейшем случае применяют два правила и за оценку погрешности менее точного из них принимают модуль разности двух приближений. Например, если применить составное правило трапеций и составное правило Симпсона, то разность полученных приближений можно использовать для оценки погрешности первого. Есть и немало других возможностей. Ниже будет показано, как использовать два правила трапеций с различным числом элементарных отрезков. В следующем параграфе мы рассмотрим еще два правила – пару Гаусса-Кронрода. Существенными здесь являются следующие два момента.

(а) В способах получения оценок погрешности большую роль играет эвристический компонент. Выбор и комбинация двух правил – это скорее искусство, чем наука.

(б) Получить и приближенное значение интеграла, и оценку погрешности значительно труднее, чем одно только приближенное значение интеграла.

Составные правила хороши тем, что позволяют легко строить правила с произвольным числом узлов. Преимущество составных правил связано с поведением их остатков при возрастании s. Рассмотрев R_{s+1} в составном правиле трапеций, мы видим, что при возрастании числа элементарных отрезков меняется $h = (b-a)/s$ и меняется точка ξ, в которой вычисляется $f^{(2)}$, но не меняется порядок производной. Налицо сходство между рассматриваемой ситуацией и ситуацией из § 5 гл. 2, где речь шла об экстраполяции. Здесь можно использовать ту же идею. Дважды применив составное правило трапеций, – с s и $2s$ элементарными отрезками, – получим

$$|R_{s+1}| = \frac{b-a}{12}h^2|f^{(2)}(\xi_1)|, \quad |R_{2s+1}| = \frac{b-a}{12}\left(\frac{h}{2}\right)^2|f^{(2)}(\xi_2)|.$$

Если считать, что значения $f^{(2)}$ в точках ξ_1 и ξ_2 совпадают, то

$$|R_{2s+1}| = \frac{|R_{s+1}|}{4}.$$

Удвоение числа элементарных отрезков привело к уменьшению погрешности приблизительно в четыре раза. Говорят, что составное правило

трапеций имеет *второй порядок*, так как в выражении для остатка содержится множитель h^2. Составное правило Симпсона имеет *четвертый порядок*. При применении составного правила Симпсона удвоение числа элементарных отрезков уменьшает погрешность приблизительно в шестнадцать раз. Мы уже наблюдали аналогичные изменения погрешности интерполяции, когда рассматривали кусочно-полиномиальную интерполяцию в §8, 11 гл. 4. В самом деле, между кусочно-полиномиальной интерполяцией и квадратурными правилами существует тесная связь. Мы еще убедимся в этом в § 9.

Пример 5.4. Вычисление erf (1) **с помощью составного правила трапеций.**

Табл. 5.2 содержит результаты приближенного вычисления erf(1) с помощью составного правила трапеций при различных значениях s.

Таблица 5.2 Вычисление erf (1) с помощью составного правила трапеций

Число элементарных отрезков s	Правило трапеций	Погрешность R_{s+1}	R_{s+1}/R_{2s+1}
2	0.82526295	1.74×10^{-2}	–
4	0.83836777	4.33×10^{-3}	0.2485
8	0.84161922	1.08×10^{-3}	0.2496
16	0.84243051	2.70×10^{-4}	0.2499
32	0.84263323	6.76×10^{-5}	0.2500
64	0.84268390	1.69×10^{-5}	0.2500
128	0.84269657	4.22×10^{-6}	0.2500

В таблице приведены также множители, на которые умножаются погрешности при удвоении s. Как и следует ожидать, эти множители близки к 0.25. □

Эти идеи лежат в основе следующего практического способа оценки погрешности. Если вычислить два приближения, используя составное правило трапеций с s и $2s$ элементарными отрезками, то, как мы видели, $|R_{2s+1}| \approx |R_{s+1}|/4$. Обозначив вычисленные приближения через T_{s+1}, T_{2s+1}, получим

$$I = T_{s+1} + R_{s+1} = T_{2s+1} + R_{2s+1},$$

откуда

$$|T_{s+1} - T_{2s+1}| = |R_{s+1} - R_{2s+1}| \approx \frac{3}{4}|R_{s+1}|.$$

Таким образом, модуль разности двух последовательных приближений, умноженный на 4/3, представляет собой оценку погрешности квадратурного правила T_{s+1}. Оценка погрешности квадратурного правила T_{2s+1}

должна быть в 4 раза меньше. Практичный, но весьма осторожный метод заключается в том, чтобы принять за приближение к интегралу T_{2s+1}, а за оценку погрешности $\frac{4}{3}|T_{s+1} - T_{2s+1}|$.

Вот несколько соображений, которые связаны с созданием надежной программы, использующей описанный прием.

(1) Формулы трапеций и Симпсона относятся к формулам замкнутого типа, требующим вычисления $f(a)$ и $f(b)$. Они неприменимы в случае интегралов, обладающих неустранимыми особенностями, таких, как

$$\int_0^1 \ln x \cos x^2\, dx \quad \text{или} \quad \int_0^1 \frac{e^{\sin x}}{\sqrt{x}}\, dx.$$

Каждый из этих интегралов существует, т. е. площадь под каждой из кривых конечна, однако их подынтегральные функции обращаются в левом конце отрезка интегрирования в бесконечность. Многие практические задачи приводят к подобным интегрируемым особенностям. Возникающих здесь трудностей можно избежать, если использовать составное правило прямоугольников (см. задачу 5.2 (б)) или составное правило Гаусса.

(2) Вернемся к выводу правила трапеций (или правила Симпсона) в разд. 2.1. Мы исходили из предположения о том, что разложение подынтегральной функции в ряд Тейлора существует. Если функция f имеет на $[a, b]$ особенность, такую, как $f = \sqrt{x}$ на $[0, 1]$, то ее нельзя разложить в ряд Тейлора на всем отрезке и представление погрешности, а вместе с ним и сама формула трапеций окажутся необоснованными. Аналогичное замечание относится и к составным правилам; кроме того, в случае составных правил мы дополнительно предполагаем, что сумму большого числа значений производной в разных точках можно заменить значением производной в одной точке, умноженным на s. Действительно, для описанного выше способа оценки погрешности требуется, чтобы значения $f^{(2)}$ не сильно отличались друг от друга в разных точках ξ_i. Это не так, если f быстро осциллирует или каким-либо иным образом сильно меняется. В каждом из этих двух случаев ($f^{(2)}$ существует не на всем отрезке $[a, b]$ или $f^{(2)}$ слишком сильно меняется) составное правило трапеций все же можно применять, но прежние выводы о величине погрешности сделать нельзя. Вообще если квадратурное правило является суммой Римана, подобно правилу трапеций, то и соответствующее составное правило является суммой Римана. Такие правила с ростом числа узлов могут давать сколь угодно точные приближения, но скорость сходимости при этом зависит от гладкости f. В частности, удвоение числа элементарных отрезков может и не уменьшить погрешность в четыре раза.

(3) Если мы располагаем приближением, полученным с помощью составного правила трапеций с n узлами (т. е. с $n - 1$ элементарных

отрезков) и хотим получить более точное приближение, то естественно выбрать один из следующих способов.

(а) Увеличить число узлов на единицу, т. е. вычислить T_{n+1} (для n элементарных отрезков).

(б) Удвоить число узлов, т. е. вычислить T_{2n} (для $2n-1$ элементарных отрезков).

(в) Удвоить число элементарных отрезков, т. е. вычислить T_{2n-1} (для $2n-2$ элементарных отрезков).

При этом T_{n+1} лишь немногим точнее T_n, в то время как T_{2n} и T_{2n-1} точнее T_n приблизительно в четыре раза (для гладких f). Но легко заметить, что при переходе от T_n к T_{n+1} нужно вычислить $n-1$ *новых* значений функции f, при переходе к T_{2n} нужно вычислить $2n-1$ таких значений, а при переходе к T_{2n-1} – всего лишь $n-1$. В самом деле, если h – расстояние между узлами правила T_n, то

$$T_{2n-1} = \frac{1}{2}T_n + \frac{h}{2}\left[f\left(a+\frac{h}{2}\right) + f\left(a+\frac{3h}{2}\right) + \ldots + f\left(b-\frac{3h}{2}\right) + f\left(b-\frac{h}{2}\right)\right].$$

Поэтому в программах, которые осуществляют интегрирование с помощью составного правила трапеций, почти всегда используются способ (в) и приведенная выше формула.

В течение 60-х годов на основе составных правил были разработаны программы, в которых трудности (1) и (2) были преодолены. Одно из наиболее плодотворных направлений исследований состояло в использовании экстраполяции при оценке погрешности составного правила трапеций; результаты можно найти по названию «квадратуры Ромберга» в нескольких работах, указанных в гл. 1. Но с 1970 года наиболее популярными методами численного интегрирования в программах общего назначения стали адаптивные алгоритмы, описываемые в § 6. Однако интерес к экстраполяции не пропал; были разработаны усовершенствованные методы, которые можно использовать в сочетании с адаптивными алгоритмами. См., например, статью [de Doncker, Robinson, 1984].

5.5. Квадратурные правила Гаусса-Кронрода

В § 4 мы видели, что с помощью удвоения числа элементарных отрезков составное правило трапеций позволяет получить два приближения T_n и T_{2n-1}. Для гладких функций T_{2n-1} приблизительно в четыре раза точнее T_n, а общий объем работы по получению T_n и T_{2n-1} составляет $2n-1$ вычислений значений подынтегральной функции. В разд. 2.1 мы установили, что формулы Гаусса являются «наилучшими» при заданном числе узлов, а теперь нас интересует, как удачнее выбрать пару формул Гаусса, чтобы с их помощью получить и приближенное значение интеграла, и оценку погрешности. Обозначим n-точечное правило Гаусса через G_n; мы вправе ожидать, что G_{2n-1} представ-

ляет собой гораздо более точное приближение и что величину $|G_n - G_{2n-1}|$ можно принять за оценку погрешности. Но, как было отмечено выше, разные правила Гаусса не имеют общих узлов за исключением середины отрезка интегрирования. Поэтому общий объем работы по получению G_n и G_{2n-1} состоит из $3n-1$ (или $3n-2$, если n нечетно) вычислений значений подынтегральной функции. Аналогичная ситуация складывается при использовании G_n и G_{2n}. Если же использовать G_n и G_{n+1}, то общий объем работы составит $2n+1$ вычислений значений подынтегральной функции, что сравнимо с общим объемом работы при использовании составного правила трапеций, но при этом G_{n+1} будет лишь немногим точнее G_n.

В 1965 году задачу получения оценок погрешности квадратурных правил Гаусса исследовал с этой точки зрения советский специалист в области вычислительной математики Кронрод[1]. Идея Кронрода состояла в построении, наряду с правилом G_n, нового правила, узлами которого были бы как все узлы G_n, так и некоторые другие точки. Если

$$G_n = \sum_{i=1}^{n} w_i f(x_i),$$

Таблица 5.3 Пара Гаусса-Кронрода (7.15) для отрезка $[-1, 1]$

Узлы 7-точечного правила Гаусса	Веса
±0.94910 79123 42758	0.12948 49661 68870
±0.74153 11855 99394	0.27970 53914 89277
±0.40584 51513 77397	0.38183 00505 05119
0.00000 00000 00000	0.41795 91836 73469

Узлы 15-точечного правила Кронрода	Веса
±0.99145 53711 20813	0.02293 53220 10529
±0.94910 79123 42759	0.06309-20926 29979
±0.86486 44233 59769	0.10479 00103 22250
±0.74153 11855 99394	0.14065 32597 15525
±0.58608 72354 67691	0.16900 47266 39267
±0.40584 51513 77397	0.19035 05780 64785
±0.20778 49550 07898	0.20443 29400 75298
0.00000 00000 00000	0.20948 21410 84728

[1] Результаты А.С. Кронрода по уточняющим квадратурам опубликованы в книге [Кронрод А.С. Узлы и веса квадратурных формул. – М.: Наука, 1964]. – *Прим. перев.*

Кронрод предложил строить правила вида

$$K_{2n+1} = \sum_{i=1}^{n} a_i f(x_i) + \sum_{j=1}^{n+1} b_j f(y_j).$$

Отметим, что G_n и K_{2n+1} имеют n общих узлов. Кроме того, K_{2n+1} имеет $n+1$ дополнительных узлов и новые веса a_i, b_j. Алгебраическая степень точности правила Гаусса G_n равна $2n-1$. Значения $3n+2$ параметров a_i, b_j и y_j были найдены Кронродом так, чтобы алгебраическая степень точности правила K_{2n+1} была равна $3n+1$. В табл. 5.3 приведено правило K_{15} с 15 десятичными знаками[2]. Два правила (G_n, K_{2n+1}) называются *парой Гаусса-Кронрода*. Цена вычисления пары составляет $2n+1$ значений подынтегральной функции. Ту же цену имеет правило G_{2n+1}, алгебраическая степень точности которого равна $4n+1$, но оно не дает оценки погрешности. Та же цена и у двух правил G_n и G_{n+1}, с помощью которых можно получить оценку погрешности. Однако K_{2n+1} гораздо точнее, чем G_{n+1}. За оценку погрешности приближения G_n можно взять величину $|G_n - K_{2n+1}|$, а K_{2n+1} принять за приближенное значение интеграла. Такой подход представляется вполне разумным. Однако опыт показывает, что оценка $|G_n - K_{2n+1}|$ существенно превосходит погрешность более точного приближения K_{2n+1}. Экспериментально установлено, что следующий подход будет реалистическим и в то же время осторожным:

приближенное значение интеграла $= K_{2n+1}$;
оценка погрешности $= (200\,|G_n - K_{2n+1}|)^{1.5}$.

(На самом деле при компьютерных вычислениях оценка погрешности слегка масштабируется в зависимости от величины f, но при $f \approx 1$ она будет такой, какая указана здесь.) С первого взгляда может показаться, что такая оценка погрешности больше величины $|G_n - K_{2n+1}|$, но последняя обычно много меньше 1.0, и степень 1.5 делает ее еще меньше.

Если с помощью указанного правила вычислить с двойной точностью erf (1), перейдя от отрезка $[-1, 1]$ к отрезку $[0, 1]$, то получим

$$G_7 = 0.842700792948824892,$$
$$K_{15} = 0.842700792949714861,$$
$$(200\,|G_7 - K_{15}|)^{1.5} = 2 \cdot 10^{-15}.$$

Действительная погрешность составляет приблизительно $2 \cdot 10^{-17}$. Для сравнения отметим, что составное правило трапеций дает гораздо меньшую точность. Последняя строка табл. 5.2 в примере 5.4 содержит результат с погрешностью $4 \cdot 10^{-6}$, для получения которого потребовалось 129 вычислений значений подынтегральной функции.

Пара Гаусса-Кронрода вместе с приведенной выше оценкой погреш-

[2] В табл. 5.3 значения первого узла правила G_7 и второго узла правила K_{15} различаются в последнем разряде, чего не должно быть. *Прим. перев.*

ности сейчас представляет собой один из самых эффективных методов вычисления интегралов общего вида. Пара (G_7, K_{15}) является стандартной. Использование ее в сочетании с адаптивным квадратурным алгоритмом позволяет получить надежные и эффективные программы общего назначения.

Несмотря на столь положительные утверждения, читатель должен сознавать, что приближенные значения интеграла и погрешности, вычисленные указанным способом, могут все же оказаться неверными. Существуют функции со сравнительно хорошим поведением, для которых оба этих приближенных значения сколь угодно плохи, т.е. имеют неверный порядок величины и неверный знак. Если перейти к иному правилу или к иной оценке погрешности или сделать и то и другое, то этим можно лишь уменьшить вероятность, но нельзя полностью ликвидировать возможность получения неверного результата.

5.6. Автоматические и адаптивные квадратурные алгоритмы

Автоматический квадратурный алгоритм на основе входной информации – функции f, отрезка $[a, b]$ и требуемой точности ε – вычисляет результат Q и оценку погрешности E, которые, как можно надеяться, удовлетворяют неравенствам

$$|Q - I| < E < \varepsilon.$$

О функции f не делается никаких предположений, кроме одного: с помощью внешней подпрограммы можно вычислять значения $f(x)$ для любых заданных x. Большинство пользователей предпочитают алгоритмы, в которых задается *относительная точность*

$$|Q - I| < E \cdot |I| < \varepsilon \cdot |I|,$$

поэтому многие программы, например Q1DA из этой главы, работают именно с ней. Но поскольку случай $I \approx 0$ требует особой логики, мы рассмотрим только абсолютную точность.

Автоматический квадратурный алгоритм можно получить на основе составного правила трапеций. Будем вычислять T_{s+1} при $s = 1, 2, 4, 8, 16, \ldots$ до тех пор, пока не станет справедливым неравенство $\frac{4}{3}|T_{2s+1} - T_{s+1}| < \varepsilon$. При таком алгоритме точки, в которых вычисляются значения f, расположены на $[a, b]$ равномерно. От подынтегральной функции зависит лишь скорость сходимости алгоритма.

Применение автоматических квадратурных алгоритмов всегда требует больших затрат по сравнению с теми, которые понадобятся, если, изучив конкретную задачу, воспользоваться ее характерными особенностями и специальным образом организовать процесс вычислений. Часто дополнительные затраты составляют до нескольких сотен, а иногда и тысяч процентов. Однако автоматические квадратурные алгоритмы

удобны, и для многих задач общее компьютерное время оказывается все же достаточно малым, чтобы оправдать затраты. В частности, такая ситуация обычно имеет место на начальной стадии вычислительного эксперимента. Затем, на более поздних стадиях эксперимента, возникает потребность определить, какую выгоду в экономии человеческого и машинного времени приносит дополнительный анализ.

Адаптивный автоматический квадратурный алгоритм – это особый вид автоматического квадратурного алгоритма, пытающийся приспособить выбор узлов квадратурной формулы к каждой конкретной подынтегральной функции. Двумя важными компонентами такого алгоритма являются локальный квадратурный модуль (ЛКМ) и общая стратегия.

ЛКМ представляет собой процедуру, на входе которой находятся f и отрезок $[\alpha, \beta]$, а на выходе – $Q_{[\alpha,\beta]}$ и $E_{[\alpha,\beta]}$, называемые локальной квадратурой и локальной оценкой погрешности. Следующие примеры иллюстрируют два варианта выбора локальной квадратуры и локальной оценки погрешности.

Пример 5.5. Простые локальная квадратура и локальная оценка погрешности.

$$Q_{[\alpha,\beta]} = T_{15},$$
$$E_{[\alpha,\beta]} = \frac{4}{3}|T_{15} - T_8|.$$

Локальная оценка погрешности выбрана на основе рассуждений § 4. □

Пример 5.6. Реалистические локальная квадратура и локальная оценка погрешности.

$$Q_{[\alpha,\beta]} = K_{15},$$
$$E_{[\alpha,\beta]} = (200|G_7 - K_{15}|)^{1.5}.$$

Эта локальная оценка погрешности основана на формуле из § 5. □

Локальные квадратурные модули должны быть надежными. Обычно в них предусмотрен анализ ошибок округления и выявление нестандартного поведения. Тем не менее мы относимся к ним скорее как к «строительным блокам», а не как к завершенным алгоритмам.

Одна из наилучших общих стратегий, называемая *глобально адаптивной*, такова. ЛКМ применяется к исходному отрезку $[a, b]$. Если $E_{[a,b]} < \varepsilon$, то работа прекращается и выдаются $Q = Q_{[a,b]}$, $E = E_{[a,b]}$. В противном случае отрезок $[a, b]$ делится пополам, и ЛКМ применяется к каждой из половин. Если $E = E_{\left[a, \frac{a+b}{2}\right]} + E_{\left[\frac{a+b}{2}, b\right]} < \varepsilon$, то работа прекращается и выдаются E и $Q = Q_{\left[a, \frac{a+b}{2}\right]} + Q_{\left[\frac{a+b}{2}, b\right]}$. В противном случае делится

пополам та из половин, которая имеет бо́льшую локальную оценку погрешности. Общий шаг алгоритма состоит в следующем. Исходный отрезок $[a, b]$ заменяется множеством подотрезков различной длины; если сумма всех локальных оценок погрешности превосходит ε, то подотрезок, имеющий наибольшую локальную оценку погрешности, делится пополам. Когда сумма наконец становится меньше ε, работа прекращается и за результат Q принимается сумма локальных квадратур, а за оценку погрешности E – сумма локальных оценок погрешности.

Такая стратегия предполагает, что на отрезках достаточно малой длины подынтегральная функция имеет хорошее поведение и поэтому локальный квадратурный модуль обеспечивает достаточно точную локальную квадратуру и малую локальную оценку погрешности. Если поведение функции $f(x)$ в точке x_0 плохое, то эта точка будет заключаться в подотрезки все меньшей и меньшей длины. В конечном счете малой станет и локальная квадратура, пропорциональная длине подотрезка.

В практическом алгоритме нужно предусмотреть механизм, который позволит избежать излишнего измельчения подотрезков: либо посредством ограничения числа половинных делений, либо с помощью какой-либо иной, более сложной стратегии. Также необходимо учитывать влияние ошибок округления при вычислении значений подынтегральной функции, которое может оказаться существенным даже и на больших подотрезках. Способы, предлагаемые для этой цели, оказываются весьма специальными и меняются от программы к программе. Хороший общий обзор соответствующих алгоритмов, в который включены также и программы, можно найти в книге [Piessens et al., 1983].

Если изменить значения подынтегральной функции в конечном числе точек, то величина интеграла не изменится. Тем не менее результат, полученный с помощью автоматического квадратурного алгоритма, полностью определяется некоторым конечным набором значений подынтегральной функции. Поэтому среди автоматических квадратурных программ нет такой, которая действительно гарантировала бы, что приближенное значение интеграла или оценка погрешности хоть сколько-нибудь точны. Какова бы ни была программа, для нее легко подобрать задачу, которая заставит программу либо «сдаться», либо выдать неверный результат. Так может случиться, если набор значений $f(x_i)$ не отражает специфических черт подынтегральной функции. Пусть, например, требуется вычислить

$$I = \frac{1}{\sqrt{2\pi}} \int_{-200000}^{200000} t^2 \exp(-t^2/2)\, dt \approx 1.0.$$

Здесь при некотором обращении к подпрограмме EXP произойдет исчезновение порядка (аргумент -10^{10} слишком мал для любой подпрограммы, вычисляющей экспоненту). Если же предусмотреть это и полагать в таких случаях подынтегральную функцию равной нулю, то программа в конце концов почти наверняка выдаст в качестве значения

интеграла число 0. Подынтегральная функция почти на всем отрезке интегрирования практически равна нулю, за исключением небольшой части отрезка, длина которой составляет одну сотую процента от общей длины, равной 400 000. Если только не надеяться на счастливую случайность, автоматический квадратурный алгоритм нам здесь не поможет. Бдительного пользователя ничто не заменит! (См. также задачу 5.4.)

5.7. Подпрограммы Q1DA и QK15

Автоматическая квадратурная подпрограмма Q1DA имеет общее назначение. Пользователь пишет основную программу, которая вызывает Q1DA, а также фортранную подпрограмму-функцию с именем F, вычисляющую подынтегральную функцию. Q1DA – одна из подпрограмм набора, в который входят также подпрограммы Q1DAX и GL15T. Для работы последней требуется, чтобы функцию F можно было вычислить в любой точке отрезка интегрирования. Q1DAX представляет собой глобальный адаптивный квадратурный алгоритм, вызывающий локальный квадратурный модуль GL15T. Подпрограмма Q1DA служит драйвером для Q1DAX. Никаких вычислений Q1DA не производит; она лишь осуществляет задание ряда переменных и массивов и вызывает Q1DAX. В математическом программном обеспечении драйверы используются часто. Подпрограмма Q1DA выдает результат Q, оценку погрешности E, признак ошибки IFLAG, по которому пользователь может судить об успехе или неуспехе вычислений, и число KF обращений к подпрограмме-функции F, которое служит мерой произведенной работы.

Часто задача содержит некий параметр, «естественную частоту», позволяющий заранее определить, какими будут затраты. Пусть требуется проинтегрировать функцию $e^x \sin^2 100x$ по отрезку $[0, \pi]$; на этом отрезке помещается 50 периодов синуса. Даже оптимистически настроенный пользователь должен будет предположить, что на каждый период придется несколько вычислений подынтегральной функции. Поэтому если значение KF окажется не превосходящим 100, то это свидетельствует о возможной ошибке либо при постановке задачи, либо при использовании подпрограммы.

Работа подпрограммы Q1DA следует общей схеме § 6, но на два момента полезно обратить особое внимание.

(1) Элемент случайности. В описанном в § 6 алгоритме отрезок $[a, b]$ при первом разбиении делится пополам. Q1DA выбирает точку первого разбиения случайным образом так, чтобы она была близка к точке $(b + a)/2$, но не совпадала с ней; все последующие разбиения являются бисекциями. Из-за этого при всяком новом обращении к подпрограмме узлы будут слегка отличаться от тех, которые использовались при предыдущих обращениях. Если подынтегральная функция имеет острые пики, то можно обратиться к Q1DA два-три раза и сравнить результаты.

Это уменьшит вероятность того, что какая-то важная особенность поведения функции будет упущена.

(2) Сглаживание особенностей в концевых точках. В практических задачах интегралы часто обладают особенностями. Обычно подынтегральные функции имеют особенности на концах отрезка интегрирования. Бывает, что в таких случаях лучше всего сделать замену переменной. Если $x = p(t)$, $dx = p'(t)\,dt$, то

$$I = \int_a^b f(x)\,dx = \int_\alpha^\beta f(p(t))\,p'(t)\,dt = \int_\alpha^\beta g(t)\,dt,$$

где $p(\alpha) = a$, $p(\beta) = b$. Значение интеграла от f совпадает со значением интеграла от g, но последний может оказаться более простым для численного интегрирования. В Q1DA используется следующая замена:

$$p(t) = b - (b - a)\,u^2(2u + 3), \text{ где } u = \frac{t - b}{b - a}.$$

Отметим, что $p(a) = a$, $p(b) = b$ и $p'(a) = p'(b) = 0$. Если $f(x)$ имеет особенность в точке a или в точке b, то $g(t)$ может ее и не иметь. Например, если $f(x) = \ln x$ и $[a, b] = [0, 1]$, то

$$\int_0^1 \ln x\,dx = \int_0^1 \ln(t^2(3 - 2t))\,6t(1 - t)\,dt.$$

Хотя второй интеграл выглядит сложнее, численное нахождение его значения оказывается более простым. Подпрограмма Q1DA выполняет замену переменной автоматически. Если f не имеет особенностей в концевых точках, то замена переменной не оказывает существенного влияния. Кроме того, поскольку $|p'(t)| \leqslant p'((a + b)/2) = 3/2$, новая подынтегральная функция слегка увеличивается по модулю вблизи середины отрезка интегрирования, в то время как в концевых точках обращается в нуль.

Подпрограмма Q1DAX обладает и такими возможностями, которых подпрограмма Q1DA не имеет. Это возможность повторно обращаться к подпрограмме для вычислений с большей точностью без повторения предыдущих вычислений и возможность осуществлять интегрирование по более чем одному отрезку (что полезно, когда подынтегральная функция терпит разрыв в некоторой точке отрезка интегрирования; как правило, Q1DAX тратит значительно больше времени на то, чтобы локализовать эту точку.)

Глобальный адаптивный квадратурный алгоритм имеет и недостаток: число подотрезков растет. Для их хранения нужно выделить место в памяти, и в случае сложной задачи может наступить момент, когда памяти не хватит. К этому моменту среди подотрезков обычно уже есть такие, которые имеют небольшую локальную оценку погрешности и, по-видимому, не повлияют существенно на дальнейший ход вычислений. Вполне возможно, что эти подотрезки уже не придется подвергать

разбиению. Напомним, что разбиению подвергается тот подотрезок, который имеет наибольшую локальную оценку погрешности. Подпрограмма Q1DAX перестает хранить подотрезки с малыми локальными оценками погрешности, если отведенная ей память исчерпана. Часто этого бывает достаточно для того, чтобы близкие к завершению вычисления благополучно закончились. Подпрограмма Q1DA позволяет хранить 50 подотрезков; для многих практических задач такого количества достаточно. Пользователь может явно задать количество хранимых подотрезков в подпрограмме Q1DAX.

Подпрограмма Q1DAX также выдает на выходе наибольшее и наименьшее из вычисленных значений подынтегральной функции. Это полезно, если требуется построить график подынтегральной функции.

Следующая программа иллюстрирует использование Q1DA.

```
C     Иллюстрирующая программа для Q1DA
C
      A = 0.0
      B = 1.0
C     Задание концов отрезка интегрирования [0, 1]
      EPS = 0.001
C     Задание допустимой точности
      CALL Q1DA(A, B, EPS, R, E, KF, IFLAG)
      WRITE (*, *) A, B, EPS, R, E, KF, IFLAG
      END
C
C
      FUNCTION F(X)
C     Вычисление подынтегральной функции F
      F = SIN(2. * X)-SQRT(X)
      RETURH
      END
C
C     Результаты, полученные для этого примера
C
C     0.0 1.0 .001 .041406750 .69077 e - 07 30 0
```

Для многих квадратурных задач использование тонкой стратегии необязательно или нежелательно. Часто от программы требуется лишь, чтобы она была простой и, следовательно, быстрой. Подпрограмма QK15 – локальный квадратурный модуль, представляющий собой реализацию пары Гаусса-Кронрода (G_7, K_{15}), которая описана в § 5. Подпрограмма QK15 взята из Quadpack, унифицированного набора программ одномерного интегрирования, описанного в книге [Piessens et al., 1983]. Quadpack содержит как автоматические квадратурные программы, так и реализации шести различных пар Гаусса-Кронрода. Подпрограмма QK15 для

произвольной функции f и конечного отрезка $[a, b]$ вычисляет локальную квадратуру RESULT и локальную оценку погрешности ABSERR. Преобразования, описанные в § 3, осуществляются внутри подпрограммы, пользователь от них избавлен. Одно из преимуществ такого локального квадратурного модуля состоит в том, что заранее известен объем работы – пятнадцать вычислений значений подынтегральной функции. Те же значения QK15 использует и для вычисления двух других интегралов:

$$\text{RESABS} \approx \int_a^b |f|\,dx \text{ и } \text{RESASC} \approx \int_a^b \left| f - \frac{\text{RESULT}}{b-a} \right| dx.$$

Подпрограмма QK15, подобно подпрограмме Q1DA, требует, чтобы пользователь составил фортранную подпрограмму-функцию, вычисляющую подынтегральную функцию для произвольного x. Подпрограмма Q1DA не позволяет пользователю выбрать имя этой подпрограммы-функции, им должно быть F. Но подпрограмма QK15 является более гибкой. (Той же гибкостью обладает подпрограмма Q1DAX.) Вы можете выбрать для подпрограммы-функции любое имя, однако оно должно появиться *по крайней мере в трех местах:* (1) в операторе EXTERNAL в программе, вызывающей QK15; (2) как имя составленной вами подпрограммы-функции; (3) в списке параметров при каждом вызове QK15. В Фортране предусмотрено следующее: если один из параметров подпрограммы или подпрограммы-функции в свою очередь представляет собой имя подпрограммы или подпрограммы-функции, то с ним надлежит обращаться иначе, чем с другими параметрами; об этом и предупреждает оператор EXTERNAL. Вот пример использования подпрограммы QK15, иллюстрирующий применение оператора EXTERNAL.

```
C     Вычисление erf (1), т.е. интеграла
C     2/sqrt(pi)*exp(-x*x) от 0 до 1.0
      EXTERNAL FUNC
      REAL A, B, RESULT, ABSERR, RESABS, RESASC, PI
C
      A = 0.0
      B = 1.0
      PI = 4.0*ATAN(1.0)
      CALL QK15 (FUNC, A, B, RESULT, ABSERR, RESABS, RESASC)
      WRITE(*, *)' ЗНАЧЕНИЕ ERF(1), ВЫЧИСЛЕННОЕ QK15'
      WRITE (*, *)' 2.0/SQRT(PI) *RESULT, ABSERR',
      WRITE(*, *) 2.0/SQRT (PI)*RESULT, ABSERR
      STOP
      END
C
      REAL FUNCTION FUNC (X)
        REAL X
        FUNC = EXP(-X*X)
        RETURN
      END
```

```
C     Выходные данные (с IBM PC/AT)
C     ЗНАЧЕНИЕ ERF (1), ВЫЧИСЛЕННОЕ QK15
C     2.0*SQRT(PI)*RESULT, ABSERR
C     0.842701 0.50229e-4
```

Отметим, что каждое вычисление подынтегральной функции требует вызова подпрограммы-функции FUNC. Более эффективным было бы сделать FUNC подпрограммой и передавать ей массив абсцисс. Это заменило бы пятнадцать вызовов подпрограммы-функции на один вызов подпрограммы. Далее, в подпрограмме FUNC вычисления для каждой из абсцисс независимы, и на некоторых компьютерах Фортран-транслятор мог бы организовать процесс счета так, чтобы несколько значений подынтегральной функции вычислялись параллельно. К сожалению, квадратурных программ общего назначения, которые использовали бы подобное преимущество, до сих пор не создано, и поэтому здесь мы будем продолжать следовать традиционной технике.

5.8. Интегрирование таблично заданных функций

Пусть задана последовательность пар чисел

$$(x_i, y_i),\ x_1 < x_2 < \ldots < x_n,$$

которую мы будем интерпретировать как таблицу точных значений некоторой (неизвестной) функции $f(x)$:

$$f(x_i) = y_i.$$

Требуется вычислить интеграл

$$I = \int_{x_1}^{x_n} f(x)\,dx.$$

Поскольку данные точны, представляется разумным проинтерполировать точки (x_i, y_i) так, как это делалось в гл. 4. Тогда мы получим функцию $g(x)$, для которой $g(x_i) = y_i$, а затем вычислим

$$I \approx I' = \int_{x_1}^{x_n} g(x)\,dx.$$

К наиболее широко используемым интерполянтам относятся полиномы, кусочно-полиномиальные, и в частности кусочно-линейные функции, эрмитовы кубические функции и сплайны. Кусочно-полиномиальные функции дают наиболее простые квадратурные правила. Ими мы займемся ниже. Однако если наши данные из-за ошибок эксперимента или вычислительных погрешностей являются лишь приближенными, то использование аппарата интерполирования оказывается нецелесообразным; в этом случае следует обратиться к приемам, описанным в гл. 6.

Пусть $g(x)$ - функция, соединяющая каждую пару соседних точек

отрезком прямой. Это *кусочно-линейный интерполянт*, описанный в § 7 гл. 4 и обозначенный там через $L(x)$. Тогда

$$I \approx I' = \int_{x_1}^{x_n} g(x)\,dx = \sum_{i=1}^{n-1} \int_{x_i}^{x_{i+1}} g(x)\,dx.$$

Поскольку функция g линейна на отрезке $[x_i, x_{i+1}]$, ее можно проинтегрировать точно:

$$\begin{aligned} I \approx I' &= \frac{1}{2}\sum_{i=1}^{n-1}(x_{i+1}-x_i)(y_i+y_{i+1}) = \\ &= \frac{1}{2}((x_2-x_1)y_1 + (x_3-x_1)y_2 + \\ &+ (x_4-x_2)y_3 + \ldots + (x_n - x_{n-2})y_{n-1} + (x_n - x_{n-1})y_n). \end{aligned}$$

Отметим, что если точки x_i расположены на равном расстоянии друг от друга, то мы получаем составное правило трапеций, а в случае произвольных x_i – обобщение правила трапеций. Это правило дает *точное* значение интеграла кусочно-линейной функции $g(x)$, интерполирующей наши данные.

* 5.8.1. Интегрирование таблично заданных функций: эрмитова кубическая квадратура

Если в качестве интерполянта использовать эрмитову кубическую функцию $c(x)$, определенную в § 10 гл. 4,

$$g(x) \equiv c(x) = \sum_{i=1}^{n} (y_i c_i(x) + d_i \hat{c}_i(x)),$$

то получим

$$I \approx I' = \int_{x_1}^{x_n} c(x)\,dx = \sum_{i=1}^{n} \left(y_i \int_{x_1}^{x_n} c_i(x)\,dx + d_i \int_{x_1}^{x_n} \hat{c}_i(x)\,dx\right).$$

Интегралы в правой части можно вычислить, так как мы располагаем явными формулами для c_i и $\hat{c}_i$. Можно показать, что

$$I' = \sum_{i=1}^{n} \alpha_i y_i + \sum_{i=1}^{n} \beta_i d_i,$$

где

$$\alpha_i = \begin{cases} (x_2 - x_1)/2, & \text{если } i = 1, \\ (x_{i+1} - x_{i-1})/2, & \text{если } i = 2, \ldots, n-1, \\ -(x_n - x_{n-1})/2, & \text{если } i = n; \end{cases}$$

$$\beta_i = \begin{cases} (x_2 - x_1)^2/12, & \text{если } i = 1, \\ (x_{i+1} - x_{i-1})(x_{i+1} - 2x_i + x_{i-1}), & \text{если } i = 2, \ldots, n-1, \\ -(x_n - x_{n-1})^2/12 & \text{если } i = n. \end{cases}$$

На практике величины d_i нам неизвестны. Но если определить их с помощью сплайн-интерполяции, как в гл. 4, или каким-либо иным способом, то мы получим правило, дающее весьма точное приближение.

Эрмитову кубическую квадратуру можно интерпретировать следующим образом. Заметим, что первая из сумм $\sum_{i=1}^{n} \alpha_i y_i$ в точности совпадает с обобщенным правилом трапеций. Ей соответствует площадь под отрезками, соединяющими точки (x_i, y_i). При интерполировании эрмитовой кубической функцией осуществляется сглаживание. Между отрезками, соединяющими точки (x_i, y_i) с точками (x_{i+1}, y_{i+1}), и кубическими параболами, соединяющими те же точки, имеются небольшие «шапочки» (или «чашечки»). Вторая сумма $\sum_{i=1}^{n} \beta_i d_i$ представляет собой сумму их площадей.

Единственный важный частный случай – это случай равноотстоящих узлов. При этом все β_i, за исключением β_1 и β_n, обращаются в нуль, а сумма $\sum_{i=1}^{n} \alpha_i y_i$ становится составным правилом трапеций. Таким образом, мы приходим к важному результату:

$$I' = h(f(x_1)/2 + f(x_2) + \ldots + f(x_{n-1}) + f(x_n)/2) + \frac{h^2}{12}(d_1 - d_n);$$

составное правило трапеций + поправочный член = эрмитова кубическая квадратура.

Отметим, что для любого набора значений d_i производной во внутренних узлах будет получено одно и то же приближенное значение интеграла, несмотря на то, что интерполянты различны.

Если физическая модель предполагает, что исходная функция $f(x)$ периодическая и $f'(x_1) = f'(x_n)$, то поправочный член исчезает, и мы получаем в точности составное правило трапеций. Из рассуждений § 4 никак не следует, что случай периодической функции $f(x)$ является особым. Однако можно показать, что составное правило трапеций часто дает исключительно точное приближение, когда гладкая периодическая подынтегральная функция задана в равноотстоящих точках.

Пример 5.7. Составное правило трапеций в случае периодических функций.

Функция $f(x) = 1/(1 + 0.5 \sin 10\pi x)$ является периодической с периодом 1/5. Строго говоря, мы имеем дело не с таблично заданной функцией, так как $f(x)$ можно вычислить в любой точке. Но свойство периодичности указывает на то, что вычисление функции в равноотстоящих узлах и использование правила трапеций дадут хороший результат. Приближенные значения интеграла от $f(x)$ по отрезку [0, 1], полученные с помощью составного правила трапеций с 2, 4, 8 и 16

узлами, равны 1.0, 1.16666650, 1.15476180 и 1.15470050. Погрешность последнего приближения составляет $4 \cdot 10^{-8}$. Приближения, полученные с помощью квадратур Гаусса (также с 2, 4, 8 и 16 узлами), оказываются гораздо менее точными. □

5.8.2. Подпрограмма PCHQA

Для многих задач формулы разд. 5.8 и 5.8.1 настолько просты, что их можно запрограммировать непосредственно. Но поскольку интегрирование связано с эрмитовыми кубическими функциями, соответствующие подпрограммы есть в пакете PCHIP, описанном в § 9 гл. 4. Здесь мы рассмотрим простой в обращении драйвер PCHQA. Эта подпрограмма требует, чтобы пользователь сформировал входные массивы, содержащие абсциссы, соответствующие им значения функции, а также значения d_i производной, соответствующие тем же абсциссам. Кроме того, должен быть задан отрезок интегрирования $[a, b]$, такой, что $x_1 \leqslant a \leqslant b \leqslant x_n$. Следовательно, PCHQA можно использовать для интегрирования по отрезку, начало и конец которого не обязательно принадлежат множеству заданных абсцисс. Обычно перед вызовом PCHQA приходится обращаться к подпрограмме PCHEZ, чтобы вычислить неизвестные значения производной (см. § 9 гл. 4), но PCHQA можно вызвать и непосредственно. В распространенном случае, когда абсциссы являются равноотстоящими и $[a, b] = [x_1, x_n]$, величины d_2, ..., d_{n-1}, как нам известно из разд. 8.1, не входят в квадратурную формулу, и их можно положить равными нулю. Недостаток PCHQA состоит в том, что поскольку программа рассчитана на общий случай, то даже равноотстоящие абсциссы при обращении к ней должны задаваться так, как если бы они были неравноотстоящими. Пример использования этой программы можно найти в § 9 гл. 4.

5.9. Бесконечные и полубесконечные отрезки

С помощью подпрограммы QAGI, входящей в Quadpack, можно вычислить многие несобственные интегралы первого рода. Подпрограмма QAGI принадлежит к небольшому числу высококачественных программ вычисления интегралов по бесконечным отрезкам, ни одна из которых все же не отличается той надежностью и той универсальностью, которой обладают программы интегрирования по конечным отрезкам. Выбор наилучших способов интегрирования существенно зависит от индивидуальных особенностей задачи. Здесь описаны основные применяющиеся на практике методы, самыми распространенными из которых являются метод усечения (разд. 9.1) и метод весовых функций (разд. 9.4). К сожалению, невозможно указать какой-либо один способ или одно правило, которые во всех ситуациях работали бы настолько хорошо, что ими можно было бы ограничиться.

5.9.1. Усечение

Если один из концов отрезка интегрирования $[a, b]$ или оба они бесконечны, то обычно поступают так: бесконечные концы заменяют конечными, после чего интеграл по конечному отрезку вычисляют с помощью какого-либо стандартного метода. Часто удается показать, что вклад, который вносят в значение интеграла «хвосты», пренебрежимо мал, но еще чаще интегрирование производят по конечным отрезкам разной длины и, сравнив полученные результаты, убеждаются в том, что приближенное значение интеграла не меняется. Такой способ может в общем случае дать неверный результат, но часто, будучи подкреплен физической интуицией, он себя оправдывает.

Пример 5.8. Вычисление несобственного интеграла методом усечения. Вычислим

$$I = \int_0^\infty \exp(-x)\cos^2(x^2)\,dx = 0.70260\ldots .$$

Обозначив подынтегральную функцию через $f(x)$, запишем

$$I - \int_0^A f(x)\,dx = \int_A^\infty f(x)\,dx < \int_A^\infty \exp(-x)\,dx = \exp(-A).$$

Итак, если отбросить «хвост», то допущенная при этом ошибка не превзойдет $\exp(-A)$. Рис. 5.2, на котором показан график подынтегральной функции, иллюстрирует применение метода усечения. □

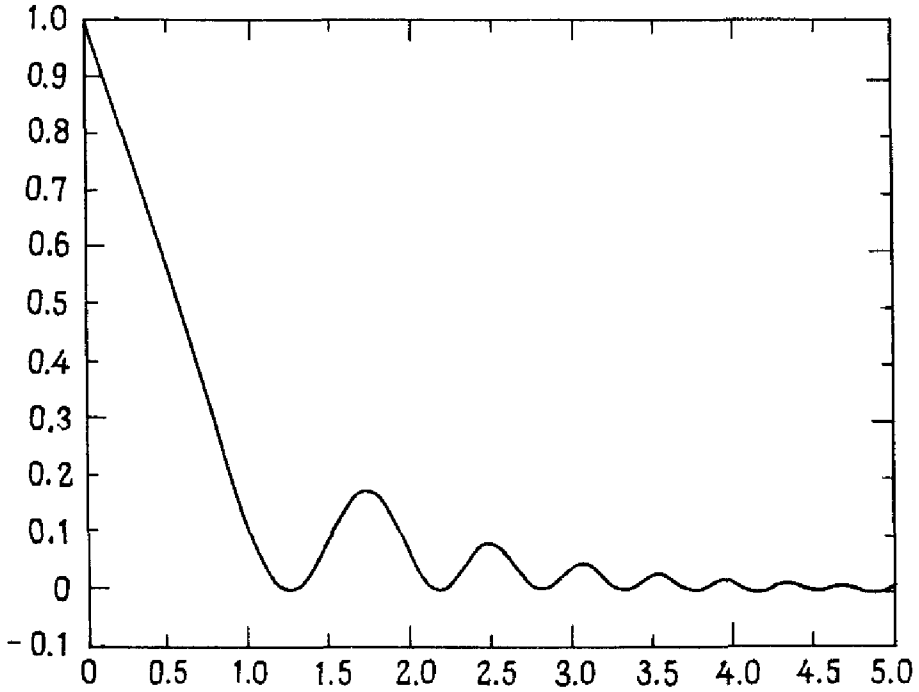


Рис. 5.2. Подынтегральная функция из примера 5.8.

5.9.2. Замена переменной

Замена $x = p(t)$ позволяет перейти от одного отрезка интегрирования к другому. Для отрезка $[0, \infty]$ типичны замены $x = -\ln t$ и $x = t/(1-t)$. В первом случае получим

$$\int_0^\infty f(x)\,dx = -\int_1^0 f(-\ln t)\frac{dt}{t} = \int_0^1 f(-\ln t)\frac{dt}{t}.$$

Для интеграла из примера 5.8 это дает

$$I = \int_0^1 \cos^2(\ln^2 t)\,dt.$$

Здесь подынтегральная функция совершает бесконечно много колебаний от 0 до 1 на любом отрезке, содержащем точку 0. Если использовать замену $x = p(t) = -2\ln t$, то придем к подынтегральной функции $2t\cos^2(4\ln^2 t)$. Она также осциллирует, но, поскольку она ограничена функцией $y = t$, ее предел при t, стремящемся к нулю, равен нулю. Если пренебречь интегралом по отрезку $[0, \varepsilon]$, то оставшийся интеграл вычислить не так уж сложно. Но, с другой стороны, подпрограмма Q1DA справится с каждым из полученных в результате замен интегралов одинаково хорошо. Например, если обратиться к Q1DA с первым из интегралов, задав точность 10^{-5}, то получим $I = 0.7026013 \pm 6\cdot 10^{-6}$ при 870 вычислениях значений подынтегральной функции. Для второго интеграла получим $I = 0.7026032 \pm 7\cdot 10^{-6}$ при 930 вычислениях значений подынтегральной функции.

Замену переменной следует производить весьма тщательно, иначе интеграл будет сведен к интегралу по конечному отрезку, но не станет проще. С другой стороны, хорошо подобранная замена способна вызвать поистине чудесные превращения. Этот вопрос по-прежнему активно изучается. Полезные сведения на эту тему можно найти в работе [Murota, Iri, 1982].

*5.9.3. Правило трапеций для бесконечного отрезка

Выбрав сетку с шагом h, рассмотрим правило трапеций в применении к интегралам по полубесконечным или бесконечным отрезкам

$$\int_0^\infty f(x)\,dx \approx h\sum_{k=1}^{\infty} f(kh),$$

$$\int_{-\infty}^{\infty} f(x)\,dx \approx h\sum_{k=-\infty}^{\infty} f(kh).$$

На практике нужно выбирать не только шаг h, но и конечный предел суммирования N. Одна из возможных стратегий состоит в вычислении последовательности приближений при убывании h и возрастании N, когда произведение Nh^2 остается приблизительно постоянным. С по-

мощью подобных правил весьма эффективно решаются многие задачи, особенно такие, в которых при стремлении x к $-\infty$ или к $+\infty$ велика скорость стремления подынтегральной функции к нулю. Например, если для задачи $I = 1/\sqrt{\pi} \int_{-\infty}^{\infty} \exp(-x^2)\,dx$ выбрать $N = 10$ и $h = 1/\sqrt{N}$, то правило трапеций даст результат с погрешностью $2 \cdot 10^{-6}$. С другой стороны, для подынтегральной функции $1/(1 + 10x^2)$ результат при $N = 100$ будет иметь погрешность 0.001, а при $N = 1000$ – погрешность $6 \cdot 10^{-3}$.

Успех метода зависит от поведения подынтегральной функции $f(x)$ при стремлении x к $-\infty$ или к $+\infty$. В связи с этим некоторые исследователи предлагают использовать замену переменной $x = p(t)$, которая оставляет отрезок интегрирования бесконечным, но увеличивает скорость стремления к нулю новой подынтегральной функции $p'(t)f(p(t))$ при стремлении t к $-\infty$ или к $+\infty$. Идея эта сходна с той, которая использовалась при осуществлении замены переменной в подпрограмме Q1DA; напомним, что там имели место равенства $p'(a) = p'(b) = 0$. Например, замена переменной

$$x = p(t) = \exp(t) - \exp(-t)$$

не меняет отрезка интегрирования $(-\infty, +\infty)$, но может увеличить скорость убывания подынтегральной функции при стремлении t к $\pm\infty$. Интеграл примет вид

$$\int_{-\infty}^{\infty} f(x)\,dx = \int_{-\infty}^{\infty} (e^t + e^{-t}) f(e^t - e^{-t})\,dt.$$

Если осуществить такое преобразование для подынтегральной функции $1/(1 + 10x^2)$, а затем воспользоваться правилом трапеций, то при $N = 100$ будет получен результат с погрешностью $8 \cdot 10^{-5}$, а при $N = 1000$ результат будет содержать 15 верных знаков. Иногда подобное преобразование можно применять несколько раз.

5.9.4. Весовые функции и квадратуры Гаусса – Лагерра

В интегралах по бесконечным отрезкам подынтегральные функции часто содержат множители определенного вида. Два из наиболее распространенных множителей – это $\exp(-x)$ и $\exp(-x^2)$. Обозначим любой из них через $q(x)$. Метод введения *весовой функции* состоит в построении формул вида

$$I = \int_a^b q(x) f(x)\,dx = \sum_{i=1}^{n} w_i \quad f(x_i) + R_n.$$

Пределы интегрирования a, b могут быть как конечными, так и бесконечными. Веса и узлы в подобных формулах как бы содержат аналитическую информацию о $q(x)$. Таким образом, мы получаем обобщение подхода из раздела 2.1, где неявно полагалось $q(x) = 1$.

В качестве примера правила с весовой функцией рассмотрим правило

$$\int_0^\infty \exp(-x) f(x)\,dx \approx w_1 f(x_1) + w_2 f(x_2).$$

Как и в разделе 2.1, два узла и два веса мы считаем неизвестными и пытаемся найти их из условия точности формулы для $f(x) = 1$, x, x^2 и x^3. Отсюда, как и раньше, мы получаем систему четырех нелинейных уравнений

$$i! = \int_0^\infty \exp(-x)\, x^i dx = w_1 (x_1)^i + w_2 (x_2)^i,\ i = 0,\ 1,\ 2,\ 3.$$

Эта система имеет единственное решение. Многое из того, что было справедливым в разделе 2.1, остается справедливым и сейчас. Для двухточечного правила мы получаем

$$\int_0^\infty \exp(-x) f(x)\,dx \approx \frac{2+\sqrt{2}}{4} f(2-\sqrt{2}) + \frac{2-\sqrt{2}}{4} f(2+\sqrt{2}).$$

Это двухточечное правило называют *квадратурным правилом Гаусса–Лагерра*, так как его узлы являются нулями полинома Лагерра [1] второй степени. Три подобных квадратурных правила приведены в табл. 5.4, однако в большинстве программных библиотек научного назначения имеются подпрограммы, которые вычисляют узлы и веса правил Гаусса–Лагерра для любого n.

Аналогичные правила получены для произвольного n и произвольной весовой функции $q(x)$; существует полная теория построения таких правил, по крайней мере для положительных $q(x)$. Правила эти дают хорошие результаты, когда функция f хорошо приближается полиномами, а иногда и в более общих ситуациях. Другие примеры подобных формул приводятся в книге [Davis, Rabinowitz, 1984].

Пример 5.9. Вычисление несобственного интеграла с помощью правила Гаусса–Лагерра.

Найдем приближенное значение интеграла

$$I = 1.046\ldots = \int_0^\infty \exp(-x)\, x^{1.2}\,dx.$$

[1] Эдмон Никола Лагерр (1834–1886) – французский математик, вся профессиональная деятельность которого проходила в Политехнической школе в Париже. Большинство его работ принадлежит к той области математики, которую называют теперь аналитической геометрией, хотя важных успехов добился он и в анализе. Помимо квадратурных формул, о которых идет речь, с его именем связан также класс дифференциальных уравнений. О нем говорили как о спокойном, мягком человеке, страстно преданном научным исследованиям, преподаванию и воспитанию двух своих дочерей.

Таблица 5.4 Некоторые квадратурные правила Гаусса-Лагерра для интеграла $\int_0^\infty \exp(-x)f(x)dx$

Узлы	Веса
$n = 2$	
0.585786437626905	0.853553390593274
$0.341421356237310 \times 10^{1}$	0.146446609406726
$n = 4$	
0.322547689619392	0.603154104341634
$0.174576110115835 \times 10^{1}$	0.357418692437800
$0.453662029692113 \times 10^{1}$	$0.388879085150054 \times 10^{-1}$
$0.939507091230113 \times 10^{1}$	$0.539294705561327 \times 10^{-3}$
$n = 8$	
0.170279632305101	0.369188589341638
0.903701776799380	0.418786780814343
$0.225108662986613 \times 10^{1}$	0.175794986637172
$0.426670017028766 \times 10^{1}$	$0.333434922612157 \times 10^{-1}$
$0.704590540239347 \times 10^{1}$	$0.279453623522567 \times 10^{-2}$
$0.107585160101810 \times 10^{2}$	$0.907650877335821 \times 10^{-4}$
$0.157406786412780 \times 10^{2}$	$0.848574671627253 \times 10^{-6}$
$0.228631317368893 \times 10^{2}$	$0.104800117487151 \times 10^{-8}$

Двухточечное правило Гаусса–Лагерра дает

$$I \approx \frac{2+\sqrt{2}}{4}(2-\sqrt{2})^{1.2} + \frac{2-\sqrt{2}}{4}(2+\sqrt{2})^{1.2} = 1.089\ldots. \quad \square$$

*5.9.5. Правило th

Правило трапеций так хорошо работает в применении к интегралам по бесконечным отрезкам, что можно предложить такой подход к вычислению интегралов *по конечным отрезкам*: сводить эти интегралы к интегралам по бесконечным отрезкам таким образом, чтобы новые подынтегральные функции стремились к нулю при стремлении аргумента к бесконечности. Одна из возможных замен переменной

$$x = p(t) = \operatorname{th}\left(\frac{t}{2}\right) = \frac{e^{t/2} - e^{-t/2}}{e^{t/2} + e^{-t/2}}$$

преобразует интеграл к виду

$$I = \int_{-1}^{1} f(x)\,dx = \int_{-\infty}^{\infty} f(p(t))p'(t)\,dt = 2\int_{-\infty}^{\infty} \frac{1}{(e^{t/2}+e^{-t/2})^2} f\left(\frac{e^{t/2}-e^{-t/2}}{e^{t/2}+e^{-t/2}}\right) dt.$$

Теперь интеграл можно приблизить выражением

$$I \approx h \sum_{k=-N}^{N} p'(kh) f(p(kh)) = 2h \sum_{k=-N}^{N} \frac{1}{(e^{kh/2}+e^{-kh/2})^2} f\left(\frac{e^{kh/2}-e^{-kh/2}}{e^{kh/2}+e^{-kh/2}}\right),$$

в котором в качестве шага выбрана величина

$$h = \pi\sqrt{\frac{2}{N} - \frac{1}{N}}.$$

Такое преобразование выглядит странным, так как получившееся в результате правило вычисления интегралов по отрезку $[-1, 1]$ не является точным ни для какого полинома – даже для константы – ни при каких положительных h и конечных N. Тем не менее оно дает замечательную точность. Для примера положим $N = 5$, $h = \pi\sqrt{0.4} - \frac{1}{5}$. В табл. 5.5 приведены значения $p(kh)$ и $2hp'(kh)$, являющиеся нулями и весами квадратуры. Отметим, что правило имеет $2N + 1$ узлов. Узлы расположены симметрично относительно нуля, который сам входит в число узлов.

Таблица 5.5 Узлы и веса квадратуры th для $N = 5$

Узлы	Веса
0.000000	0.893459
±0.713098	0.439127
±0.945434	0.094844
±0.990649	0.016631
±0.998428	0.002807
±0.999737	0.000471

Мы видим из таблицы, что большинство нулей группируются вблизи точек ± 1. Иногда это приводит к неприятностям вычислительного характера: подынтегральные функции, которые содержат такие множители, как $1 - x$, нельзя непосредственно вычислять на некоторых компьютерах из-за исчезновения порядка: кроме того, многие подынтегральные функции вблизи точек ± 1 нужно вычислять особым образом, чтобы избежать чрезмерной потери точности.

Пример 5.10. Вычисление интеграла по конечному отрезку с помощью правила th.

Вычислим с помощью правила th интеграл

$$\int_{-1}^{1} \exp(-x^2)\ln(1-x)\,dx.$$

Ниже приведены результаты при различных значениях N и h.

Таблица 5.6

N	h	Правило
5	1.787	−0.306715
10	1.304	−0.315748
20	0.943	−0.316688
30	0.778	−0.316713
40	0.677	−0.316714

В приближении, соответствующем $N = 40$ (81 вычисление подынтегральной функции), все цифры верные. □

5.10. Двойные интегралы

Часто требуется вычислить интеграл

$$I = \int\int_D f(x, y)\,dA,$$

где D — область в плоскости $x - y$. Задача эта гораздо сложнее, чем задача вычисления интеграла от функции одной переменной. Здесь остается немало «белых пятен» и продолжаются активные исследования.

По аналогии с определением из § 2, n-точечной квадратурной формулой называется формула

$$I = \sum_{i=1}^{n} w_i f(x_i, y_i) + R_n.$$

Будем говорить, что формула имеет алгебраическую степень точности d, если $R_n = 0$ для произвольного полинома от двух переменных степени d и $R_n \neq 0$ для некоторого полинома степени $d + 1$. Полиномом от двух переменных степени d называется линейная комбинация выражений вида $x^p y^q$, где $p + q \leqslant d$.

Большие семейства квадратурных формул построены для прямоугольника, треугольника, круга и других областей, считающихся стан-

дартными. Что касается областей более общего вида, то с ними иногда поступают одним из следующих способов.

(1) Область интегрирования помещают в прямоугольник и доопределяют функцию $f(x, y)$ так, чтобы вне области D она равнялась нулю. Тогда интеграл по прямоугольнику равен интегралу по D. К сожалению, новая подынтегральная функция в новой прямоугольной области имеет скачки.

(2) Область D аппроксимируют объединением треугольников. Если граница D не является ломаной, то для такой аппроксимации может потребоваться много маленьких треугольников. Поскольку интегрирование по каждому из треугольников осуществляется с некоторой погрешностью, нужно, чтобы погрешность, возникающая из-за того, что часть области осталась непокрытой треугольниками, и погрешность интегрирования по объединению треугольников приблизительно совпадали.

(3) Осуществляют преобразование, переводящее область D в стандартную. Такой способ не является столь общим, как два предыдущих; однако в следующем разделе мы покажем, как интеграл по треугольнику можно преобразовать в интеграл по квадрату.

*5.10.1. Мультипликативные правила для прямоугольника и треугольника

Если

$$\int_{\alpha}^{\beta} f(x)\,dx = \sum_{i=1}^{n} w_i f(x_i) + R_1$$

и

$$\int_{a}^{b} g(y)\,dy = \sum_{j=1}^{m} p_j g(y_j) + R_2,$$

то

$$\begin{aligned}
\int_a^b \int_\alpha^\beta f(x, y)\,dx\,dy &= \int_a^b \Big(\sum_{i=1}^{n} w_i f(x_i, y) + R_1\Big)\,dy = \\
&= \sum_{i=1}^{n} w_i \int_a^b f(x_i, y)\,dy + \int_a^b R_1\,dy = \\
&= \sum_{i=1}^{n} w_i \Big(\sum_{j=1}^{m} p_j f(x_i, y_j) + R_2\Big) + \int_a^b R_1\,dy = \\
&= \sum_{i=1}^{n}\sum_{j=1}^{m} w_i p_j f(x_i, y_j) + \sum_{i=1}^{n} w_i R_2 + \int_a^b R_1\,dy = \\
&= \sum_{i=1}^{n}\sum_{j=1}^{m} w_i p_j f(x_i, y_j) + R.
\end{aligned}$$

Такая формула называется *мультипликативной формулой*, так как использует nm узлов. На рисунке 5.3 показаны $2 \times 3 = 6$ узлов мульти-

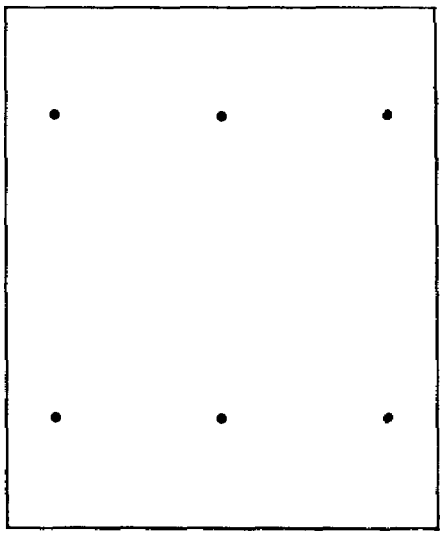


Рис. 5.3. Мультипликативное правило, полученное из двухточечного и трехточечного правил Гаусса.

пликативной формулы для прямоугольника, которая получена из трехточечной и двухточечной квадратур Гаусса. Веса мультипликативной формулы представляют собой произведения двух «одномерных» весов.

Если формулы одномерного интегрирования имеют алгебраические степени точности d_1 и d_2, то мультипликативная формула будет точно интегрировать полиномы $x^p y^q$, где $p \leqslant d_1$, $q \leqslant d_2$. Поэтому мультипликативная формула точна для любых полиномов степени $\min(d_1, d_2)$; кроме того, она точна для некоторых полиномов степени $d_1 + d_2$. Мультипликативные правила легко выводятся и в связи с этим пользуются популярностью. Их недостаток в слишком большом числе узлов. Например, мультипликативное правило, полученное из двух трехточечных правил Гаусса, имеет девять узлов, а его алгебраическая степень точности равна 5. Между тем существует немультипликативное правило, алгебраическая степень точности которого также равна 5, но которое имеет лишь семь узлов.

Число областей, которые могут быть преобразованы в прямоугольник, удивительно велико. Хотя осуществление преобразования иногда влечет за собой дополнительные трудности, это не умаляет ценности приема. Для примера рассмотрим интеграл по треугольнику

$$I = \iint_{\Delta} f(x, y)\, dA, \qquad \Delta = \{(x, y): 0 \leqslant x \leqslant 1,\ 0 \leqslant y \leqslant x\}.$$

В результате замены

$$u = x, \qquad v = y/x$$

интеграл принимает вид

$$I = \int_0^1 \int_0^x f(x, y)\, dy\, dx = \int_0^1 \int_0^1 f(u, uv)\, u\, dv\, du.$$

Теперь мы можем применить мультипликативное правило для квадрата. Конечно, это своего рода алгебраический трюк; его полезность, как

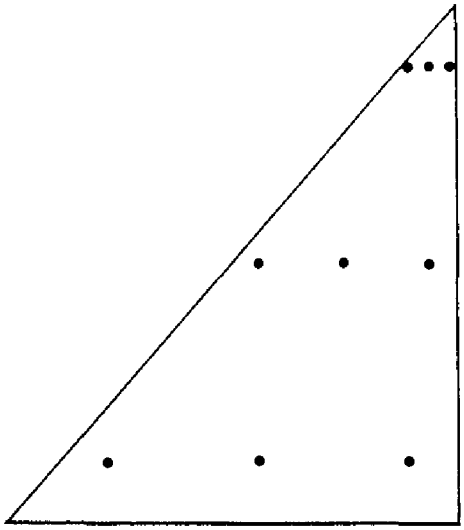


Рис. 5.4. Мультипликативное правило для треугольника.

и полезность всякого преобразования, зависит от конкретной задачи. Например, на рис. 5.4 показаны узлы мультипликативного правила для треугольника, полученного из двух трехточечных правил Гаусса. Плотность узлов такого правила велика в верхней части треугольника и мала в нижней. Из-за этого подобные правила с большим числом узлов применяются редко.

5.10.2. Использование программ одномерного интегрирования для вычисления двойных интегралов

До недавнего времени высокоэффективных программ для вычисления двойных интегралов не существовало, тогда как хорошие программы одномерного интегрирования уже были. Поэтому, естественно, делались попытки использовать для вычисления двойных интегралов программы одномерного интегрирования. В этом разделе мы опишем приемы, которые позволяют осуществлять такие вычисления и по-прежнему широко применяются.

Пусть в нашем распоряжении есть две программы одномерного интегрирования

```
SUBROUTINE Q1(G,A1,B1,EPS1,RES1,ERR1,KF1,IFLAG1),
SUBROUTINE Q2(F,A2,B2,EPS2,RES2,ERR2,KF2,IFLAG2).
```

Решая задачу 5.11, вы будете использовать подпрограмму Q1DA и ее копию, переименованную в Q1, но метод допускает применение любых двух надежных подпрограмм. Мы должны решить, какие функции выбрать в качестве одномерных подынтегральных функций F и G и как задать допустимые погрешности EPS1 и EPS2. Если интеграл можно представить в виде

$$I = \int_a^b \left(\int_\alpha^\beta f(x, y)\, dy \right) dx,$$

то примем выражение в скобках за функцию $g(x)$:

$$I = \int_a^b g(x)\,dx, \quad g(x) = \int_\alpha^\beta f(x, y)\,dy;$$

тогда двойной интеграл можно вычислить с помощью наших двух программ. По заданному x с помощью Q2 вычислим интеграл, обозначенный через $g(x)$. Получим

$$\mathrm{RES2} = g(x) - E_2,$$

где, как мы надеемся,

$$|E_2| < \mathrm{ERR2} < \mathrm{EPS2}.$$

Величины RES2, ERR2 и E_2 зависят от x. Теперь с помощью Q1 вычислим интеграл от g. Конечно, в действительности мы интегрируем не g, а $\mathrm{RES2} = g(x) - E_2$. На выходе программы Q1 получим

$$\mathrm{RES1} = \int_a^b (g(x) - E_2(x))\,dx - E_1,$$

где

$$|E_1| < \mathrm{ERR1} < \mathrm{EPS1}.$$

Таким образом,

$$I = \int_a^b g(x)\,dx = \mathrm{RES1} + \int_a^b E_2(x)\,dx + E_1.$$

Сумма двух последних членов не должна превосходить допустимой погрешности ε. Поскольку

$$\left|\int_a^b E_2(x)\,dx + E_1\right| < (b-a)\,\mathrm{EPS2} + \mathrm{EPS1},$$

допустим любой выбор EPS1 и EPS2, при котором выражение в правой части не превосходит ε. Впрочем, пока мы не принимали во внимание объем работы, произведенной программами Q1 и Q2. Интуитивно ясно (и может быть доказано), что если внутренний интеграл вычисляется неточно, то функция, которую должна проинтегрировать внешняя программа Q1, будет представляться ей «пилообразной». Интегрирование такой функции потребует лишних затрат, а иногда и сделает невозможным завершение работы программы Q1. Поэтому на практике рекомендуется пользоваться следующим правилом: внутренний интеграл должен вычисляться раз в десять точнее внешнего. Мы положим

$$\mathrm{EPS1} = 0.9\varepsilon, \quad \mathrm{EPS2} = \frac{\varepsilon}{10(b-a)}.$$

Оценка погрешности вычисляется по формуле

$$\mathrm{ERR1} + (b-a)\max \mathrm{ERR2}.$$

Пример 5.11. Вычисление двойного интеграла с помощью пары программ одномерного интегрирования.

Двойной интеграл по прямоугольнику

$$I = \int_0^1 \int_0^2 \exp(-x^2 y^2)\, dx\, dy$$

можно вычислить по следующей программе.

```
C Вычисление двойного интеграла с помощью двух программ
C одномерного интегрирования
C
      EXTERNAL G
      COMMON ERMAX, XX
C
      ERMAX = 0.0
      EPS   = 1.E - 4
      EPS1  = 9.0/10.0 * EPS
      A1    = 0.0
      B1    = 1.0
      CALL Q1 (G, A1, B1, EPS1, RES1, ERR1, KF1, IFLAG1)
C
      ERR   = ERR1 + (B1 - A1) * ERMAX
      WRITE (*,*) RES1, ERR, KF1, IFLAG1
      STOP
      END
C
C
      FUNCTION G(X)
      EXTERNAL F
      COMMON ERMAX, XX
C
      XX    = X
      EPS2  = 0.1 * 1.E - 4
      A2    = 0.0
      B2    = 2.0
      CALL Q2 (F, A2, B2, EPS2, RES2, ERR2, KF2, IFLAG2)
C
C Проверка IFLAG2
      IF  (IFLAG2.NE.0)
     *    WRITE (*,*) 'ERROR IN Q2, IFLAG2, KF2 =', IFLAG2, KF2
C
      ERMAX = MAX (ERMAX, ERR2)
      G = RES2
      RETURN
      END
C
```

```
C
      FUNCTION F(Y)
      COMMON ERMAX, X
C
      F = EXP (-X*X*Y*Y)
      RETURN
      END □
```

Следует обратить внимание на оператор COMMON и на оператор присваивания XX = X в подпрограмме-функции G. Они необходимы для того, чтобы в подпрограмму-функцию F, являющуюся подынтегральной по отношению к Q2, было передано значение переменной X – входного параметра подпрограммы-функции G. Использование оператора COMMON – наиболее подходящее средство, предоставляемое Фортраном для работы с *глобальными переменными*. Переменной X в FUNCTION F(Y) и переменной XX в FUNCTION G(X) соответствует одна и та же ячейка памяти. Оператор COMMON понадобился из-за того, что подпрограмма-функция F имеет только один параметр, а ей должно быть передано значение переменной X. В Фортране одна и та же переменная не может встречаться в списке параметров и в операторе COMMON, поэтому в операторе COMMON мы использовали переменную XX, которой присваивается значение переменной X.

* 5.10.3. Немультипликативные правила и автоматические программы двумерного интегрирования

Немультипликативным правилом называют квадратурное правило, полученное любым способом, отличным от последовательного применения правил одномерного интегрирования по каждой переменной. Обычно немультипликативные правила более эффективны по сравнению с обеспечивающими ту же точность мультипликативными, и это создает сильный стимул к построению таких правил. Естественно попытаться ответить на следующий вопрос: если правило имеет m узлов, то как выбрать его узлы и веса, чтобы оно оказалось точным для полиномов от двух переменных наиболее высокой степени? В случае одной переменной такой подход привел нас к квадратурам Гаусса. Однако в случае двух и большего числа переменных ситуация становится гораздо более сложной. Возможно существование нескольких различных правил с m узлами, которые точны для полиномов одной и той же степени, тогда как правил с тем же числом узлов, точных для полиномов более высокой степени, не существует. Далее, при одном и том же m ответы на поставленный вопрос могут быть совершенно разными для разных областей. Для большинства же областей наилучшие правила просто неизвестны. Возникающие здесь проблемы обсуждаются в книге [Stroud, 1972]. Наиболее подробно в ней рассматриваются треугольные области, поскольку они важны для приложений. Великолепным руководством,

содержащим рекомендации по выбору и использованию квадратурных правил, является технический отчет [Lyness, 1983].

Мы уже видели, насколько полезными оказались пары Гаусса – Кронрода в случае одномерных квадратур. Делались попытки распространить идею их построения на случай двух переменных. Из результатов, полученных к настоящему времени, упомянем результат Лори [Laurie, 1980]: исходя из правила с 7 узлами алгебраической степени точности 5, он, добавив 12 узлов (так что общее число узлов стало равным 19), построил правило, алгебраическая степень точности которого равна 8. Известно, что квадратурное правило, точное для полиномов, степень которых не превосходит 8, должно иметь по крайней мере 16 узлов; правила Лори используют на 3 узла больше, но зато позволяют получить оценку погрешности. На рис. 5.5 показано расположение семи узлов правила алгебраической степени точности 5, а также двенадцати дополнительных узлов, которые входят в правило Лори алгебраической степени точности 8.

Благодаря доступности автоматических программ одномерного интегрирования двойные интегралы в течение многих лет вычислялись при помощи алгоритмов и программ, подобных описанным в разд. 10.2. Но с появлением немультипликативных правил, позволяющих получать оценки погрешности, стали создаваться и программы двумерного интегрирования. Самые ранние из них использовали в качестве основных

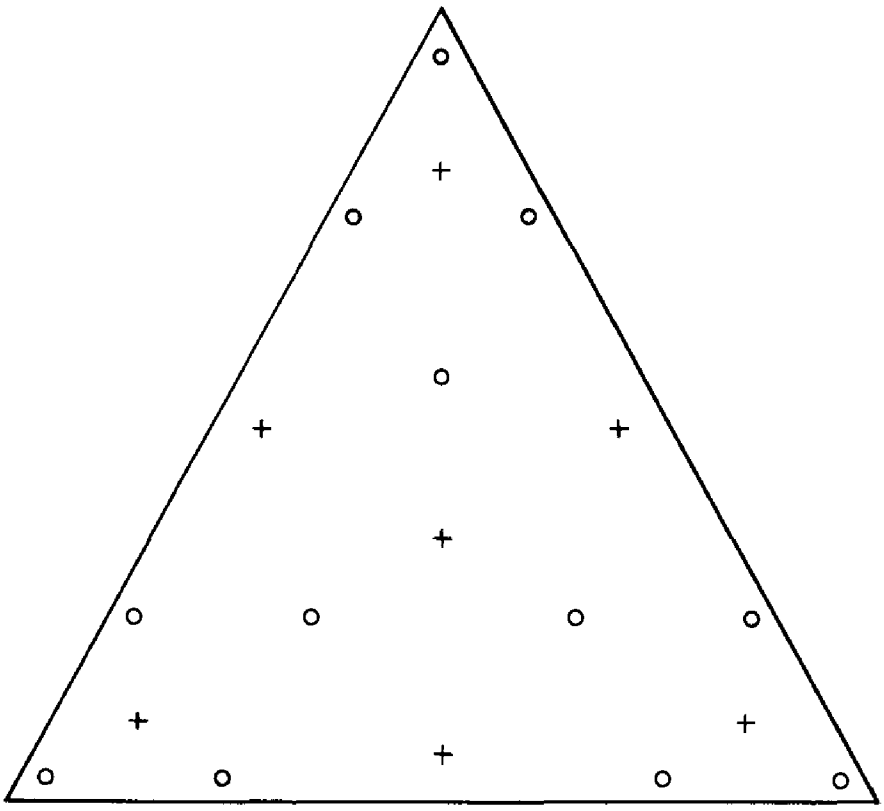


Рис. 5.5. Немультипликативное правило для треугольника:
+ 7 узлов формулы алгебраической степени точности 5;
○ 12 дополнительных узлов формулы алгебраической степени точности 8.

прямоугольные области, в более поздних предпочтение было отдано треугольным областям. Общая стратегия, применяемая в этих программах,–глобальная адаптивная стратегия. Одна из наиболее привлекательных черт глобальных алгоритмов состоит в том, что они обобщаются и на многомерный случай. Вместо отрезков при этом рассматриваются треугольники, а деление отрезка пополам заменяется разбиением треугольника на два (или четыре) подтреугольника. Соответствующие программы требуют большего объема памяти, чем программы одномерного интегрирования и программа из разд. 10.2. Однако применение их обычно более эффективно по сравнению с применением двух программ одномерного интегрирования, так как дает выигрыш в количестве вычислений значений подынтегральной функции. И наконец, такие программы способны справляться с более сложными интегралами. С типичной и хорошей программой вы можете познакомиться в статье [de Doncker, Robinson, 1984].

5.11. Методы Монте-Карло

Термин «метод Монте-Карло» был введен во время второй мировой войны фон Нейманом и Уламом в Лос-Аламосе в связи с моделированием нейтронной диффузии в расщепляемом материале. Статистикам эта идея была известна еще в 1910 году (например, У. С. Госсету, подписывавшему свои работы псевдонимом Стьюдент). Однако окончательно название утвердилось благодаря Нику Метрополису. Он сообщил, что в начале 30-х годов в Римском университете Энрико Ферми, воспользовавшись этой идеей и проведя расчеты вручную, предсказал результаты экспериментов, чем изумил своих коллег. Метод использовался также для приближенного вычисления сложных многомерных интегралов. С тех пор появилось около 3000 статей и книг, посвященных методу Монте-Карло и его обобщениям. Здесь мы приведем простой пример, основанный на ранних идеях о приближенном вычислении одномерных интегралов. Дальнейшие примеры вы встретите в § 9 гл. 10. Хорошим справочным материалом являются книга [Rubinstein, 1981] и [Haber, 1970].

Для примера рассмотрим простой вариант метода Монте-Карло, который легко реализуется при наличии датчика равномерно распределенных случайных чисел. О том, как генерировать псевдослучайные числа, мы расскажем в гл. 10. Выберем N случайных точек $0 \leqslant U_i \leqslant 1$ и осуществим переход к отрезку $[a, b]$ по формуле $u_i = a + U_i(b - a)$. Рассмотрим величину

$$\theta_N = (b - a)\frac{1}{N}\sum_{i=1}^{N} f(u_i),$$

которая представляет собой среднее арифметическое значений функции f, умноженное на $b - a$. Интуиция подсказывает, а элементарный статистический анализ подтверждает, что полученная величина аппрокси-

мирует интеграл I. Для приближения интеграла по двумерной области нужно вычислить среднее арифметическое значений подынтегральной функции в точках, случайным образом выбранных в этой области, и умножить результат на площадь области.

Насколько точным будет полученное приближение? Поскольку выбор u_i случаен, в принципе не исключено, что при конкретном испытании все u_i попадут в некий малый подотрезок отрезка $[a, b]$. Поэтому с определенностью можно утверждать лишь то, что величина θ_N заключена между максимальным и минимальным значениями f,—факт достаточно бесполезный. Однако вероятность того, что все u_i соберутся на небольшом участке, мала, особенно при больших N. Поэтому мы все же можем делать вероятностные утверждения о точности приближения θ_N: например, можно утверждать, что с определенной большой вероятностью величина $|\theta_N - I|$ будет меньше некоторой заданной величины. Действительно, с помощью теории вероятностей легко показать, что θ_N, являясь случайной величиной, имеет математическое ожидание I и среднеквадратическое отклонение

$$\sigma(\theta_N) = |b - a| N^{-1/2} \sigma(f),$$

где $\sigma(f)$—константа, зависящая от f, но не зависящая от N, значение которой можно вычислить. Далее, если предположить, как обычно, что распределение θ_N при больших N близко к нормальному, то с вероятностью, приблизительно равной 19/20,

$$|\theta_N - I| \leqslant 2 |b - a| \sigma(f) N^{-1/2} = O(N^{-1/2}).$$

В этом смысле мы и утверждаем, что погрешность стремится к нулю как $N^{-1/2}$ при $N \to \infty$.

Грубо говоря, для увеличения точности в 10 раз нужно увеличить N в 100 раз. Это не слишком впечатляет, особенно в свете некоторых примеров из данной главы. Однако замечательно то, что рассмотренный простой вариант метода Монте-Карло можно распространить на случай нескольких переменных почти для любой функции и любой области, если только объем области известен и имеется удобный способ выбора случайных точек в области. Основное свойство—убывание погрешности как $N^{-1/2}$—не зависит от размерности области и справедливо при весьма слабых предположениях о гладкости функции f. Требуется лишь существование интеграла, а функция f при этом даже не обязана быть непрерывной.

Конечно, в случае одной переменной обычно можно добиться более быстрой сходимости по сравнению с $N^{-1/2}$. Если подынтегральная функция гладкая, то достичь этого просто; при наличии же особенностей, которые подынтегральная функция имеет в изолированных точках, можно применить различные средства, позволяющие справиться и с ними. В связи с этим методы Монте-Карло в случае одной переменной почти никогда не используются. В случае двух и большего числа переменных подынтегральная функция может иметь особенности

не только в точках, но и вдоль кривых и поверхностей сложной формы; при этом избавиться от них удается крайне редко. Поэтому для интегрирования разрывных функций многих переменных подходят лишь весьма немногие методы; одним из них и является метод Монте-Карло.

Пример 5.12. Вычисление одномерного интеграла методом Монте-Карло.

С помощью простого варианта метода Монте-Карло найдем приближенное значение интеграла из примера 5.8 разд. 9.1, пользуясь при этом заменами переменной $x = -\ln t$ и $x = -2\ln t$.

Таблица 5.7

N	Оценка для $x = -\ln t$	Оценка для $x = -2\ln t$
100	0.713998	0.710285
1000	0.710525	0.734007
10000	0.712493	0.705209
100000	0.700767	0.698311
1000000	0.703014	0.702628

Отметим, что для метода Монте-Карло ни одна из замен переменной не является предпочтительной; при 1 000 000 узлов оба приближенных значения имеют 3–4 верные цифры, что согласуется с убыванием погрешности как $N^{-1/2}$. Напомним, что $I = 0.70260\ldots$. Читателя может шокировать то, что для решения задачи подынтегральную функцию потребовалось вычислить в 1 000 000 точек. Действительно, для одномерного интеграла такое число узлов чрезмерно велико. Но, с другой стороны, если бы мы вычисляли шестикратный интеграл, картина была бы приблизительно такой же. В шестимерном случае использование миллиона точек означает, что по каждой переменной их используется в среднем около десяти. При вычислении многомерных интегралов часто требуются миллионы значений подынтегральной функции. □

Стоит упомянуть два обобщения метода Монте-Карло. **(1)** Вычислители давно заметили, что общий объем работы сократится, если сосредоточить узлы в тех частях области интегрирования, где подынтегральная функция меняется наиболее быстро. Осуществить это можно автоматически, что в некотором смысле превращает метод Монте-Карло в адаптивный алгоритм. К сожалению, составить соответствующие программы значительно сложнее, чем реализовать основной метод, а простота всегда была привлекательной чертой метода Монте-Карло; поэтому упомянутая заманчивая идея находит применение не всегда. **(2)** С 1970 года стали активно использоваться «квазислучайные», или теоретико-числовые, методы вычисления многомерных интегралов. В их

основе – выбор специальных узлов, которые не являются случайными, но обладают хорошими теоретическими свойствами, что позволяет получать погрешности, убывающие не как $N^{-1/2}$ в случае метода Монте-Карло, а как N^{-1}. Подынтегральная функция должна при этом удовлетворять определенным ограничениям, более строгим по сравнению с предыдущими, но не представляющим существенного препятствия на практике. Читателей, которые хотят познакомиться с этими идеями более подробно, мы адресуем к указанным выше работам.

5.12. Историческая справка: Улам (1909 – 1984) и фон Нейман (1905 – 1957)

Джон фон Нейман и Станислав Марцин Улам, американские иммигранты из Венгрии и Польши, достигли во время второй мировой войны значительных успехов в интересующей нас области. Работая вместе над особо секретным Манхэттенским проектом в Лос-Аламосе, Нью-Мехико, они применили метод Монте-Карло для приближенного решения задач гидродинамики, которые из-за своей сложности не поддавались решению точными математическими методами, а при решении традиционными численными методами требовали слишком больших затрат времени. Исходная задача состояла в том, чтобы определить, «насколько глубоко нейтрон может проникать в различные экранирующие вещества». При этом исследователям была известна следующая информация: средняя длина пробега нейтрона в данном веществе до столкновения с атомным ядром, шансы нейтрона отразиться от ядра или быть поглощенным им и т. д. Но несмотря на то, что вероятность каждого такого отдельного события легко могла быть найдена, вывести точное уравнение, позволяющее предсказать исход длинной последовательности событий, оказалось невозможно. Чтобы получить ответ, Улам и фон Нейман предложили рассмотреть большое число нейтронов и проследить прохождение каждого из них через экран. Но для осуществления этого они решили воспользоваться не физическим экспериментом, а моделью жизни нейтрона. Все перемещения нейтрона были разбиты на дискретные шаги, такие, как столкновение, поглощение и т. д. Что именно должно произойти на каждом шагу, определялось с помощью таблицы случайных чисел. В результате модель жизни нейтрона представлялась последовательностью шагов, а все множество нейтронов описывалось миллионами таких моделей. Вычисления должны были производиться на компьютерах, совершенствование которых шло параллельно военным разработкам. Методы Монте-Карло и в настоящее время представляют собой важный инструмент для изучения процессов, происходящих в ядерных реакторах, а также многих других явлений.

И Улам, и фон Нейман, чистые математики с традиционным европейским образованием, внесли вклад в решение многих других приклад-

ных задач. Например, Улам обнаружил несовершенство проекта создания первой водородной бомбы и внес в него коррективы, способствовавшие окончательному успеху. До сих пор горячо обсуждается, кто был изобретателем цифровой вычислительной машины; относительно же того, кто первым в полной мере осознал возможности компьютера, сомнений нет. Это было значительным достижением фон Неймана. Действительно, его коллега Герман Голдстайн утверждал, что «авторитет фон Неймана в физических и общественных науках был столь велик, что, когда он предложил людям обратиться к численному моделированию,.. они поверили ему, и этим в значительной мере объясняется раннее признание компьютеров. Если бы не он, это признание шло бы значительно медленнее». Предложения фон Неймана по использованию компьютеров для предсказания погоды, анализа запасов нефти, решения задач гидродинамики послужили главным толчком к финансированию учреждений, заинтересованных в соответствующих исследованиях.

О «Джонни» говорили как о самом выдающемся и разностороннем математике двадцатого века. И, без сомнения, он был еще и самым быстрым. Его способность впитывать и осваивать гигантские объемы самой разнообразной информации была просто исключительной. Хотя для представителей его профессии быстрый ум не является редкостью, поразительная быстрота фон Неймана вошла в поговорку. Его дарование сравнивали с дарованием Гаусса; во всей математической науке 30-х годов не нашлось бы, пожалуй, ни одного раздела, с которым он не был бы по крайней мере бегло знаком; по-видимому, так же обстояло дело и с теоретической физикой. Есть достоверный рассказ о том, как фон Нейман зашел однажды к коллеге и застал того за беседой с аспирантом; обсуждались результаты расчетов, которые аспирант выполнял в течение нескольких месяцев и недавно закончил. Послушав разговор, фон Нейман подумал о задаче и, не зная, что результаты уже получены, стал объяснять, какими они должны быть. С большинством выводов аспирант согласился, но все же отметил, что в некоторых числах были небольшие ошибки. После секундного раздумья фон Нейман, который всегда был не прочь блеснуть своим талантом, подправил результаты, но он был поражен тем, что этот новый «мальчик» сумел продублировать его расчеты.

Фон Нейман, сын преуспевающего банкира, обучался математике по индивидуальной программе, поскольку его учителя поняли, что занятия по обычной гимназической программе являются для него напрасной тратой времени. В начале своей карьеры он занимался теоретическими исследованиями по логике, формальным системам и основаниям математики, что в дальнейшем, когда предметом его интереса сделались компьютеры, сослужило ему хорошую службу. В 1928 году им была заложена основа нового раздела математики – теории игр. Ключевым моментом новой дисциплины стало впервые полученное фон Нейманом доказательство того, что для любой стратегической игры из определенного класса существует по крайней мере одна оптимальная стратегия,

которая в среднем гарантирует минимальные потери. Обнародованная и приведенная в систему фон Нейманом, теория игр быстро нашла признание и применение в экономике, военной науке и многих общественных науках. С энтузиазмом поддерживал фон Нейман создание игровых программ, в частности программы игры в шахматы. Ранний ее вариант, созданный в Лос-Аламосе, пришлось упростить, чтобы он соответствовал существующему оборудованию. В нем использовалась доска уменьшенных размеров, а среди фигур отсутствовали слоны. Из-за этого игра получила название «антиклерикальной» [1]. В период с 1925 по 1940 г. фон Нейман, казалось, не переводя дыхания, делал стремительные успехи одновременно в таких науках, как логика и анализ, не говоря уже о математической физике. Многие его работы по теории операторов и сейчас остаются непревзойденными, а его результаты по спектральной теории до сих пор являются основой нерелятивистской квантовой теории. Достижения фон Неймана настолько впечатляли, что в 1933 году его пригласили в Институт перспективных исследований Принстонского университета, в котором он в то время был самым молодым постоянным сотрудником.

Его здоровье стало ухудшаться в 1955 году. Период, когда с развитием болезни знаменитая легкость его ума пошла на убыль, был для фон Неймана серьезным испытанием. Он умер от рака два года спустя.

5.13. Задачи

5.1. *Интеграл, определяющий функцию ошибок*

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2)\, dt,$$

очень просто находится с помощью численных квадратур.

(а) Напишите программу, которая, используя Q1DA, печатает таблицу значений $\operatorname{erf}(x)$ для $x = 0.0, 0.1, 0.2, \ldots, 1.9, 2.0$. Задайте $\mathrm{EPS} = 10^{-5}$. Как правило, вы не должны запрашивать значения интеграла, точность которых превосходила бы точность значений подынтегральной функции. Последние же вычисляются с точностью, в небольшое число (5 или 10) раз превышающей $\varepsilon_{\text{маш}}$. Сравните вашу таблицу с опубликованными значениями или со значениями, полученными по надежной программе, имеющейся на вашем компьютере. Интеграл нужно вычислить для различных значений верхнего предела интегрирования. Каждое обращение к Q1DA должно быть независимым от остальных. Большей эффективности можно добиться, если рассматривать данную

[1] Название «антиклерикальная» для шахматной игры, в которой отсутствуют слоны, объясняется игрой слов: английское слово bishop имеет два значения: епископ и шахматный слон. – *Прим. перев.*

задачу как задачу решения дифференциального уравнения; см. задачу 8.1 в гл. 8.

(б) Повторите вычисления из (а), используя программу QK15. Сравните точность результатов и оценки погрешности.

5.2. Используя интегральное представление

$$\pi = \int_0^1 \frac{4}{1 + x^2}\, dx,$$

найдите приближения к числу π.

(а) Используйте составное правило трапеций или составное правило Симпсона с 2, 4, 8, 16, 32, 64 и 128 элементарными отрезками. Затабулируйте также погрешность. Как уменьшается погрешность при удвоении числа элементарных отрезков? (Программу для составного правила трапеций напишите с учетом замечания (3) из § 4). В зависимости от величины $\varepsilon_{маш}$ вашего компьютера, вы можете обнаружить, что с некоторого момента удвоение числа элементарных отрезков не улучшает результата. Погрешность может даже возрасти. Как вы это объясните?

(б) Используя метод из § 4, выведите составную формулу прямоугольников. Сравните ее с составной формулой трапеций с точки зрения (**i**) «открытости», (**ii**) остатка, (**iii**) объема работы, (**iv**) замечания, аналогичного замечанию (3) из § 4. Повторите вычисления из (а) с составным правилом прямоугольников.

(в) Используя те же углы, что в (а), вычислите приближенные значения интеграла либо с помощью эрмитовой кубической квадратуры, либо с помощью PCHQA. Как уменьшается погрешность при удвоении числа элементарных отрезков? Обратите внимание на то, что при равноотстоящих узлах из значений производной нужны только d_1 и d_n. Найдите их либо непосредственно, дифференцируя подынтегральную функцию, либо приближенно способом из гл. 2.

(г) Вычислите приближенные значения интеграла, используя квадратурные формулы Гаусса с 2, 4 и 8 узлами. Разделите отрезок интегрирования пополам и примените на каждом из подотрезков 4-точечное правило. Сравните результат, полученный с помощью такого составного правила (с 8 узлами), с результатом применения 8-точечной формулы Гаусса.

(д) Вычислите приближенные значения интеграла и погрешности, используя QK15. Насколько надежна полученная оценка погрешности?

(е) Примените Q1DA при различных значениях параметра EPS. Для этой программы данный интеграл является простым. До тех пор пока EPS существенно превышает $\varepsilon_{маш}$, зависимость количества вычислений подынтегральной функции от EPS будет слабой.

5.3. Решите задачу 5.2 для интеграла

$$-\frac{4}{9} = \int_0^1 \sqrt{x} \ln x \, dx.$$

5.4. Если использовать Q1DA для интегрирования функции, тождественно равной нулю на [0, 1], то результат будет равен нулю. В каких точках будет вычисляться подынтегральная функция? Попробуйте определить это, интегрируя с помощью Q1DA функцию

```
REAL FUNCTION F(X)
REAL X
PRINT *,X
F = 0.0
RETURN
END
```

Пусть на печать выданы числа $x_1, x_2, \ldots, x_k$. Какой результат будет получен, если применить Q1DA для вычисления интеграла

$$\int_0^1 (x - x_1)^2 (x - x_2)^2 \ldots (x - x_k)^2 \, dx?$$

Объясните ваш ответ. Как Q1DA пытается обойти возникающее здесь затруднение?

5.5. Выполните следующие задания.

(а) Для интеграла

$$\int_0^1 f(x)\,dx, \quad f(x) = \begin{cases} \exp(x), & \text{если } 0 < x \leqslant r, \\ \sin x, & \text{если } r < x < 1, \end{cases}$$

где $r = 0.5$ и $r = 0.3$, используйте Q1DA при значении параметра $\mathrm{EPS} = 10^{-4}$. Объясните различия (если они имеются) в объеме работы. Если бы элемент случайности в Q1DA отсутствовал и точкой первого деления была бы середина отрезка интегрирования, облегчило бы это вычисление интеграла в каком-либо из случаев?

(б) Для задачи (а) в случае $r = 0.3$ найдите длину наименьшего элементарного отрезка из использованных Q1DA. Если бы отрезок [0, 1] был разбит на подотрезки такой длины, каким было бы общее количество вычислений $f(x)$?

5.6. Рассмотрим 11-точечное правило Ньютона–Котеса

$$\int_0^1 f(x)\,dx \approx \sum_{i=0}^{10} w_i f(i/10),$$

где w_i определяются из условия точности правила для $f(x) = 1, x, x^2, \ldots, x^{10}$.

(а) Найдите веса w_i с помощью подпрограммы SGEFS. Является ли полученное правило суммой Римана?

(б) Воспользуйтесь полученным в (а) правилом для вычисления

интегралов из задач 5.2, 5.3 и 5.5. Вы увидите, что квадратуры, в которых некоторые веса отрицательны, также могут оказаться полезными.

(в) Если у вас есть таблицы, например, такие, как [Abramowitz, Stegun, 1965], сравните значения весов, вычисленные вами, с точными значениями. Какова наибольшая погрешность? Повторите (б), используя точные значения весов. Насколько отличаются результаты от результатов из (б)?

5.7. Используя данные задачи 4.3, вычислите приближенное значение интеграла с помощью

(а) обобщенного правила трапеций;

(б) подпрограммы PCHQA. Вам потребуются значения производных d_i. Найдите их, применив подпрограмму PCHEZ.

5.8. Выберите двадцать точек вида $(x_i, \operatorname{erf}(x_i))$ с неравноотстоящими абсциссами, принадлежащими отрезку [0, 1].

а) Применив PCHEZ, постройте интерполирующий сплайн, а затем с помощью PCHQA проинтегрируйте его по отрезку $[x_1, x_n]$.

(б) Если бы у вас не было подпрограммы PCHQA, то для интегрирования сплайна из (а) можно было бы использовать Q1DA, предусмотрев вызов PCHEV при каждом обращении к фортранной подпрограмме-функции F(X). Напишите такую программу и, получив результат, сравните его с результатом из (а). Если ваш компьютер позволяет измерять время работы программы, определите затраты для каждого из подходов. К каким выводам вы придете?

5.9. Какая из следующих задач потребует большого количества вычислений подынтегральной функции или окажется непосильной для подпрограммы Q1DA при значении параметра EPS $= 10^{-4}$? Сначала попробуйте ответить без реального вычисления интегралов, а затем вычислите их и сравните ответы. Проделайте то же самое с подпрограммой QK15. Насколько надежны полученные с ее помощью оценки погрешности?

(а) $\int_0^1 \exp(x^2)\,dx$;

(б) $\int_0^1 x^x\,dx$;

(в) $\int_0^{10} \sin 50x\,dx$;

(г) $\int_0^1 \ln x\,dx$;

(д) $\int_0^5 x^{-1/2} \exp(x^2)\,dx$;

(е) $\int_1 (x-1)^{0.2} \cdot (x^2+1)^{-1}\,dx$.

5.10. Найдите приближенное значение интеграла

$$\int_0^\infty \exp(-x)\cos^2 x\,dx.$$

(а) Осуществите усечение и примените Q1DA на конечном отрезке.

(б) Сделайте замену переменной $x = -\ln t$ и вычислите получившийся интеграл с помощью Q1DA. Затем повторите вычисления при замене переменной $x = t/(1-t)$ и сравните результаты.

(в) Используйте квадратурные правила Гаусса–Лагерра с 2, 4 и 8 узлами (табл. 5.4). Сравните результаты с результатами из (а) и (б).

(г) Используйте правило трапеций для полубесконечного отрезка. Выполните вычисления при разных значениях N и h.

(д) Сделав замену переменной $x = \exp(t)$, используйте правило трапеций для бесконечного отрезка. Выполните вычисления при разных значениях N и h.

5.11. Выполните следующие задания:

(а) Вычислите

$$\iint_D \exp(x^2y^2)\,dA, \qquad D = \{(x, y):\ 0 \leqslant x \leqslant 1,\ 0 \leqslant y \leqslant 1\},$$

используя подпрограмму Q1DA и ее копию, переименованную в Q1. *Указания:* (1) последовательный вызов подпрограмм осуществляется в следующем порядке: основная программа, Q1, ..., G, Q1DA, Q1DAX,GL15T,F, поэтому Q1 не должна вызывать Q1DAX или GL15T; (2) сменив имена программ, проверьте также, что Q1 не обращается к F; (3) сперва протестируйте программу на функции, интеграл от которой известен, например $f(x, y) = 1$.

(б) Используя программу из (а), вычислите интеграл от той же функции по области D, представляющей собой четверть круга:

$$D = \{(x, y):\ 0 \leqslant x \leqslant 1,\ 0 \leqslant y \leqslant \sqrt{1-x^2}\}.$$

(в) Используя метод Монте-Карло, вычислите интегралы от $f(x, y) = 1$ и $f(x, y) = \exp(x^2y^2)$ по четверти круга. Сравните количество вычислений подынтегральной функции и затраты на программирование с аналогичными показателями в (а). *Указание:* используйте UNI из гл. 10; в частности, см. задачу 10.5.

5.12. Проверьте результаты из примера 5.10. Затем решите ту же задачу, используя Q1DA. Какой из способов оказался более трудоемким?

5.14. Прологи: QK15, Q1DA и PCHQA

```
      SUBROUTINE QK15(F,A,B,RESULT,ABSERR,RESABS,RESASC)
C*** НАЧАЛО ПРОЛОГА QK15
C*** ДАТА НАПИСАНИЯ 800101 (ГГММДД)
C*** ДАТА ПЕРЕСМОТРА 870530 (ГГММДД)
```

```
C*** КАТЕГОРИЯ NO. H2A1A2
C*** КЛЮЧЕВЫЕ СЛОВА: 15-точечные правила Гаусса - Кронрода
C*** АВТОРЫ: PIESSENS R., DE DONCKER E.
                APPLIED MATH.AND PROGR.DIV.- K. U. LEUVEN
C*** НАЗНАЧЕНИЕ: QK15  вычисляет I = интеграл для F по (A,B)
C                                   (с оценкой погрешности)
C                      и  J = интеграл для ABS(F) по (A,B)
C*** ОПИСАНИЕ
C
C    Из книги D. Kahaner, C. Moler, S. Nash
C             "Numerical Methods and Software"
C             Prentice-Hall, 1988
C
C    Версия для вещественной арифметики одинарной точности.
C
C    Входные параметры:
C      F      - REAL
C               Имя подпрограммы-функции FUNCTION F(X), вы-
C               числяющей подынтегральную функцию.
C               Фактическое имя для F должно быть указано в опера-
C               торе EXTERNAL в вызывающей программе.
C
C      A      - REAL
C               Нижний предел интегрирования.
C      B      - REAL
C               Верхний предел интегрирования
C    Выходные параметры:
C      RESULT - REAL
C               Приближение к интегралу I.
C               Вычисляется с помощью 15-ТОЧЕЧНОГО ПРАВИ-
C               ЛА КРОНРОДА (RESK), полученного оптимальным
C               добавлением абсцисс к 7-ТОЧЕЧНОМУ ПРАВИЛУ
C               ГАУССА (RESG).
C
C      ABSERR - REAL
C               Оценка модуля абсолютной погрешности.
C
C
C      RESABS - REAL
C               Приближение к интегралу J.
C      RESASC - REAL
C               Приближение к интегралу для ABS(F - I/(B - A)).
C*** ССЫЛКИ: PIESSENS R. ET AL., QUADPACK: A SUBROUTINE
C             PACKAGE FOR AUTOMATIC INTEGRATION,
C             SPRINGER, BERLIN, 1983
C    ВЫЗЫВАЕМЫЕ ПОДПРОГРАММЫ: R1MACH
```

```
C     КОНЕЦ ПРОЛОГА QK15
      SUBROUTINE Q1DA(A,B,EPS,R,E,KF,IFLAG)
C*** НАЧАЛО ПРОЛОГА Q1DA
C*** ДАТА НАПИСАНИЯ 821018 (ГГММДД)
C*** ДАТА ПЕРЕСМОТРА 870525 (ГГММДД)
C*** КАТЕГОРИЯ NO. H2A1A1
C*** КЛЮЧЕВЫЕ СЛОВА: адаптивная квадратура, автоматическая
                      квадратура
C*** АВТОР: KAHANER D.K., SCIENTIFIC COMPUTING DIVI-
            SION, NBS.
C*** НАЗНАЧЕНИЕ: Q1DA находит приближение к одномерному ин-
C                тегралу для заданной пользователем подынте-
C                гральной функции (простая в использовании
C                программа).
C*** ОПИСАНИЕ
C     Q1DA вычисляет определенный интеграл для заданной пользова-
C     телем функции одной переменной.
C
C     Из книги: D. Kahaner, C. Moler, S. Nash
C               "Numerical Methods and Software"
C               Prentice-Hall, 1988
C
C     Описание параметров:
C
C       A
C       B       (на входе) пределы интегрирования.
C       EPS     (на входе) допустимая погрешность вычисления интеграла.
C                  чтобы получить точность до двух знаков, задайте
C                  EPS = .01, до трех знаков - EPS = .001 и т. д.
C                  значение EPS должно быть положительным.
C       R       (на выходе) наиболее точное из вычисленных програм-
C                  мой Q1DA значений вашего интеграла.
C       E       (на выходе) оценка ABS (интеграл - R).
C       KF      (на выходе) мера трудоемкости: количество вычислений
C                  вашей подынтегральной функции.
C                  KF всегда не меньше 30.
C       IFLAG   (на выходе) признак ошибки. Принимает значения:
C                     0 Нормальное завершение, выполняются неравенства
C                       E < EPS и E < EPS * ABS(R).
C                     1 Нормальное завершение, выполняются неравенства
C                       E < EPS, но E > EPS * ABS(R).
C                     2 Нормальное завершение, выполняются неравенства
C                       E < EPS * ABS(R), но E > EPS.
C                     3 Нормальное завершение, но значение EPS задано
C                       слишком малым, чтобы можно было удовлетво-
C                       рить требованиям абсолютной или относительной
```

```
C                погрешности.
C
C              4 Вычисления не завершены из-за больших ошибок
C                округления. По-видимому, E и R не далеки от
C                истинных.
C              5 Вычисления не завершены из-за нехватки памяти.
C                R и E не далеки от истинных.
C              6 Вычисления не завершены из-за трудностей,
C                вызванных слишком жесткими требованиями к
C                погрешности.
C              7 Вычисления не завершены из-за того, что значение
C                EPS задано < = 0.0.
C
C          Замечание. Если IFLAG = 3, 4, 5 или 6, попробуйте
C                     вместо Q1DA воспользоваться Q1DAX.
C
C     ГДЕ ВАША ПОДЫНТЕГРАЛЬНАЯ ФУНКЦИЯ?
C
C     Вы должны написать фортранную подпрограмму-функцию с име-
C     нем F, вычисляющую подынтегральную функцию. Обычно она
C     выглядит так:
C       FUNCTION F(X)
C             F-(вычисление подынтегральной функции в точке X)
C             RETURN
C       END
C
C
C     ИЛЛЮСТРИРУЮЩАЯ ПРОГРАММА
C
C       A = 0.0
C       B = 1.0        (ЗАДАНИЕ КОНЦОВ ОТРЕЗКА ИНТЕГРИРО-
C                       ВАНИЯ [0, 1])
C       EPS = 0.001  (ЗАДАНИЕ ДОПУСТИМОЙ ПОГРЕШНОСТИ)
C       CALL Q1DA(A,B,EPS,R,E,KF,IFLAG)
C       END
C       FUNCTION F(X)
C             F = SIN(2.* X) - SQRT(X) (НАПРИМЕР)
C             RETURN
C       END
C     ВЫХОДНЫЕ ДАННЫЕ ДЛЯ ЭТОГО ПРИМЕРА
C     0.0 1.0 .001 .041406750 .69077E-07 30 0
C
C     Замечание 1.
C
C       Программа содержит небольшой элемент случайности. При
C       нескольких последовательных обращениях к ней с одними и
```

```
C     теми же входными параметрами результаты окажутся разными,
C     но, как правило, близкими друг к другу.
C
C  Замечание 2.
C
C     Программа предназначена для интегрирования по конечному
C     отрезку. Поэтому входные параметры A и B должны быть
C     вещественными числами из диапазона, который допустим на ва-
C     шем компьютере. Если требуется произвести интегрирование
C     по бесконечному отрезку, то задайте A или B или оба этих
C     параметра такими, чтобы интеграл по отрезку [A, B] почти
C     не отличался от интеграла по бесконечному отрезку. Здесь,
C     однако, необходима осторожность. Например, чтобы проинте-
C     грировать EXP(-X*X) по вещественной прямой, можно поло-
C     жить A = -20., B = 20. или присвоить этим параметрам сходные
C     значения, которые обеспечат хороший результат. Если вы
C     выберете A = -1.E10 и B = +1.E10, то вы столкнетесь с двумя
C     неприятными явлениями. Во-первых, вы обязательно получите
C     сообщение об ошибке из программы EXP, так как обращаться
C     к ней придется при слишком малых значениях аргумента. Воз-
C     можны и другие неприятности, например, произойдет исчезно-
C     вение порядка. Во-вторых, даже если арифметика будет работать
C     должным образом, Q1DA, скорее всего, выдаст неверный ре-
C     зультат, потому что по сравнению со столь большим отрезком
C     интегрирования отрезок [-20, 20], который вносит основной
C     вклад в значение интеграла, почти «бесконечно мал», и вряд ли
C     точки, в которых будет вычисляться подынтегральная функция,
C     попадут в этот отрезок.
C
C  Более гибкая программа
C
C     Q1DA представляет собой простой в использовании драйвер для
C     другой программы Q1DAX. Q1DAX обладает по сравнению с
C     Q1DA дополнительными возможностями.
C
C  Ссылки: нет
C  ВЫЗЫВАЕМЫЕ ПОДПРОГРАММЫ: Q1DAX
C  КОНЕЦ ПРОЛОГА Q1DA
      REAL FUNCTION PCHQA(N,X,F,D,A,B,IERR)
C*** НАЧАЛО ПРОЛОГА PCHQA
C*** ДАТА НАПИСАНИЯ 870829 (ГГММДД)
C*** ДАТА ПЕРЕСМОТРА 870829 (ГГММДД)
C*** КАТЕГОРИЯ NO. E3,H2A2
C*** КЛЮЧЕВЫЕ СЛОВА: простое в использовании эрмитово куби-
C                     ческое или сплайн-интегрирование, чис-
C                     ленное интегрирование, квадратура
```

```
C*** АВТОР: KAHANER D.K., (NBS)
C           SCIENTIFIC COMPUTING DIVISION
C           NATIONAL BUREAU OF STANDARDS
C           ROOM A161, TECHNOLOGY BUILDING
C           GAITHERSBURG, MARYLAND, 20899
C           (301) 975-3808
C*** НАЗНАЧЕНИЕ: PCHQA вычисляет определенный интеграл для
C                кусочно-полиномиальной эрмитовой кубической
C                функции или сплайна по произвольному отрезку
C                (простая в использовании программа).
C*** ОПИСАНИЕ
C
C    PCHQA: Кусочно-полиномиальное эрмитово кубическое или
C           сплайн-интегрирование, произвольные пределы интегри-
C           рования, простота в использовании.
C
C    Из книги: D. Kahaner, C. Moler, S. Nash
C              "Numerical Methods and Software"
C              Prentice-Hall, 1988
C
C    PCHQA вычисляет по отрезку [A, B] определенный интеграл для
C    кусочно-полиномиальной эрмитовой кубической функции или
C    сплайна, заданных параметрами N, X, F, D. Представляет собой
C    простой в использовании драйвер для программы PCHIA
C    Ф.Н. Фритча, описание которой приведено в указанной ниже
C    работе (2). Программа обладает также другими возможностями.
C
C    Обращение к подпрограмме:
C
C      VALUE = PCHQA (N, X, F, D, A, B, IERR)
C
C      INTEGER N, IERR
C      REAL X(N), F(N), D(N), A, B
C
C    Описание параметров:
C
C      VALUE - (на выходе) ЗНАЧЕНИЕ интеграла.
C
C      N     - (на входе) число заданных точек. (Сообщение об
C              ошибке, если N.LT.2.)
C
C      X     - (на входе) вещественный массив значений независимой
C              переменной. Элементы массива X должны быть
C              заданы в строго возрастающем порядке:
C              X(I - 1) .LT. X(I), I = 2(1)N.
C              (Сообщение об ошибке в противном случае.)
```

```
C
C       F       -(на входе) вещественный массив значений функции.
C               Значение F(I) соответствует X(I).
C
C       D       -(на входе) вещественный массив значений производной.
C               Значение D(I) соответствует X(I).
C
C       A, B    -(на входе) пределы интегрирования.
C               Замечание. Здесь не требуется, чтобы отрезок [A, B]
C                          принадлежал отрезку [X(1), X(N)]. Однако
C                          если это не так, к результату нужно отно-
C                          ситься с большой долей недоверия.
C
C       IERR    -(на выходе) признак ошибки.
C                 Нормальный выход:
C                   IERR = 0 (нет ошибок).
C                 Предупреждения:
C                   IERR = 1, если A не принадлежит отрезку
C                             [X(1), X(N)].
C                   IERR = 2, если B не принадлежит отрезку
C                             [X(1), X(N)].
C                   IERR = 3, если выполняются оба вышеуказанных
C                             условия. (Отсюда следует, что либо
C                             отрезок [A, B] содержит отрезок
C                             [X(1), X(N)], либо эти отрезки не пере-
C                             секаются.)
C                 Устранимые ошибки:
C                   IERR = -1, если N.LT.2.
C                   IERR = -3, если элементы массива X заданы не
C                              в строго возрастающем порядке. (В
C                              каждом из этих случаев интеграл не
C                              вычисляется.)
C                 Замечание. Проверка параметров осуществляется в
C                            указанном выше порядке; при обнаруже-
C                            нии ошибки остальные параметры не
C                            проверяются.
C
C*** ССЫЛКИ: 1. F. N. FRITSCH, R. E. CARLSON, MONOTONE
C              PIECEWISE CUBIC INTERPOLATION, SIAM.
C              J. NUMER. ANAL. 17, 2 (APRIL 1980), 238-246
C            2. F. N. FRITSCH, PIECEWISE CUBIC HERMITE
C              INTERPOLATION PACKAGE, FINAL SPECI-
C              FICATIONS, LAWRENCE LIVERMORE NA-
C              TIONAL LABORATORY, COMPUTER DOCU-
C              MENTATION UCID-30194, AUGUST 1982
C    ВЫЗЫВАЕМЫЕ ПОДПРОГРАММЫ: PCHIA
C    КОНЕЦ ПРОЛОГА PCHQA
```

Глава 6

Аппроксимация данных методом наименьших квадратов

6.1. Введение

Рассмотрим следующий эксперимент: воду прогоняют сквозь контейнер, в который добавлено некоторое количество краски. Через каждые несколько секунд измеряется концентрация краски в воде, вытекающей из контейнера. Ожидается, что концентрация краски будет линейно уменьшаться со временем. Результаты измерений показаны на рис. 6.1.

Заметим, что точки данных не лежат на прямой линии. Это не так уж неожиданно. Измерительные приборы могут быть не совсем точными, может оказаться невозможным точно интерпретировать измерения, а смешивание может происходить не совсем так, как предсказано. Чтобы определить скорость, с которой убывает концентрация, экспериментатору следовало бы аппроксимировать данные прямой линией, которая в некотором смысле «наилучшим образом» аппроксимирует данные. На рис. 6.1 вычерчена одна из таких аппроксимаций.

Ситуации такого типа встречаются весьма часто. Одна из причин, по которой требуется знать скорость смешивания, заключается в том, что

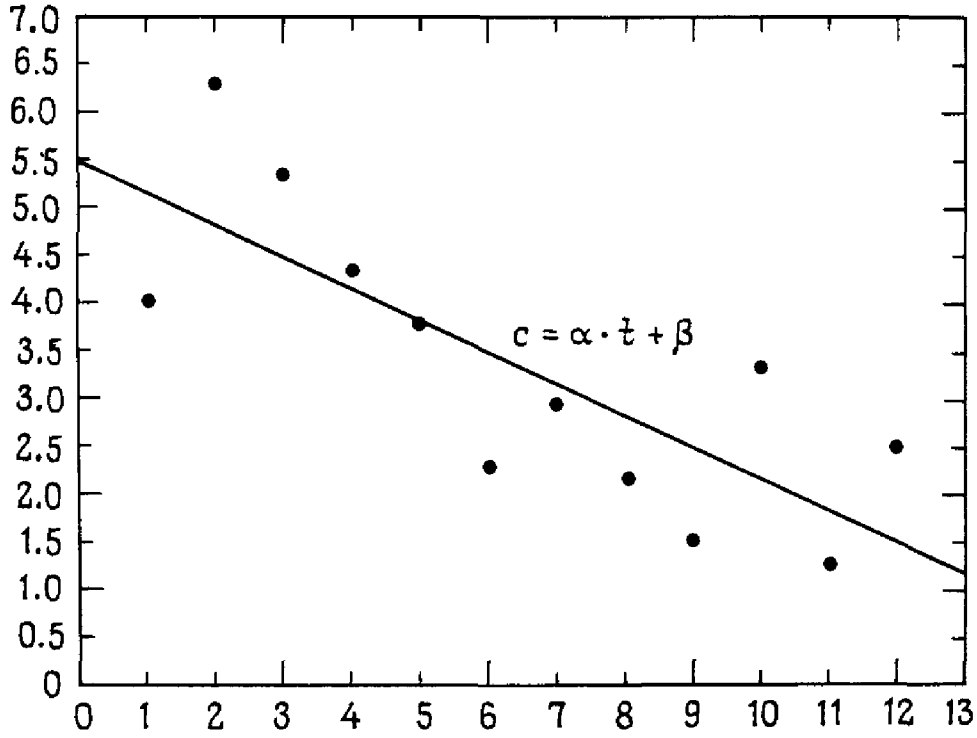


Рис. 6.1. Эксперимент с изменением концентрации.

мы хотим уметь предсказывать, как будут протекать другие эксперименты. При других обстоятельствах нам может потребоваться смоделировать инфляцию в экономике, распространение эпидемии или рост населения страны. Когда данные «зашумлены», т. е. полны случайных ошибок, тогда аппроксимация данных простой функцией позволяет нам изучать тренды (тенденции изменения) в данных; это называют *сглаживанием*.

Возможны две качественно различные причины, по которым требуется найти линию, аппроксимирующую данные.

(1) Скорость смешивания α необходима, например, чтобы определить, не перегружено ли оборудование для инжекции краски.

(2) Аппроксимирующая линия необходима, чтобы предсказать концентрацию краски в моменты времени, для которых не проводилось измерений. В этом случае нас интересуют не столько значения α и β, сколько значения приближающей функции.

В иные времена такая работа выполнялась предсказателями будущего и волшебниками, которые испрашивали мудрости у хрустальных шаров и карт Таро. Древние задавали вопросы дельфийскому оракулу, как описано в пьесе Шекспира «Зимняя сказка». Сейчас приближение данных утратило романтику, но это отчасти компенсируется улучшением научного обоснования.

Прежде чем идти дальше, сформулируем задачу приближения данных более четко. Предположим, что нам известны данные (t_i, b_i), $i = 1, \ldots, m$, представляющие некоторую лежащую в их основе функцию $b(t)$, для которой $b_i = b(t_i)$. В описанном выше эксперименте величины t_i представляют собой моменты времени, когда были сделаны измерения, а b_i – концентрацию краски. Мы предполагаем, что задана некоторая модель этих данных, т. е.

$$\text{наблюдение} = \text{модель} + \text{ошибка}.$$

Более определенно, мы предполагаем, что модель имеет вид

$$b_i \approx x_1 \phi_1(t_i) + x_2 \phi_2(t_i) + \ldots + x_n \phi_n(t_i),$$

где $\phi_j(t)$ – заданные *модельные функции*. В нашем примере, где модель является прямой линией, мы предполагаем, что

$$b_i \approx x_1 + x_2 t_i,$$

где $\phi_1(t) = 1$ и $\phi_2(t) = t$. Коэффициенты x_j называются *параметрами* модели, именно их нужно определить. Если бы модель была точной и не было ошибок измерения, мы могли бы заменить знак $\approx$ на $=$, однако это бывает редко.

Вышеприведенная модель называется *линейной моделью*, потому что она представляет собой линейную комбинацию модельных функций $\phi_i(t)$. Это не означает, что модельные функции являются линейными. Даже если бы наши данные должны были аппроксимироваться квадратичным полиномом, так что $b(t) \approx x_1 + x_2 t + x_3 t^2$, и тогда модель все же была бы линейной, хотя одна из модельных функций $\phi_3(t) =$

$= t^2$ – нелинейная функция параметра t. Решающее обстоятельство заключается в том, что модель является линейной функцией параметров x. Нелинейные модели также полезны, и они обсуждаются в гл. 9. Распространенный пример – экспоненциальная модель типа

$$b(t) \approx x_1 e^{x_2 t}.$$

Здесь модель является нелинейной функцией параметра x_2.

Большинство моделей, обсуждавшихся до сих пор, имели полиномиальные модельные функции, в которых данные аппроксимировались полиномом от переменной t. В качестве модельных могут использоваться любые функции. В случае кубической эрмитовой интерполяции модельные функции были бы кусочно-кубическими функциями $c_i(t)$ и $\bar{c}_i(t)$, описанными в § 10 гл. 4. Модели могут содержать более одной независимой переменной. Это возможно, например, при анализе данных переписи населения, где независимые переменные могут включать возраст человека, доход и размер семьи. В этом случае модельные функции были бы такими: $\phi_1(t_i)$ = возраст (i), $\phi_2(t_i)$ = доход (i), а $\phi_3(t_i)$ = размер семьи (i).

Мы можем поставить задачу приближения данных более четко, используя матрично-векторные обозначения. Определим матрицу A размера $m \times n$ как

$$a_{ij} = (A)_{ij} = \phi_j(t_i),$$

и пусть b и x – векторы наблюдений и параметров соответственно. Тогда обсуждавшиеся выше условия можно записать в виде

$$b \approx Ax \text{ или } b - Ax \approx 0;$$

вектор $b - Ax$ называют вектором *невязок*.

Пример 6.1. Формирование матрицы данных.

Предположим, что модель является квадратичным полиномом $b(t) \approx x_1 + x_2 t + x_3 t^2$ и что имеются данные для четырех значений t: $t_1 = 1$, $t_2 = 2$, $t_3 = 3$, $t_4 = 4$. Модельными функциями являются $\phi_1(t) = 1$, $\phi_2(t) = t$ и $\phi_3(t) = t^2$.

Тогда матрица коэффициентов имеет вид

$$A = \begin{bmatrix} 1 & t_1 & t_1^2 \\ 1 & t_2 & t_2^2 \\ 1 & t_3 & t_3^2 \\ 1 & t_4 & t_4^2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 4 & 16 \end{bmatrix}. \quad \square$$

Параметры выбираются так, чтобы сделать невязки $b - Ax$ как можно меньше. Обычный подход, который применяется и здесь, заключается в решении задачи

$$\min_x \sum_{j=1}^{m} [(b - Ax)_j]^2.$$

Для модели из примера 6.1 она принимает вид

$$\min_{x_1,x_2,x_3} \sum_{j=1}^{4} [b_j - (x_1 + x_2 t_j + x_3 t_j^2)]^2.$$

Поскольку мы *минимизируем сумму квадратов*, этот способ называется *аппроксимацией данных методом наименьших квадратов*. Используя нормы векторов (см. разд. 2.1 гл. 3), можно записать задачу в эквивалентной форме:

$$\min_x \|b - Ax\|_2^2 = (b - Ax)^T (b - Ax);$$

здесь используется евклидова 2-норма.

Если число данных и число модельных функций равны, матрица будет квадратной. Если она к тому же невырожденная, то решение задачи наименьших квадратов представляет собой интерполянт, т.е. невязка равна нулю. Таким образом, мы видим, что сформулированная как матричная задача, аппроксимация методом наименьших квадратов включает в себя задачу решения системы линейных уравнений $Ax = b$ с квадратной невырожденной матрицей A в качестве частного случая. Методы и программы, описанные в этой главе, можно использовать и для решения линейных уравнений, однако они требуют примерно вдвое больше арифметических операций, чем гауссово исключение. Однако методы наименьших квадратов в машинной арифметике дают несколько более точные решения. Вдобавок они устраняют необходимость выбора главного элемента, делая алгоритмы более пригодными для многих параллельных и векторных компьютеров. На параллельном компьютере со многими процессорами выбор ведущих элементов может потребовать больше времени, чем арифметика, и, следовательно, замедлить алгоритм.

Задачу наименьших квадратов можно графически интерпретировать как минимизацию вертикальных расстояний от точек данных до модельной кривой. Эта идея основана на предположении, что все ошибки в аппроксимации соответствуют ошибкам в наблюдениях b_i. Если имеются также ошибки в независимых переменных t_i, то может оказаться более уместным минимизировать евклидово расстояние от данных до модели. Это иллюстрирует рис. 6.2. Такой подход может быть особенно полезным, если график модельной функции крутой, как, скажем, вблизи особенности в нелинейной модели. Рисунок это показывает. В подобном случае крутизна модельной функции сделает ошибку наименьших квадратов большой, даже если точка данных близка к графику и приближение визуально удовлетворительно. Минимизация евклидова расстояния от модели называется *обобщенным методом наименьших квадратов* или *методом ортогональных проекций*. Этот метод выходит за рамки данной книги, но его подробный разбор можно найти в книге [Golub, Van Loan, 1983].

Последнее замечание касается обозначений. В этой главе мы обсуж-

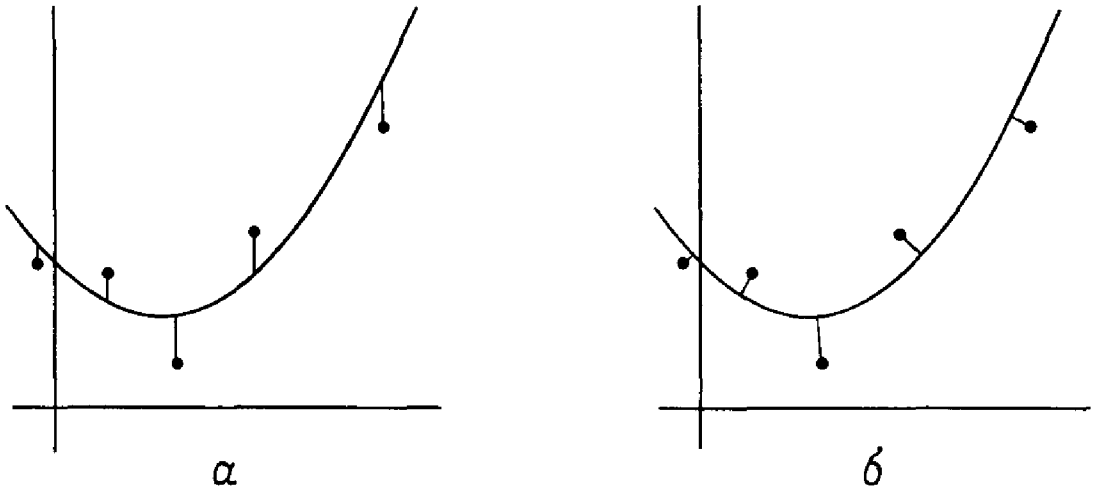


Рис. 6.2. Модели с использованием вертикальных расстояний (слева) и евклидовых расстояний (справа).

даем задачу наименьших квадратов в форме

$$\min_x \|Ax - b\|_2,$$

в то время как статистики используют запись

$$\min_\beta \|X\beta - y\|_2.$$

Обе задачи совершенно одинаковы, изменились только имена: $A \to X$, $x \to \beta$, $b \to y$. Хотя это чисто косметическое изменение, оно может вызвать путаницу при изучении аппроксимации данных. В этой книге нас прежде всего интересует вычисление параметров модели, а не статистические свойства решения, поэтому мы используем обозначение $\|Ax - b\|_2$, принятое у специалистов по численному анализу. Это обозначение также помогает подчеркнуть связь между приближением данных и решением линейных уравнений $Ax = b$.

6.1.1. Аппроксимация данных методом наименьших квадратов с весами

Во многих приложениях не все точки данных одинаково важны. Часто это происходит потому, что некоторые данные известны более точно, чем другие. Например, в астрономической модели Галактики (Млечного Пути) легче получить точные измерения от нашего Солнца, чем от звезд, удаленных на много световых лет.

Эта информация может быть включена в задачу наименьших квадратов путем взвешивания точек данных. Вместо того чтобы решать задачу

$$\min_x \sum_{j=1}^{m} [(b - Ax)_j]^2$$

относительно параметров x, мы будем решать задачу

$$\min_x \sum_{j=1}^{m} [w_j(b - Ax)_j]^2,$$

где w_j – вес, отражающий важность точки данных. Чем важнее точка, тем больше ее вес. Если ошибка в j-й точке данных приблизительно равна e_j, то выбирают $w_j = 1/e_j$. Таким образом, чем меньше ошибка, тем больше вес. Иногда все данные имеют одно и то же число верных значащих цифр, т. е. одну и ту же относительную ошибку. В этом случае хорошим выбором w_j будет $w_j = 1/b_j$, если b_j не равно нулю. Взвешивание может улучшить статистические свойства решения (см. § 2).

Любую программу для решения невзвешенной задачи наименьших квадратов можно использовать для решения взвешенной задачи после шкалирования вектора наблюдений b и матрицы коэффициентов A. Мы умножаем j-ю строку A, так же как j-е наблюдение b_j, на w_j, а затем решаем задачу

$$\min_x \|b_w - A_w x\|_2^2.$$

Коэффициенты в новой задаче определяются как

$$(b_w)_j = w_j b_j, \qquad (A_w)_{jk} = w_j A_{jk}.$$

Конечно, полученные для этой модели параметры будут меняться, когда меняются веса.

Пример 6.2. Аппроксимация данных методом наименьших квадратов с весами.

Рассмотрим модель из примера 6.1 с наблюдениями $b_1 = 2$, $b_2 = 7$, $b_3 = 9$, $b_4 = 6$. Предположим, что ошибки данных оцениваются как $e_1 = 1/4$, $e_2 = 1$, $e_3 = 1/2$, $e_4 = 1/3$. Если мы используем веса $w_j = 1/e_j$, то взвешенная задача наименьших квадратов

$$\min_x \|b_w - A_w x\|_2^2$$

имеет коэффициенты

$$A_w = \begin{bmatrix} 4 & 4 & 4 \\ 1 & 2 & 4 \\ 2 & 6 & 18 \\ 3 & 12 & 48 \end{bmatrix} \quad \text{и} \quad b_w = \begin{bmatrix} 8 \\ 7 \\ 18 \\ 18 \end{bmatrix}.$$

Веса равны $w_1 = 4$, $w_2 = 1$, $w_3 = 2$, $w_4 = 3$. □

* 6.1.2. Аппроксимация данных с другими нормами

Аппроксимация данных методом наименьших квадратов основана на компромиссе: модельная кривая не может пройти через все точки данных, и тогда 2-норма используется в качестве посредника при выборе

параметров. До того как был разработан метод наименьших квадратов, использовались различные способы аппроксимации. Многие из них соответствуют использованию различных норм в качестве посредника.

Вот два распространенных способа выбора норм:

$$\min_x \sum_{j=1}^{m} |(b - Ax)_j| \text{ (1-норма);}$$

$$\min_x \max_j |(b - Ax)_j| \text{ (∞-норма).}$$

Оптимальные значения параметров для этих норм будут отличаться от значений для 2-нормы. Кроме того, вычисления с этими нормами более трудны (решения можно получить с помощью линейного программирования). Тем не менее они имеют полезные свойства. Например, 1-норма менее чувствительна к присутствию «выбросов» (ложных точек данных). По этой причине их продолжают использовать для приближения данных в некоторых приложениях. Применение различных норм иллюстрирует простой пример 6.3.

Пример 6.3. Оценивание с различными нормами.

Предположим, что заданы наблюдения b_i и мы хотим аппроксимировать их константой x, т. е. $b_i = x +$ ошибка$_i$. Используются три способа оценки:

1-норма: медиана наблюдений b_i;
2-норма: среднее арифметическое наблюдений b_i;
∞-норма: $\frac{1}{2}(\min_i b_i + \max_i b_i)$.

Рассмотрим множество данных $\{b_i\} = \{1, 2, 5, 9, 12\}$. Три способа оценки дают следующие результаты:

1-норма: 5 (среднее число во множестве);
2-норма: 5.8 $(1 + 2 + 5 + 9 + 12)/5$;
∞-норма: $6.5 = \frac{1}{2}(1 + 12)$.

Предположим, что при сборе данных была сделана ошибка: решили, что пятый член равен 112, а не 12. Тогда будут получены такие оценки:

1-норма: 5 (среднее число во множестве то же самое);
2-норма: $25.8 = (1 + 2 + 5 + 9 + 112)/5$;
∞-норма: $56.5 = \frac{1}{2}(1 + 112)$.

Заметим, что на оценку, полученную с помощью 1-нормы, ошибка не подействовала. □

Метод наименьших квадратов (2-норма) – это наиболее часто используемый метод приближения данных. Вычислительно он наиболее прост. Он допускает также важные статистические интерпретации (во многих

случаях он дает оценку максимального правдоподобия для параметров). Подробности см. в книге [Draper, Smith, 1981]. По указанным причинам мы рассматриваем здесь именно этот подход.

6.2. Исследование данных

Анализ данных состоит из двух этапов: определения модели и вычисления параметров. В этой книге мы рассматриваем главным образом второй этап и предполагаем, что модель задана. Часто это оправданно. В описанном выше эксперименте со смешиванием поведение концентрации краски можно предсказать теоретически, и только скорость смешивания нужно определить экспериментально.

Однако во многих примерах, особенно в общественных науках, данные используются, чтобы помочь определить модель. Например, социолог может собрать данные о жителях города, такие, как доход, образовательный уровень, возраст, пол, вес, раса и род занятий. Если бы социолог хотел изучить доходы населения, можно было бы использовать следующую модель:

$$\text{доход}_i \approx x_1 \cdot \text{образование}_i + x_2 \cdot \text{возраст}_i + x_3 \cdot \text{пол}_i + x_4 \cdot \text{вес}_i + x_5 \cdot \text{раса}_i + x_6 \cdot \text{род занятий}_i.$$

Здесь все данные были использованы для формирования модели. Однако не ясно, будет ли вес человека как-то связан с его доходом, а присутствие этого члена в модели может повлиять на параметры x_j и, следовательно, исказить выводы, полученные из данных. Социолог предпочел бы включить в модель только те члены, которые окажутся значимыми по результатам *исследования данных*.

При исследовании данных возникают следующие вопросы. (1) Адекватно ли модель объясняет данные? (2) Являются ли какие-либо члены в модели избыточными (и, следовательно, их можно проигнорировать)? (3) Насколько точны параметры? (4) Насколько точны предсказания, сделанные на основе модели?

Ответы на эти вопросы основаны на статистических критериях, некоторые из которых обсуждаются ниже. Однако они опираются на важное предположение относительно данных: ошибки в модели должны быть независимы и нормально распределены. Если это предположение не выполнено, заключения, выведенные из данных, могут быть ненадежными.

Исследование данных может оказаться нелегким делом, занимающим много времени; иногда может потребоваться помощь опытного статистика. Многие проблемы можно обнаружить при рассмотрении графика невязок.[1)] Такая картина дана на рис. 6.3 для эксперимента с изменением концентрации, описанного выше.

[1)] В математической статистике чаще говорят о рассмотрении регрессионных остатков, а не невязок.– *Прим. перев.*

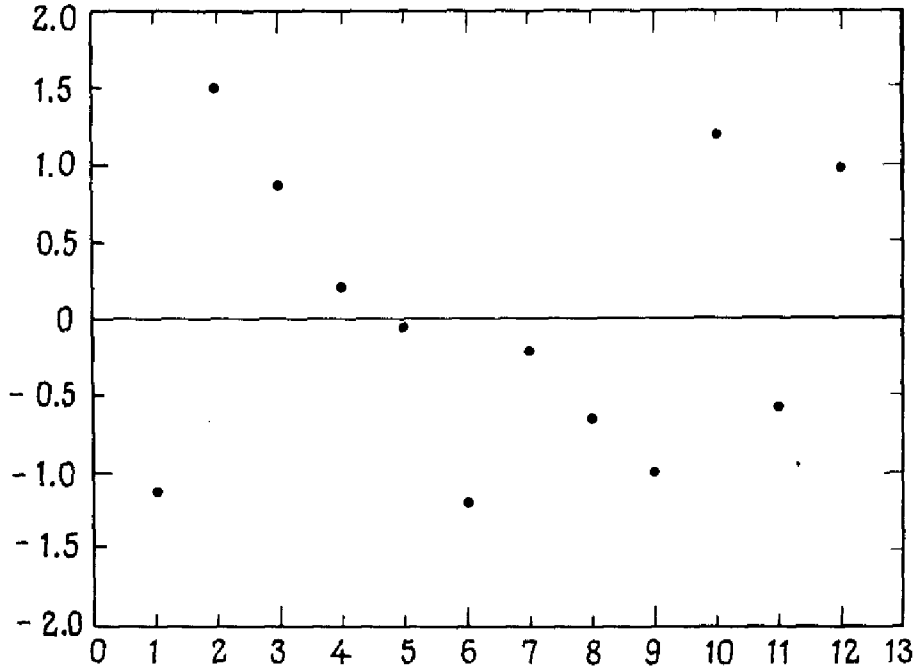


Рис. 6.3. Эксперимент с изменением концентрации: график невязок.

Важно, чтобы график невязок не имел каких-либо видимых закономерностей. Значения невязок должны быть разбросанными; они не должны «слипаться в комки». Размах невязок не должен изменяться в пределах графика. Говоря нестрого, им следует выглядеть «случайно». Некоторые примеры подозрительных графиков невязок показаны на рис. 6.4.

Дальнейшую информацию можно получить в результате шкалирования невязок с использованием стандартного отклонения s, меры разброса [1] их значений. *Стандартное отклонение* невязок определяется по формуле

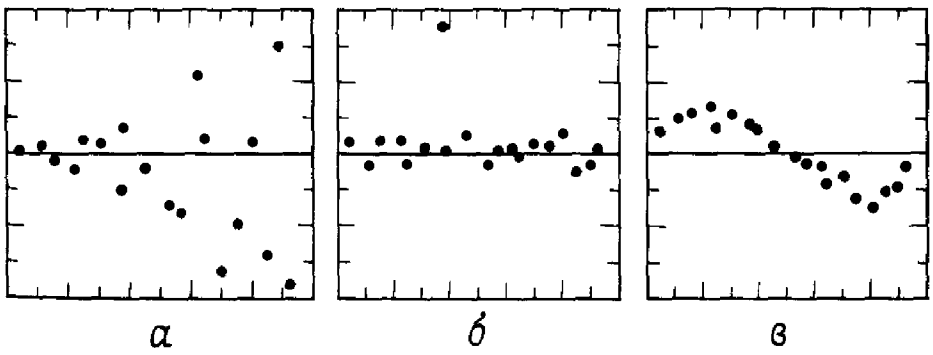


Рис. 6.4. Подозрительные графики невязок: рост дисперсии (слева), выброс (в центре), нелинейность (справа).

[1] Как видно из дальнейшего изложения, речь идет не о разбросе невязок, а об оценке стандартного отклонения ошибок в модели в предположении, что дисперсии их одинаковы.– *Прим. перев.*

$$s = \frac{\|b - Ax^*\|_2}{\sqrt{m-n}},$$

где x^* – решение задачи наименьших квадратов; шкалированные невязки равны

$$(b - Ax^*)/s.$$

Примерно 95% шкалированных невязок должно лежать в интервале $[-2, 2]$. Если какое-то значение находится вне этого интервала, такую особую точку данных следует проверить более тщательно, поскольку это может указывать на ошибку в данных или неадекватность модели. Точки с большими невязками могут оказывать сильное влияние на значение параметров, поэтому их следует обрабатывать с осторожностью. Однако важно отдавать себе отчет, что точки, оказывающие большое влияние, не обязательно имеют большие невязки. В примере с изменением концентрации $s = 0.993$, так что график шкалированных невязок выглядит практически так же, как график невязок.

Если невязки не удовлетворительны, это может говорить о проблемах с данными, с моделью или и с тем, и с другим. Есть много способов решить эти проблемы. Исчерпывающее обсуждение см. в книгах [Chatterjee, Price, 1977] и [Belsley, Kuh, Welsch, 1981].

Пакеты статистических программ часто обеспечивают возможность получения графиков невязок, а также производят дополнительные вычисления, помогающие определить, подходит ли метод наименьших квадратов для имеющихся данных. Некоторые из этих вычислений обсуждаются ниже. Многие из них зависят от сделанных выше предложений о невязках, и при нарушении этих предположений результаты могут оказаться неточными.

1. Стандартное отклонение величины b – это мера изменчивости в наблюдениях b. Чем больше значение стандартного отклонения, тем больше разброс значений b. Если наблюдения распределены нормально, то примерно 95% наблюдений лежит в пределах двух стандартных отклонений от среднего значения.
2. Стандартные ошибки в x, по аналогии со стандартным отклонением, оценивают ошибки в параметрах x. Чем больше значение, тем больше возможная ошибка в параметре.
3. Доверительные интервалы – это вероятностные границы для ошибок в параметрах. Доверительные интервалы определяют исходя из стандартных ошибок в параметрах и желаемой вероятности того, что границы правильны. В сделанных выше предположениях, истинное значение параметра будет лежать внутри интервала, скажем, с вероятностью 90 или 95%.
4. Величина R^2 указывает, какая доля изменчивости в данных объясняется моделью, и лежит в интервале $[0, 1]$. Если R^2 близко к 1, то модель адекватна. Меньшее значение может указывать на то, что есть величины, не включенные в модель. Вычисление R^2 имеет

смысл только в том случае, если модель содержит постоянный член.

5. Нормальный вероятностный график невязок – это график невязок относительно нормального распределения. Он помогает определить, подходит ли распределение невязок для приближения данных методом наименьших квадратов. В идеале этот график должен был бы быть прямой линией $y = x$. Фактические результаты не дадут этой прямой линии, но не должны и слишком далеко отклоняться от нее.
6. Статистика Дарбина–Уотсона может указать на вырождение данных (см. § 7). Это число находится в диапазоне [0, 4]. Если $d = 2$, свидетельств вырождения нет. Чем дальше d от 2, тем больше это свидетельствует о вырождении.
7. Ковариационная матрица Y – это мера изменчивости и взаимозависимости параметров x. Диагональный элемент $(Y)_{ii}$ представляет собой *дисперсию* i-го параметра x_i; дисперсия определяется как квадрат стандартного отклонения. Внедиагональный элемент $(Y)_{ij}$ представляет собой *ковариацию* параметров x_i и x_j. Если два параметра полностью независимы, их ковариация равна нулю. Если они связаны (например, если один увеличивается всякий раз, когда другой уменьшается), ковариация будет отлична от нуля. В идеале параметры независимы, а внедиагональные элементы малы. Вычисление ковариационной матрицы – относительно дорогостоящая процедура, поскольку оно включает в себя обращение матрицы $(A^T A)$, где A – матрица коэффициентов в задаче аппроксимации данных.

Таблица 6.1 Эксперимент с изменением концентрации

t	b
1.0	4.0225
2.0	6.3095
3.0	5.3522
4.0	4.3553
5.0	3.7861
6.0	2.2947
7.0	2.9492
8.0	2.1732
9.0	1.4921
10.0	3.3424
11.0	1.2596
12.0	2.4732

В примере 6.4 эти вычисления проводятся для эксперимента с изменением концентрации.

Пример 6.4. Статистический анализ эксперимента с изменением концентрации.

Экспериментальные данные приведены в табл. 6.1.

Мы используем линейную модель, описанную в § 1, с модельными функциями $\phi_1(t) = 1$ и $\phi_2(t) = t$. Матрица коэффициентов A имеет элементы $(A)_{ij} = \phi_j(t_i)$ и, таким образом,

$$A = \begin{pmatrix} 1 & 1.0 \\ 1 & 2.0 \\ 1 & 3.0 \\ 1 & 4.0 \\ 1 & 5.0 \\ 1 & 6.0 \\ 1 & 7.0 \\ 1 & 8.0 \\ 1 & 9.0 \\ 1 & 10.0 \\ 1 & 11.0 \\ 1 & 12.0 \end{pmatrix}$$

С этой моделью мы получаем следующие значения перечисленных выше статистических параметров:

стандартное отклонение $b = 1.527$,
стандартное отклонение остатков $b - Ax^* = 0.993$,
параметр $x_1 = 5.478$,
параметр $x_2 = -0.332$,
стандартная ошибка для $x_1 = 0.611$,
стандартная ошибка для $x_2 = 0.083$,
95%-ный доверительный интервал для $x_1 = [4.116, 6.840]$,
95%-ный доверительный интервал для $x_2 = [-0.517, -0.147]$,
$R^2 = 0.616$,
статистика Дарбина–Уотсона = 2.090,
ковариационная матрица $= \begin{pmatrix} 0.3735 & -0.0448 \\ -0.0448 & 0.0069 \end{pmatrix}$.

Результаты вполне удовлетворительны. Ковариация двух параметров (-0.0448) мала, как нам и хотелось. Статистика Дарбина–Уотсона почти равна идеальному значению 2. Стандартные ошибки параметров не слишком велики по отношению к их оценочным значениям. Статистика R^2 указывает, что модель прямой линии объясняет около 2/3 изменчивости в данных. Мы можем заключить, что метод наименьших квадратов в данном случае подходит. □

6.3. Нормальные уравнения

Существует много разных алгоритмов для вычисления множества коэффициентов, минимизирующего сумму квадратов. Одна из возможностей – использовать дифференциальное исчисление. Запишем задачу в векторной форме

$$\begin{aligned} r^2 &= \|b - Ax\|_2^2 = (b - Ax)^T (b - Ax) = \\ &= b^T b - b^T Ax - x^T A^T b + x^T A^T Ax = \\ &= b^T b - 2b^T Ax + x^T A^T Ax. \end{aligned}$$

Наша цель – минимизировать эту функцию по x. Для того чтобы в точке решения достигался минимум, первые производные по x в этой точке должны равняться нулю. Производные данной функции составляют

$$-2A^T b + 2A^T Ax,$$

и поэтому решение должно удовлетворять системе линейных уравнений

$$(A^T A)x = (A^T b).$$

Эти уравнения называются *нормальными уравнениями* [1].

Пример 6.5. Использование нормальных уравнений.

В качестве примера рассмотрим приближение данных (1, 1), (2, 1.5), (3, 0.75) и (4, 1.25) прямой линией. Модель имеет вид $b_i \approx x_1 + x_2 t_i$ с модельными функциями $\phi_1(t) = 1$ и $\phi_2(t) = t$. Таким образом,

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{pmatrix}, \quad b = \begin{pmatrix} 1.00 \\ 1.50 \\ 0.75 \\ 1.25 \end{pmatrix}.$$

Нормальные уравнения имеют вид

$$A^T Ax = \begin{pmatrix} 4 & 10 \\ 10 & 30 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = A^T b = \begin{pmatrix} 4.50 \\ 11.25 \end{pmatrix}.$$

Их решение дает $x = (1.125, 0)^T$, так что «наилучшая» линия, аппроксимирующая данные, определяется формулой $b(t) = 1.125$. □

Нормальные уравнения представляют собой систему линейных уравнений, поэтому их можно решить методом гауссова исключения, если матрица коэффициентов $A^T A$ обратима. Можно показать, что решения нормальных уравнений совпадают с решениями задачи наименьших

[1] Нормальные уравнения можно вывести и без матрично-векторных обозначений, если читатель к ним не привык. Функция наименьших квадратов имеет вид $\Sigma_{i=1}^m [b_i - \Sigma_{j=1}^n (A)_{ij} x_j]^2$. Первая производная этой функции по x_k равна $2\Sigma_{i=1}^m (-A)_{ik} [b_i - \Sigma_{j=1}^n (A)_{ij} x_j] = -2\Sigma_{i=1}^m (A^T)_{ki} b_i + 2\Sigma_{i=1}^m (A^T)_{ki} [\Sigma_{j=1}^n (A)_{ij} x_j] = -2\Sigma_{i=1}^m (A^T)_{ki} b_i + 2\Sigma_{i=1}^m (A^T)_{ki} (Ax)_i$. Эти два члена представляют собой k-е компоненты векторов $-2A^T b$ и $2A^T Ax$ соответственно.

квадратов. В принципе нормальные уравнения можно решать, используя подпрограмму SGEFS из гл. 3. Однако матрица коэффициентов A^TA симметрична, поэтому требуемые SGEFS время и объем памяти могут быть сокращены вдвое. Более того, можно показать, что матрица будет положительно определенной, и поэтому не требуется выбора главного элемента. Следовательно, используя вариант гауссова исключения для положительно определенных симметричных матриц, можно решить нормальные уравнения с затратами, вдвое меньшими, чем требуется для SGEFS.

Однако у использования нормальных уравнений есть и большой минус. Оказывается, матрица A^TA часто имеет большое число обусловленности, поэтому, независимо от того, каким именно способом решать нормальные уравнения, ошибки в данных и ошибки округления, вносимые в процессе решения, значительно увеличиваются в вычисляемых коэффициентах. Грубо говоря, число обусловленности матрицы A^TA равно *квадрату* числа обусловленности матрицы A.

В качестве иллюстрации предположим, что $\varepsilon_{\text{маш}} = 10^{-6}$ и что $\text{cond}(A) = 1000$ (не особенно большая величина). Тогда ошибка в решении системы $Ax = b$ с помощью гауссова исключения составила бы примерно $1000 \times 10^{-6} = 10^{-3}$, т.е. примерно три значащие цифры были бы верными. С другой стороны, если мы попытаемся решать систему $A^TAx = A^Tb$, ошибка будет порядка $1000^2 \times 10^{-6} = 1$, т.е. решение может вообще не иметь верных значащих цифр.

В крайней ситуации, когда базисные функции $\phi_i(t)$ линейно зависимы, можно показать, что матрица A^TA вырожденна и число обусловленности может рассматриваться как бесконечное. Методы, которые избегают больших чисел обусловленности, свойственных нормальным уравнениям, лучше умеют обнаруживать линейную зависимость между базисными функциями. Гауссово исключение и его варианты плохо подходят для обнаружения такой линейной зависимости. Оценка обусловленности, включенная в SGEFS, отчасти помогает, но она может только предупредить о затруднении – она не может предложить лекарство.

Нормальные уравнения не рекомендуется использовать для решения задач наименьших квадратов общего вида. Наиболее надежные методы основаны на матричных факторизациях, использующих ортогональные матрицы. В вычислительном отношении они несколько дороже, но зато дают более точные решения. Вдобавок, они позволяют решать вырожденные и недоопределенные системы уравнений и могут применяться к решению задач оптимизации с ограничениями. Эти методы обсуждаются в следующем параграфе.

Тем не менее есть ситуации, в которых нормальные уравнения обладают преимуществом. Заманчиво использовать нормальные уравнения, когда данных намного больше, чем параметров; например, если есть 10 000 данных и только 2 параметра, матрица A^TA имеет размер всего лишь 2×2, тогда как A имеет размер $10\,000 \times 2$. Хотя для

процедуры, описанной в этой главе, требуется, чтобы матрица A хранилась в памяти, подход, основанный на ортогональных факторизациях, можно сделать столь же эффективным по памяти, как и нормальные уравнения.

Заранее может быть трудно решить, какой метод использовать. Правильный выбор зависит от таких факторов, как обусловленность матрицы, величина невязок в решении и форма данных. Для библиотек программ общего назначения программы, основанные на ортогональных факторизациях, можно рассматривать как наилучший выбор.

6.4. Ортогональные факторизации

Наша цель – свести задачи наименьших квадратов общего вида к более простым, которые можно легко решить. Поскольку линейные системы уравнений являются специальным случаем задач наименьших квадратов, не удивительно, что определение «более простой» задачи имеет почти один и тот же смысл для обоих типов задач.

Пример 6.6. Треугольная задача наименьших квадратов.

Рассмотрим следующую задачу с 3 параметрами и 5 наблюдениями:

минимизировать $\| Ax - b \|_2^2$,

где

$$A = \begin{bmatrix} 3 & 4 & 1 \\ 0 & 2 & 2 \\ 0 & 0 & 7 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad b = \begin{bmatrix} 14 \\ 10 \\ 21 \\ 6 \\ 2 \end{bmatrix}.$$

Поскольку нижняя часть матрицы A нулевая, эта задача сводится к следующей:

$$\|Ax - b\|_2^2 = \left\| \begin{bmatrix} 3 & 4 & 1 \\ 0 & 2 & 2 \\ 0 & 0 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} - \begin{bmatrix} 14 \\ 10 \\ 21 \end{bmatrix} \right\|_2^2 + \left\| \begin{bmatrix} 6 \\ 2 \end{bmatrix} \right\|_2^2 =$$

$$= \| Rx - b_{(1)} \|_2^2 + \| -b_{(2)} \|_2^2,$$

где

$$\mathrm{R} = \begin{bmatrix} 3 & 4 & 1 \\ 0 & 2 & 2 \\ 0 & 0 & 7 \end{bmatrix}, \quad b_1 = \begin{bmatrix} 14 \\ 10 \\ 21 \end{bmatrix}, \quad b_2 = \begin{bmatrix} 6 \\ 2 \end{bmatrix}.$$

Второй член $\| b_{(2)} \|_2$ не зависит от x и, таким образом, не может быть уменьшен. Первый член можно сделать равным нулю, решив систему $Rx = b_{(1)}$ размера 3×3 с помощью обратной подстановки, что дает

$$x_3 = 21/7 = 3,$$
$$x_2 = (10 - 2x_3)/2 = 4/2 = 2,$$
$$x_1 = (14 - 4x_2 - 1x_3)/3 = 3/3 = 1.$$

Итак, $x^* = (1, 2, 3)^T$ – решение задачи наименьших квадратов. □

Если задача наименьших квадратов имеет треугольную матрицу коэффициентов, т. е. матрицу с нулями ниже главной диагонали, как в примере 6.6, ее можно решить с помощью обратной подстановки. Алгоритм тот же самый, что и для линейных уравнений. Для системы размера $m \times n$ первые n компонент вектора невязок будут нулями, а оставшиеся $m - n$ компонент не будут зависеть от параметров.

Прямоугольную матрицу A можно привести к верхнетреугольной форме с помощью гауссова исключения. Однако это не поможет в решении задачи наименьших квадратов, поскольку решение приведенной задачи не будет решением первоначальной задачи. Это показывает пример 6.7.

Пример 6.7. Гауссово исключение для прямоугольных систем. Рассмотрим следующую задачу размера 3×2:

$$\text{минимизировать} \left\| \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} \right\|_2^2.$$

Решение этой задачи методами, описанными ниже, дает $x^* = (1.6, -0{,}2)^T$ и $\|Ax^* - b\|_2 \approx 0.63$. Мы можем применить к этой системе гауссово исключение: вычтем удвоенную первую строку из второй и утроенную первую строку из третьей. Полученная задача будет иметь такой вид:

$$\text{минимизировать} \left\| \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 3 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \right\|_2^2.$$

Ее решение $\bar{x} = (1,1)^T$ и $\|A\bar{x} - b\|_2 = 2$. Это намного больше, чем оптимальное значение 0.63, а параметры далеки от верных значений. Проблема возникает потому, что метод исключения не сохраняет значения 2-нормы и не приводит к эквивалентной задаче наименьших квадратов. □

Здесь можно использовать преобразования другого типа, так называемые *ортогональные преобразования*. Ортогональное преобразование – это матрица P размера $n \times m$, удовлетворяющая соотношению $P^T P = I$, где I – единичная матрица размера $n \times n$. Простой пример ортогонального преобразования дает единичная матрица, поскольку $I^T I = I \times I = I$. Не обязательно, чтобы матрица преобразования P была квадратной; например, $P = \frac{1}{2}(1, 1, 1, 1)^T$ – ортогональное преобразование, поскольку

$P^TP = \frac{1}{4}[1^2 + 1^2 + 1^2 + 1^2] = 1$. Еще один пример ортогонального преобразования – симметричная матрица

$$P = \begin{pmatrix} 3/5 & -4/5 \\ -4/5 & 3/5 \end{pmatrix}.$$

Это пример ортогонального преобразования специального типа, называемого *преобразованием Хаусхолдера*. Такие преобразования описаны в следующем разделе; они будут для нас основным инструментом решения задач наименьших квадратов.

Ортогональные преобразования ценны тем, что сохраняют 2-норму:

$$\|Px\|_2 = \sqrt{(Px)^T(Px)} = \sqrt{x^TP^TPx} = \sqrt{x^TIx} = \sqrt{x^Tx} = \|x\|_2.$$

Поскольку

$$\min_x \|Ax - b\|_2 = \|P(Ax - b)\|_2 = \|(PA)x - (Pb)\|_2,$$

наш подход к решению задачи наименьших квадратов будет состоять в нахождении последовательности ортогональных преобразований P, которая сводит A к треугольному виду: $A \to R$, где R – верхнетреугольная матрица. Каждый член последовательности будет преобразованием Хаусхолдера. Сведение A к верхнетреугольному виду можно рассматривать как матричную факторизацию: мы факторизуем A как произведение $Q * R$ ортогональной и верхнетреугольной матриц. Такое разложение часто называют *QR-факторизацией*.

Наиболее важное свойство этого подхода состоит в его точности: решения, полученные с помощью QR-факторизации, обычно намного точнее, чем решения нормальных уравнений. Неожиданно то, что QR-факторизация почти так же эффективна, как нормальные уравнения: если A – матрица размера $m \times n$, то метод QR-факторизации требует $n^2(m - n/3)$ операций с плавающей точкой, а чтобы сформировать и решить нормальные уравнения, нужно $\frac{1}{2}n^2(m + n/3)$ операций. Следовательно, улучшение точности, которое дает QR-разложение, не требует высокой платы.

*6.4.1. Преобразование Хаусхолдера

Преобразованием Хаусхолдера P называется матрица размера $m \times n$ вида

$$P = I - 2\frac{vv^T}{v^Tv},$$

где v – ненулевой вектор-столбец $m \times 1$; см. пример 6.8.

Пример 6.8. Преобразование Хаусхолдера.

Если строить преобразование Хаусхолдера, используя вектор $v = (1, 2)^T$, то

$$P = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - 2 \times \frac{1}{1 \cdot 1 + 2 \cdot 2} \begin{pmatrix} 1 \\ 2 \end{pmatrix} (1 \quad 2) =$$

$$= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \frac{2}{5} \begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix} =$$

$$= \begin{pmatrix} 3/5 & -4/5 \\ -4/5 & 3/5 \end{pmatrix}. \square$$

Легко проверяется, что любая матрица этого типа симметрична. Кроме того, любая матрица P подобного вида автоматически является ортогональной:

$$P^T P = P \times P =$$

$$= \left(I - 2 \frac{vv^T}{v^T v} \right) \left(I - 2 \frac{vv^T}{v^T v} \right) =$$

$$= I - 4 \frac{vv^T}{v^T v} + 4 \frac{vv^T}{v^T v} \times \frac{vv^T}{v^T v} =$$

$$= I - 4 \frac{vv^T}{v^T v} + 4 \frac{v (v^T v) v^T}{(v^T v)^2} =$$

$$= I - 4 \frac{vv^T}{v^T v} + 4 \frac{vv^T}{v^T v} =$$

$$= I.$$

Для матрицы 2×2 в примере 6.8 простые вычисления показывают, что, действительно, $P \times P = I$.

Преобразования Хаусхолдера имеют важное значение, потому что позволяют легко привести матрицу к верхнетреугольному виду. Если $a = (a_1, \ldots, a_m)^T$ – произвольный вектор, можно найти такой вектор v, что преобразование Хаусхолдера, построенное с использованием v, аннулирует все компоненты вектора a, кроме одной:

$$P \begin{bmatrix} a_1 \\ a_2 \\ \cdot \\ \cdot \\ \cdot \\ a_m \end{bmatrix} = \begin{bmatrix} \alpha \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix}.$$

Поскольку ортогональное преобразование сохраняет норму, должно

выполняться равенство $\alpha = \|a\|_2$. Ниже мы покажем, как выбрать v.

Чтобы показать, как привести матрицу A к треугольному виду, предположим, что A – матрица размера 5×3:

$$A = \begin{pmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{pmatrix},$$

где $\times$ обозначает произвольное число. Если a – первый столбец A и P выбирается так, как описано выше, то

$$PA = \begin{pmatrix} \alpha & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{pmatrix}.$$

Элементы, обозначенные символом $\times$, были модифицированы матрицей P.

На втором шаге выберем в качестве a четыре нижних элемента второго столбца матрицы PA и построим соответствующее преобразование Хаусхолдера $\bar{P}$. Тогда

$$P_1 PA = \begin{pmatrix} 1 & 0 \\ 0 & \bar{P} \end{pmatrix} \begin{pmatrix} \alpha & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{pmatrix} = \begin{pmatrix} \alpha & \times & \times \\ 0 & \bar{\alpha} & \times \\ 0 & 0 & \times \\ 0 & 0 & \times \\ 0 & 0 & \times \end{pmatrix}.$$

Чтобы завершить приведение, на третьем шаге выберем в качестве a три нижних элемента последнего столбца преобразованной матрицы и построим соответствующее преобразование Хаусхолдера $\tilde{P}$:

$$P_2 P_1 PA = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \tilde{P} \end{pmatrix} \begin{pmatrix} \alpha & \times & \times \\ 0 & \bar{\alpha} & \times \\ 0 & 0 & \times \\ 0 & 0 & \times \\ 0 & 0 & \times \end{pmatrix} = \begin{pmatrix} \alpha & \times & \times \\ 0 & \bar{\alpha} & \times \\ 0 & 0 & \tilde{\alpha} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

Поскольку матрицы $\bar{P}$ и $\tilde{P}$ ортогональны, ортогональными являются и матрицы P_1 и P_2, а также $P_2 P_1 P$. Итак, матрица A приводится к верхнетреугольному виду умножением слева на ортогональную матрицу.

Прежде чем дать численный пример, покажем, как строить преобразования Хаусхолдера. Пусть a – вектор, подлежащий преобразованию. Мы хотим, чтобы

$$Pa = \alpha(1, 0, \ldots, 0)^T.$$

Тогда

$$\begin{aligned} Pa &= \left(I - 2\frac{vv^T}{v^Tv}\right)a = \\ &= a - \left(2\frac{v^Ta}{v^Tv}\right)v = \\ &= \alpha e_1. \end{aligned}$$

где $e_1 = (1, 0, \ldots, 0)^T$. Отсюда

$$\left(2\frac{v^Ta}{v^Tv}\right)v = a - \alpha e_1.$$

Другими словами, вектор v коллинеарен вектору $a - \alpha e_1$. Если умножить v на отличную от нуля константу, то отвечающее ему преобразование Хаусхолдера не изменится (почему?). Это значит, что мы можем выбрать $v = a - \alpha e_1$. Поскольку $\alpha = \mp \|a\|_2$, получаем $v = a \mp \|a\|_2 e_1$.

Пример 6.9. Построение преобразования Хаусхолдера.

В качестве примера рассмотрим $a = (3, 0, 4)^T$. Тогда

$$\|a\|_2 = (3^2 + 0^2 + 4^2)^{1/2} = 5 \text{ и } v = \begin{bmatrix} 3 \\ 0 \\ 4 \end{bmatrix} \mp 5 \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$$

Вектор v можно выбрать двумя способами, каждый из которых приводит к правильному преобразованию Хаусхолдера: $v = (-2,0,4)^T$ или $v = (8,0,4)^T$. Поскольку вычитание может иногда приводить к чувствительной потере значащих цифр, более безопасно выбирать знак $\|a\|_2$ так, чтобы избежать вычитания. В нашем случае предпочтительнее $v = (8,0,4)^T$. Построение преобразования Хаусхолдера дает

$$P = I - 2\frac{vv^T}{v^Tv} = \begin{bmatrix} -0.6 & 0. & -0.8 \\ 0. & 1. & 0. \\ -0.8 & 0. & 0.6 \end{bmatrix}$$

и

$$Pa = \begin{bmatrix} -0.6 & 0. & -0.8 \\ 0. & 1. & 0. \\ -0.8 & 0. & 0.6 \end{bmatrix} \begin{bmatrix} 3 \\ 0 \\ 4 \end{bmatrix} = \begin{bmatrix} -5 \\ 0 \\ 0 \end{bmatrix},$$

что и требовалось. □

Теперь мы можем использовать эти методы для решения задачи наименьших квадратов. Это решение описывается в примере 6.10.

Пример 6.10. Решение задачи наименьших квадратов с помощью преобразований Хаусхолдера.

Рассмотрим задачу

$$\min_x \|Ax - b\|_2,$$

где

$$A = \begin{bmatrix} 3 & -1 \\ 0 & 0 \\ 4 & 7 \end{bmatrix}, \qquad b = \begin{bmatrix} 0 \\ 18 \\ 25 \end{bmatrix}.$$

Прежде всего приведем матрицу A к верхнетреугольному виду. На первом шаге рассматриваем первый столбец матрицы A, $a = (3,0,4)^T$ – вектор, использованный в предыдущем примере. Применение преобразования Хаусхолдера к A дает

$$PA = \begin{bmatrix} -5 & -5 \\ 0 & 0 \\ 0 & 5 \end{bmatrix}.$$

На втором шаге в качестве a выбирается нижняя часть второго столбца матрицы PA, т.е. $a = (0,5)^T$. Тогда $\|a\|_2 = 5$ и

$$v = \begin{pmatrix} 0 \\ 5 \end{pmatrix} \mp 5 \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

В этом случае выбор знака не имеет значения. Возьмем вектор $v = (5,5)^T$ и получим преобразование Хаусхолдера

$$\bar{P} = \begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix}.$$

Применение этого преобразования к матрице PA дает верхнетреугольную матрицу

$$\begin{bmatrix} 1 & 0 \\ 0 & \bar{P} \end{bmatrix} PA = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} -5 & -5 \\ 0 & 0 \\ 0 & 5 \end{bmatrix} = \begin{bmatrix} -5 & -5 \\ 0 & -5 \\ 0 & 0 \end{bmatrix}.$$

Для того чтобы решить задачу наименьших квадратов, мы должны также применить преобразование Хаусхолдера к вектору правой части b:

$$\begin{pmatrix} 1 & 0 \\ 0 & \bar{P} \end{pmatrix} Pb = \begin{pmatrix} 1 & 0 \\ 0 & \bar{P} \end{pmatrix} \begin{bmatrix} -20 \\ 18 \\ 15 \end{bmatrix} = \begin{bmatrix} -20 \\ -15 \\ -18 \end{bmatrix}.$$

Решение треугольной системы

$$\begin{pmatrix} -5 & -5 \\ 0 & -5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} -20 \\ -15 \end{pmatrix}$$

путем обратной подстановки дает $x^* = (1,3)^T$. Невязка для найденного решения равна $Ax^* - b = (0, -18,0)^T$. □

Приведение матрицы A к верхнетреугольному виду можно истолковать как матричную факторизацию. На каждом шаге приведения матрица умножается на ортогональную матрицу, построенную на основе преобразования Хаусхолдера. Если обозначить итоговую верхнетреугольную матрицу через R и если A – матрица размера $m \times n$, то

$$R = P_n P_{n-1} \ldots P_2 P_1 A.$$

Любая из матриц P_i удовлетворяет соотношению $P_i \times P_i = I$, поэтому

$$A = P_1 P_2 \ldots P_{n-1} P_n R.$$

Пусть $Q = P_1 \ldots P_n$. Поскольку произведение ортогональных матриц также является ортогональной матрицей, матрица Q ортогональна, т. е. мы факторизовали A в произведение $Q \times R$ ортогональной и верхнетреугольной матриц. В примере 6.10

$$A = \begin{bmatrix} 3 & -1 \\ 0 & 0 \\ 4 & 7 \end{bmatrix} = \begin{bmatrix} -0.6 & 0.8 & 0 \\ 0 & 0 & -1 \\ -0.8 & -0.6 & 0 \end{bmatrix} \begin{bmatrix} -5 & -5 \\ 0 & -5 \\ 0 & 0 \end{bmatrix} = Q \times R.$$

Хотя мы получили эту факторизацию с помощью преобразований Хаусхолдера, сходный результат можно получить, применяя процедуру ортогонализации Грама – Шмидта к столбцам матрицы A; этот подход обсуждается во многих книгах по линейной алгебре.

6.5. Подпрограмма SQRLS

Подпрограмма SQRLS является драйвером для подпрограмм из библиотеки Linpack, которая подробно описана в гл. 3. Подпрограмма SQRLS строит QR-факторизацию матрицы A размера $m \times n$, а затем решает задачу наименьших квадратов, используя эту факторизацию. Если нужно решить несколько задач с одной и той же матрицей коэффициентов, но разными правыми частями, то нужно факторизовать матрицу A только один раз и сообщить SQRLS о том, что для последующих правых частей этот шаг можно опустить.

Многие из параметров подпрограммы нам уже знакомы. На входе SQRLS требуются данные A, M, N, B, описывающие задачу, а на выходе получается массив параметров X и массив невязок RSD. Некоторые другие формальные параметры нуждаются в небольшом комментарии.

LDA – это ведущий (строчный) размер матрицы A, объявленный в вызывающей программе; он обсуждается в § 3 гл. 3. Подпрограмме SQRLS требуется несколько дополнительных массивов памяти JPVT, QRAUX и WORK; последний является рабочим массивом. Выходные массивы JPVT и QRAUX используются, чтобы запомнить выбор главных элементов и другую информацию, накапливаемую при QR-фак-

торизации; когда решают задачи с несколькими правыми частями, эти массивы не должны изменяться или использоваться для посторонних целей, поскольку они нужны для построения ортогональных преобразований, необходимых для факторизации.

Входной параметр TOL и выходное значение KR взаимосвязанны. Они используются для того, чтобы определить, является ли задача наименьших квадратов вырожденной. Это решение основывается на величине ошибок округления, которые предполагаются пропорциональными TOL. Чтобы такая оценка была эффективной, элементы A должны быть, грубо говоря, одного порядка. Один из способов обеспечить это – поделить каждую строку A на ее наибольший элемент (заметим, что соответствующая компонента B также должна быть шкалирована на это значение). Если это выполнено, разумно положить TOL = $\varepsilon_{\text{маш}}$. Во многих задачах наименьших квадратов данные являются неточными, потому что они поступают с измерительных приборов, имеющих ограниченную точность. В этом случае ошибки измерения будут преобладать над ошибками округления и может оказаться разумным положить TOL равным относительной ошибке измерения элементов матрицы A. Возможно, полезной будет модель ортогональных проекций (см. § 1).

Если входное значение TOL приводит подпрограмму SQRLS к выводу, что задача вырожденная, то некоторые из коэффициентов решения будут произвольными (задача не имеет единственного решения). Подпрограмма SQRLS присвоит этим произвольным коэффициентам нулевые значения, а затем однозначно определит оставшиеся коэффициенты. Число однозначно определяемых параметров равняется рангу матрицы (числу независимых столбцов), который оценивается в SQRLS и присваивается выходному параметру KR; при этом N – KR параметров произвольны и положены равными нулю. Следует всегда проверять параметр KR после вызова SQRLS. Если его значение меньше чем N, модель вырождена. Возможно, необходимо шкалировать данные или переосмыслить выбор модельных функций. Во всяком случае, эта ситуация не должна пройти мимо вашего внимания!

Методы, использованные в SQRLS, похожи на описанные в предыдущих параграфах, однако в них внесены некоторые изменения, чтобы можно было решать вырожденные и недоопределенные задачи. Мы молчаливо подразумевали, что число данных больше, чем число параметров (т. е. $m > n$). Это обычно верно для задач приближения данных, но есть другие приложения, в которых $m < n$ и, следовательно, решение не определено однозначно; см. § 9. Мы также предполагали, что модель невырожденна (т. е. матрица A имеет полный ранг, так что верхнетреугольная матрица, которая получается в результате QR-факторизации, обратима). Однако применение метода наименьших квадратов является разумным и для недоопределенных и вырожденных задач, и QR-факторизацию можно приспособить для их решения. Подпрограмму SQRLS можно использовать и в этих случаях. Для вырожденных задач требуется выбор главных элементов, точно так же, как частичный

выбор главных элементов был необходим в гауссовом исключении. Если матрица коэффициентов слегка изменяется, то можно просто обновить QR-факторизацию, и это будет намного эффективнее, чем вычислять ее заново. В библиотеке Linpack имеются подпрограммы для обновления QR-факторизации.

Ковариационную матрицу $(A^TA)^{-1}$ также можно вычислять с помощью подпрограмм из библиотеки Linpack. Если вы работаете в одинарной точности, то нужно обратиться к процедуре SPODI. Она подробно описана в руководстве по библиотеке Linpack (см. список литературы).

Ниже решается простая задача наименьших квадратов. Она основана на квадратичной модели, обсуждавшейся в примере 6.1. Результат работы программы имеет следующий вид:

```
МАТРИЦА КОЭФФИЦИЕНТОВ
      1.000000 1.000000  1.000000
      1.000000 2.000000  4.000000
      1.000000 3.000000  9.000000
      1.000000 4.000000 16.000000
      1.000000 5.000000 25.000000
ПРАВАЯ ЧАСТЬ
      1.000000 2.300000 4.600000 3.100000 1.200000

РАНГ МАТРИЦЫ = 3
ПАРАМЕТРЫ
   -3.020001   4.491428   -0.728571
НЕВЯЗКИ
    0.257143   -0.748572   0.702858   -0.188571   -0.022857
```

Вызывающая программа имеет следующий вид:

```
      PARAMETER (MM = 5, NN = 3)
      REAL A(MM,NN), B(MM), X(NN), QRAUX(NN), WORK(NN), TOL
      INTEGER JPVT(NN)
      DATA B /1.0, 2.3, 4.6, 3.1, 1.2/
C
C  ПОСТАНОВКА ЗАДАЧИ НАИМЕНЬШИХ КВАДРАТОВ
C  КВАДРАТИЧНАЯ МОДЕЛЬ, РАВНООТСТОЯЩИЕ ТОЧКИ
C
      M = 5
      N = 3
      DO  10 I = 1,M
          A(I,I) = 1.0
          DO  10 J = 2,N
              A(I,J) = A(I,J - 1) * I
10    CONTINUE
      TOL = 1.E - 6
      WRITE (*,*) 'МАТРИЦА КОЭФФИЦИЕНТОВ'
```

```
      WRITE (*,800) (A(I,J), J = 1,N), I = 1,M)
      WRITE (*,*) 'ПРАВАЯ ЧАСТЬ'
      WRITE (*,810) (B(I), I = 1,M)
C
C  РЕШЕНИЕ ЗАДАЧИ НАИМЕНЬШИХ КВАДРАТОВ
C
      ITASK = 1
      CALL SQRLS (A, MM, M, N, TOL, KR, B, X, B, JPVT, QRAUX,
                  WORK, ITASK, IND)
C
C  ПЕЧАТЬ РЕЗУЛЬТАТОВ
C
      IF (IND .NE. 0) WRITE (*,*) 'КОД ОШИБКИ = ', IND
      WRITE (*,*) 'РАНГ МАТРИЦЫ = ', KR
      WRITE (*,*) 'ПАРАМЕТРЫ'
      WRITE (*,800) (X(J), J = 1,N)
      WRITE (*,*) 'НЕВЯЗКИ'
      WRITE (*,810) (B(I), I = 1,M)
C
      STOP
800   FORMAT (3F12.6)
810   FORMAT (5F12.6)
      END
```

Документацию к подпрограмме SQRLS можно найти в конце главы.

6.6. Историческая справка: Гаусс

Карл Фридрих Гаусс считается одним из величайших математиков всех времен, сравнимым по значению с Архимедом и Ньютоном. Его исследования по теории чисел, геометрии и анализу, как опубликованные, так и оставшиеся в рукописях, преобразовали математику XIX столетия. Открытия в астрономии принесли ему славу среди естествоиспытателей, широкой публике он стал известен своими работами по электромагнетизму и геодезии.

Гаусс родился в 1777 г. в бедной семье. Его родители были мало образованны – только отец мог читать и писать. Гаусса послали в ничем не примечательную школу, но в первом классе он удивил учителя арифметики, мгновенно решив трудную задачу суммирования первых 100 натуральных чисел; ему было тогда 8 лет. Со временем его способности привлекли к нему внимание герцога Брунсвика, который сделал Гаусса своим протеже, оплачивал его обучение и поддерживал его вплоть до своей смерти в 1806 г. в эпоху наполеоновских войн.

Подростком Гаусс добился больших успехов в геометрии (решив 2000-летнюю проблему Евклида), алгебре (представив строгое доказательство основной теоремы алгебры) и теории чисел (написав работу,

вошедшую в его "Disquisitiones Arithmeticae", или «Арифметические исследования»). В каждом случае он не только решил конкретную проблему, но и дал полный анализ предмета и предложил новые методы решения, которые оказали далеко идущее влияние на математику.

В 1801 г. Гаусс заинтересовался прикладными аспектами математики. Двадцатью годами раньше Уильямом Гершелем была открыта седьмая планета Уран. Немецкий философ Гегель рассматривал семь как философски совершенное число и саркастически критиковал астрономов, которые, с его точки зрения, зря тратили время на поиски восьмой планеты. Почти тотчас же после опубликования этих рассуждений Гегеля, 1 января 1801 г., Джузеппе Пиацци открыл астероид Цереру.

Церера была видима только сорок дней перед тем, как пропасть из виду за Солнцем. Астрономы хотели знать, в какой точке небосклона она появится вновь. К этому времени астрономы умели вычислять орбиты комет и больших планет, но их методы не позволяли предсказать орбиту Цереры. Однако Гаусс, использовав три точки наблюдения, обширный анализ и метод наименьших квадратов, сумел определить орбиту с такой точностью, что Церера была легко обнаружена, когда вновь появилась в конце 1801 г. Эта работа принесла Гауссу необычайную славу среди ученых всей Европы (и дала побочные эффекты, проявившиеся в его личной жизни: Гаусс назвал своего старшего сына в честь Пиацци, а двух других детей от первого брака – в честь астрономов Ольберса и Хардинга).

Гаусс был одним из первых, кто использовал методы наименьших квадратов, однако еще задолго до их изобретения ученые пытались моделировать данные. Такое моделирование часто использовалось в астрономии, как в нашем примере с Церерой. Было сделано несколько оценок ее положения, но поскольку телескопы и другие инструменты были не очень точными, измерения не согласовывались между собой и некоторое приближенное значение приходилось принимать за «истинное» положение.

Другой областью приложения была картография, где наблюдатели должны были проводить измерения на местности, чтобы определить расстояния между ориентирами. Измерения не могли быть точными как из-за недостатков аппаратуры, так и из-за ошибок при выполнении измерений. Это приводило к противоречивости данных, которую следовало устранить, прежде чем чертить карту.

Уже в Европе шестнадцатого столетия, а вероятно, и более раннего времени предлагалось множество подходов к решению этой задачи. Обычно они были основаны на свойствах невязок, в наших обозначениях, вектора $Ax - b$. Одной из первых идей было выбирать параметры так, чтобы сумма остатков равнялась нулю. Более двух столетий этот метод с различными усовершенствованиями использовался для того, чтобы обеспечить вычислимость и однозначность решения. Другая идея состояла в том, чтобы при отборе решений руководствоваться 1-нормой невязки. В 1873 г. Лаплас предложил использовать ∞-норму для опреде-

ления решения. Удивительно, что именно эти подходы рассматривались первыми. Наука того времени была тесно связана с практическими вычислениями, но как раз с этими нормами вычисления проводить довольно трудно. Однако интуитивно они кажутся привлекательными как методы согласования ошибок измерений; видимо, это и сделало их популярными.

Концепция оценки наименьших квадратов впервые была открыта восемнадцатилетним Гауссом в 1795 г. Ему нужен был практический способ анализа данных топографической съемки, а позднее он регулярно использовал этот метод в астрономических вычислениях. Гаусс считал эту идею такой простой, что не опубликовал ее; он полагал, что она использовалась за много лет до него. Фактически первое опубликованное описание метода наименьших квадратов появилось в 1805 г. в приложении к книге Лежандра «Новые методы определения орбит комет». Позднее Гаусс несколько раз писал о свойствах оценок наименьших квадратов, используя для их обоснования только что появившиеся идеи из области теории вероятностей.

Когда Гаусс опубликовал описание своих астрономических вычислений, Лежандр обвинил его в краже идеи метода наименьших квадратов. Лежандр был неправ, но Гаусс не стал приводить доказательства того, что он использовал этот метод уже десять лет. Не в первый раз Гаусс предвосхищал работу других математиков. Он предъявил к своим собственным работам очень высокие требования и не публиковал их, если результаты не были завершенными и элегантными, идеи – сопряженными с глубокими приложениями, а доказательства – безукоризненными. Его сыновья рассказывали, что Гаусс отговаривал их от пути в науку, потому что не хотел, чтобы с его фамилией ассоциировались какие-либо второстепенные работы.

После смерти Гаусса в его записных книжках и неизданных рукописях обнаружилось много результатов, которые, будь они в обращении, сберегли бы другим ученым десятилетия работы. При его жизни математики часто посылали Гауссу свои работы на отзыв, и часто его откликом был вежливый комплимент вместе с замечанием, что он сделал те же самые открытия несколькими годами ранее. Это не усиливало любви к нему со стороны студентов и других математиков.

После смерти герцога и, следовательно, потери финансовой поддержки Гаусс стал профессором астрономии Гёттингенского университета; место, которое он сохранил до конца жизни. Он занял эту должность частично потому, что его преподавательские обязанности не были слишком обременительными. Идея обучения математике отталкивала его – главным образом потому, что это означало натаскивание плохо подготовленных и незаинтересованных студентов в элементарнейших преобразованиях.

Поздние работы Гаусса включали исследования по теории вероятностей, дифференциальной геометрии (результаты, которые позднее привели к теории относительности Эйнштейна), электромагнетизму,

неевклидовой геометрии и геодезии. Побочным результатом его работ по электричеству и магнетизму явилось то, что в 1833 г. он впервые в мире – за пять лет до Морзе – передал сообщения по электрическому телеграфу.

Гаусс дважды женился, в первый раз в 1805 г., а затем в 1810 г. Смерть первой жены погрузила его в одиночество, от которого он так никогда и не оправился. Он повелевал своими дочерьми и ссорился с младшими сыновьями, которые эмигрировали в Соединенные Штаты. Гаусс прожил долгую и здоровую, хотя и беспокойную, жизнь. Умер он 23 февраля 1855 г.

* 6.7. Вырожденные задачи наименьших квадратов

Вырожденная задача наименьших квадратов – это такая задача, в которой решение не единственно. Чаще всего проблема связана с данными; обычно это означает, что данных недостаточно, чтобы определить все модельные функции. Однако иногда это может указывать на недостаточность модели. Приведем простой пример.

Пример 6.11. Вырожденная задача наименьших квадратов.

Рассмотрим модель

$$b(t) \approx x_1 \cdot (1) + x_2 \cdot (t) + x_3 \cdot (2t + 1).$$

Третья модельная функция $2t + 1$ является линейной комбинацией остальных, поэтому существует бесконечно много способов аппроксимации данных. Чтобы убедиться в этом, заметим, что

$$\alpha[1 \cdot (1) + 2 \cdot (t) - 1 \cdot (2t + 1)] = 0$$

при любом α, и тогда любой набор параметров вида $x = (x_1, x_2, x_3)$ можно заменить на $x + \alpha(1, 2, -1)$ без изменения качества приближения (т. е. выбранное частное значение α не будет влиять на $b(t)$). □

Вообще вырождение имеет место, если модельные функции линейно зависимы, т. е. если некоторая ненулевая линейная комбинация модельных функций равна нулю, как в примере 6.11. Точнее, задача будет вырожденной, если столбцы матрицы коэффициентов A линейно зависимы. Когда модельные функции линейно зависимы, невязки все еще определены однозначно, даже если параметры определены неоднозначно. Таким образом, любое решение задачи наименьших квадратов дает одно и то же приближение данных. Так случилось в нашем простом примере. Если главная цель аппроксимации методом наименьших квадратов состоит в том, чтобы сгладить данные, то вырожденность может и не создавать серьезных трудностей. Однако, когда важно получить оценки параметров, вырожденность может оказаться главной проблемой.

Поскольку вырожденная задача имеет множество решений вместо единственного решения, может показаться, что задача упрощается. В действительности вырожденность делает нашу жизнь более трудной.

В точной арифметике есть четкое различие между вырожденными и невырожденными задачами, но в арифметике с округлениями может оказаться трудно решить, является ли задача вырожденной или нет. Если мы сделаем ошибку, параметры будут совершенно неточными. При этом чем ближе задача к вырожденной, тем хуже будет эффект, к которому приведет ошибка. Решение задачи наименьших квадратов

$$\min \left\| \begin{pmatrix} 1 & 1 \\ 0 & 10^{-k} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\|_2$$

равно $x = (1 - 10^k, 10^k)^T$. Предположим теперь, что эти коэффициенты были получены в результате QR-факторизации другой матрицы. Тогда элемент 10^{-k} может представлять собой всего лишь ошибку факторизации и, таким образом, в пределах точности компьютера задача может быть вырожденной. Если мы решим, что это действительно так, то нам следует решать задачу

$$\min \left\| \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\|_2.$$

Эта задача имеет много решений. Обычно полагают $x_2 = 0$, чтобы устранить неопределенность, что дает $x = (1, 0)^T$. Два полученных решения очень различны. Чем больше берется k и, следовательно, 10^{-k} делается ближе к нулю, тем больше становится различие этих решений.

В каком-то смысле это не вычислительные трудности. Даже если задача наименьших квадратов вырожденна, QR-факторизация матрицы коэффициентов A не является проблемой. Единственная трудность в том, что верхнетреугольная матрица R больше не будет обратимой, поскольку у нее будут нулевые элементы на диагонали. Однако нам следует использовать диагональные элементы R для определения, является ли задача наименьших квадратов вырожденной.

Даже в точной арифметике при QR-факторизации матрицы можно не обнаружить, что она почти вырожденна. Матрица размера $n \times n$

$$A = \begin{pmatrix} 1 & -1 & -1 & \dots & -1 \\ & 1 & -1 & \dots & -1 \\ & & 1 & \dots & -1 \\ & & & \ddots & \vdots \\ & & & & 1 \end{pmatrix}$$

уже является верхнетреугольной, поэтому она не меняется при QR-факторизации. Выглядит она вполне безобидно. Все диагональные элементы и определитель равны единице, и матрица явно невырожденная. Однако она очень близка к вырожденной. Если мы положим $a_{n,1} = 2^{-(n-2)}$, матрица становится вырожденной.

Чтобы лучше распознавать вырождение, введем в факторизацию выбор главного элемента. В гауссовом исключении при решении линейных уравнений применялся выбор главной строки. Здесь мы будем использовать выбор главного столбца. На первом шаге путем вычисления 2-норм всех столбцов матрицы определяется «наибольший» столбец. Это столбец перемещается в начало матрицы перестановкой с первым столбцом. Затем выполняется первый шаг ортогональной факторизации, аннулирующий в новом первом столбце все элементы, кроме первого. В начале каждой последующей итерации «наибольший» из оставшихся столбцов (в смысле 2-нормы) перемещается к началу матрицы, а затем применяется обычное ортогональное преобразование. Начиная со второй итерации, учитываются только нижние части столбцов. Таким образом, на втором шаге мы рассматриваем элементы со 2-го по m-й, на третьем шаге – элементы с 3-го по m-й и т. д. Результирующая матрица R будет иметь диагональные элементы, монотонно убывающие по абсолютным значениям, которые будут служить более точным индикатором вырожденности.

Пример 6.12. QR-факторизация с выбором главного столбца.

Пусть выбор главного элемента применяется к матрице

$$A = \begin{bmatrix} 1 & -1 & -1 & -1 & -1 \\ & 1 & -1 & -1 & -1 \\ & & 1 & -1 & -1 \\ & & & 1 & -1 \\ & & & & 1 \end{bmatrix}.$$

Наибольшим является последний столбец, и на первом шаге он будет переставлен с первым столбцом. В итоге QR-факторизация приводит к матрице

$$R = \begin{bmatrix} 2.2361 & 0.8944 & 0.4472 & 0. & -0.4472 \\ 0. & 1.7889 & 0.3354 & 0. & -0.3354 \\ 0. & 0. & -1.6394 & 0. & 0.4194 \\ 0. & 0. & 0. & -1.4142 & 0.7071 \\ 0. & 0. & 0. & 0. & 0.1078 \end{bmatrix}.$$

Последний диагональный элемент теперь мал ($.1078 \approx .125 = 2^{-(n-2)}$). □

Факторизацию с выбором главных столбцов можно представить в виде

$$A = QRP,$$

где P – матрица перестановок, которая регистрирует сделанные перестановки столбцов. Матрица R будет иметь вид

$$R = \begin{bmatrix} R_{(1)} & R_{(2)} \\ 0 & 0 \end{bmatrix},$$

где $R_{(1)}$ – верхнетреугольная, а $R_{(2)}$ – просто некоторая ненулевая прямоугольная матрица. Число ненулевых строк в R равно рангу матрицы A, т.е. числу ее линейно независимых столбцов.

С помощью факторизации можно получить приведенную задачу наименьших квадратов

$$\| Ax - b \|_2 = \| RPx - Q^T b \|_2 .$$

Чтобы упростить формулы, определим новые параметры $y = Px$ и новые данные $c = Q^T b$. Кроме того, разобьем y и c на части, как в матрице R, т.е. $y = (y_{(1)}, y_{(2)})^T$ и $c = (c_{(1)}, c_{(2)})^T$. Тогда получим

$$\| Ax - b \|_2^2 = \left\| \begin{bmatrix} R_{(1)} & R_{(2)} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y_{(1)} \\ y_{(2)} \end{bmatrix} - \begin{bmatrix} c_{(1)} \\ c_{(2)} \end{bmatrix} \right\|_2^2 =$$

$$= \| R_{(1)} y_{(1)} + R_{(2)} y_{(2)} - c_{(1)} \|_2^2 + \| - c_{(2)} \|_2^2 .$$

Второй член $-c_{(2)}$ не зависит от параметров y. Однако первый член может быть обращен в нуль бесконечным числом способов. Компоненты подвектора $y_{(2)}$ можно выбрать произвольными; тогда $y_{(1)}$ определяется из уравнения

$$R_{(1)} y_{(1)} = c_{(1)} - R_{(2)} y_{(2)} .$$

Эта верхнетреугольная система решается обратной подстановкой. Обычно полагают $y_{(2)} = 0$. Резюмируем наш алгоритм:

факторизовать $A = QRP$;
построить $c = Q^T b$;
решить $R_{(1)} y_{(1)} = c_{(1)}$ и положить $y = (y_{(1)}, 0)^T$;
положить $x = P^T y$, т.е. переупорядочить переменные.

Алгоритм получился почти таким же, как и в невырожденном случае. Воспользуемся им в следующем примере.

Пример 6.13. Решение вырожденной задачи наименьших квадратов.

В качестве иллюстрации рассмотрим задачу

$$A = \begin{bmatrix} 4 & 2 & 3 \\ 0 & 1 & 5 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad b = \begin{bmatrix} 8 \\ 2 \\ 5 \\ 7 \end{bmatrix}.$$

Матрица A уже имеет искомый вид, так что QR-факторизация не нужна. Матрицы $R_{(1)}$ и $R_{(2)}$ равны

$$R_{(1)} = \begin{bmatrix} 4 & 4 \\ 0 & 1 \end{bmatrix}, \quad R_{(2)} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}.$$

Чтобы получить параметры, решаем уравнения

$$R_{(1)} y_{(1)} = \begin{bmatrix} 4 & 2 \\ 0 & 1 \end{bmatrix} y_{(1)} = c_{(1)} = \begin{bmatrix} 8 \\ 2 \end{bmatrix}$$

и получаем $y = (1, 2)^T$, следовательно, $x = (1, 2, 0)^T$. Если бы мы выбрали другое значение $y_{(2)}$, скажем $y_{(2)} = 3$, то находили бы параметры из системы

$$R_{(1)} y_{(1)} = \begin{bmatrix} 4 & 2 \\ 0 & 1 \end{bmatrix} y_{(1)} = c_{(1)} - R_{(2)} y_{(2)} =$$

$$= \begin{bmatrix} 8 \\ 2 \end{bmatrix} - \begin{bmatrix} 3 \\ 5 \end{bmatrix} (3) = \begin{bmatrix} -1 \\ -13 \end{bmatrix},$$

решение которой дает $y = (25/4, -13)^T$ и $x = (25/4, -13{,}3)^T$. В обоих случаях вектор невязки равен $(0, 0, -5, -7)^T$. □

* 6.8. Сингулярное разложение

Хотя QR-факторизация с выбором главного столбца почти всегда является эффективным способом распознавания вырождения, она все же не дает полной гарантии. Некоторые задачи, близкие к вырожденным, проскальзывают через эту проверку. Наиболее надежное из имеющихся средств – это сингулярное разложение (SVD – Singular Value Decomposition), другая ортогональная матричная факторизация. Ее надежность достигается дорогой ценой – она требует в 5–10 раз больше арифметических операций, чем QR-факторизация. Кроме того, SVD невозможно эффективно обновить, когда изменяются данные. Здесь мы лишь кратко обсудим SVD и его наиболее интересные применения. Более полную информацию см. в книге [Golub, Van Loan, 1983]. Для вычисления SVD можно воспользоваться процедурой SSVDC из библиотеки Linpack.

Если A – матрица размера $m \times n$ и $m > n$, то одна из форм SVD имеет вид

$$A = U \Sigma V^T,$$

где U и V^T – ортогональные матрицы, а Σ – диагональная матрица. То есть $U^T U = I_m$, $V V^T = I_n$, U – матрица размера $m \times m$, V – матрица

размера $n \times n$, а

$$\Sigma = \begin{bmatrix} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \ddots & & \\ & & & \sigma_{n-1} & \\ & & & & \sigma_n \\ & & \mathbf{0} & & \end{bmatrix}.$$

– диагональная матрица размера $m \times n$ (т. е. такого же размера, как A). Кроме того, $\sigma_1 \geqslant \sigma_2 \geqslant \ldots \geqslant \sigma_n \geqslant 0$. Величины σ_i называются *сингулярными числами* матрицы A. Наименьшее сингулярное число σ_n равно расстоянию в 2-норме от A до ближайшей вырожденной матрицы. Количество ненулевых сингулярных чисел равно рангу матрицы. Итак, если матрица A вырождена, то по крайней мере $\sigma_n = 0$. На практике сингулярные числа редко в точности равны нулю, однако, если матрица A «близка к вырожденной», некоторые из сингулярных чисел будут малыми. Отношение σ_1/σ_n можно рассматривать как число обусловленности матрицы A. Это не то же самое число обусловленности, которое мы обсуждали в гл. 3, но оно имеет во многом сходные свойства и обычно является величиной примерно такого же порядка. Вот пример сингулярного разложения.

Пример 6.14. Сингулярное разложение.

Если

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix},$$

то SVD имеет вид

$$A = U\Sigma V^T = \begin{bmatrix} .1409 & .8247 & -.4202 & -.3513 \\ .3439 & .4263 & .2985 & .7816 \\ .5470 & .0278 & .6638 & -.5093 \\ .7501 & -.3706 & -.5420 & .0790 \end{bmatrix} \times$$

$$\times \begin{bmatrix} \mathbf{25.4624} & 0. & 0. \\ 0. & \mathbf{1.2907} & 0. \\ 0. & 0. & \mathbf{0.} \\ 0. & 0. & 0. \end{bmatrix} \begin{bmatrix} .5045 & .5745 & .6445 \\ -.7608 & -.0571 & .6465 \\ .4082 & -.8165 & .4082 \end{bmatrix}.$$

Сингулярные числа выделены жирным шрифтом. Заметим, что $\sigma_3 = 0$; это показывает неполноту ранга матрицы. □

Сингулярному разложению около ста лет. Его независимо открыли Белтрами в 1873 г. и Жордан в 1874 г. для случая квадратных матриц. В 30-е годы XX в. Эккарт и Янг распространили это разложение на прямоугольные матрицы. Однако как вычислительное средство его стали использовать намного позже, в 60-е годы. Это легко понять, если учесть, что вычисление SVD требует применения ряда достаточно тонких численных методов. Практическое использование SVD стало возможно в первую очередь благодаря Дж. Голубу, который в серии работ продемонстрировал его полезность и применимость во многих областях.

*6.8.1. Решение линейной задачи наименьших квадратов с помощью SVD

Сингулярное разложение имеет много приложений. Прежде всего, его можно использовать для решения задачи наименьших квадратов. Имеем

$$\begin{aligned} \| Ax - b \|_2 &= \| U\Sigma V^T x - b \|_2 = \\ &= \| U^T (U\Sigma V^T x - b) \|_2 = \text{(поскольку матрица } U \text{ ортогональна)} \\ &= \| \Sigma V^T x - U^T b \|_2 . \end{aligned}$$

Обозначив $U^T b$ через d и $V^T x$ через z, получим

$$\| Ax - b \|_2^2 = \| \Sigma z - d \|_2^2 = \left\| \begin{pmatrix} \sigma_1 z_1 - d_1 \\ \sigma_2 z_2 - d_2 \\ \vdots \\ \sigma_n z_n - d_n \\ -d_{n+1} \\ -d_{n+2} \\ \vdots \\ -d_m \end{pmatrix} \right\|_2^2 =$$

$$= (\sigma_1 z_1 - d_1)^2 + \ldots + (\sigma_n z_n - d_n)^2 + d_{n+1}^2 + \ldots + d_m^2 .$$

Если ни одно из сингулярных чисел не равно нулю, мы можем однозначно выбрать z_i (или, что эквивалентно, x_i), чтобы свести эту величину к ее минимуму

$$\| Ax - b \|_2^2 = d_{n+1}^2 + \ldots + d_m^2 .$$

В этом случае задача наименьших квадратов имеет единственное решение. Однако если $\sigma_n = 0$, то возможен *произвольный* выбор z_n, и любой

выбор даст одну и ту же остаточную сумму квадратов

$$\| Ax - b \|_2^2 = d_n^2 + d_{n+1}^2 + \ldots + d_m^2.$$

В этом случае задача наименьших квадратов не имеет единственного решения. Когда некоторое сингулярное число $\sigma_i = 0$, обычно соответствующее z_i приравнивают к нулю. Оказывается, сингулярные числа не равны нулю тогда и только тогда, когда базисные функции Φ_j линейно независимы в точках данных. Поэтому, если некоторые из базисных функций близки к зависимости, среди сингулярных чисел есть близкие к нулю. Поскольку нулевое сингулярное число означает вырожденность и отсутствие единственности, не удивительно, что малые сингулярные числа являются симптомом плохой обусловленности. Это проявляется в больших изменениях вычисленного решения в результате малых изменений данных или точности машинной арифметики.

Правильное использование SVD включает некоторый допуск, отражающий точность исходных данных и используемой арифметики с плавающей точкой. Любое σ_j, превышающее допуск, приемлемо, и соответствующее z_j вычисляется как $z_j = d_j/\sigma_j$. Любое σ_j, меньшее допуска, рассматривается как пренебрежимо малое, а соответствующее z_j полагается равным нулю.

Допуск в SVD играет роль, подобную роли TOL в подпрограмме SQRLS. Увеличение допуска ведет к увеличению невязок, но дает результаты, которые меньше подвержены изменениям при изменении данных. Уменьшение допуска ведет к уменьшению невязок, но дает результаты, которые более чувствительны к изменению данных. Пренебрежение величинами σ_j, меньшими, чем допуск, дает эффект снижения числа обусловленности. Поскольку число обусловленности является фактором увеличения ошибок, это приводит к более надежному определению параметров задачи наименьших квадратов. Однако за это увеличение надежности приходится платить возможным увеличением невязок.

Пример 6.15. Приближение данных с помощью SVD.

Предположим, что мы хотим решить задачу наименьших квадратов с матрицей 4×3 из примера 6.14 и с правой частью $b = (1, 0, 0, 0)^T$. При этом

$$U^T b = \begin{bmatrix} 0.1409 \\ 0.8247 \\ -0.4202 \\ -0.3513 \end{bmatrix},$$

так что, если бы ни одно из сингулярных чисел не было нулевым, мы могли бы уменьшить остаточную сумму квадратов до $d_m^2 = d_4^2 = 0.3513^2$. Однако мы знаем из предыдущего примера, что $\sigma_3 = 0$, так что лучшее, что мы можем получить, — это $d_3^2 + d_4^2 = 0.4202^2 + 0.3513^2$. Это про-

изойдет, если $z_1 = d_1/\sigma_1$, $z_2 = d_2/\sigma_2$, $z_3 = 0$, и тогда $V^T x = z$ или $x = Vz$. Вычисления дают $z_1 = 0.005533$, $z_2 = 0.6390$, $z_3 = 0$ и

$$x_{\text{SVD}} = \begin{bmatrix} -0.4833 \\ -0.0333 \\ 0.4167 \end{bmatrix}. \quad \square$$

Более реальный пример получается при использовании данных переписи населения США из задачи 4.2. Здесь $m = 8$ точек данных, значения t равны 1900, ..., 1970, а соответствующие значения y выражаются в миллионах человек и равны 75.99, 91.97, ..., 203.21. Попытаемся приблизить их квадратичной моделью:

$$b(t) \approx c_1 + c_2 t + c_3 t^2.$$

Используя подпрограмму SSVDC с одинарной точностью из библиотеки Linpack на персональном компьютере IBM с допуском 0.0 (т. е. включая все сингулярные числа), найдем коэффициенты задачи наименьших квадратов

$$c_1 = -0.372 \times 10^5, \quad c_2 = 0.368 \times 10^2, \quad c_3 = -0.905 \times 10^{-2}$$

и сингулярные числа матрицы 8×3

$$\sigma_1 = 0.106 \times 10^8, \quad \sigma_2 = 0.648 \times 10^2, \quad \sigma_3 = 0.346 \times 10^{-3}.$$

Вычисления с двойной точностью приводят примерно к тем же сингулярным числам, но коэффициенты оказываются равными

$$c_1 = 0.373 \times 10^5, \quad c_2 = -0.402 \times 10^2, \quad c_3 = 0.108 \times 10^{-1}.$$

Даже знаки двух наборов коэффициентов не совпадают.

Если использовать эту модель для предсказания численности населения в 1980 г., то вычисления с двойной точностью дадут прогноз 227.78 миллионов, а вычисления с одинарной точностью – 145.21 миллионов. Очевидно, что вычисления с одинарной точностью бесполезны, но что можно сказать о вычислениях с двойной точностью? Насколько они надежны? Для данной задачи число обусловленности $\sigma_1/\sigma_3 = 0.306 \times \times 10^{11}$; это указывает на некоторые сложности. Для значений t, заключенных между 1900 и 1970, три базисные функции близки к линейной зависимости.

Чтобы улучшить ситуацию, заметим, что $\sigma_3 \approx 0$, и положим его равным нулю. Тогда вычисления с двойной точностью дают

$$c_1 = -0.167 \times 10^{-2}, \quad c_2 = -0.162 \times 10^1, \quad c_3 = -0.871 \times 10^{-3},$$

а при вычислениях с одинарной точностью

$$c_1 = -0.166 \times 10^{-2}, \quad c_2 = -0.162 \times 10^1, \quad c_3 = -0.869 \times 10^{-3}.$$

Теперь коэффициенты согласуются гораздо лучше. Кроме того, они стали намного меньше, а это означает, что теперь при вычислении

квадратного трехчлена потеря значащих разрядов вследствие вычитаний будет не так велика. Прогноз численности населения на 1980 г. составит 212.91 миллионов и 214.96 миллионов соответственно. Эффект использования одинарной точности все еще заметен, но результаты уже не являются катастрофическими (другой подход к моделированию данных переписи можно найти в задаче 4.2).

Когда задача наименьших квадратов решается с использованием SVD, решение x_{SVD} обладает одним ценным свойством: оно является решением минимальной длины. То есть если $\bar{x}$ – любой набор параметров, который минимизирует $\| Ax - b \|_2$, то $\| x_{\text{SVD}} \|_2 \leqslant \| \bar{x} \|_2$.

* 6.8.2. Сжатие данных с помощью SVD

Предположим, что космический спутник делает фотографии Юпитера, чтобы послать их на Землю. На спутнике фото оцифровывается путем разбиения его на крошечные квадраты, называемые *пикселами* или элементами изображения (pixel = picture element). Каждый пиксел представляется одним числом, которое характеризует среднюю световую интенсивность фотографии в этом квадрате. Если каждая фотография разбивается на 500 × 500 пикселов, спутник должен передать на Землю 250 000 чисел для каждой фотографии. Это заняло бы много времени и ограничило бы число фотографий, которые возможно передать. Может оказаться, что данную матрицу удается приблизить «более простой» матрицей, хранение которой требует меньше памяти. Простоту матрицы разумно понимать как относительно низкий ранг, т.е. наличие лишь нескольких независимых столбцов. Вдобавок к экономии памяти такие матрицы могут помочь лучше понять конкретную задачу.

Рассмотрим этот массив чисел 500 × 500 как матрицу A, найдем ее сингулярное разложение и воспользуемся им для того, чтобы получить приближенную картину.

Чтобы построить аппроксимацию, обозначим i-е столбцы матриц U и V через u_i и v_i соответственно. Тогда нетрудно убедиться, что сингулярное разложение можно представить в виде

$$A = U\Sigma V^T = \sum_{i=1}^{n} \sigma_i u_i v_i^T.$$

Матрица $u_i v_i^T$ – это *внешнее произведение* столбца матрицы U и соответствующего столбца матрицы V. Каждое внешнее произведение может храниться в памяти с использованием только $m + n$ позиций, а не mn позиций. Для примера 6.14 получим

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} = U\Sigma V^T =$$

$$= 25.4624 \begin{bmatrix} .1409 \\ .3439 \\ .5470 \\ .7501 \end{bmatrix} (.5045 \quad .5745 \quad .6445) +$$

$$+1.2907 \begin{bmatrix} .8247 \\ .4263 \\ .0278 \\ -.3706 \end{bmatrix} (-.7608 \quad -.0571 \quad .6465) +$$

$$+ \quad 0 \begin{bmatrix} -.4202 \\ .2985 \\ .6638 \\ -.5420 \end{bmatrix} (.4082 \quad -.8165 \quad .4082).$$

Для того чтобы сжать данные, малые сингулярные числа заменяют нулями. Если бы использовалось только 10 сингулярных чисел, то аппроксимация имела бы вид

$$A = \sum_{i=1}^{n} \sigma_i u_i v_i^T \approx \sum_{i=1}^{10} \sigma_i u_i v_i^T.$$

Формировать матрицу аппроксимации в явном виде нет необходимости.

Эту идею можно применить к фотографии со спутника. Чтобы получить приближение к картине, заменим все малые сингулярные числа

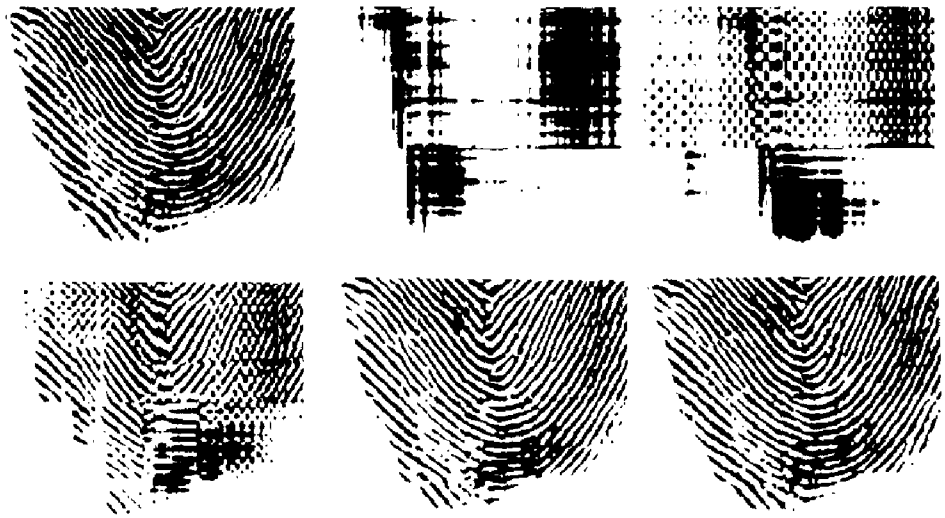


Рис. 6.5. Аппроксимация изображения с помощью SVD:
(а) исходные данные (отпечаток пальца);
(б) используется 1 сингулярное число;
(в) используется 2 сингулярных числа;
(г) используется 4 сингулярных числа;
(д) используется 10 сингулярных чисел;
(е) используется 20 сингулярных чисел.

нулями; для того чтобы сохранилось приемлемое качество изображения, нам может потребоваться оставить только 10–20 самых больших сингулярных чисел. Приближенное изображение зависит только от первых 10–20 столбцов матриц U и V и от соответствующих сингулярных чисел; оставшиеся коэффициенты будут умножаться на нуль, и ими можно пренебречь. Итак, вместо 250 000 чисел приближенная картина будет зависеть только от 10 000–20 000 чисел. На Землю при этом можно будет послать намного больше изображений. Конечно, полезность этой процедуры зависит от конкретного вида матрицы и распределения ее сингулярных чисел.

Эта идея подробно обсуждается в статье [Andrews, Patterson, 1975]. В качестве иллюстрации см. рис. 6.5.

* 6.9. Проблема нуль-пространства

Наше внимание было направлено главным образом на переопределенные системы уравнений, т. е. на задачи наименьших квадратов, в которых число точек данных больше, чем число параметров. Если задача недоопределена и система уравнений имеет множество решений, то может быть важно не только определить какое-то одно решение, но и охарактеризовать все решения системы.

Это имеет важное приложение к оптимизации с ограничениями. Предположим, что нужно решить следующую задачу:

$$\text{минимизировать } F(x_1, x_2)$$
$$\text{при ограничении } x_1 + x_2 = 1.$$

Ограничение позволяет упростить задачу путем замены $x_2 = 1 - x_1$ и привести ее к следующему виду:

$$\text{минимизировать } F(x_1, 1 - x_1).$$

Мы не только получили задачу без ограничений, но и уменьшили на 1 число переменных.

В общем случае предположим, что нужно решить n-мерную задачу

$$\text{минимизировать } F(x)$$
$$\text{при ограничении } Ax = b.$$

Хотелось бы свести ее к задаче без ограничений с меньшим числом переменных, используя ограничения для того, чтобы выразить некоторые переменные через остальные. Обычно ограничений меньше, чем переменных, поскольку в противном случае решение было бы полностью определено ограничениями и тогда не было бы нужды решать задачу оптимизации. В данном случае мы хотим определить все значения x, которые удовлетворяют ограничениям.

С помощью подпрограммы SQRLS можно найти некоторое решение $\bar{x}$ задачи $Ax = b$ (если эти уравнения не имеют решения, задача мини-

мизации теряет смысл). Если $\tilde{x}$ – любое другое решение задачи $Ax = b$, то

$$A(\bar{x} - \tilde{x}) = A\bar{x} - A\tilde{x} = b - b = 0.$$

Итак, любое решение $\tilde{x}$ задачи $Ax = b$ можно записать в виде $\tilde{x} = \bar{x} + d$, где d удовлетворяет уравнению $Ad = 0$. Говорят, что d лежит в *нуль-пространстве* (или *ядре*) матрицы A. Задача состоит в том, чтобы определить все векторы, которые лежат в нуль-пространстве матрицы A.

Для ограничения $x_1 + x_2 = 1$ матрица коэффициентов имеет вид $A = (1, 1)$. Частным решением для этого ограничения будет $\bar{x} = (0, 1)^T$. Общее решение можно записать как $\tilde{x} = \bar{x} + d$, где $d = (\alpha, -\alpha)^T$, а α произвольно. Заметим, что $Ad = 0$. Тогда исходная задача оптимизации принимает вид

$$\min_{\alpha} F(\bar{x} + \alpha d) = F(\alpha, 1 - \alpha),$$

т. е. переходит в задачу без ограничений относительно одной переменной α.

Предположим, что у нас есть QR-факторизация матрицы A^T:

$$A^T = QR = (Q_{(1)}, Q_{(2)}) \begin{bmatrix} R_{(1)} \\ 0 \end{bmatrix},$$

где $R_{(1)}$ – квадратная верхнетреугольная матрица. Преобразование этого равенства дает

$$AQ = (AQ_{(1)}, AQ_{(2)}) = R^T = (R_{(1)}^T, 0),$$

и, таким образом, $AQ_{(2)} = 0$. Итак, столбцы $Q_{(2)}$ лежат в нуль-пространстве матрицы A. Можно показать, что любой вектор из нуль-пространства матрицы A является линейной комбинацией столбцов подматрицы $Q_{(2)}$.

Пример 6.16. Определение нуль-пространства.

Для матрицы ограничений

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}$$

QR-факторизация матрицы A^T имеет вид

$$Q = \begin{bmatrix} -0.1826 & -0.8165 & -0.4001 & -0.3741 \\ -0.3651 & -0.4082 & 0.2546 & 0.7970 \\ -0.5477 & 0. & 0.6910 & -0.4717 \\ -0.7303 & 0.4082 & -0.5455 & 0.0488 \end{bmatrix},$$

$$R = \begin{bmatrix} -5.4772 & -12.7802 \\ 0. & -3.2660 \\ 0. & 0. \\ 0. & 0. \end{bmatrix}.$$

При этом $Q_{(2)}$ определяется двумя последними столбцами матрицы Q:

$$Q_{(2)} = \begin{bmatrix} -0.4001 & -0.3741 \\ 0.2546 & 0.7970 \\ 0.6910 & -0.4717 \\ -0.5455 & 0.0488 \end{bmatrix}.$$

Вычисления выполнялись с использованием библиотеки Linpack в арифметике с 16 (десятичными) цифрами, но здесь приведены только четыре цифры. Чтобы проверить, что нуль-пространство определено правильно, построим матрицу

$$A \times Q_{(2)} = \begin{bmatrix} 0.5551 \times 10^{-16} & 0.1388 \times 10^{-16} \\ 0. & 0.2776 \times 10^{-16} \end{bmatrix}.$$

Все ее элементы находятся на уровне погрешности округления. □

Теперь применим эту идею к задаче оптимизации с ограничениями. Если у нас есть некоторое решение $\bar{x}$ уравнений $Ax = b$, определяющих ограничения, то общее решение можно записать в виде $\tilde{x} = \bar{x} + Q_{(2)}p$, где p – произвольный вектор. Итак, задача оптимизации с ограничениями сводится к задаче

$$\min_p F(\bar{x} + Q_{(2)}p).$$

Если в исходной задаче было n переменных и m линейно независимых ограничений, то в новой задаче будет только $n - m$ переменных и не будет ограничений.

Хотя мы обсуждали только задачи, в которых ограничения задаются системой линейных уравнений, рассмотренные здесь идеи можно также применить к ограничениям-неравенствам вида $Ax \leqslant b$, например к задачам линейного программирования.

6.10. Задачи

6.1. Смоделируйте 11 точек данных, полагая $t_i = (i - 1)/10$ и $b_i = \mathrm{erf}(t_i)$, $i = 1, \ldots, 11$. Вычислите $\mathrm{erf}(t)$, используя алгоритм из задачи 1.1.

(а) Аппроксимируйте данные в смысле наименьших квадратов полиномами степеней от 1 до 10. Сравните аппроксимирующий полином с функцией $\mathrm{erf}(t)$ для ряда значений t, лежащих между точками данных, и посмотрите, как максимальная ошибка зависит от числа n коэффициентов полинома.

(б) Поскольку $\mathrm{erf}(t)$ – нечетная функция от t, т.е. $\mathrm{erf}(t) = -\mathrm{erf}(-t)$, имеет

смысл приближать те же данные линейной комбинацией нечетных степеней t:

$$\operatorname{erf}(t) \approx c_1 t + c_2 t^3 + \ldots + c_n t^{2n-1}.$$

Снова посмотрите, как ошибка в промежуточных точках зависит от n. Поскольку в этой задаче t изменяется на [0, 1], нет необходимости рассматривать использование других базисных полиномов.

(в) Полиномы не слишком хорошо аппроксимируют $\operatorname{erf}(t)$, поскольку они не ограничены при больших t, тогда как $\operatorname{erf}(t)$ при больших t стремится к 1. Используя те же данные, аппроксимируйте их моделью вида

$$\operatorname{erf}(t) \approx c_1 + e^{-t^2}(c_2 + c_3 z + c_4 z^2 + c_5 z^3),$$

где $z = 1/(1 + t)$. Какова ошибка в промежуточных точках по сравнению с полиномиальными моделями?

6.2. Воспользуйтесь методом наименьших квадратов для аппроксимации данных переписи населения из задачи 4.2.

(а) Аппроксимируйте данные переписи полиномами различных степеней. Используйте полученные приближения для предсказания численности населения в 1980 г. Насколько прогноз численности населения чувствителен к выбору базисных полиномов? К выбору допуска для пренебрежимо малых сингулярных чисел? К точности арифметики, если вы можете ее выбирать?

(б) Попробуйте приблизить данные переписи квадратичной функцией

$$b(t) \approx c_1 + c_2 t + c_3 t^2,$$

используя нормальные уравнения. Каково число обусловленности получившейся матрицы? Каков прогноз численности населения на 1980 г.?

6.3. Рассмотрим следующие данные с интервалом 1 с, полученные в ходе физического эксперимента (первое наблюдение выполнено в момент $t = 1.0$):

Таблица 6.2

t : 1 – 9	t : 10 – 18	t : 19 – 25
5.0291	7.5677	14.5701
6.5099	7.2920	17.0440
5.3666	10.0357	17.0398
4.1272	11.0708	15.9069
4.2948	13.4045	15.4850
6.1261	12.8415	15.5112
12.5140	11.9666	17.6572
10.0502	11.0765	
9.1614	11.7774	

Чтобы побольше узнать о данных, попытайтесь приблизить данные различными моделями.

(а) Аппроксимируйте данные прямой линией, используя подпрограмму SQRLS. Шкалируйте невязки, как это описано в § 2, и постройте график невязок (либо используя компьютерную графику, либо от руки). Невязка для одной из точек данных намного больше, чем другие. Можно

предположить, что эта точка не согласуется с остальными данными, т.е. является *выбросом*.

(б) Удалите выброс и снова аппроксимируйте данные прямой линией. Снова шкалируйте невязки и представьте их на графике. Какую картину вы видите на графике невязок? Выглядят ли невязки как случайные числа?

(в) Чтобы избавиться от тренда в невязках, аппроксимируйте данные новой моделью

$$y(t) \approx x_1 \cdot 1 + x_2 \cdot t + x_3 \cdot \sin(t).$$

Нанесите шкалированные невязки на график. Выглядят ли они случайными теперь?

6.4. Выброс может иногда очень сильно влиять на параметры модели. Вот почему важно выявлять такие данные и проверять, являются ли они правильными. Чтобы проиллюстрировать это замечание, рассмотрим следующее искусственное множество данных:

$$t_i = \begin{cases} i, & i = 1, 2, \ldots, 10, \\ 20, & i = 11, \end{cases} \qquad y_i = \begin{cases} 0, & i = 1, \ldots, 10, \\ M, & i = 11. \end{cases}$$

Поэкспериментируйте с различными значениями M. Аппроксимируйте данные прямой линией с помощью подпрограммы SQRLS при $M = 0, 5, 10, 15, 20$. Насколько быстро меняются параметры в зависимости от M? Можно ли идентифицировать точку 11 как выброс, используя график шкалированных остатков?

6.5. Рассмотрим следующее множество данных:

Таблица 6.3

t	y	Граница ошибки
0.00	20.00	20.00
0.25	51.58	24.13
0.50	68.73	26.50
0.75	75.46	27.13
1.00	74.36	26.00
1.25	67.09	23.13
1.50	54.73	18.50
1.75	37.98	12.13
2.00	17.28	4.00

Заметим, что границы ошибки различны для разных точек данных. Аппроксимируйте данные квадратичным полиномом, используя взвешенный метод наименьших квадратов, описанный в § 1.

6.6. В следующей таблице представлены уровни смертности (число смертей на сто тысяч человек) для возрастов 20–45 лет в Англии начала столетия (читайте каждый столбец сверху вниз):

Таблица 6.4

20–26 лет	27–33 лет	34–40 лет	41–45 лет
431	499	746	956
409	526	760	1014
429	563	778	1076
422	587	828	1134
530	595	846	1024
505	647	836	
459	669	916	

(а) Нанесите данные на график. Используйте подпрограмму SQRLS для аппроксимации данных прямой линией и начертите эту линию на том же графике. Считаете ли вы, что прямая хорошо аппроксимирует данные?

(б) График позволяет предположить, что для возрастных интервалов [20, 28], [28, 39] и [39, 45] данные можно приблизить различными прямыми. С помощью SQRLS аппроксимируйте данные тремя прямыми и вычертите их на том же графике. Поскольку мы не делали никаких предположений о взаимосвязи между этими прямыми, данные на каждом интервале можно при этом рассматривать совершенно независимо.

(в) Приближение, которое вы построили в (б), не будет непрерывным в точках 28 или 39. Один из способов обеспечить непрерывность – выбрать модельные функции, которые имеют это свойство. Поскольку для трех прямых линий нужно шесть коэффициентов, а непрерывность в точках 28

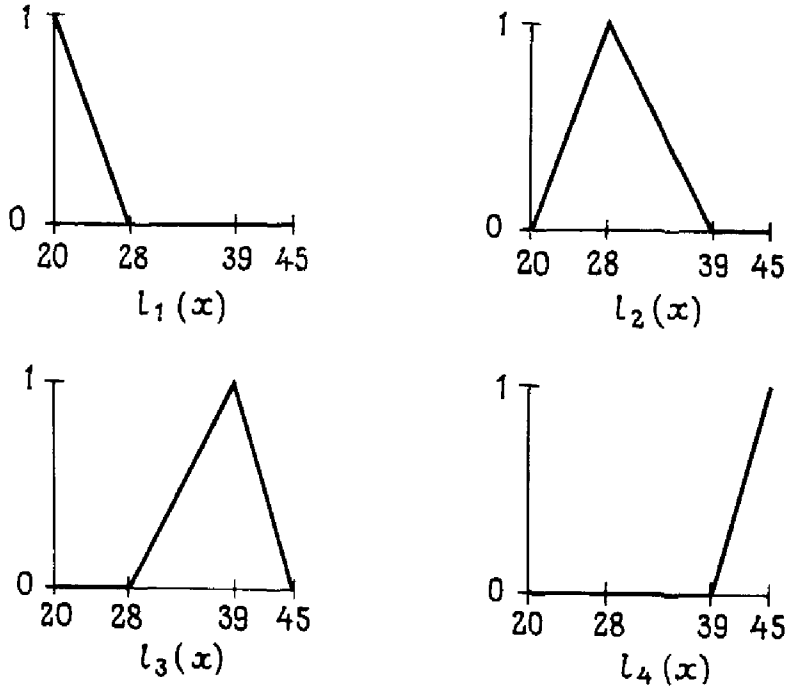


Рис. 6.6.

и 39 налагает два ограничения, можно ожидать, что потребуется $6 - 2 = 4$ модельные функции. Мы предлагаем вам использовать четыре функции $l_i(x)$, $i = 1, \ldots, 4$, изображенные на рис. 6.6. Все они определены и непрерывны при $20 \leqslant x \leqslant 45$, и тем же свойством обладает любая их линейная комбинация. Используя эти модельные функции, поставьте и решите задачу наименьших квадратов с помощью SQRLS. Начертите полученную аппроксимацию на графике, построенном в (а) и (б). Какая из трех аппроксимаций дает наилучшее приближение?

6.7. Рассмотрим следующее множество данных:

Таблица 6.5

t	y
0.00	20.00
0.25	51.58
0.50	68.73
0.75	75.46
1.00	74.36
1.25	67.09
1.50	54.73
1.75	37.98
2.00	17.28

Допустим, что мы хотим приблизить данные моделью

$$y(t) \approx x_1 \cdot 1 + x_2 \cdot \sin(t) + x_3 \cdot \cos(t) + x_4 \cdot \sin(2t) + x_5 \cdot \cos(2t).$$

Дополнительно предположим, что параметры должны удовлетворять следующим двум линейным ограничениям:

$$x_1 + x_2 + x_3 + x_4 + x_5 = 1,$$
$$3x_1 + 2x_2 + x_3 = 4.$$

Воспользуйтесь для решения этой задачи методами из § 9.

(а) Найдите частное решение $\bar{x}$ для ограничений, решив линейную систему

$$x_1 + x_2 = 1,$$
$$3x_1 + 2x_2 = 4$$

с помощью SGEFS и положив $\bar{x} = (x_1, x_2, 0, 0, 0)^T$.

(б) Пусть

$$B = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 3 & 2 & 1 & 0 & 0 \end{bmatrix}$$

есть матрица ограничений. Проверьте, что $B^T = QR$, где

$$Q = \begin{bmatrix} -.04472141 & -0.6902676 & -0.3827515 & -0.2975130 & -0.2975130 \\ -0.4472141 & -0.3067856 & 0.1717229 & 0.5815458 & 0.5815458 \\ -0.4472141 & 0.0766956 & 0.8048115 & -0.2705498 & -0.2705498 \\ -0.4472141 & 0.4601789 & -0.2968903 & 0.4932585 & -0.5067396 \\ -0.4472141 & 0.4601789 & -0.2968903 & -0.5067396 & 0.4932585 \end{bmatrix},$$

$$R = \begin{bmatrix} -2.2360649 & -2.6832657 \\ 0. & -2.6076736 \\ 0. & 0. \\ 0. & 0. \\ 0. & 0. \end{bmatrix},$$

а Q – ортогональная матрица (эту факторизацию можно найти с помощью подпрограмм из библиотеки Linpack, вызываемых подпрограммой SQRLS).

(в) Общее решение для ограничений можно записать в виде

$$x = \bar{x} + Q_{(2)} d$$

с использованием обозначений из § 9. Исходя из этой формулы, запишите исходную задачу наименьших квадратов *без ограничений* с использованием новых параметров d.

(г) Решите эту задачу наименьших квадратов относительно d с помощью SQRLS, а затем определите параметры x, которые решают исходную задачу наименьших квадратов с ограничениями.

6.8. Многие циклические явления природы можно моделировать синусоидой и ее высшими гармониками:

$$y(t) = \sum_{i=1}^{n} A_i \sin\left[\frac{2\pi}{(T/i)}(t + \phi_i)\right].$$

Величины A_i, T и ϕ_i называются соответственно *амплитудой*, *периодом* и *фазой* i-й гармоники. Аппроксимация данных такой моделью в смысле наименьших квадратов является нелинейной задачей, которая не решается методами этой главы, но для ее решения можно применить методы, которые мы будем изучать в гл. 9. Однако во многих ситуациях период T либо известен, либо его можно приближенно оценить, т. е. в качестве неизвестных остаются только амплитуды и фазы. Однако задача остается нелинейной, поскольку ϕ_i находятся под знаком синуса.

(а) Используя формулу синуса суммы, покажите, что

$$A_i \sin\left[\frac{2\pi}{(T/i)}(t + \phi_i)\right] = a_i \cos\left(\frac{2\pi t}{T/i}\right) + b_i \sin\left(\frac{2\pi t}{T/i}\right),$$

где $a_i = A_i \sin(\theta_i)$, $b_i = A_i \cos(\theta_i)$, $\theta_i = 2\pi\phi_i/(T/i)$. Итак, если a_i и b_i можно найти, то

$$A_i = \sqrt{a_i^2 + b_i^2}, \quad \phi_i = \frac{T/i}{2\pi} \operatorname{arc\,tg}(a_i/b_i).$$

Поскольку a_i и b_i входят сюда линейно, для их нахождения можно использовать подпрограмму SQRLS.

(б) Интересно изучить воздействие сжигания горючих веществ на погоду. Одним из важных показателей является концентрация двуокиси углерода в атмосфере. Файл данных C02.DAT на диске программного обеспечения содержит величины концентрации CO_2 (в долях на миллион), которые измерялись в обсерватории Мауна-Лоа (Гавайи) ежемесячно с 1958 по 1974 г. Нанесите данные на график. Вы заметите, что имеется циклическая компонента с периодом примерно 12 месяцев ($T = 12$). Имеется также тренд данных вверх, который не выражается только синусами и косинусами. Почему этот тренд является обоснованным? Исследование статистики общемировой добычи горючих ископаемых с середины 60-х годов показало, что приемлемой моделью для этих данных будет

$$y(t) = B + d\exp(\alpha t) + \sum_{i=1}^{n} A_i \sin\left[\frac{2\pi}{(12/i)}(t + \phi_i)\right],$$

где t измеряется в месяцах (т. е. $t = 1$ для первой точки, $t = 2$ для второй и т. д.), $\alpha \approx 0.0037$/месяц, а остальные параметры неизвестны. Поэкспериментируйте, сначала фиксируя n, а затем используя методику из (а) и подпрограмму SQRLS, чтобы найти другие параметры. Постройте графики приближения и невязок, варьируя $n < 4$. «Случайны» ли невязки или в них видна некая регулярность?

6.11. Пролог: SQRLS

```
      SUBROUTINE SQRLS (A, LDA, M, N, TOL, KR, B, X, RSD,
     JPVT, QRAUX, WORK, ITASK, IND)
C***  НАЧАЛО ПРОЛОГА SQRLS
C***  ДАТА НАПИСАНИЯ 870911 (ГГММДД)
C***  ДАТА ПЕРЕСМОТРА 871016 (ГГММДД)
C***  КАТЕГОРИЯ N0. D9
C***  КЛЮЧЕВЫЕ СЛОВА: Наименьшие квадраты, переопределенные
C***                  системы, линейные уравнения
C***  АВТОР: STEPHEN NASH (GEORGE MASON UNIVERSITY)
C***  НАЗНАЧЕНИЕ: SQRLS решает переопределенные, недоопре-
C     деленные и вырожденные системы линейных
C     уравнений методом наименьших квадратов.
C     Решение получается с помощью QR-факториза-
C     ции матрицы A коэффициентов размера M на N.
C***  ОПИСАНИЕ
C     Из книги D. Kahaner, C. Moler, S. Nash
C              "Numerical Methods and Software"
C              Prentice-Hall, 1988
C
C     SQRLS используется для решения (в смысле наименьших квадра-
C     тов) переопределенных, недоопределенных и вырожденных линей-
C     ный систем. Система вида A*X приближенно равна B, где A-
C     матрица размера M на N, B- заданный вектор длины M, а X-
C     вектор длины N, который нужно вычислить. Ищется решение X,
C     которое минимизирует сумму квадратов (2-норму) невязки A*
C     *X - B.
C     Численный ранг A определяется с использованием допуска TOL.
C     SQRLS использует подпрограмму SQRDC из библиотеки LIN-
C     PACK, чтобы вычислить QR-факторизацию с выбором главных
C     столбцов для матрицы A размера M на N. Дальнейшую информа-
C     цию см. в гл. 9 указанной книги.
C
C
C     Входные параметры:
C
```

```
C       A       REAL  (LDA, N)
C               Матрица, разложение которой нужно найти в задаче
C               приближения данных методом наименьших квадра-
C               тов A(I, J) есть значение J-й базисной (модельной)
C               функции в точке I.
C       LDA     INTEGER
C               Строчная размерность массива A.
C
C       M       INTEGER
C               Число строк в матрице A.
C
C       N       INTEGER
C               Число столбцов в матрице A.
C
C       TOL     REAL
C               Относительный допуск, используемый для определе-
C               ния численного ранга. Задачу следует шкалировать
C               так, чтобы все элементы A имели примерно одинако-
C               вую абсолютную точность EPS, тогда разумное
C               значение для TOL примерно равно EPS, деленному на
C               величину наибольшего по модулю матричного эле-
C               мента.
C
C       JPVT    INTEGER(N)
C       QRAUX   REAL(N)
C       WORK    REAL(N)
C               Три вспомогательных массива, используемых для
C               факторизации матрицы A (не требуются, если ITASK
C               больше 1).
C
C       B       REAL(M)
C               Правая часть линейной системы.
C               В задаче приближения данных методом наименьших
C               квадратов B(I) содержит значение I-го наблюдения.
C
C       ITASK   INTEGER
C               Если ITASK = 1, SQRLS факторизует матрицу A и
C               решает задачу наименьших квадратов.
C               Если ITASK = 2, для SQRLS это означает, что
C               матрица A была факторизована в одном из предыду-
C               щих обращений к SQRLS, и в этом случае решается
C               только задача наименьших квадратов.
C
C   Выходные параметры:
```

```
C
C
C     X       REAL(N)
C             Полученное методом наименьших квадратов решение
C             линейной системы.
C
C     RSD     REAL(M)
C             Невязка B – A*X. Может быть помещена в памяти
C             на место B.
C
C     IND     INTEGER
C             Код ошибок: IND = 0: ОШИБОК НЕТ.
C                         IND = –1: N > LDA  (ФАТАЛЬНАЯ
C                                             ОШИБКА).
C                         IND = –2: N < 1    (ФАТАЛЬНАЯ
C                                             ОШИБКА).
C                         IND = –3: ITASK < 1 (ФАТАЛЬНАЯ
C                                             ОШИБКА).
C
C     A       Содержит выходную матрицу программы SQRDC.
C             Треугольная матрица R из QR-факторизации содер-
C             жится в верхнем треугольнике, а информация, нужная
C             для восстановления ортогональной матрицы Q, хра-
C             нится ниже диагонали, а также в векторе QRAUX.
C
C     KR      INTEGER
C             Численный ранг.
C
C     JPVT    Информация из SQRDC о выборе главных столбцов.
C
C             Столбцы JPVT(1), ... , JPVT(KR) исходной матрицы
C             линейно независимы в пределах допуска TOL, а оста-
C             вшиеся столбцы зависимы от указанных KR столбцов.
C             ABS(A(1,1))/ABS(A(KR,KR)) есть оценка числа обус-
C             ловленности для матрицы, составленной из незави-
C             симых столбцов, и для R. Эта оценка будет не больше
C             1/TOL.
C
C     Использование:
C     SQRLS может быть эффективно использована для решения не-
C     скольких задач наименьших квадратов с одной и той же матрицей
C     A. Первая система решается с ITASK = 1. Последующие системы
C     решаются с ITASK = 2, чтобы избежать повторного вычисления
C     матричной факторизации, параметры KR, JPVT и QRAUX не
C     должны изменяться между вызовами SQRLS.
C
```

```
C*** ССЫЛКИ: DONGARRA ET AL, LINPACK USERS GUIDE,
                  SIAM, 1979
C*** ВЫЗЫВАЕМЫЕ ПОДПРОГРАММЫ: SQRANK, SQRLSS,
                                XERROR
C*** КОНЕЦ ПРОЛОГА SQRLS
```

Глава 7

Решение нелинейных уравнений

7.1. Введение

В гл. 3 мы изучили, как решать линейные уравнения. Во многих приложениях линейные уравнения достаточно эффективны при приближенном описании реального явления. Но существует предел того, что могут дать линейные модели. В биологических моделях, например, популяции животных часто подчиняются уравнениям типа

$$\text{популяция } (t) = 2e^{0.1t},$$

где t – время, а числа 2 и 0.1 – константы, задающие начальный размер популяции и скорость ее роста соответственно. В этом уравнении связь между временем и размером популяции нелинейная. Если мы хотим узнать, когда популяция мышей, скажем, достигнет величины 1000, мы должны решить нелинейное уравнение

$$1000 - 2e^{0.1t} = 0$$

относительно t. Для этого простого примера решение можно записать явно с помощью логарифмов; но если модель становится более сложной, например

$$\text{популяция } (t) = 2e^{0.1t} - 0.05t^2 - 0.001t^{1.3},$$

то сделать это не представляется возможным.

Вообще нас будет интересовать решение уравнения

$$f(x) = 0.$$

Для приведенного выше примера $f(x) = 1000 - 2e^{0.1x}$. Любое нелинейное уравнение можно записать в такой форме путем простого переупорядочения членов. В основном нас будут интересовать уравнения относительно одной переменной, но мы будем также рассматривать системы из n уравнений с n неизвестными:

$$\begin{aligned} f_1(x_1, \ldots, x_n) &= 0, \\ f_2(x_1, \ldots, x_n) &= 0, \\ &\vdots \\ f_n(x_1, \ldots, x_n) &= 0. \end{aligned}$$

Если положить $x = (x_1, \ldots, x_n)^T$ и $f(x) = (f_1(x), \ldots, f_n(x))^T$, то можно

представить эту систему уравнений в том же, что и выше, виде $f(x) = 0$.

Системы нелинейных уравнений возникали при построении квадратурных формул Гаусса (см. раздел 2.1 гл. 5). Они также возникают, например, в усложненных моделях популяций, включающих более одного животного. Если мыши и олени конкурируют в некотором регионе из-за запасов одной и той же пищи, то мы можем получить формулы типа

$$\text{мыши}\ (t) = 2e^{0.1t} - 0.4\ \text{олени}\ (t),$$
$$\text{олени}\ (t) = 5e^{0.2t} - 0.6\ \text{мыши}\ (t).$$

Если нужно узнать, когда популяция мышей достигнет 1000 и какова будет соответствующая популяция оленей, необходимо решить систему уравнений

$$1000 - 2e^{0.1t} + 0.4\ \text{олени}\ (t) = 0,$$
$$\text{олени}\ (t) - 5e^{0.2t} + 0.6(1000) = 0.$$

Эта система состоит из двух уравнений относительно двух неизвестных.

Между линейными и нелинейными уравнениями существуют некоторые фундаментальные различия. Прежде всего, любая невырожденная система линейных уравнений имеет единственное решение. Для нелинейных уравнений это не так. Вещественных решений может не существовать, как в уравнении

$$f(x) = \sin x + 2 = 0,$$

или их может быть много, как в уравнении

$$f(x) = \sin x = 0.$$

Можно привести еще более хитроумные примеры, для которых каждая точка некоторого интервала является решением.

Во-вторых, любая система линейных уравнений может быть решена с затратой фиксированного конечного числа арифметических операций, примерно $n^3/3$ для метода исключения Гаусса, примененного к системе из n уравнений. И только некоторые нелинейные уравнения специального вида могут быть решены точно. Если $f(x)$ – полином от одной переменной степени не выше четырех, то существуют формулы для его нулей (некоторые из них могут быть комплексными числами). Формулы для квадратичного трехчлена хорошо известны, формула Кардано для кубического полинома обсуждается в задачах, приведенных в конце этой главы. Однако, как показал примерно в 1830 г. Э. Галуа, если используются обычные арифметические операции и операция извлечения корня, то не существует формул для нулей произвольных полиномов степени 5 и выше. Так как любой конечной последовательности вычислений на ЭВМ соответствует формула такого типа, решить произвольные нелинейные уравнения, даже с одной переменной, не представляется возможным. Поистине тупиковая ситуация!

Мы обойдем эту трудность, расширив определение понятия «решение». Для простоты рассмотрим одно уравнение относительно одного неизвестного. Пусть x^* удовлетворяет уравнению $f(x^*) = 0$. Будем говорить, что $\bar{x}$ «решает» уравнение $f(x) = 0$, если либо

$$|f(\bar{x})| \approx 0, \quad \text{либо } |\bar{x} - x^*| \approx 0.$$

Таким образом, либо $\bar{x}$ приближенно удовлетворяет нелинейному уравнению, либо $\bar{x}$ приближенно равно решению уравнения. Эти определения не совсем эквивалентны, и иногда какое-то из них оказывается предпочтительнее. Смысл символа $\approx$ намеренно оставлен не вполне определенным, обычно он зависит от точности арифметики компьютера. Так как арифметические операции на компьютере выполняются с ошибками округления, то эти определения не должны казаться неестественными. В арифметике с округлениями может не найтись такого числа с плавающей точкой x, что $f(x) = 0$, и, следовательно, может не существовать ни одного решения уравнения, если придерживаться строгого определения понятия «решение».

Второе определение может показаться странным, так как оно основано на знании точки x^*, удовлетворяющей условию $f(x^*) = 0$. Если мы знаем такую точку, то зачем нам решать уравнение? А если мы не знаем такой точки, то как мы можем проверить это условие? Если функция $f(x)$ непрерывна, то нам не нужно знать x^*. Предположим, что есть две такие точки $a < b$, что $f(a) \cdot f(b) < 0$. Тогда, так как функция $f(x)$ непрерывна, на интервале $[a, b]$ должна существовать точка x^*, удовлетворяющая условию $f(x^*) = 0$. Поэтому любая точка x этого интервала удовлетворяет неравенству $|x - x^*| \leqslant |b - a|$. Если точка a близка к b, то указанное выше условие будет выполнено.

Во многих важных приложениях эти два определения решения эквивалентны. Предположим для примера, что функция $f'(x)$ непрерывна и ограничена, т. е. $|f'(x)| \leqslant L$. Если мы разложим $f(x)$ по формуле Тейлора в точке x^*, то получим

$$|f(\bar{x})| = |f(x^*) + f'(\eta)(\bar{x} - x^*)| = |f'(\eta)| \cdot |\bar{x} - x^*| \leqslant L|\bar{x} - x^*|,$$

где η – некоторая точка между $\bar{x}$ и x^*. Поэтому $|f(\bar{x})| \approx 0$, если $|\bar{x} - x^*| \approx 0$. Если мы предположим также, что $|f'(x)| \geqslant c > 0$, то дополнительно получим

$$|\bar{x} - x^*| \leqslant \frac{1}{c} |f(\bar{x})|,$$

так что если $|f(\bar{x})| \approx 0$, то и $|\bar{x} - x^*| \approx 0$.

Наши методы решения нелинейных уравнений будут итерационными; это значит, что они будут строить последовательность приближенных решений $x_1, x_2, \ldots, x_n, x_{n+1}, \ldots$ уравнения $f(x) = 0$. Обычно невозможно заранее гарантировать, что метод будет сходиться к приближенному

решению нелинейного уравнения. Однако если мы знаем точки a и b, удовлетворяющие условию $f(a) \cdot f(b) < 0$, то можно не только гарантировать сходимость, но и указать максимальное число итераций, требуемых для получения решения с заданной точностью. Это предположение не является нереалистическим. Реальная задача часто содержит информацию, позволяющую найти требуемые верхние границы (в приведенном выше примере с популяцией мышей мы можем знать начальную популяцию колонии). Если это не так, то вычисление нескольких пробных значений функции может помочь определить эти границы.

Для систем уравнений ситуация оказывается более сложной. Как и раньше, возможно отсутствие решений или наличие многих решений. В общем случае не существует алгоритма нахождения решения за конечное число шагов, и в качестве замены приходится принимать приближенные решения. Поэтому методы решения задачи будут итерационными. Однако гарантировать существование решения или сходимость к решению для системы уравнений намного сложнее, чем для одного уравнения.

При оценке эффективности алгоритмов решения нелинейных уравнений обычно предполагается, что затраты на вычисление значения функции $f(x)$ велики; настолько велики, что превосходят все остальные арифметические затраты алгоритма. Например, если вычисление $f(x)$ включает решение дифференциального уравнения или интегрирование функции, то это действительно так для любого алгоритма, обсуждаемого в данной главе. Поэтому, хотя делается все возможное, чтобы алгоритмы оказались максимально эффективными, иногда выполняют дополнительные вычисления, если они дают надежду сократить количество требуемых вычислений функции. Если значения нелинейной функции вычисляются легко, может оказаться, что алгоритмы этой главы будут работать медленнее, чем менее сложные методы. Однако если вычисление нелинейной функции стóит дорого или если нелинейное уравнение является трудным для решения (например, оно плохо обусловлено, см. гл. 3), то описываемые здесь алгоритмы, скорей всего, превзойдут по эффективности своих более простых собратьев.

Одной из наиболее известных нелинейных задач является нахождение нулей полинома. Это классическая задача, но ее практическая значимость не соответствует ее большой известности. Существует не так уж много ситуаций, в которых требуется найти нули полинома. Методы из этой главы можно использовать для полиномов специального вида. Если требуются более эффективные алгоритмы, их можно найти в собрании алгоритмов журнала ACM (см. список литературы в конце книги). Методы построены так, чтобы они быстро сходились и находили все нули (вещественные и комплексные) полинома с вещественными или комплексными коэффициентами. Дополнительную информацию см. в статье [Jenkins, Traub, 1970].

Собственные значения матрицы A можно определить как решения полиномиального уравнения $\det(A - xI) = 0$. Этот полином не нужно

строить в явном виде, так как его корни могут быть чувствительными к малым ошибкам в коэффициентах (см. § 4, гл. 3). Более подходящие для этой задачи методы можно найти в пакете EISPACK (см. руководство [Smith et al., 1976]). С другой стороны, можно вычислить корни полинома путем нахождения собственных значений соответствующей матрицы. Этот способ неэффективен с точки зрения затрат арифметических вычислений и памяти, но он работает.

Один из обычных источников систем нелинейных уравнений связан с задачей минимизации функции, но такой подход к ее решению не рекомендуется. Если мы пытаемся найти

$$\min F(x_1, \ldots, x_n)$$

и если $F(x)$ – непрерывно дифференцируемая функция, то в точке решения вектор первых производных будет нулевым:

$$\left.\frac{\partial F(x)}{\partial x_1}\right|_{x=x^*} = 0,$$
$$\left.\frac{\partial F(x)}{\partial x_2}\right|_{x=x^*} = 0,$$
$$\vdots$$
$$\left.\frac{\partial F(x)}{\partial x_n}\right|_{x=x^*} = 0.$$

Следовательно, если мы сможем найти решение x^* этой системы нелинейных уравнений, то получим возможное решение задачи минимизации. Может также оказаться, что это точка максимума функции или седловая точка. Однако решать задачу минимизации с помощью методов из этой главы обычно нецелесообразно. Для этого больше подходят методы, обсуждаемые в главе по оптимизации (гл. 9). При использовании методов оптимизации можно гарантировать сходимость к приближенному решению при довольно слабых предположениях. При этом уменьшатся затраты памяти и арифметические затраты оптимизационного алгоритма.

Нелинейные уравнения возникают также в задачах интерполяции данных в нелинейных моделях (см. гл. 6). При этом возникает система

$$f_1(x_1, \ldots, x_n) = 0,$$
$$f_2(x_1, \ldots, x_n) = 0,$$
$$\vdots$$
$$f_m(x_1, \ldots, x_n) = 0,$$

где $m > n$. Поскольку уравнений здесь больше, чем неизвестных, система

обычно не имеет решения. Точно так же, как и в случае линейной задачи наименьших квадратов, необходим компромисс, и приближенное решение находят из условия

$$\min \sum_{i=1}^{m} [f_i(x_1, \ldots, x_n)]^2 .$$

Поскольку мы минимизируем сумму квадратов нелинейных функций, эта задача называется *нелинейной задачей наименьших квадратов*. В этом случае для поиска решения также больше подходят методы, обсуждаемые в гл. 9.

В этой главе нас будет, как правило, интересовать только поиск некоторого вещественного решения одного уравнения или системы нелинейных уравнений. Нахождение всех решений нелинейного уравнения все еще остается предметом исследований, за исключением случая полиномиального уравнения.

7.2. Методы вычисления вещественных корней

Существует много алгоритмов решения нелинейных уравнений, но почти все они основаны на одном и том же принципе. Вместо непосредственного решения самого нелинейного уравнения решается последовательность более простых задач в расчете на то, что последовательность решений этих задач сходится к решению исходного уравнения. Более простыми, как правило, будут линейные уравнения, они соответствуют аппроксимации нелинейной функции прямой линией. Методы будут отличаться качеством линейной аппроксимации, скоростью сходимости к решению исходной задачи, надежностью приближения и предположениями, которые делаются относительно нелинейной функции. Для задач с одной переменной существует также возможность аппроксимировать нелинейное уравнение последовательностью квадратичных уравнений, однако этот подход мы не будем рассматривать подробно.

7.2.1. Метод бисекции

Одним из простейших и наиболее надежных методов является метод *бисекции*. Для использования бисекции нужно, чтобы были заданы две точки a и b, такие, что $a < b$ и $f(a) \cdot f(b) < 0$; таким образом, функция $f(x)$ меняет знак на интервале $[a, b]$. Обычно, но не всегда предполагают, что функция $f(x)$ непрерывна на интервале $[a, b]$, так что $f(x^*) = 0$ для некоторого $x^* \in [a, b]$. Метод бисекции систематически сужает интервал $[a, b]$, разбивая его на каждой итерации пополам до тех пор, пока он не станет меньше заданного допуска (tol_1) и, следовательно, решение x^* не окажется в пределах этого допуска. Если на некотором шаге величина $|f(x)|$ мала (меньше, чем tol_2), то итерации заканчиваются. Алгоритм имеет следующий вид.

1. Если $b - a \leqslant \mathrm{tol}_1$, то останов.
2. Положить $m = \frac{1}{2}(a + b)$ средняя точка интервала $[a, b]$. Вычислить $f(m)$. Если $|f(m)| \leqslant \mathrm{tol}_2$, то останов.
3. Если $f(a) \cdot f(m) < 0$, то положить $b \leftarrow m$, иначе положить $a \leftarrow m$. Перейти к шагу 1.

Заметим, что обе интерпретации понятия «решения» включены в алгоритм. Если мы случайно найдем нуль, то алгоритм остановится на шаге 2, если нет, мы будем продолжать процесс до тех пор, пока интервал неопределенности $[a, b]$ не достигнет заданной степени малости.

Пример 7.1. Метод бисекции.

Вот несколько итераций алгоритма бисекции для функции $f(x) = x^3$ на интервале $[a, b] = [-1, 2]$:

Задание начальных значений: $a = -1$, $b = 2$, $f(a) = -1$, $f(b) = 8$.
Итерация 1: $m = (-1 + 2)/2 = 1/2$, $f(m) = 1/8$, $b \leftarrow 1/2$, $[a, b] = [-1, 1/2]$.
Итерация 2: $m = (-1 + 1/2)/2 = -1/4$, $f(m) = -1/64$, $a \leftarrow -1/4$, $[a, b] = [-1/4, 1/2]$.
Итерация 3: $m = (-1/4 + 1/2)/2 = 1/8$, $f(m) = 1/512$, $b \leftarrow 1/8$, $[a, b] = [-1/4, 1/8]$.
...

□

На каждой итерации интервал делится пополам. Если мы берем среднюю точку m в качестве приближения к решению, то ошибка $|m - x^*|$ на каждой итерации уменьшается в среднем в два раза. Другими словами, на каждой итерации получается еще одна верная двоичная цифра. Для 32-битового числа с плавающей точкой и с 24-битовой мантиссой при условии, что начальный интервал был [1, 2], полная точность будет получена за 24 итерации. На каждой итерации функция вычисляется один раз и, следовательно, потребуется всего лишь 24 вычисления функции. (Если бы функция вычислялась для каждого числа с плавающей точкой из этого интервала, то потребовалось бы 16 777 216 вычислений, что примерно в 700 000 раз больше, чем для метода бисекции.)

Если обозначить через e_i ошибку на i-й итерации, $e_i = m - x^*$, то для метода бисекции

$$\frac{|e_{i+1}|}{|e_i|} \approx \frac{1}{2}.$$

В общем случае говорят, что метод сходится со скоростью r, если

$$\lim_{i \to \infty} \frac{|e_{i+1}|}{|e_i|^r} = C,$$

где C – некоторая конечная ненулевая константа. Используя обозначения из § 5 гл. 2, запишем $|e_{i+1}| = O(|e_i|^r)$. Для метода бисекции $r = 1$ и $C = 1/2$. Если $r = 1$, то скорость сходимости называется *линейной*; если $r > 1$ – *сверхлинейной*; если $r = 2$ – *квадратичной*.

Часто скорость сходимости r имеет большее значение, чем константа C, методы с большими значениями r более привлекательны. Тем не менее константа C также может быть важна. Если скорость сходимости линейна и $C \geqslant 1$, то нет никакой гарантии, что $e_i \to 0$, т. е. метод может не сходиться (почему?). Далее, линейно сходящийся метод с $C \approx 0$ вначале может сходиться быстрее, чем квадратично сходящийся метод с большой константой C. Таблица 7.1 иллюстрирует эти рассуждения в идеальной ситуации, когда $e_{i+1} = Ce_i^r$ для различных значений C и r. Во всех случаях $e_0 = 1$. Нулевое значение в таблице соответствует исчезновению порядка.

Таблица 7.1 Скорость сходимости в идеальной ситуации

i	$r = 1$ $C = .1$	$r = 1$ $C = .5$	$r = 1$ $C = .95$	$r = 1.3$ $C = .95$	$r = 1.6$ $C = .95$	$r = 2$ $C = .95$
1	1.0(−01)	5.0(−01)	9.5(−01)	9.5(−01)	9.5(−01)	9.5(−01)
2	1.0(−02)	2.5(−01)	9.0(−01)	8.9(−01)	8.8(−01)	8.6(−01)
3	1.0(−03)	1.2(−01)	8.6(−01)	8.1(−01)	7.7(−01)	7.0(−01)
4	1.0(−04)	6.2(−02)	8.1(−01)	7.3(−01)	6.2(−01)	4.6(−01)
5	1.0(−05)	3.1(−02)	7.7(−01)	6.3(−01)	4.4(−01)	2.0(−01)
6	1.0(−06)	1.5(−02)	7.4(−01)	5.2(−01)	2.6(−01)	3.9(−02)
7	1.0(−07)	7.8(−03)	7.0(−01)	4.1(−01)	1.1(−01)	1.5(−03)
8	1.0(−08)	3.9(−03)	6.6(−01)	2.9(−01)	2.8(−02)	2.1(−06)
9	1.0(−09)	2.0(−03)	6.3(−01)	1.9(−01)	3.1(−03)	4.1(−12)
10	1.0(−10)	9.8(−04)	6.0(−01)	1.1(−01)	9.0(−05)	1.6(−23)
11	1.0(−11)	4.9(−04)	5.7(−01)	5.5(−01)	3.2(−07)	2.5(−46)
12	1.0(−12)	2.4(−04)	5.4(−01)	2.2(−01)	3.9(−11)	6.0(−92)
13	1.0(−13)	1.2(−04)	5.1(−01)	6.7(−01)	2.1(−17)	0.0(−00)
14	1.0(−14)	6.1(−05)	4.9(−01)	1.4(−01)	1.9(−27)	0.0(−00)
15	1.0(−15)	3.0(−05)	4.6(−01)	1.9(−04)	1.7(−43)	0.0(−00)
16	1.0(−16)	1.5(−05)	4.4(−01)	1.4(−05)	3.5(−69)	0.0(−00)
17	1.0(−17)	7.6(−06)	4.2(−01)	4.5(−07)	0.0(−00)	0.0(−00)
18	1.0(−18)	3.8(−06)	4.0(−01)	5.3(−09)	0.0(−00)	0.0(−00)
19	1.0(−19)	1.9(−06)	3.8(−01)	1.7(−11)	0.0(−00)	0.0(−00)
20	1.0(−20)	9.5(−07)	3.6(−01)	9.2(−15)	0.0(−00)	0.0(−00)

Заметим, что, хотя большие значения r в конечном счете обеспечивают быструю сходимость, линейные скорости сходимости будут вполне приемлемыми, если константа C мала. Но если константа C близка

к единице, то линейная сходимость будет недопустимо медленной.

Приведенное выше описание метода бисекции является традиционным. Бисекцию можно также интерпретировать как аппроксимацию $f(x)$ специальной линейной функцией на каждой итерации и вычислении нуля этой линейной функции. График этой функции представляет собой прямую, проходящую через две точки $(a, \operatorname{sign} f(a))$ и $(b, \operatorname{sign} f(b))$. Если $f(a) < 0$ и $f(b) > 0$, то прямая задается уравнением

$$y = -1 + 2(x - a)/(b - a)$$

и $y = 0$, если $x = m = \frac{1}{2}(a + b)$. Это – грубое приближение к $f(x)$. Оно полностью игнорирует значения функции $f(x)$ и учитывает лишь их знаки. В примере 7.1 на первой итерации $a = -1$ и $b = 2$, так что для линейной аппроксимации $y = -1 + 2(x + 1)/3$. Полагая $y = 0$ и решая уравнение относительно x, получаем $x = \frac{1}{2}$, как и раньше; см. рис. 7.1.

Используя значения функции, можно построить более хорошие аппроксимации. Методы, которые мы будем сейчас рассматривать, имеют более высокую скорость сходимости (большие значения r), но они могут сходиться медленно, если не задано хорошее начальное приближение, и могут вообще не сходиться, если в алгоритме не предусмотрены меры предосторожности. Тем не менее во многих случаях они будут работать лучше, чем метод бисекции, быстро сходясь к решению для

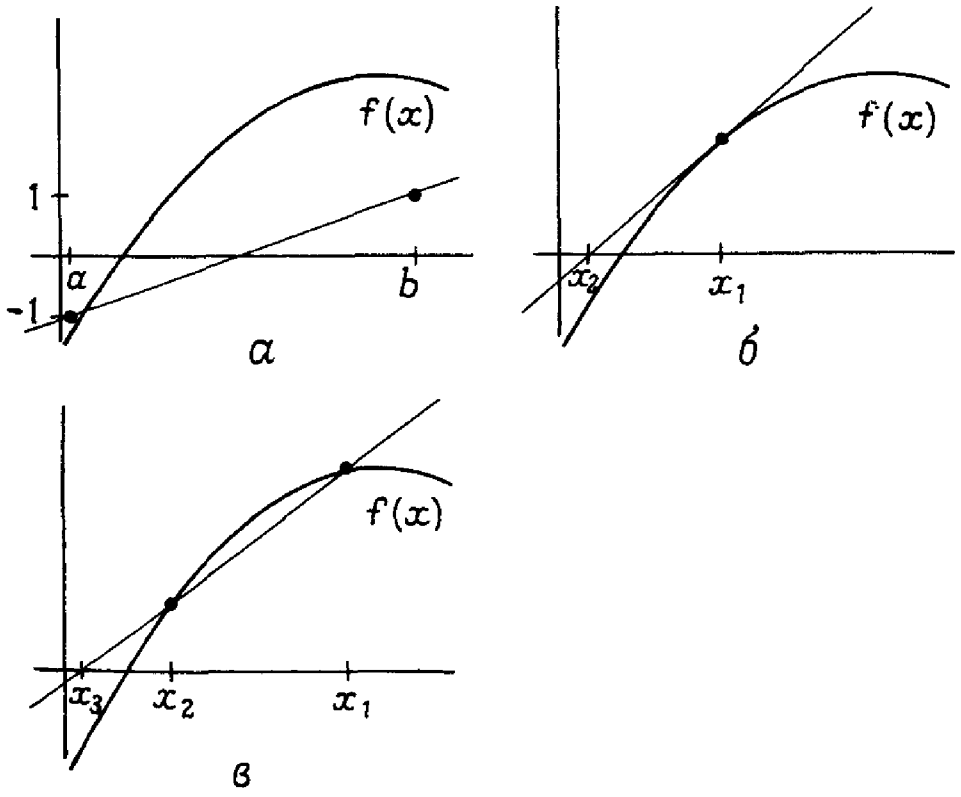


Рис. 7.1. Линейные аппроксимации нелинейной функции.

широкой области начальных приближений. Эти методы обычно предполагают, что функция $f(x)$ имеет одну или несколько непрерывных производных даже в тех случаях, когда производные явно в вычислениях не используются.

7.2.2. Метод Ньютона

Одним из лучших общих методов решения уравнения $f(x) = 0$ является метод Ньютона. Если есть некоторое приближение x_i к решению x^*, то метод Ньютона аппроксимирует функцию $f(x)$ касательной к ее графику в точке x_i. Точка пересечения касательной с осью абсцисс принимается за новое приближение. Метод Ньютона часто работает так, как показано на рис. 7.2, и приближения быстро сходятся к решению.

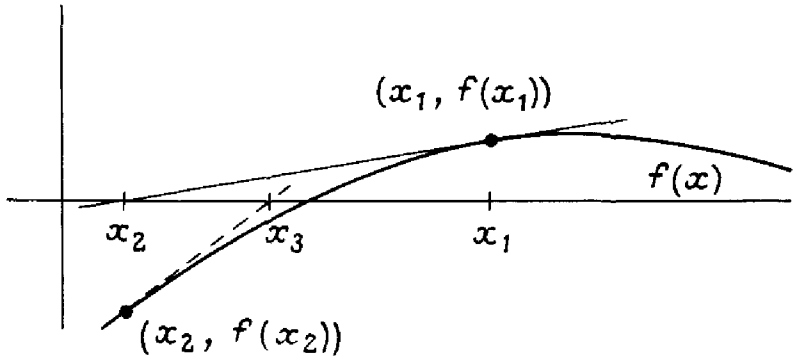


Рис. 7.2. Метод Ньютона.

Для вывода формул метода Ньютона разложим функцию $f(x)$ в ряд Тейлора в точке x_i:

$$f(x) = f(x_i) + f'(x_i)(x - x_i) + \ldots .$$

Касательная задается при помощи первых двух членов ряда

$$y = f(x_i) + f'(x_i)(x - x_i).$$

Полагая $y = 0$, получаем

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}.$$

Итерационная формула примерно в таком виде была получена Рафсоном в 1690 г., но за методом закрепилось название метода Ньютона, поскольку Ньютон предложил аналогичный процесс на несколько лет раньше.

Пример 7.2. Метод Ньютона.

Если мы применим метод Ньютона к задаче $f(x) = x^2 - 2 = 0$ с $x_0 = 1$, то он будет работать так (поскольку $f'(x) = 2x$):

Итерация 0: $x_1 = x_0 - (x_0^2 - 2)/(2x_0) = 1 - (1 - 2)/2 = 3/2$.
Итерация 1: $x_2 = 3/2 - (9/4 - 2)/3 = 17/12$.

Итерация 2: $x_3 = 17/12 - (289/144 - 2)/(17/6) = 577/408 \approx 1.414216$.

Решением нелинейного уравнения является $x^* = \sqrt{2} \approx 1.414214$. Если ньютоновскую итерационную формулу для $f(x) = x^2 - 2$ несколько преобразовать, то она примет вид

$$x_{i+1} = \frac{1}{2}\left(x_i + \frac{2}{x_i}\right).$$

Это формула для ручного вычисления квадратных корней, которому вас, возможно, когда-то обучали. В арифметике с округлениями она менее эффективна, чем формула, полученная в § 3. □

Метод Ньютона хорош тем, что быстро сходится, точнее, имеет квадратичную скорость сходимости. Это не трудно показать. Вновь разложим $f(x)$ в ряд Тейлора в окрестности точки x_i:

$$f(x) = f(x_i) + f'(x_i)(x - x_i) + \frac{1}{2}f''(\eta)(x - x_i)^2,$$

где η – некоторая неизвестная точка между x и x_i. Если положить $x = x^*$, получим $f(x) = f(x^*) = 0$:

$$0 = f(x^*) = f(x_i) + f'(x_i)(x^* - x_i) + \frac{1}{2}f''(\eta)(x^* - x_i)^2.$$

Разделим теперь это равенство на $f'(x_i)$ и преобразуем результат:

$$x^* - \left(x_i - \frac{f(x_i)}{f'(x_i)}\right) = \frac{f''(\eta)}{2f'(x_i)}(x^* - x_i)^2.$$

Согласно формулам итераций Ньютона, левая часть как раз и есть $x^* - x_{i+1}$. Если мы определим ошибки как $e_i = x^* - x_i$ и $e_{i+1} = x^* - x_{i+1}$, то получим

$$e_{i+1} = C_i e_i^2,$$

где $C_i = f''(\eta)/(2f'(x_i))$. Если итерации сходятся, то

$$\lim_{i \to \infty} \frac{|e_{i+1}|}{|e_i|^2} = C,$$

где $C = |f''(x^*)/(2f'(x^*))|$. Но это и есть определение квадратичной сходимости.

Если применить метод Ньютона к задаче $f(x) = x^2 - 2 = 0$, то мы получим результаты, приведенные в табл. 7.2. Поскольку нелинейная функция достаточно проста, мы получаем полное согласие с теоретическим результатом.

Однако метод Ньютона не всегда работает так хорошо. Он может и не сходиться. Например, если $f'(x_i) = 0$, метод не определен. Если $f'(x_i) \approx 0$, могут возникнуть трудности, так как новое приближение x_{i+1}

Таблица 7.2

x_i	e_i	e_{i+1}/e_i^2	C
1.500000000000	8.6(−02)	0.5000	0.3536
1.416666666667	2.5(−03)	0.3333	0.3536
1.414215682745	2.1(−06)	0.3529	0.3536
1.414213452374	1.6(−12)	0.3536	0.3536

может оказаться значительно худшим приближением к решению, чем x_i. Эту ситуацию иллюстрирует рис. 7.3. Тем не менее метод Ньютона можно модифицировать так, чтобы ситуации такого типа возникнуть не могли. В гл. 9 этот вопрос обсуждается более детально.

Еще одним недостатком метода Ньютона является необходимость вычисления $f'(x)$. Это может потребовать много времени, может оказаться трудным или даже невозможным делом, особенно если вычисление $f(x)$ включает, например, вычисление интеграла, решение дифференциального уравнения или если значение функции определяется

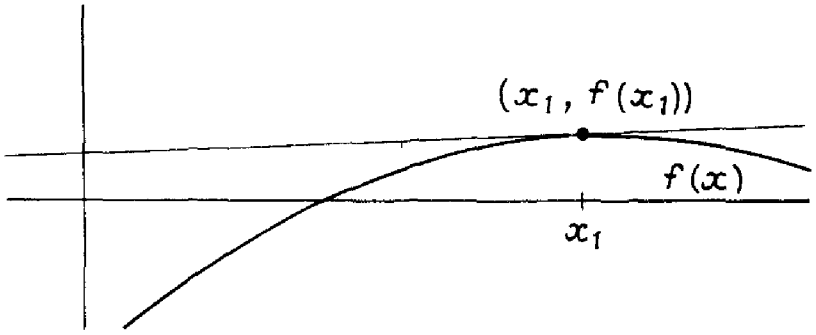


Рис. 7.3. Неудачное применение метода Ньютона.

результатом работы некоторого физического устройства. Один из способов преодоления этого недостатка состоит в аппроксимации $f'(x)$ конечными разностями, как это описано в гл. 2. Построенный таким способом метод будет вести себя почти так же, как и метод Ньютона с точными значениями производной, но его недостатком будет чрезмерное количество вычислений функций. Иной путь состоит в использовании другого метода, который не требует вычисления значений производных. Один из таких методов обсуждается ниже.

7.2.3. Метод секущих

В *метод секущих* линейная аппроксимация функции $f(x)$ строится с использованием двух предыдущих итераций x_i и x_{i-1}. Идея метода состоит в том, что через две точки $(x_i, f(x_i))$ и $(x_{i-1}, f(x_{i-1}))$ проводится секущая и нуль этой секущей берется в качестве нового приближения к решению, как показано на рис. 7.4.

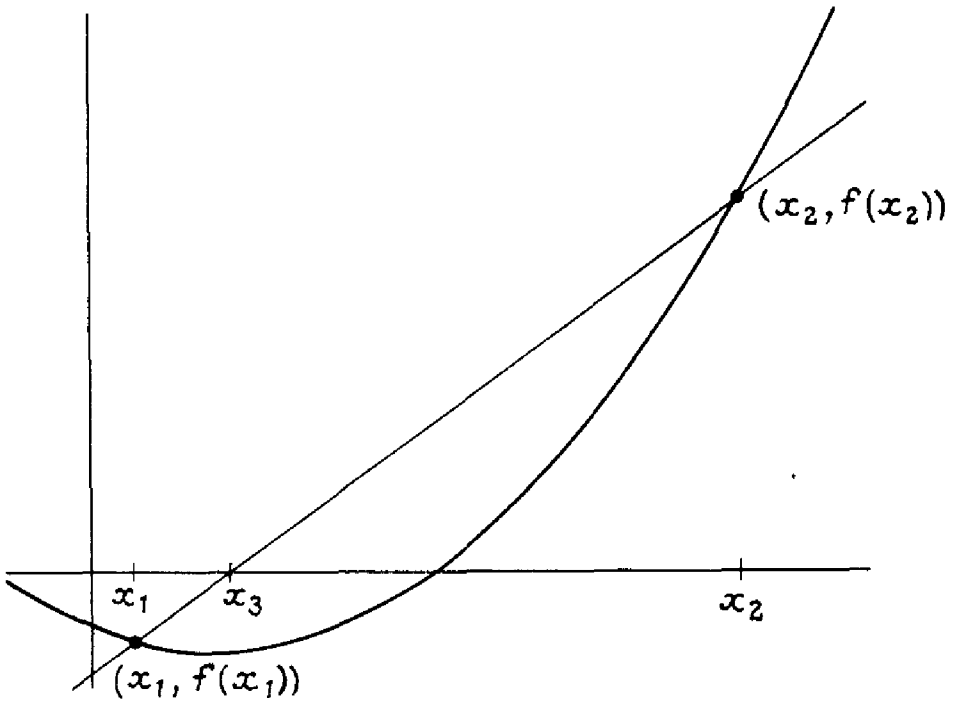


Рис. 7.4. Метод секущих.

Чтобы получить формулу метода секущих, рассмотрим уравнение прямой, проходящей через точки $(x_i, f(x_i))$ и $(x_{i-1}, f(x_{i-1}))$:

$$y = f(x_i) + (x - x_i)\frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}.$$

Полагая $y = 0$ и $x = x_{i+1}$, после перегруппировки получаем

$$x_{i+1} = x_i - f(x_i)\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}.$$

Это и есть формула метода секущих. Заметим, что она соответствует формуле метода Ньютона, в которой используется аппроксимация

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}.$$

Использование этой аппроксимации избавляет нас от необходимости вычисления производной, но приводит к замедлению сходимости алгоритма.

Если мы применим метод секущих к задаче $f(x) = x^2 - 2 = 0$ с $x_0 = 1$ и $x_1 = 2$, то получим следующие результаты:

Итерация 1: $x_2 = x_1 - f(x_1)(x_1 - x_0)/(f(x_1) - f(x_0)) =$
$= 2 - (2)(2 - 1)/(2 - (-1)) = 4/3.$

Итерация 2: $x_3 = 4/3 - (-2/9)(4/3 - 2)(-2/9 - 2) = 7/5.$

Итерация 3: $x_4 = 7/5 - (-1/25)(7/5 - 4/3)/((-1/25) - (-2/9)) =$
$= 58/41 \approx 1.414634.$

Как и раньше, решением является $x^* = \sqrt{2} \approx 1.414214$.

Метод секущих сходится сверхлинейно со скоростью $r = \frac{1}{2}(1 + \sqrt{5}) \approx 1.618$ и константой $C = |f''(x^*)/(2f'(x^*)|^{1/r}$. Доказать это довольно трудно, поэтому здесь мы этого делать не будем. Доказательство можно найти в книге [Ortega, Rheinboldt, 1971]. Однако можно продемонстрировать сходимость метода на примере задачи $f(x) = x^2 - 2 = 0$. Для метода секущих требуется два начальных приближения; мы возьмем $x_0 = 1$ и $x_1 = 2$. В данном случае решение заключено между этими двумя точками, но это не обязательно. Результаты приведены в табл. 7.3. Здесь практические результаты немного отличаются от теоретических, но характер сходимости по существу совпадает с предсказанным выше. Обратите внимание на столбец значений e_i. Заметим, что погрешность стремится к нулю достаточно быстро, со скоростью, гораздо более высокой, чем линейная, но все же не квадратичной. Такое поведение типично для метода секущих. Числа в третьем столбце не постоянны, но все они имеют примерно одну и ту же величину; только этого и можно ожидать в большинстве задач.

Таблица 7.3

x_i	e_i	e_{i+1}/e_i^r	C
1.333333333333333	8.1(−02)	0.1921	0.5259
1.400000000000000	1.4(−02)	0.8314	0.5259
1.414634146341463	4.2(−04)	0.4100	0.5259
1.414211438474870	2.1(−06)	0.6163	0.5259
1.414213562057320	3.1(−10)	0.4767	0.5259
1.414213562373095	4.4(−16)	1.0466	0.5259

Метод секущих может не сходиться, во многом по тем же самым причинам, по которым не сходится метод Ньютона. Например, если $f(x_i) = f(x_{i-1})$, то метод просто не определен. Однако, как и в методе Ньютона, многие проблемы можно обойти, если включить в алгоритм определенные предупредительные меры. Обобщения метода секущих входят в число наиболее широко используемых методов решения систем нелинейных уравнений.

Как для метода Ньютона, так и для метода секущих мы неявно предполагали, что x^* является простым нулем $f(x)$, т. е. что $f'(x^*) \neq 0$. Если x^* является кратным нулем ($f'(x^*) = 0$), то в большинстве случаев решить нелинейное уравнение оказывается намного труднее. Мы видели в гл. 2, что поиск нуля функции $f(x) = (x - a)^n$ представляет собой плохо обусловленную задачу; это также имеет место для кратных нулей произвольных нелинейных функций. Метод Ньютона и метод секущих

по-прежнему можно применять, но решение в общем случае будет определяться менее точно. Более того, эти методы будут сходиться гораздо медленнее, с линейной скоростью. Правда, в тех редких случаях, когда кратность корня известна, можно восстановить быструю сходимость. Эти замечания не означают, что указанные методы имеют какие-то изъяны; трудность заключена в самой задаче.

Другие методы можно получить, рассматривая квадратичные аппроксимации функции $f(x)$. Если значения $f(x)$ вычислены более чем в двух точках, то для улучшения последующих оценок корня можно использовать *обратную квадратичную интерполяцию*. Пусть $g(y)$ является квадратичной относительно переменной y функцией, для которой $x_j = g(f_j)$, $j = i-2, i-1, i$. Как и в § 3 гл. 4,

$$g(y) = x_{i-2}\frac{(y-f_{i-1})(y-f_i)}{(f_{i-2}-f_{i-1})(f_{i-2}-f_i)} + x_{i-1}\frac{(y-f_{i-2})(y-f_i)}{(f_{i-1}-f_{i-2})(f_{i-1}-f_i)} + x_i\frac{(y-f_{i-2})(y-f_{i-1})}{(f_i-f_{i-2})(f_i-f_{i-1})}.$$

В качестве очередного приближения нуля берется $x_{i+1} = g(0)$. Его можно непосредственно выразить через три значения x и три значения f, но сама формула для нас здесь не важна. Необходимо, чтобы все три значения f были различными. Если это не выполнено, то в формуле появляется деление на нуль.

Скорость сходимости обратной квадратичной интерполяции равна 1.839, что немного выше, чем в методе секущих. Однако необходимы три стартовых значения, и если начальные значения расположены недостаточно близко к нулю, то поведение алгоритма может оказаться хаотическим.

7.3. Подпрограмма FZERO

Один из лучших существующих машинных алгоритмов для нахождения вещественного корня функции одной переменной сочетает надежность метода бисекции с предельной скоростью сходимости метода секущих, если функция гладкая. Этот метод, который называется FZERO, был предложен ван Вайнгаарденом, Деккером и другими сотрудниками Математического центра в Амстердаме в 60-х годах. Впервые алгоритм был опубликован в статье [Dekker, 1969]; обсуждаемая здесь программа написана Шэмпайном и Уоттсом. Описание и анализ даны в работе [Wilkinson, 1967], см. особенно с. 8–12.

Пролог для FZERO приводится в конце этой главы. Типичное обращение к FZERO выглядит так:

```
CALL FZERO (F, B, C, B, RE, AE, IFLAG).
```

Второй и третий параметры B и C обозначают концевые точки интервала, в котором ищется нуль; FZERO изменяет их. На выходе параметр

B представляет собой окончательное приближение к решению x^*. Параметр F является именем вещественной подпрограммы-функции, имеющей один вещественный параметр. Параметры AE и RE указывают допустимые абсолютную и относительную погрешности результата. В программе предполагается, что F(B) и F(C) имеют разные знаки.

Подпрограмма FZERO итерационно улучшает оценки B и C для корня. Нуль все время заключен между B и C. При этом $|F(B)| \leqslant |F(C)|$. Алгоритм прекращает работу, когда длина интервала $|B - C|$ уменьшается настолько, что начинает удовлетворять условию

$$|B - C| \leqslant 2(RE|B| + AE).$$

Кроме того, чтобы застраховаться на случай, когда заданные значения RE и AE слишком малы, в проверке на сходимость используется машинный эпсилон $\varepsilon_{маш}$.

Четвертый параметр подпрограммы FZERO – это ваша самая точная догадка о положении нуля в интервале $[B, C]$. Если никакой дополнительной информации о положении нуля у вас нет, мы рекомендуем задавать в качестве значения этого параметра один из концов интервала B или C. Этот параметр на выходе из FZERO сохраняется.

На каждом шаге FZERO выбирает следующую итерацию из двух кандидатов, один из которых получен с помощью алгоритма бисекции, а другой – с помощью алгоритма секущих. Если точка, полученная по методу секущих, оказывается «приемлемой», то выбирают ее; в противном случае выбирается точка бисекции. Определение «приемлемой точки» является довольно сложным, но по существу оно означает, что точка лежит внутри текущего интервала и не слишком близка к его концам. Следовательно, длина интервала гарантированно уменьшается на каждом шаге, притом для хороших функций она уменьшается сильно.

Необходимо отметить некоторые детали подпрограммы FZERO, так как они используются и в других ситуациях. Точка бисекции вычисляется по формуле

$$XM = B + 0.5*(C - B)$$

вместо обычной формулы

$$XM = 0.5*(C + B).$$

Чтобы понять, почему, возьмем C = 0.982 и B = 0.984. Предположим, что вычисления проводятся на машине, позволяющей с округлением выполнять арифметические операции над трехразрядными десятичными числами с плавающей точкой. Тогда C + B вычисляется как 1.97 и обычная формула дает среднюю точку 0.985, которая оказывается вне интервала. Общий принцип состоит в том, что лучше строить формулы так, чтобы они представляли нужную величину в виде небольшой поправки к хорошей аппроксимации.

Особое внимание уделяется проблемам переполнения и исчезновения

порядка. Например, способ проверки того, что F(B) и F(C) имеют разные знаки, отличается от обычного теста

$$F(B) * F(C) < 0.0.$$

Если $F(B) = 10^{-30}$ и $F(C) = -10^{-30}$, то эти числа имеют противоположные знаки, но на большинстве компьютеров при вычислении произведения произойдет исчезновение порядка, результат будет положен равным нулю, и, таким образом, тест не пройдет. Точка, полученная с помощью алгоритма интерполяции, записывается в виде

$$B + P/Q,$$

но деление производится только тогда, когда оно необходимо и безопасно. Эта величина вовсе не требуется в том случае, если применяется бисекция.

Что же можно в итоге сказать о подпрограмме FZERO? Во-первых, алгоритм всегда будет сходиться, даже для арифметики с плавающей точкой. Во-вторых, число вычислений значений функции можно оценить через $\varepsilon_{маш}$ и параметры подпрограммы. В-третьих, гарантируется, что функция F меняет знак внутри итогового интервала [B, C], найденного алгоритмом. В-четвертых, если F является достаточно гладкой функцией с непрерывной второй производной в окрестности простого корня F, то алгоритм FZERO (если он стартует достаточно близко от нуля) в конечном счете прекратит выполнение бисекции и с помощью метода секущих найдет корень, причем скорость сходимости будет по крайней мере 1.618. Хотя алгоритм довольно сложен, можно гарантировать, что он работает хорошо.

Следующая программа иллюстрирует использование подпрограммы FZERO для простой функции $f(x) = x^3 - 2x - 5$. На графике $f(x)$ легко видеть, что функция имеет только один вещественный нуль, который находится между 2 и 3. На выходе программы мы получим следующие результаты:

НАЧАЛЬНЫЙ ИНТЕРВАЛ	2.000000	3.000000
ДОПУСКИ	1.0000000E–06	1.0000000
ОЦЕНКА НУЛЯ	2.094552	
ЗНАЧЕНИЕ ФУНКЦИИ	1.4305115E–06	

В этом примере $f(x)$ является полиномом, к которому можно применить более специализированные алгоритмы, но мы выбрали эту функцию по историческим причинам. Вот отрывок из письма, которое де Морган в 1861 году написал Уивеллу (цитируется по книге [Whittaker, Robinson, 1921]: «Я потому называю $x^3 - 2x - 5 = 0$ прославленным уравнением, что именно к нему Уоллис применил метод Ньютона, когда тот впервые опубликовал его, вследствие чего каждый численный метод должен проявить себя в том числе и на этом примере. Изобрести численный метод и не показать, как он работает для этого уравнения,

означает оказаться паломником, не желающим войти в маленькую дверцу»[1]).

Вот текст вызывающей программы.

```
C ИЛЛЮСТРИРУЮЩАЯ ПРОГРАММА ДЛЯ FZERO
C
      REAL       B, C, AE, RE
      EXTERNAL F
C
      B  = 2.0
      C  = 3.0
      AE = 1.E-6
      RE = 1.E-6
      WRITE (*,*) 'НАЧАЛЬНЫЙ ИНТЕРВАЛ: ', B, C
      WRITE (*,*) 'ДОПУСКИ:                ', AE, RE
C
      CALL FZERO (F, B, C, C, RE, AE, IFLAG)
C
      IF (IFLAG .NE. 1) WRITE (*,*) 'ERROR CODE =', IFLAG
      WRITE (*,*) 'ОЦЕНКА НУЛЯ =', B
      WRITE (*,*) 'ЗНАЧЕНИЕ ФУНКЦИИ =', F(B)
C
      STOP
      END
C
      REAL FUNCTION F(X)
      REAL X
C
      F = X*(X*X - 2.0) - 5.0
C
      RETURN
      END
```

Обратим внимание на оператор Фортрана

```
EXTERNAL  F
```

в начале программы. Такой оператор необходим, когда в подпрограмму передается имя функции. При вызове подпрограммы имена функций должны обрабатываться иначе, чем параметры других типов, и оператор EXTERNAL как раз указывает на это различие.

7.4. Историческая справка: Эварист Галуа

Именно Эваристу Галуа мы обязаны пессимистическим выводом о невозможности решать нелинейные уравнения общего вида. Этот

[1]) Де Морган использует скрытую цитату из книги "Pilgrim's Progress" английского религиозного писателя Джона Баниана (1628–1688).– *Прим. перев.*

результат не был изолированным умозаключением. Напротив, идеи и методы, разработанные Галуа, стали основой важного направления в алгебре, развиваемого современными математиками.

Галуа родился 25 октября 1811 г. в местечке Бур-ла-Рен под Парижем. Его отец должен был стать мэром, его мать происходила из семьи известных юристов, но, несмотря на свое происхождение, Галуа прожил жизнь, полную неудач. Он был таким невезучим, что, будь эта история не подлинной, а вымышленной, мы, возможно, рассмеялись бы над его жизненными неудачами, как если бы это происходило с героем мелодрамы Диккенса.

До двенадцати лет Галуа обучала дома мать. В 1823 г. его отправили в Париж в образцовый, классического направления лицей Людовика Великого со строгими порядками. Сначала дела шли хорошо. Но в течение следующих двух лет учеба, видимо, наскучила ему, успеваемость ухудшилась, и он был переведен в младший класс. Когда ему исполнилось четырнадцать, он начал изучать математику сперва по учебнику Лежандра, а позднее – читая самостоятельно труды Лежандра и Абеля. Он очень увлекся математикой и ничем больше не интересовался. Никаких успехов в школе у него не было.

В шестнадцать лет Галуа предпринял попытку сдать вступительные экзамены в Политехническую школу, ведущее математическое учебное заведение Франции. И провалился. (Полагают, что экзаменаторы не сумели распознать в нем гения, а сам Галуа был слишком нетерпелив, чтобы подробно разъяснять свои идеи.) Тем не менее он проходил обучение самостоятельно под руководством хорошего учителя и опубликовал свою первую научную статью (по непрерывным дробям) в 1829 году.

Спустя некоторое время Галуа послал одну свою важную статью Коши, ведущему французскому математику того времени, для представления ее во Французскую Академию наук. Коши забыл о ней и потерял ее резюме. В 18 лет Галуа снова попробовал сдать вступительные экзамены в Политехническую школу. Чувствуя, что терпит крушение, он швырнул тряпку для вытирания доски в одного из экзаменаторов. Он попал в экзаменатора, но не попал в школу.

Вскоре вследствие политического скандала отец Галуа покончил жизнь самоубийством. Похороны вылились в массовые выступления общественности.

В 1830 году в возрасте 19 лет Галуа отправил статью по алгебраическим функциям в Академию наук для участия в конкурсе на получение гран-при по математике. Секретарь Академии Жан-Батист-Жозеф Фурье взял статью домой и потерял ее.

После революции 1830 г. Галуа стал проявлять политическую активность, присоединившись к артиллеристам Национальной гвардии. 10 мая его арестовали за революционную деятельность, но суд признал его невиновным. Его арестовали снова 14 июля 1831 г., но обвинили только через два месяца в совершении глупого преступления, а именно

в незаконном ношении военной формы. Он был приговорен к шести месяцам тюрьмы. Почти в то же время его третья научная статья, посланная им в Академию наук, была оценена Пуассоном как «непонятная».

Галуа был освобожден 29 мая 1832 г. и в тот же день был вызван на дуэль. Обстоятельства этого вызова недостаточно ясны. Накануне дуэли он всю ночь записывал разработанные им математические идеи: эти записи он доверил своему другу Огюсту Шевалье. (Они были опубликованы лишь в 1846 г., руководство публикацией осуществлял Жозеф Лиувилль.) Вслед за тем он написал два последних письма. Во втором, адресованном двум неназванным друзьям, говорилось:

> Я был вызван на дуэль двумя патриотами – отказаться было невозможно. Я прошу прощения, что не посоветовался ни с кем из вас. Но мои противники взяли с меня слово чести, что никто из патриотов не узнает об этом. Ваша задача очень проста: засвидетельствовать, что я дрался против воли после того, как перепробовал все средства уладить конфликт. .. Сохраните память обо мне, так как судьба не отвела мне достаточного срока, чтобы моя страна узнала мое имя. Я умираю вашим другом.
>
> Э. Галуа

(Цитируется по книге [Bell, 1975], с. 375.)

Рано утром 30 мая он был ранен в живот. Через несколько часов его нашли и привезли в госпиталь, где он умер от воспаления брюшины вследствие пулевого ранения. Его похоронили в безымянной могиле.

7.5. Системы нелинейных уравнений

Рассмотрим теперь задачу решения системы n нелинейных уравнений с n неизвестными

$$\begin{aligned} f_1(x_1, \ldots, x_n) &= 0, \\ f_2(x_1, \ldots, x_n) &= 0, \\ &\cdot \\ &\cdot \\ &\cdot \\ f_n(x_1, \ldots, x_n) &= 0. \end{aligned}$$

Для упрощения изложения введем векторы $x = (x_1, \ldots, x_n)^T$ и $f(x) = (f_1(x), \ldots, f_n(x))^T$. В этих обозначениях наша задача имеет вид

$$f(x) = 0.$$

Между методами решения одного нелинейного уравнения и системы нелинейных уравнений существует много общего; запись задачи в такой форме поможет сделать это сходство более явным.

Мы рассмотрим аналогии метода Ньютона для нелинейных уравнений. В одномерном случае итерация метода Ньютона аппроксимирует нелинейное уравнение линейным, полученным с использованием первых двух членов ряда Тейлора. Эту же идею можно применить и в n-мерном случае. Пусть $x^{(k)}$ – текущее приближение к решению x^* системы $f(x) = 0$. Тогда ряд Тейлора в точке $x^{(k)}$ можно записать в виде

$$f(x^{(k)} + p) = f(x^{(k)}) + J(x^{(k)})p + \dots,$$

где $J(x^{(k)}) = \nabla f(x^{(k)})$ – матрица Якоби первых производных в точке $x^{(k)}$, т. е.

$$(J(x^{(k)}))_{ij} = \left(\frac{\partial f_i(x)}{\partial x_j}\right)_{x = x^{(k)}}.$$

Мы будем использовать обозначения $J^{(k)} = J(x^{(k)})$ для матрицы Якоби и $f^{(k)} = f(x^{(k)})$ для вектора значений функции.

Пример 7.3. Вычисление матрицы Якоби.

Рассмотрим модель мыши–олени из введения. Пусть численность популяции мышей равна 1000, и пусть $x_1 = t$ и $x_2 =$ олени (t). Тогда система нелинейных уравнений имеет вид

$$f_1(x_1, x_2) = 1000 - 2e^{0.1x_1} + 0.4x_2 = 0,$$
$$f_2(x_1, x_2) = x_2 - 5e^{0.2x_1} + 600 = 0.$$

Матрица Якоби имеет следующий вид:

$$J = \begin{bmatrix} -2(0.1)e^{0.1x_1} & 0.4 \\ -5(0.2)e^{0.2x_1} & 1 \end{bmatrix} = \begin{bmatrix} 0.2e^{0.1x_1} & 0.4 \\ -e^{0.2x_1} & 1 \end{bmatrix}. \quad \square$$

Если рассмотреть только первые два члена ряда Тейлора, то мы получим аппроксимацию нелинейных функций

$$f(x^{(k)} + p) \approx y = f^{(k)} + J^{(k)}p.$$

Как и в одномерном случае, будем использовать нуль нашей линейной аппроксимации для определения следующего приближения $x^{(k+1)}$ к решению. Положив $y = 0$, получим систему линейных уравнений

$$J^{(k)}p = -f^{(k)}.$$

Эти уравнения можно решить с помощью методов гл. 3. Новое приближение к решению получаем по формуле $x^{(k+1)} = x^{(k)} + p$, т. е.

$$x^{(k+1)} = x^{(k)} - (J^{(k)})^{-1}f^{(k)}.$$

Это и есть формула метода Ньютона.

Пример 7.4. Метод Ньютона.

Испытаем метод Ньютона на примере

$$f_1(x_1, x_2) = x_1x_2 - x_2^3 - 1 = 0,$$

$$f_2(x_1, x_2) = x_1^2 x_2 + x_2 - 5 = 0,$$

с $x^{(k)} = (2, 3)^T$. Матрица Якоби имеет вид

$$J^{(k)} = \begin{bmatrix} x_2 & x_1 - 3x_2^2 \\ 2x_1 x_2 & x_1^2 + 1 \end{bmatrix}_{x=(2,3)^T} = \begin{bmatrix} 3 & -25 \\ 12 & 5 \end{bmatrix},$$

и уравнение Ньютона имеет вид

$$J^{(k)} p = \begin{bmatrix} 3 & -25 \\ 12 & 5 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} 22 \\ -10 \end{bmatrix} = -f^{(k)}.$$

Использование гауссова исключения дает $p = (-0.4444, -0.9333)^T$, и поэтому новым приближением к решению будет $x^{(k+1)} = x^{(k)} + p = (1.5556, 2.0667)^T$. Решением системы нелинейных уравнений является $x^* = (2, 1)^T$. Полный процесс итераций показан в табл. 7.4. □

Таблица 7.4

$x^{(k)}$	$\|x^{(k)} - x^*\|_2$	$\|f(x^{(k)})\|_2$
(2.000000000000000, 3.000000000000000)	2.0(+00)	2.4(+01)
(1.555555555555556, 2.066666666666667)	1.2(+00)	6.9(+00)
(1.547205413881141, 1.477793334960415)	6.6(−01)	1.9(+00)
(1.780535029261150, 1.158864811005650)	2.7(−01)	5.2(−01)
(1.952843000499447, 1.028442688075807)	5.5(−02)	9.3(−02)
(1.997762973104506, 1.001240409084899)	2.6(−03)	4.4(−03)
(1.999995236211894, 1.000002599617500)	5.4(−06)	9.5(−06)
(1.999999999978874, 1.000000000011532)	2.4(−11)	4.2(−11)
(2.000000000000000, 1.000000000000000)	0.0(+00)	0.0(+00)

Метод Ньютона будет сходиться квадратично, так же как и в примере 7.4, и эта быстрая сходимость делает метод привлекательным. С другой стороны, каждая итерация требует решения системы линейных уравнений – относительно трудоемкой процедуры. Кроме того, метод Ньютона требует вычисления всех n^2 первых частных производных нелинейных функций. Если производные трудно вычислить, то это может оказаться серьезным препятствием. Заметим, что производные можно аппроксимировать, используя численное дифференцирование (см. § 5 гл. 2); можно также воспользоваться обобщениями метода секущих, не требующими вычисления значений производных.

Как можно ожидать на основании обсуждения в разд. 2.2, если используются обобщения метода секущих, квадратичной сходимости может не быть. Кроме того, как и в одномерном случае, метод Ньютона

различные его варианты могут вообще не сходиться. Это может произойти, если матрица Якоби будет вырожденной. Так, если в примере 7.4 взять $x^{(k)} = (0, 0)^T$, то обе компоненты первой строки матрицы будут равны нулю.

Для улучшения свойств метода Ньютона при решении задач общего вида принимаются определенные меры предосторожности. Возможны различные подходы. В следующем разделе мы обсудим некоторые детали одного из таких подходов, названного стратегией *доверительной области*. Другой подход, основанный на *линейном поиске*, обсуждается в гл. 9.

*7.5.1. Меры предосторожности в методе Ньютона

Меры предосторожности для метода Ньютона основаны на двух идеях. Во-первых, мы хотим иметь гарантию, что на каждой итерации происходит приближение к решению. Во-вторых, мы хотим предотвратить использование больших шагов, которые могут привести к катастрофе. Продвижение к решению будет измеряться с помощью значений нелинейных функций; мы хотим добиться выполнения условия

$$\| f(x^{(k+1)}) \|_2 < \| f(x^{(k)}) \|_2 .$$

С целью предотвращения больших шагов наложим ограничение на шаг p:

$$\| p \|_2 \leqslant \delta ,$$

где δ – некоторый ограничитель, выбираемый алгоритмом. Ограничитель δ выражает нашу степень доверия к модели $f(x^{(k)} + p) \approx f(x^{(k)}) + J^{(k)} p$. Если модель хорошая, то аппроксимация будет эффективной при больших значениях p и можно выбрать большой ограничитель δ. Если модель плохая, то аппроксимация будет приемлемой лишь при малых значениях p и нужно использовать малое значение δ. Множество $\{p: \| p \|_2 \leqslant \delta\}$ называется *доверительной областью*.

Если ограничитель δ мал, трудно надеяться, что решение уравнения Ньютона будет удовлетворять нашему ограничению. Необходим некоторый компромисс. Один подход, имеющий как теоретические, так и практические преимущества, состоит в отказе от требования, чтобы шаг действительно удовлетворял уравнению $Jp = -f$; вместо этого указанное уравнение решается настолько хорошо, насколько это возможно при выполнении ограничения $\| p \|_2 \leqslant \delta$. Другими словами, требуется решить следующую задачу оптимизации:

$$\begin{array}{ll} \text{минимизировать по } p & \| J^{(k)} p + f^{(k)} \|_2 \\ \text{при условии} & \| p \|_2 \leqslant \delta . \end{array}$$

Если бы никаких ограничений не было и если бы матрица $J^{(k)}$ была невырожденной, то, очевидно, выбор $p = -(J^{(k)})^{-1} f^{(k)}$ минимизировал

бы $\| J^{(k)}p + f^{(k)} \|_2$, так как это значение равнялось бы нулю. Мы не будем рассматривать здесь методы решения этой минимизации задачи; дальнейшую информацию можно найти в гл. 9 и в книге [Dennis, Schnabel, 1983].

Опишем простейшую версию всего алгоритма в целом. На k-й итерации выполняются следующие действия:

1. Если $f(x^{(k)}) \approx 0$, то останов.
2. Вычислить шаг p, решая приведенную выше задачу оптимизации с ограничением.
3. Если $\| f(x^{(k)} + p) \|_2 < \| f(x^{(k)}) \|_2$, то шаг принимается. Положить $x^{(k+1)} \leftarrow x^{(k)} + p$, $k \leftarrow k + 1$. Перейти к шагу 1.
4. Если шаг отвергается, то уменьшить ограничитель δ. Положить $x^{(k+1)} \leftarrow x^{(k)}$, $k \leftarrow k + 1$. Перейти к шагу 1.

На первый взгляд алгоритм может циклиться бесконечно, не давая приемлемого шага p и поэтому не делая сдвига в направлении решения. Когда шаг отвергается, алгоритм показывает, что ряд Тейлора не дает хорошей аппроксимации данной функции в точке $x^{(k)}$. Однако для точек, достаточно близких к $x^{(k)}$, основанная на ряде Тейлора модель будет хорошо аппроксимировать функцию. Поэтому, когда ограничитель δ и, следовательно, $\| p \|_2$ станут малыми, шаг все-таки окажется приемлемым. Для хорошо написанного алгоритма, вероятно, потребуется только одно или два повторения.

Невозможно гарантировать, что алгоритмы доверительной области будут сходиться к решению системы нелинейных уравнений. Для задач размерности больше единицы методы с гарантированной глобальной сходимостью все еще остаются предметом исследования. Эта проблема тесно связана с задачей глобальной оптимизации.

Как бы то ни было, если используется стратегия доверительной области, то можно гарантировать сходимость к локальному решению. Локальное решение $\bar{x}$ является локальным минимизатором функции (более подробную информацию о локальных решениях см. в гл. 9). Локальное решение не обязательно должно удовлетворять уравнению $f(\bar{x}) = 0$.

Пример 7.5. Локальное решение нелинейного уравнения.
В качестве иллюстрации рассмотрим одномерную задачу

$$f(x) = x^3 - 3x + 18.$$

Решение нелинейного уравнения $f(x) = 0$ есть $x^* = -3$. Однако точка $\bar{x} = 1$ является его локальным решением. Чтобы убедиться в этом, заметим, что

$$\| f(x) \|_2^2 = f^2(x) = x^6 - 6x^4 + 36x^3 + 9x^2 - 108x + 326.$$

В точке $x = 1$ производные имеют вид

$$\frac{d}{dx}(\|f(x)\|_2^2) = 6x^5 - 24x^3 + 108x^2 + 18x - 108 = 0,$$

$$\frac{d^2}{dx^2}(\|f(x)\|_2^2) = 30x^4 - 72x^2 + 216x + 18 = 192 > 0,$$

поэтому $x = 1$ – локальный минимизатор $\|f(x)\|_2^2$. Но поскольку $f(1) = 16$, точка $x = 1$ не является решением нелинейного уравнения, см. рис. 7.5. □

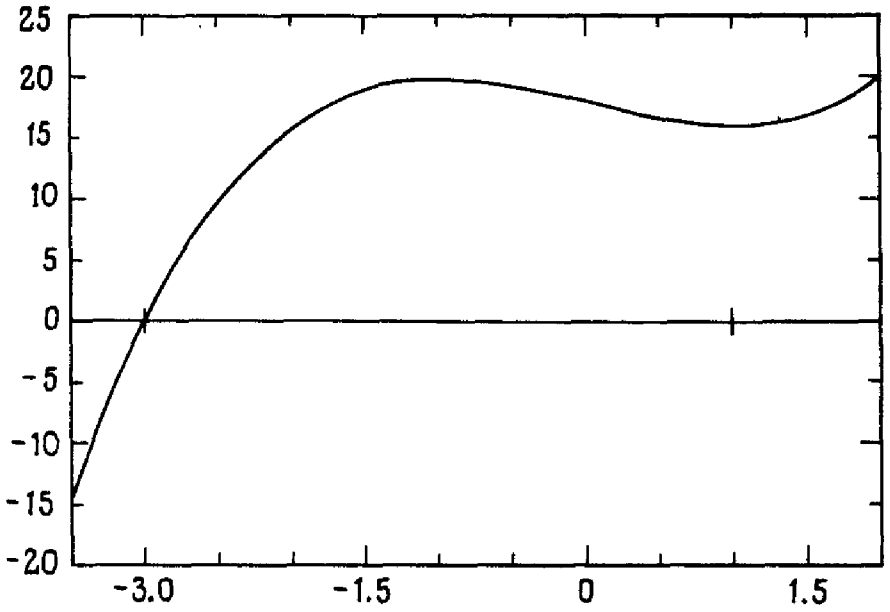


Рис. 7.5. Локальное решение нелинейного уравнения.

Чтобы избежать нахождения только локальных решений, можно использовать более сложные стратегии. Одна группа методов, которые называются *методами гомотопии*, описана в обзорной статье [Watson, 1986]. Соответствующие программы входят в пакет Hompack (см. список литературы в конце книги). Эти методы имеют большую вычислительную сложность, чем описанная выше стратегия, поскольку они требуют значительных вспомогательных вычислений, таких, как решение дифференциального уравнения или последовательности систем нелинейных уравнений. Однако они позволяют успешно решать трудные задачи, не поддающиеся решению более традиционными методами.

Еще один возможный подход основан на интервальном анализе (см. § 3 гл. 2) и обсуждается в книге [Hansen, 1969].

7.6. Подпрограмма SNSQE

Подпрограмма SNSQE разработана для решения систем нелинейных уравнений. Ее можно использовать и для решения одного уравнения с одним неизвестным, но более эффективной в этом случае будет подпрограмма FZERO. Заметим, что для FZERO требуется, чтобы

пользователь указал интервал, содержащий нуль функции. Подпрограмма SNSQE построена на основе идей предыдущего параграфа; более точное описание алгоритма можно найти в отчете [More et al., 1980]. Как для FZERO, так и для SNSQE пользователь должен написать подпрограмму вычисления значений нелинейных функций.

Дать гарантию, что подпрограмма SNSQE найдет решение системы нелинейных уравнений, нельзя. Возможно, она найдет лишь точку локального минимума функционала $\| f(x) \|_2^2$; эту точку иногда называют локальным решением уравнений. Переменная INFO используется для указания на нарушение сходимости процесса.

В приведенном ниже примере требуется решить систему двух уравнений с двумя неизвестными

$$f_1(x_1, x_2) = x_1 x_2 - x_2^3 - 1,$$
$$f_2(x_1, x_2) = x_1^2 x_2 + x_2 - 5.$$

В качестве начального приближения к решению берется $x = (2, 3)^T$, а точным решением является $x^* = (2, 1)^T$.

На выходе программа дает следующий результат:

```
НАЧАЛЬНЫЕ ДАННЫЕ
     2.000000          3.000000
ОЦЕНКА РЕШЕНИЯ
     1.999997          1.000001
ЗНАЧЕНИЯ НЕЛИНЕЙНЫХ ФУНКЦИЙ
    -0.3457E-05       -0.7153E-05
```

Ниже приводится вызывающая программа. Массив W является рабочим вектором длины LW. Для задачи с n неизвестными рабочий вектор должен иметь длину не меньше, чем $n(3n + 13)/2$. Параметр TOL представляет собой допуск, задающий требуемую точность решения. Параметр INFO указывает, успешно или нет решена задача. Если INFO = 1, то задача решена. По поводу других значений INFO см. пролог к подпрограмме SNSQE.

```
C  ОСНОВНАЯ ПРОГРАММА ДЛЯ РЕШЕНИЯ СИСТЕМ НЕ-
   ЛИНЕЙНЫХ УРАВНЕНИЙ
C
      PARAMETER  (N = 2,  LW = 19)
      REAL       X(N), FVEC(N), W(LW), TOL
      EXTERNAL   F
C
C  ЗАДАНИЕ ПАРАМЕТРОВ ДЛЯ SNSQE
C
      TOL = 1.E-6
      X (1) = 2.0
      X (2) = 3.0
      WRITE  (*, 800)  X(1), X(2)
```

```
      IOPT = 2
      NPRINT = 0
C
C  РЕШЕНИЕ НЕЛИНЕЙНЫХ УРАВНЕНИЙ
C
      CALL SNSQE (F, F, IOPT, N, X, FVEC, TOL, NPRINT,
     INFO, W, LW)
C
C  ПЕЧАТЬ РЕЗУЛЬТАТОВ
C
      IF (INFO .NE. 1) WRITE (*, 810)  INFO
      WRITE (*, 820) X(1), X(2)
      WRITE (*, 830) FVEC(1), FVEC(2)
C
      STOP
800   FORMAT ('INITIAL GUESS', /, 2F12.6)
810   FORMAT ('INFO =', 13)
820   FORMAT ('ESTIMATE OF SOLUTION', /, 2F12.6)
830   FORMAT ('VALUES OF NONLINEAR FUNCTIONS', /, 2E12.4)
      END
C
C
      SUBROUTINE F (N, X, FVEC, IFLAG)
      REAL X(N), FVEC(N)
C
C  ВЫЧИСЛЕНИЕ НЕЛИНЕЙНЫХ ФУНКЦИЙ
C
      FVEC(1) = X(1)*X(2) - X(2)**3 - 1.0
      FVEC(2) = X(1)**2*X(2) + X(2) - 5.0
C
      RETURN
      END
```

Второй параметр SNSQE – это имя подпрограммы JAC, написанной пользователем для вычисления матрицы Якоби. Подпрограмма SNSQE работает лучше, если такая подпрограмма действительно имеется. Заметим, что при IOPT = 2 элементы матрицы Якоби аппроксимируются разностными отношениями, и поэтому подпрограмма JAC может отсутствовать. В этом случае мы предполагаем, что формальное имя, задаваемое в качестве первого аргумента, также используется и для задания второго аргумента, как это сделано в приведенном выше примере. Для небольших задач, особенно если они должны решаться лишь один раз, мы рекомендуем использовать подпрограмму SNSQE с IOPT = 2. Значение IOPT = 1 используйте в том случае, когда вам необходимо повысить быстродействие подпрограммы. Написание подпрограммы для вычисления матрицы Якоби может потребовать много

времени и, кроме того, существует шанс внести алгебраическую ошибку в производные. Подпрограмму CHKDER можно использовать для проверки вашей подпрограммы вычисления производных; она сравнивает вашу «точную» матрицу Якоби с ее численной аппроксимацией и пытается определить, согласуются ли они. CHKDER используется независимо от SNSQE.

7.7. Задачи

7.1. При прокладывании водопроводных сетей коммунальные службы должны учитывать возможность промерзания. Несмотря на сложность структуры почвы и погодных условий, можно получить приемлемые приближения, основываясь на предположении, что почва является однородной во всех направлениях. В этом случае температура в градусах Цельсия $T(x, t)$ на глубине x (в метрах) через t секунд после начала резкого похолодания приближенно определяется по формуле

$$\frac{T(x, t) - T_s}{T_i - T_s} = \operatorname{erf}\left(\frac{x}{2\sqrt{\alpha t}}\right),$$

где T_s – постоянная температура на поверхности в течение холодного периода, T_i – начальная температура почвы перед похолоданием, α – коэффициент теплопроводности почвы (в $м^2/с$). Предположим, что $T_i = 20\,^\circ C$, $T_s = -15\,^\circ C$, $\alpha = 0.138 \cdot 10^{-6}$ $м^2/с$. Используйте подпрограмму FZERO, чтобы определить, как глубоко должен находиться водопровод, чтобы вода в нем замерзла только после шестидесятидневного воздействия этой постоянной поверхностной температуры. Напомним, что вода замерзает при температуре $0\,^\circ C$.

7.2. Цель этой задачи – напомнить вам о том, что точные решения иногда не столь полезны, как численные методы. Уравнение Кеплера для вычисления орбиты имеет вид

$$M = E - e \sin E.$$

Символы M, E и e обозначают среднюю аномалию, эксцентрическую аномалию и эксцентриситет орбиты соответственно.

(а) Если $M = 24.851090$ и $e = 0.1$, используйте FZERO для нахождения E.

(б) Величина E описывается формулой

$$E = M + 2 \sum_{m=1}^{\infty} \frac{1}{m} J_m(m e) \sin(m M),$$

где $J_m(x)$ – функция Бесселя 1-го рода порядка m. Воспользуйтесь этой формулой, чтобы определить E для значений из (а).

(в) Найдите E по формуле (б), используя равенство

$$J_m(m e) = \sum_{n=0}^{\infty} \frac{(-1)^n (m e/2)^{2n+m}}{n!\,(m+n)!}.$$

Проведите сравнение вычислительных затрат и точности результатов, полученных в (б) и (в), с результатами, полученными в (а).

7.3. Длинный проводящий стержень диаметром D метров с электрическим сопротивлением R на единицу длины помещен в большой замкнутый объем, чья граница (на большом расстоянии от стержня) поддерживается при температуре T_s° С. Стержень обтекается воздухом при температуре T_∞. Если через стержень проходит электрический ток I, то температура стержня со временем стабилизируется и становится равной T, где T удовлетворяет уравнению

$$Q(T) = \pi D h(T - T_\infty) + \pi D \varepsilon \sigma (T^4 - T_s^4) - I^2 R = 0 .$$

Здесь

σ $= 5.67 \cdot 10^{-8}$ ватт/м^{2}к^4 – константа Стефана–Больцмана.
ε $= 0.8$ – коэффициент поверхностной эмиссии стержня,
h $= 20$ ватт/м^{2}к – коэффициент теплопереноса воздушного потока,
T_∞ $= T_s = 25$ °С,
D $= 0.1$ м,
$I^2 R = 100$.

Воспользуйтесь FZERO для нахождения установившейся температуры T стержня.

7.4. Воспользуйтесь FZERO для нахождения десяти наименьших положительных значений x, для которых прямая $y = x$ пересекает график $y = \operatorname{tg} x$. Решение этой задачи имеет большое значение для определения максимального груза, который может нести стержень без изменения формы. См., например, [Timoshenko, 1956], с. 154.

7.5. Найдите вещественные корни уравнения

$$x^4 - 3x^2 + 75x - 10\,000 = 0 .$$

7.6. Определите, что дает на выходе следующая программа. Объясните, почему.

```
C ОСНОВНАЯ ПРОГРАММА
C
      REAL B, C, AE, RE, F
      EXTERNAL F
C
      B  = 0.0
      C  = 1.0
      AE = 1.E-6
      RE = 1.E-6
C
      CALL FZERO (F, B, C, B, RE, AE, IFLAG)
C
      WRITE (*, *) 'ESTIMATE OF ZERO =', B
      STOP
      END
```

```
C
C ПОДПРОГРАММА-ФУНКЦИЯ
C
      FUNCTION F(X)
      REAL X, Y
C
      IF (X .EQ. 0.) Y = - 1.0
      IF (X .GT. 0.) Y =   9.0
      WRITE (*, *) X, Y
      F = Y
C
      RETURN
      END
```

7.7. Решите следующую систему уравнений относительно x_1, x_2, x_3:

$$(1 - s_1)^2 + (1.5 - s_2)^2 + (c_1 - c_2)^2 - (x_3 + 0.25)^2 = 0,$$
$$(x_3 + 0.25)s_1 + 2x_3(c_2 s_1 - c_1) = 0,$$
$$(x_3 + 0.25)s_2 + 2x_3(c_1 s_2 - 1.5c_2) = 0,$$

где s_i и c_i обозначает $\sin(x_i)$ и $\cos(x_i)$. Эти уравнения получаются при решении следующей задачи. Стальная балка единичной длины и массы закреплена на шарнире в точке $u = 0$, $v = 1.5$, $w = 0$ таким образом, что она может свободно колебаться в плоскости $u\,w$. Другая такая же балка закреплена в точке $u = 1$, $v = 0$, $w = 0$ и может колебаться в плоскости $v\,w$. Свободные концы балок соединены пружиной, длина которой в ненагруженном состоянии равна 0.25, а коэффициент упругости равен 1. Когда балки перестают колебаться, угол между первой балкой и плоскостью $v\,w$ составляет x_1, угол между второй балкой и плоскостью $u\,w$ составляет x_2, а расстояние между свободными концами балок (вдоль пружины) составляет $x_3 + 0.25$. Решите эту систему, используя подпрограмму SNSQE и предполагая, что $0 \leqslant x_1, x_2 \leqslant \pi/2$ и $x_3 > 0$. (Ответ: $x_1 = 35.55°$, $x_2 = 44.91°$, $x_3 = 0.65$.)

7.8. Суперкомпьютер Gray 1 не имеет специального устройства для деления чисел. Чтобы вычислить $1/R$, $R > 0$, он формирует аппроксимацию обратного числа, точную примерно в половине слова с плавающей точкой, а затем использует ее в качестве начального приближения для одной ньютоновской итерации. Пусть $f(x) = (1/x) - R$. Покажите, что формула метода Ньютона имеет вид

$$x_{k+1} = x_k(2 - R\,x_k).$$

(а) Пользуясь этой формулой, вычислите $1/R$ для $R = 1, 2, \ldots, 10$. Число итераций, необходимое для получения результата с шестью верными знаками после запятой, занесите в таблицу. Положите $x_0 = 0.01$.

(б) Используя ту же функцию $f(x)$, повторите вычисления, но уже

методом бисекции, и сравните объем работы с затратами в (а). Возьмите начальный интервал [0.01, 2].

7.9. Примените метод Ньютона к уравнению $f(x) = (x-1)^2$, которое имеет нуль кратности два. Покажите, что для этого уравнения итерации сходятся линейно, а не квадратично. Проверьте численно, что для кратного корня сходимость будет линейной, решая уравнение $f(x) = \exp(2x) - 2\exp(x) + 1 = 0$ и печатая погрешность на каждой итерации.

7.10. Мы использовали метод Ньютона для поиска вещественных решений нелинейных уравнений. Этот метод также можно применять для нахождения комплексных корней, просто выбирая начальное приближение x_0 комплексным. Воспользуйтесь методом Ньютона для вычисления комплексного корня в нашем историческом примере

$$x^3 - 2x - 5 = 0.$$

7.11. Одним из классических методов решения кубического уравнения является формула Кардано. Кубическое уравнение $x^3 + ax^2 + bx + c = 0$ преобразуется к «неполному» виду

$$y^3 + py + q = 0$$

с помощью замены $x = y - a/3$. Коэффициенты в неполной форме уравнения имеют вид

$$p = b - \frac{a^2}{3}, \quad q = c - \frac{ab}{3} + 2\left(\frac{a}{3}\right)^3.$$

Один вещественный корень неполного уравнения можно найти так:

$$s = \left[\left(\frac{p}{3}\right)^3 + \left(\frac{q}{2}\right)^2\right]^{1/2}, \quad y_1 = \left[-\frac{q}{2} + s\right]^{1/3} + \left[-\frac{q}{2} - s\right]^{1/3},$$

и тогда вещественный корень исходного уравнения определяется как

$$x_1 = y_1 - \frac{a}{3}.$$

Два других корня можно либо найти аналогичным способом, либо разделить уравнение на $x - x_1$ и решить полученное квадратное уравнение.

(а) Воспользуйтесь методом Кардано для вычисления вещественного корня уравнения

$$x^3 + 3x^2 + \alpha^2 x + 3\alpha^2 = 0$$

при различных значениях α. Исследуйте потерю точности из-за ошибок округления для больших α, скажем, для α порядка величины, обратной машинному эпсилону.

(б) Примените метод Ньютона к этому же уравнению с теми же значениями α. Исследуйте эффект ошибок округления и выбора начального значения.

7.12. В разд. 2.1 гл. 5 двухточечная квадратурная формула Гаусса была получена с помощью решения следующей системы нелинейных уравнений:

$$\begin{aligned}(b-a) &= w_1 + w_2,\\ 0 &= w_1(x_1 - m) + w_2(x_2 - m),\\ (b-a)^3/12 &= w_1(x_1 - m)^2 + y_2(x_2 - m)^2,\\ 0 &= w_1(x_1 - m)^3 + w_2(x_2 - m)^3,\end{aligned}$$

где $m = (a + b)/2$. Положите $[a, b] = [-1, 1]$ и решите эту систему с помощью SNSQE. Сравните ваши результаты с правильным решением

$$x_1 = 1/\sqrt{3},\quad x_2 = -1/\sqrt{3},\quad w_1 = w_2 = 1.$$

7.8. Прологи: FZERO и SNSQE

```
      SUBROUTINE FZERO (F, B, C, R, RE, AE, IFLAG)
C*** НАЧАЛО ПРОЛОГА  FZERO
C*** ДАТА НАПИСАНИЯ  700901  (ГГММДД)
C*** ДАТА ПЕРЕСМОТРА  860411  (ГГММДД)
C*** КАТЕГОРИЯ N0.       F1B
C*** КЛЮЧЕВЫЕ СЛОВА: Бисекция, нелинейная, корни, нули
C*** АВТОРЫ: SHAMPINE L. F., SNLA,
C              WATTS H. A., SNLA
C*** НАЗНАЧЕНИЕ: FZERO ищет нуль функции F(X) на заданном
C                интервале (B, C). Программа предназначена
C                главным образом для задач, в которых F(B) и
C                F(C) имеют противоположные знаки.
C*** ОПИСАНИЕ
C
C    Из книги D. Kahaner, C. Moler, S. Nash
C             "Numerical Methods and Software"
C             Prentice-Hall, 1988
C
C    Основана на методе, предложенном Деккером.
C    Написана Шэмпайном и Уотсом.
C
C       FZERO ищет нуль функции F(X) между заданными значения-
C       ми B и C до тех пор, пока ширина интервала не станет меньше
C       допуска, определяемого критерием останова,
C       ABS(B - C) .LE. 2.*(RW*ABS(B) + AE). Используемый метод
C       является эффективной комбинацией бисекции и метода секу-
C       щих.
C
C    Описание параметров:
C       F, B, C, R, RE и AE - входные параметры.
C
C       B, C и IFLAG - выходные параметры (по-
C                      меченные ниже символом *).
```

C
C F – Имя вещественнозначной внешней функции. Это имя должно быть указано в операторе EXTI вызывающей программе. F должна быть функцией одного вещественного аргумента.
C
C *B – Один из концов интервала (B, C). На выходе содержит обычно лучшее приближение к нулю функции F.
C
C *C – Другой конец интервала (B, C).
C
C R – (Лучшее) приближение нуля функции F, которое может помочь в ускорении сходимости. Если F(B) и F(R) имеют противоположные знаки, то нуль будет искаться на интервале (B, R); если нет, но F(R) и F(C) имеют противоположные знаки, то нуль ищется на интервале (R, C); в противном случае для поиска возможного нуля анализируется интервал (B, C). Если лучшее приближение не известно, то рекомендуется положить R равным B или C, поскольку в случае, если значение R не принадлежит интервалу (B, C), оно игнорируется.
C
C RE – Относительная погрешность, используемая для RW в критерии останова. Если требуемое значение RE меньше машинной точности, то RW полагается приближенно равным машинной точности.
C AE – Абсолютная погрешность, используемая в критерии останова. Если заданный интервал (B, C) содержит начало координат, то для AE должно быть задано ненулевое значение.
C
C *IFLAG – Код статуса вычислений. После каждого обращения к подпрограмме пользователь должен проверить значение IFLAG. Во всех случаях управление возвращается из FZERO в программу пользователя.
C
C 1 B с требуемой точностью является нулем функции. Интервал (B, C) сжат до требуемого допуска, функция меняет знак на (B, C) и величина F(X) уменьшается при сжатии (B, C).
C
C 2 F(B) = 0. Однако интервал (B, C), возможно, еще не сжат до требуемого допуска.
C
C 3 B может находиться в окрестности сингулярной точки функции F(X). Интервал (B, C) сжат до

```
C                  требуемого допуска, функция меняет знак на
C                  (B, C), но величина F(X) увеличилась при сжатии
C                  (B, C), т.е. abs(F(B out)) .GT. max(abs(F(B in)),
C                  abs(F(C in))).
C
C                4 Хотя интервал (B, C) сжат до требуемого допуска,
C                  изменения знака у F(X) не обнаружено. Пользова-
C                  тель должен проанализировать этот случай и ре-
C                  шить, находится ли B вблизи точки локального
C                  минимума функции F(X) или нуля четной крат-
C                  ности, или же ни один из этих вариантов не имеет
C                  места.
C
C                5 Использовано слишком много ( .GT. 500) вычисле-
C                  ний значений функции.
C
C*** ССЫЛКИ: 1. L.F. SHAMPINE, H.A. WATTS, FZERO, A ROOT-
C                SOLVING CODE, SC-TM-70-631, SEPTEMBER
C                1970.
C              2. T.J. DEKKER, FINDING A ZERO BY MEANS OF
C                SUCCESSIVE LINEAR INTERPOLATION, CON-
C                STRUCTIVE ASPECTS OF THE FUNDAMENTAL
C                THEOREM OF ALGEBRA EDITED BY B. DEJON,
C                P. HENRICI, 1969.
C
C*** ВЫЗЫВАЕМЫЕ ПОДПРОГРАММЫ: R1MACH
C*** КОНЕЦ ПРОЛОГА FZERO

      SUBROUTINE SNSQE (FCN, JAC, IOPT, N, X, FVEC, TOL,
     NPRINT, INFO, WA, LWA)
C*** НАЧАЛО ПРОЛОГА     SNSQE
C*** ДАТА НАПИСАНИЯ     800301  (ГГММДД)
C*** ДАТА ПЕРЕСМОТРА    840405  (ГГММДД)
C*** КАТЕГОРИЯ N0.      F2A
C*** КЛЮЧЕВЫЕ СЛОВА: простота использования, нелинейная
C                квадратная система, гибридный метод
C                Пауэлла, нуль.
C*** АВТОР: HIEBERT K.L. (SNLA)
C*** НАЗНАЧЕНИЕ: SNSQE – простая в использовании версия под-
C                программы SNSQ, которая находит нуль систе-
C                мы из N нелинейных функций с N переменными
C                при помощи модификации гибридного метода
C                Пауэлла. Эта подпрограмма является комбина-
C                цией подпрограмм HYBRD1 и HYBRJ1 пакета
C                MINPACK (Аргоннская национальная лабора-
C                тория).
```

```
C
C*** ОПИСАНИЕ
C     Из книги D. Kahaner, C. Moler, S. Nash
C              "Numerical Methods and Software"
C              Prentice-Hall, 1988
C
C
C
C
C
C     SNSQE предназначена для нахождения нуля системы N нелинейных функций с N переменными при помощи модификации гибридного метода Пауэлла. Для этого используется более общая программа SNSQ решения нелинейного уравнения. Пользователь должен написать программу, которая вычисляет значения функций. У пользователя есть выбор – самому написать подпрограмму, вычисляющую матрицу Якоби, или позволить программе SNSQE вычислять ее, используя разностную аппроксимацию вперед. Эта подпрограмма является комбинацией подпрограмм HYBRD1 и HYBRJ1 из пакета MINPACK (Аргоннская национальная лаборатория).
C
C
C     Подпрограмма и операторы описания типа
C
C        SUBROUTINE SNSQE (FCN, JAC, IOPT, N, X, FVEC, TOL,
C        IPRINT, INFO, WA, LWA)
C        INTEGER IOPT, N, NPRINT, INFO, LWA
C        REAL TOL
C        REAL X(N), FVEC(N), WA(LWA)
C        EXTERNAL FCN, JAC
C
C
C     Описание параметров:
C
C        Параметры, указанные в качестве входных, должны быть определены на входе в SNSQE и не меняются на выходе, тогда как параметры, указанные в качестве выходных, не нуждаются в определении на входе и получают соответствующие значения на выходе из SNSQE.
C
C        FCN  – имя подпрограммы, добавленной пользователем, которая вычисляет функции. FCN должна быть указана в операторе EXTERNAL в вызывающей программе пользователя и оформлена следующим образом.
```

```
C
C                SUBROUTINE FCN(N, X, FVEC, IFLAG)
C                INTEGER N, IFLAG
C                REAL X(N), FVEC(N)
C                ----------
C                Вычислить значение функций в точке X и поме-
C                стить этот вектор в массив FVEC.
C                ----------
C                RETURN
C                END
C
C           Значение IFLAG не должно изменяться подпрограм-
C           мой FCN, если только пользователь не хочет окон-
C           чить выполнение подпрограммы SNSQE. В этом
C           случае положите IFLAG равным отрицательному
C           целому числу.
C
C      JAC  -имя подпрограммы, добавленной пользователем,
C           которая вычисляет матрицу Якоби. Если IOPT = 1,
C           то JAC должна быть указана в операторе EXTERNAL
C           в вызывающей программе пользователя и оформле-
C           на следующим образом.
C
C                SUBROUTINE JAC(N, X, FVEC, FJAC, LDFJAC,
C                IFLAG)
C                INTEGER N, LDFJAC, IFLAG
C                REAL X(N), FVEC(N), FJAC(LDFJAC, N)
C                ----------
C                Вычислить матрицу Якоби в точке X и поместить
C                ее в FJAC. Массив FVEC содержит значения
C                функции в точке X и не должен меняться.
C                ----------
C                RETURN
C                END
C
C           Значение IFLAG не должно изменяться подпро-
C           граммой JAC, если только пользователь не хочет
C           окончить выполнение подпрограммы SNSQE. В
C           этом случае положите IFLAG равным отрицатель-
C           ному числу.
C           Если IOPT = 2, то параметр JAC может быть про-
C           игнорирован (трактуется как фиктивный аргумент).
C
C      IOPT -входная переменная, которая определяет способ
C           вычисления матрицы Якоби. Если IOPT = 1, то
C           пользователь должен генерировать матрицу Якоби
C           при помощи подпрограммы JAC. Если IOPT = 2,
```

```
C          то программа будет аппроксимировать матрицу
C          Якоби, используя разностное дифференцирование
C          вперед.
C
C     N    - целая положительная входная переменная, равная
C          числу функций и переменных.
C
C     X    - массив длины N. На входе X должен содержать
C          начальное приближение к вектору решения. На
C          выходе X содержит конечное приближение к векто-
C          ру решения.
C
C     FVEC - выходной массив длины N, который содержит зна-
C          чения функций, вычисленные в выходной точке X.
C
C     TOL  - неотрицательная входная переменная. Процесс
C          оканчивается, когда алгоритм находит, что относи-
C          тельная погрешность между X и решением не пре-
C          восходит TOL. Раздел 4 содержит более подробную
C          информацию относительно TOL.
C
C     NPRINT - целая входная переменная, которая позволяет управ-
C          лять печатью итераций, если ее значение положи-
C          тельно. В этом случае FCN вызывается с IFLAG = 0
C          в начале первой итерации, а затем каждые NPRINT
C          итераций и, кроме того, непосредственно перед вы-
C          ходом; при этом для печати доступны X и FVEC.
C          Соответствующие операторы вывода должны быть
C          добавлены к FCN (см. пример). Если NPRINT не
C          положительно, то никакого специального вызова
C          FCN с IFLAG = 0 не делается.
C
C     INFO - целая выходная переменная. Если пользователь до-
C          срочно окончил выполнение подпрограммы, INFO
C          полагается равным (отрицательному) значению
C          IFLAG. См. описание подпрограмм FCN и JAC.
C          Иначе значение INFO определяется следующим
C          образом.
C
C               INFO = 0 - неподходящие входные параметры.
C
C               INFO = 1 - по оценке подпрограммы относительная
C                          погрешность между X и решением не
C                          превосходит TOL.
C
C               INFO = 2 - число обращений к FCN достигло или
```

```
C                                превысило 100*(N + 1) для IOPT = 1
C                                или 200*(N + 1) для IOPT = 2.
C
C                     INFO = 3 - TOL слишком мало. Дальнейшее улуч-
C                                шение приближенного решения X не-
C                                возможно.
C
C                     INFO = 4 - итерации не дают хорошего улучшения.
C
C                     Разделы 4 и 5 содержат дополнительную информа-
C                     цию относительно INFO.
C
C          WA         - рабочий массив длины LWA.
C
C          LWA        - целая положительная входная переменная, не мень-
C                       шая чем (3*N**2 + 13*N)/2.
C
C
C     Успешное окончание
C
C          Точность SNSQE контролируется при помощи параметра схо-
C          димости TOL. Этот параметр используется в тесте, осуществля-
C          ющем сравнение приближенного решения X с решением XSOL.
C          Выполнение SNSQE заканчивается, если сравнение оказалось
C          успешным. Если TOL меньше машинной точности (определяе-
C          мой при помощи функции R1MACH(4)), то SNSQE проводит
C          сравнение только с машинной точностью. Дальнейший про-
C          гресс обычно невозможен. Если не требуется очень большая
C          точность, то рекомендуемое значение TOL равно корню квад-
C          ратному из машинной точности.
C
C          Тест предполагает, что функции ведут себя достаточно хорошо,
C          и если матрица Якоби вычисляется пользователем, то функции
C          и матрица Якоби программируются согласованно. Если эти
C          условия не выполнены, SNSQE может некорректно указывать
C          на сходимость. Программирование вычисления матрицы Якоби
C          можно проверить при помощи подпрограммы CHKDER. Если
C          матрица Якоби запрограммирована правильно или IOPT = 2,
C          тогда правильность результата можно проверить, например,
C          повторным запуском SNSQE с более жестким допуском.
C
C          Тест на сходимость. Если ENORM(Z) обозначает евклидову
C          норму вектора Z, то тест пытается гарантировать, что
C
C               ENORM(X - XSOL)  .LE.  TOL*ENORM(XSOL).
```

```
C
C     Если это условие выполнено с TOL = 10**(-K), то наибольшая
C     компонента X имеет K верных десятичных цифр и INFO
C     полагается равным 1. Существует опасность, что меньшие
C     компоненты X могут иметь бóльшую относительную погреш-
C     ность, но быстрая сходимость SNSQE обычно исключает эту
C     возможность.
C
C
C  Неуспешное окончание
C
C     Неуспешное окончание может быть следствием неадекватного
C     задания входных параметров, арифметических прерываний,
C     чрезмерного числа вычислений функций, ошибок в функциях
C     или отсутствия прогресса.
C
C     Несоответствующие входные параметры. INFO полагается рав-
C     ным 0, если или IOPT .LE. 1 или IOPT .GT. 2, или N .LE. 0,
C     или TOL .LT. 0.E0, или LWA .LT. (3*N**2 + 13*N)/2.
C
C     Арифметические прерывания. Если эти прерывания происходят
C     в подпрограмме FCN на ранних шагах вычислений, то они
C     могли быть вызваны неприемлемым выбором X в подпро-
C     грамме SNSQE. В этом случае можно исправить ситуацию,
C     не вычисляя функции здесь, а полагая компоненты FVEC
C     равными числам, которые превосходят числа в начальном
C     FVEC.
C
C     Чрезмерное число вычислений функций. Если число обращений
C     к FCN достигает 100*(N + 1) для IOPT = 1 или 200*(N + 1)
C     для IOPT = 2, то это указывает, что итерации сходятся очень
C     медленно, если оценивать сходимость уменьшением нормы
C     вектора FVEC, и INFO полагается равным 2. Эта ситуация
C     должна встречаться нечасто, поскольку, как указано ниже,
C     отсутствие прогресса обычно диагностируется подпрограм-
C     мой SNSQE раньше и вызывает останов с INFO = 4.
C
C     Ошибки в функциях. При IOPT = 2 выбор длины шага в конеч-
C     но-разностной аппроксимации вперед матрицы Якоби пред-
C     полагает, что относительные ошибки в функциях имеют
C     порядок машинной точности. Если это не так, то работа
C     SNSQE может окончиться неудачей (обычно с INFO = 4).
C     В этом случае пользователь должен либо использовать
C     подпрограмму SNSQ и задать длину шага, либо использо-
C     вать IOPT = 1 и добавить подпрограмму вычисления мат-
C     рицы Якоби.
```

```
C
C        Отсутствие прогресса. SNSQE ищет нуль системы, минимизи-
C          руя сумму квадратов функций. Поступая таким образом,
C          программа может попасть в ловушку в области, где мини-
C          мум не соответствует нулю системы, и в этой ситуации со
C          временем прекратится заметное продвижение в смысле век-
C          тора FVEC. В частности, это случится, если система не имеет
C          нуля. Если система имеет нуль, то может быть полезен
C          повторный запуск SNSQE с другой стартовой точкой.
C
C
C     Характеристики алгоритма
C
C        SNSQE является модификацией гибридного метода Пауэлла.
C        Две его основные характеристики – это выбор коррекции как
C        выпуклой комбинации ньютоновского направления и направле-
C        ния масштабированного градиента и пересчет матрицы Якоби
C        при помощи однорангового метода Бройдена. Выбор коррек-
C        ции гарантирует (при разумных предположениях) глобальную
C        сходимость для стартовых точек, расположенных далеко от
C        решения, и быструю сходимость. Матрица Якоби вычисляется
C        в стартовой точке при помощи либо подпрограммы пользова-
C        теля, либо аппроксимации разностями вперед, но она не
C        пересчитывается до тех пор, пока одноранговый метод не
C        перестанет давать удовлетворительного прогресса.
C
C        Временны́е характеристики. Время, требуемое SNSQE для
C          решения данной проблемы, зависит от N, поведения функ-
C          ций, требуемой точности и стартовой точки. Число арифме-
C          тических операций, необходимое для SNSQE, составляет
C          около 11,5*(N**2) на выполнение каждого вычисления
C          функций (вызов FCN) и 1,3*(N**3) на выполнение каждого
C          вычисления матрицы Якоби (вызов JAC, если IOPT = 1).
C          Если FCN и JAC не могут быть выполнены быстро, время
C          SNSQE будет в большей мере определяться временем, затра-
C          ченным в FCN и JAC.
C
C        Память. SNSQE требует (3*N**2 + 17*N)/2 слов памяти
C          одинарной точности в дополнение к памяти, требуемой для
C          программы. В программе нет внутренних описаний мас-
C          сивов.
C
C*** ССЫЛКИ: POWELL M.J.D.,
C                 A HYBRID METHOD FOR NONLINEAR EQUATI-
C                 ONS, NUMERICAL METHODS FOR NONLINEAR
C                 ALGEBRAIC EQUATIONS, P. RABINOWITZ (EDI-
```

```
C                    TOR), GORDON AND BREACH, 1970.
C*** ВЫЗЫВАЕМЫЕ ПРОГРАММЫ: SNSQ, XERROR
C*** КОНЕЦ ПРОЛОГА     SNSQE
```

Глава 8

Обыкновенные дифференциальные уравнения

8.1. Введение

Дифференциальное уравнение описывает, как изменяется функция. Это позволяет нам моделировать движения и процессы, непрерывно меняющиеся во времени. В качестве простого примера допустим, что с крыши здания брошен мяч. Тогда высота, на которой находится мяч, удовлетворяет дифференциальному уравнению

$$\frac{d}{dt}\,\text{высота}\,(t) = -g\,t,$$

где g – ускорение свободного падения, а переменная t представляет собой время, отсчитываемое от начала падения мяча. Это уравнение можно решить, интегрируя его правую часть, и получить

$$\text{высота}\,(t) = \text{высота}\,(0) - \frac{1}{2}\,g\,t^2.$$

Отметим несколько обстоятельств. Во-первых, решение зависит от высоты здания, т.е. от величины высота (0), и, следовательно, не определяется однозначно дифференциальным уравнением. Во-вторых, решением является не число, а функция.

Математические модели, основанные на дифференциальных уравнениях, широко применяются во многих научных дисциплинах для описания окружающего нас мира. От анализа химических реакций на атомарном уровне до глобального предсказания погоды, изучения комет, планет и галактик – всюду дифференциальные модели играют важную роль. Даже системы, которые по своей природе дискретны, такие, как численность кроликов в биологической экосистеме, часто хорошо аппроксимируются дифференциальными уравнениями. В этом параграфе вводятся некоторые основные идеи, необходимые для понимания обыкновенных дифференциальных уравнений. В последующих параграфах мы вернемся к каждой из этих идей и рассмотрим их более подробно.

Обыкновенное дифференциальное уравнение (*ОДУ*) – это уравнение, содержащее функцию и ее производные. Простейшее ОДУ имеет вид

$$\frac{dy}{dt} = f(t).$$

В примере с мячом y – это высота, а $f(t) = -g\,t$. Пусть нас интересует

решение начиная с некоторого $t = t_0$. Если функция $f(t)$ непрерывна на отрезке $[t_0, T]$, то решение рассматриваемого уравнения записывается в виде

$$y(t) = c + \int_{t_0}^{t} f(\tau)\, d\tau, \quad t_0 \leqslant t \leqslant T.$$

функция $y(t)$ определена с точностью до константы. В нашем примере константа c равна начальной высоте, т. е. величине высота(0). Таким образом, наше ОДУ имеет семейство решений, каждое из которых отличается от другого на аддитивную константу. Вид данного уравнения таков, что в фиксированной точке t все кривые из семейства решений имеют одинаковый наклон $\frac{dy}{dt}$. Чтобы выделить единственную кривую, возьмем то решение, которое принимает заданное значение A в точке $t = t_0$. Величина A определяется конкретной постановкой задачи; в нашем примере она выражает высоту здания.

Простейшее уравнение иллюстрирует тесную связь между задачами интегрирования и решения дифференциальных уравнений. Однако есть и определенные различия. Одно из важнейших различий в том, что при интегрировании результатом является единственное число — величина интеграла. Здесь же ищется функция, определяемая на отрезке. Иногда решение дифференциального уравнения требуется определить только в одной точке, например узнать высоту мяча на третьей секунде падения. Тогда, как и при интегрировании, результатом будет число. Однако гораздо чаще ищется решение на отрезке. Так, если мы хотим выразить зависимость высоты мяча от времени, то лучше искать решение в виде функции. Явную формулу для решения удается выписать только в простейших случаях, а в общем случае это представляется невозможным или слишком сложным. Другое различие между обыкновенными дифференциальными уравнениями и интегралами заключается в том, что первые одномерны, т. е. зависят от одного независимого переменного, а интегралы часто бывают многомерными [1]).

Поскольку решением ОДУ является функция, следует прийти к соглашению, какой смысл мы будем вкладывать в выражение «решить уравнение на компьютере». Общепринятый подход заключается в *дискретизации* задачи. Это значит, что решение $y(t)$ вычисляется только в конечном числе *дискретных* точек (узлов сетки), а не на всем *непрерывном* отрезке. Например, если $[t_0, T] = [0, 1]$, то решение можно вычислить в узлах 0.0, 0.01, 0.02, ..., 0.99, 1.0. Значение в какой-либо промежуточной точке можно получить интерполированием (см. гл. 4). Узлы сетки не обязательно размещать равномерно. Бывает, что узлы задаются заранее, поскольку требуется найти решение в определенных

[1] Многомерные интегралы можно было бы сравнить с дифференциальными уравнениями в частных производных.– *Прим. перев.*

точках или построить график или таблицу по специальным наборам узлов. Однако лучше позволить программе определить узлы самостоятельно, поскольку, учитывая информацию о конкретном дифференциальном уравнении, удается вычислить приближенное решение точнее и эффективнее. Например, если решение сильно меняется в некоторой подобласти, то необходимо разместить в такой подобласти много узлов сетки, чтобы отследить изменение решения. В этом отношении задачи численного решения ОДУ и интегрирования схожи.

Существует несколько источников возникновения ошибок при решении дифференциальных уравнений. Так, ошибки появляются при дискретизации дифференциальной задачи, подобно случаю разностной аппроксимации производной (см. пример 2 в гл. 2). Во многих, но, к сожалению, не во всех случаях повторное решение задачи на более частой сетке позволяет добиться большей точности. Однако иногда малые ошибки, внесенные в начале вычислений, могут совершенно исказить решение, если только не подобрать подходящий численный метод. Это явление иногда называется «неустойчивостью». Далее, вычислениям, как правило, сопутствуют ошибки округления. В численных методах приходится идти на компромисс, сочетая стремление уменьшить отмеченные виды ошибок с необходимостью решить задачу с меньшими вычислительными затратами.

Часто возникает потребность в решении систем обыкновенных дифференциальных уравнений, содержащих несколько неизвестных функций. Многие методы решения одного уравнения можно применить и к системе уравнений, хотя реализация такого алгоритма может оказаться утомительной и труднее будет использовать геометрическую интуицию. Дифференциальные уравнения высокого порядка, содержащие вторую, третью и более высокие производные, можно записать в виде системы дифференциальных уравнений первого порядка (см. разд. 1.2), поэтому нет особых причин рассматривать уравнения порядка выше первого.

В том случае, когда дифференциальная задача описывает два взаимодействующих процесса, один из которых претерпевает быстрые изменения, а другой — медленные, могут возникнуть определенные затруднения. Примером может служить модель удара по барабану, в которой соседствуют быстрые колебания, отвечающие тону инструмента, и медленное ослабление силы звука. Чтобы отследить быстрые колебания, потребуется взять частую сетку, что приведет к большому объему вычислений. Однако уже через весьма малое время после удара вклад быстрых колебаний в решение уменьшается, поэтому для расчета силы звука можно было бы обойтись редкой сеткой. Задачи такого рода принято называть *жесткими*. Некоторые численные методы неприемлемы для решения жестких задач, ибо они неуклонно стремятся следовать за быстрыми движениями, даже если вклад последних не столь важен, как общий характер решения. Для решения жестких задач разработаны специальные алгоритмы; некоторые из них мы рассмотрим ниже.

В этой книге мы обсудим только начальные задачи для обыкновен-

ных дифференциальных уравнений. Под начальной задачей (НЗ) мы подразумеваем дифференциальное уравнение, решение которого в некоторой точке (обозначеной выше через t_0) дано, а при t, бóльших или меньших, чем t_0, его требуется найти. Есть и другие типы задач для ОДУ. Некоторые из них перечислены в § 20.

Численные методы решения начальных задач стартуют из начальной точки и стараются отследить или «учуять» кривую решения. Часто они называются *маршевыми* методами. Чтобы решить задачу, в любом методе приходится вычислять значения правой части уравнения f. Если в какой-либо точке f обращается в бесконечность, то метод может не сработать. Такое может случиться, например, в окрестности точки $t = 0$ при решении задачи

$$y'(t) = \frac{1}{\sin\sqrt{|t|}}, \quad y(-1) = 0.$$

Тем не менее соответствующий интеграл

$$\int_{-1}^{2} \frac{dt}{\sin\sqrt{|t|}} = I = 5.3141\ldots$$

нетрудно вычислить с помощью современных программ интегрирования, таких, как Q1DA. В этом заключается еще одно различие между дифференциальными уравнениями и интегралами. Задача 8.1 дает хорошую возможность продолжить сопоставление начальных задач и квадратур.

8.1.1. Основная терминология

Как отмечено выше, простейшее обыкновенное дифференциальное уравнение записывается в виде

$$\frac{dy}{dt} = f(t).$$

Это ОДУ *первого порядка*, поскольку наивысший порядок производных, содержащихся в уравнении, равен единице. Решение этого уравнения можно найти, интегрируя правую часть, и все решения будут различаться на аддитивную константу. Рассмотрим ОДУ первого порядка более общего вида

$$\frac{dy}{dt} = f(t, y).$$

Это уравнение также имеет семейство решений (при соответствующих предположениях об f), но отдельные решения различаются уже не на аддитивную константу, поскольку при фиксированном t каждое из

решений имеет свой наклон. Мы не будем рассматривать вопросы существования решения, которые подробно обсуждаются в книге [Shampine, Gordon, 1975][1, 2].

Пример 8.1. Семейства решений.

Будем обозначать производную $\frac{dy}{dt}$ через y'. Рассмотрим два разных уравнения

$$y' = e^{-t} \quad \text{и} \quad y' = -y,$$

которые обладают соответственно решениями

$$y(t) = c - e^{-t} \quad \text{и} \quad y(t) = ce^{-t}.$$

Семейства решений этих уравнений представлены на рис. 8.1. Отметим, что при фиксированном t производная y' зависит от y во втором случае и не зависит в первом. □

Независимая переменная t часто имеет смысл времени; при этом можно считать, что ОДУ описывает эволюцию физической системы. Начальное состояние такой системы обычно бывает известным, что и поясняет смысл термина «начальная задача». Если t_0 и A – заданные числа, то начальную задачу записывают в виде

$$y' = f(t, y), \quad y(t_0) = A.$$

Пара (t_0, A) называется *начальной точкой.*

Если t представляет собой время, то решение $y(t)$ требуется найти на фиксированном интервале $t_0 \leqslant t \leqslant T$, а если t выражает расстояние, то решение можно искать для $T \leqslant t \leqslant t_0$. И теория, и алгоритмы применимы в обоих случаях, но мы опишем подробно только первый случай.

Различие между семействами решений уравнений $y' = f(t)$ и $y' = f(t, y)$ показывает еще одно отличие дифференциальных уравнений от квадратур. При вычислении определенного интеграла, скажем, от 0 до 1 мы вольны вычислять значения подынтегральной функции в узлах квадратурной формулы на [0, 1] в любой последовательности. Действительно, адаптивный квадратурный алгоритм из гл. 5 выбирает подынтервалы, исходя из оценки погрешности, а не из положения в исходном интервале интегрирования. Это допустимо в случае квадратур, поскольку подынтегральную функцию всюду можно вычислять непосредственно. Но это

---

[1] Л. Шэмпайн (L. F. Shampine), ранее работавший в Лаборатории Сандия (Sandia Laboratories), а ныне преподающий в Южном методистском университете (Southern Methodist University), является одним из наиболее активных исследователей в области численных методов решения ОДУ. Его программы DEABM и DERKF считаются лучшими реализациями методов Адамса–Бэшфорта и Рунге–Кутты (см. § 16, 19) для решения нежестких задач.

[2] Читатель может обратиться также к книге [Хайрер Э., Нерсетт С., Ваннер Г. Решение обыкновенных дифференциальных уравнений. Нежесткие задачи.– М.: Мир, 1990].– *Прим. перев.*

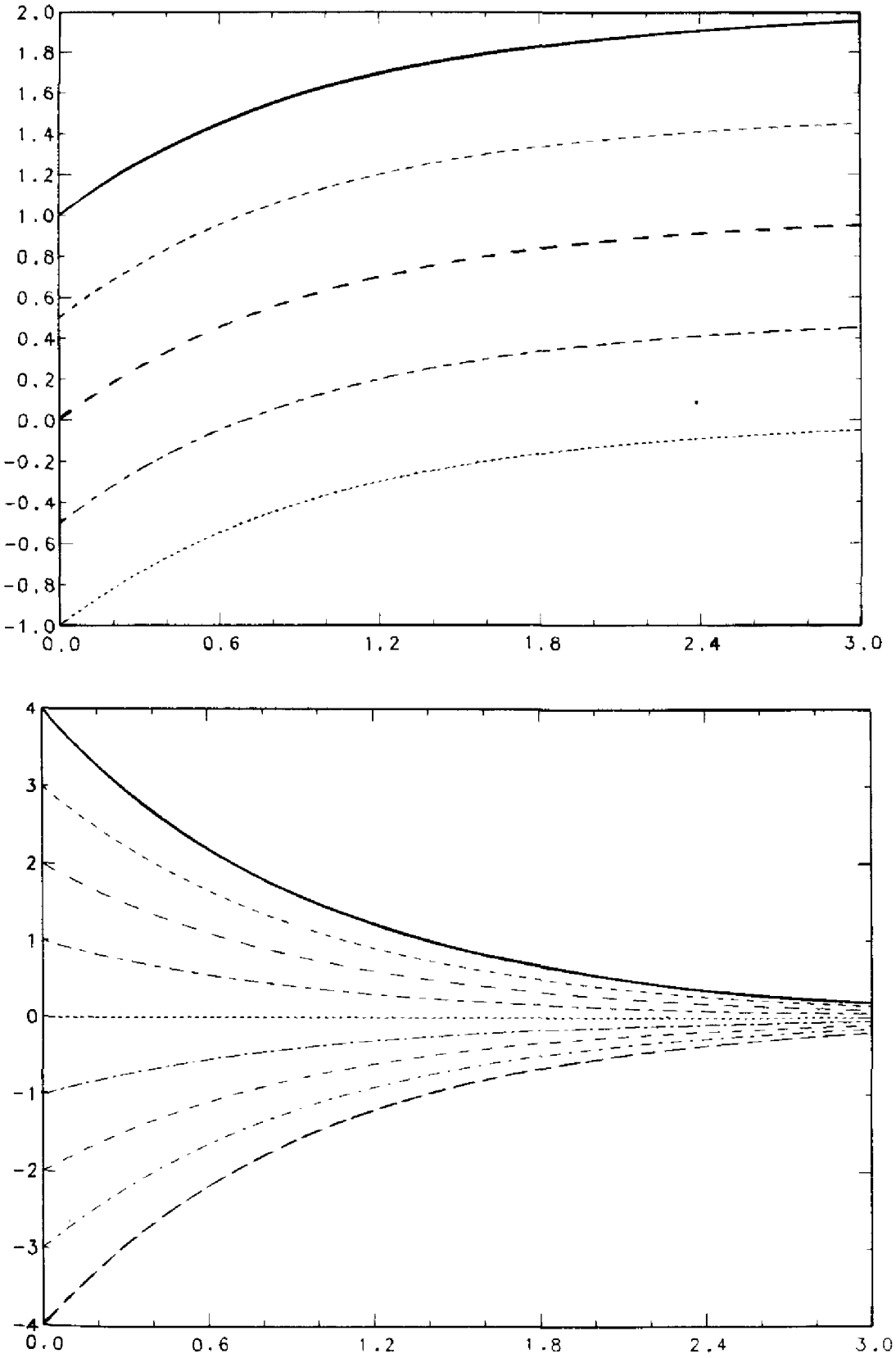


Рис. 8.1. Семейства решений.

невозможно для $y' = f(t, y)$: нельзя вычислить значение производной решения в точке $t = 0.5$ до тех пор, пока в этой точке не будет вычислено решение, так как $y'(0.5) = f(0.5, y(0.5))$.

8.1.2. Уравнения высокого порядка и системы уравнений

Большинство практических задач приводит не к одному ОДУ первого порядка. Системы уравнений первого порядка могут возникать непосредственно, если в задаче имеется более одного неизвестного. Например, простая модель «хищник — жертва», описывающая популяции кроликов $r(t)$ и лис $f(t)$, поедающих только кроликов, записывается в виде системы двух уравнений первого порядка (см. задачу 8.5)

$$r' = 2r - \alpha rf,$$
$$f' = -f + \alpha rf.$$

Взаимодействие этих двух популяций пропорционально произведению их численностей с коэффициентом пропорциональности α. Задача состоит в том, чтобы определить обе численности $r(t)$ и $f(t)$, исходя из их значений, известных в начальный момент времени t_0.

В более общей ситуации имеется n неизвестных функций $y_i(t)$, $i = 1, \ldots, n$, для которых выписано n дифференциальных уравнений

$$\begin{aligned} y_1' &= f_1(t, y_1, y_2, \ldots, y_n), \\ y_2' &= f_2(t, y_1, y_2, \ldots, y_n), \\ &\vdots \\ y_n' &= f_n(t, y_1, y_2, \ldots, y_n). \end{aligned}$$

В примере «хищник – жертва» $n = 2$, $y_1(t) = r(t)$, $y_2(t) = f(t)$ и

$$f_1(t, y_1, y_2) = 2y_1 - \alpha y_1 y_2, \quad f_2(t, y_1, y_2) = -y_2 + \alpha y_1 y_2.$$

Начальные условия для системы уравнений первого порядка записываются следующим образом:

$$y_i(t_0) = A_i \quad \text{(задано)}, \ i = 1, \ldots, n.$$

Стандартный прием – запись системы в векторной форме – позволяет упростить обозначения. Введем

$$\mathbf{y}(t) = \begin{bmatrix} y_1(t) \\ y_2(t) \\ \vdots \\ y_n(t) \end{bmatrix}, \quad \mathbf{f}(t, \mathbf{y}) = \begin{bmatrix} f_1(t, y_1, \ldots, y_n) \\ f_2(t, y_1, \ldots, y_n) \\ \vdots \\ f_n(t, y_1, \ldots, y_n) \end{bmatrix}, \quad \mathbf{y}_0 = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_n \end{bmatrix}.$$

Тогда начальная задача для системы записывается в виде

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}), \quad \mathbf{y}(t_0) = \mathbf{y}_0.$$

Если начальные условия не заданы, то система обладает n-параметриче-

ским семейством решений; каждое решение представляет собой набор из n функций.

Поразительно, но теория и численные алгоритмы для одного уравнения в основном переносятся и на случай системы. Задача 8.10 позволит вам убедиться в этом самостоятельно — запрограммировать простой алгоритм и решить систему двух уравнений. Вот почему при изложении большинства последующих вопросов мы будем брать в качестве модели одно уравнение.

ОДУ *второго порядка* — это уравнение вида

$$u'' = f(t, u, u').$$

Например, перевернутый маятник (см. задачу 8.13) моделируется нелинейным уравнением второго порядка

$$\theta'' = \frac{3}{2L}(g - A\omega^2 \sin \omega t) \sin \theta.$$

Здесь $\theta(t)$ – угол отклонения маятника в момент времени t; A, ω и L – физические параметры; g – ускорение свободного падения. Начальная задача для этого уравнения содержит также угол θ и скорость θ', заданные в момент времени t_0.

Далее, уравнения высокого порядка обычно преобразуют к системам уравнений первого порядка с помощью стандартной замены неизвестных. Введем две новые неизвестные функции $y_1(t)$ и $y_2(t)$ следующим образом:

$$y_1(t) = \theta(t) \quad \text{и} \quad y_2(t) = \theta'(t).$$

Тогда для новых неизвестных получим систему двух уравнений первого порядка

$$\begin{aligned} y_1' &= y_2, \\ y_2' &= \frac{3}{2L}(g - A\omega^2 \sin \omega t) \sin y_1 \end{aligned}$$

с начальными условиями

$$y_1(t_0) = \theta(t_0) \ \text{(задано)}, \ y_2(t_0) = \theta'(t_0) \ \text{(задано)}.$$

В случае уравнения n-го порядка

$$u^{(n)} = f(t, u, u', \ldots, u^{(n-1)})$$

введем n новых неизвестных $y_1(t)$, $y_2(t)$, ..., $y_n(t)$:

$$\begin{aligned} y_1(t) &= u, \\ y_2(t) &= u', \\ &\vdots \\ y_n(t) &= u^{(n-1)}. \end{aligned}$$

Для новых неизвестных система из n уравнений первого порядка принимает вид

$$\begin{aligned} y_1' &= y_2, \\ y_2' &= y_3, \\ &\vdots \\ y_{n-1}' &= y_n, \\ y_n' &= f(t, y_1, y_2, \ldots, y_n). \end{aligned}$$

Для задачи о маятнике $f_1(t, y_1, y_2) = y_2$ и $f_2(t, y_1, y_2) = 3(g - A\omega^2 \sin \omega t)(\sin y_1)/(2L)$. Отметим, что в системе «лисы–кролики» переменная t не входит в правые части f явно, а в случае маятника входит (в f_2). В первом случае системы называют *автономными*, а во втором — *неавтономными*. Для наглядности при решении ОДУ второго порядка бывает полезно строить график одной зависимой переменной как функции от другой, оставляя независимой переменной роль параметра. Если зависимыми переменными являются положение и скорость, то такой график называется *фазовой плоскостью*, поскольку положение и скорость именуют двумя *фазами* системы. Кривая, изображающая решение, называется *траекторией* в фазовой плоскости. Это термин широко используется и при решении систем ОДУ произвольного вида; он означает кривую на графике, по осям которого откладываются зависимые переменные. Например, если в качестве зависимых переменных взяты координаты x и y спутника на плоской орбите (см. задачу 8.6), то фазовой плоскостью будет орбита (рассматриваемая «сверху»); эта картина несет гораздо больше информации, чем два отдельных графика для x и y как функций от t.

Почти все библиотечные программы для начальных задач позволяют решать системы из n уравнений первого порядка. Если уравнение пользователя имеет второй или более высокий порядок, то это уравнение следует привести к системе (вручную). Каждый пользователь, желающий работать с наиболее современным программным обеспечением, должен уметь выполнять такие преобразования. Системы, полученные из уравнения n-го порядка, гораздо проще систем общего вида, однако лишь немногие из современных программ способны воспользоваться этим преимуществом.

Преобразование уравнения n-го порядка к системе из n уравнений первого порядка сулит определенную выгоду. Хотя основной интерес представляет искомая функция $y_1(t)$, однако и ее производные, особенно первая производная $y_1'(t)$, также физически значимы. Эти производные вычисляются в процессе решения системы «бесплатно».

8.2. Устойчивые и неустойчивые уравнения и численные методы

Большинство начальных задач, возникающих на практике, не удается решить аналитически. С XV в. до середины 50-х годов XX в. их решали приближенно с помощью весьма изощренных механических интеграторов. Возможно, этот период оказался столь долгим потому, что результатом работы интеграторов была функция, а не набор чисел. Численные методы восходят к семнадцатому столетию — к Ньютону, который предложил использовать их для расчета траекторий комет. В 1748 году французские ученые предсказали возврат кометы Галлея ценой шести месяцев расчетов вручную «с утра до вечера и даже подчас во время еды». Вычисления вручную продолжали использоваться до 60-х годов XX века. Сегодня наиболее популярная техника численного интегрирования заключается в следующем: вместо того чтобы искать приближенное решение во всех точках t отрезка $[t_0, T]$, мы ограничиваемся поиском приближенных значений решения для набора точек t_0, $t_1, \ldots, T$. Если точное решение в точке t_k равно $y(t_k)$, то приближенное значение обозначим через y_k. Мы также будем пользоваться обозначением $y'_k \equiv f(t_k, y_k)$. Отметим, что y'_k не совпадает с $y'(t_k)$. (В разделе 1.2 мы также использовали обозначение y_k для k-й компоненты вектора решения системы. Оба обозначения общеприняты, и их употребление редко приводит к недоразумениям.)

Приближенное значение y_k вычисляется по найденным ранее величинам $y_{k-1}, \ldots, y_1, y_0$. Если формула, по которой вычисляется y_k, зависит явно только от y_{k-1}, то метод называется *одношаговым*. Если y_k вычисляется по двум значениям y_{k-1} и y_{k-2}, то метод называется *двухшаговым*. Многошаговые методы доставляют определенные трудности, например на первом шаге интегрирования.

Набор узлов t_k может быть задан пользователем или построен автоматически самой программой. В обоих случаях расстояние между узлами, т. е. шаг $h_k \equiv t_{k+1} - t_k$, зависит от конкретной задачи, численного метода и точности, с которой мы хотим вычислить результат y_k. Мы знаем, что $y_0 = y(t_0)$ и во многих случаях $y_1 \approx y(t_1)$, но $y_1 \neq y(t_1)$. Поэтому точка (t_1, y_1) уже не будет лежать на кривой решения, исходящей из начальной точки (t_0, y_0). Скорее всего, она будет лежать на другой кривой из семейства решений. Если мы проследим такую кривую обратно до t_0, то не окажемся в точке (t_0, y_0). Таким образом, точка (t_1, y_1) отвечает решению данного дифференциального уравнения, но при другом начальном условии. Рис. 8.2 иллюстрирует эту ситуацию.

Лучшее, на что можно надеяться при переходе от t_1 к t_2, — это что y_2 удержится на кривой, проходящей через (t_1, y_1). В действительности такое маловероятно. Еще менее вероятно возвращение на исходную кривую, проходящую через начальную точку (t_0, y_0). Таким образом, если на первом шаге допущена ошибка, то, даже при отсутствии ошибок в дальнейшем, числа $y_2, y_3, \ldots$ также будут неточны. На величины их

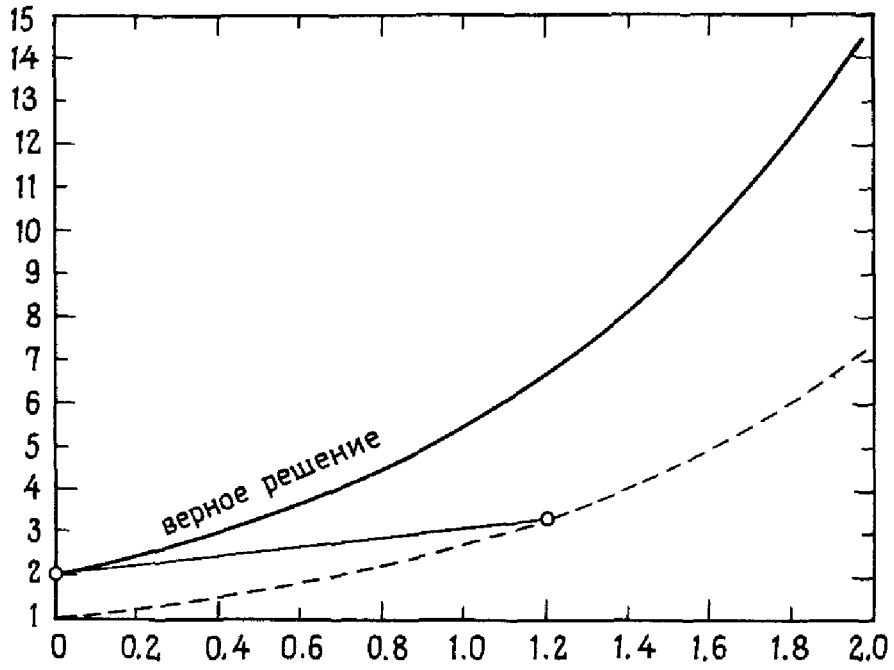


Рис. 8.2. Соскальзывание с кривой решения.

погрешностей влияют не только численный метод и шаг интегрирования, но и само семейство решений.

Обратимся к семейству решений ce^{-t}, изображенному на рис. 8.1. Заметим, что кривые из этого семейства сходятся с ростом t. Вообще, если кривые решений уравнения $y' = f(t, y)$ расходятся по мере удаления t от начальной точки, то уравнение называется *неустойчивым*. Если кривые приближаются друг к другу, то говорят, что уравнение *устойчиво*. Уравнение общего вида может демонстрировать на различных частях отрезка интегрирования оба типа поведения. Системы уравнений характеризуются такими же свойствами, хотя их труднее интерпретировать графически. Если уравнение неустойчиво, то ошибки, вообще говоря, имеют тенденцию к росту [1]). Устойчивые уравнения подавляют ошибки. Что происходит на самом деле, определяется не отдельным решением, а всем семейством решений. Две задачи $y' = y$, $y(0) = 1$ и $y' = \exp(t)$, $y(0) = 1$ имеют общее решение, но семейства решений соответствующих уравнений различны. Как вы полагаете, какую из этих задач труднее решить численно?

При фиксированном t угол наклона касательных к кривым из семейства решений, вообще говоря, изменяется как функция от y. Это изменение определяется величиной f_y, которая служит мерой устойчиво-

[1]) Употребляя термин «ошибка», мы имеем в виду абсолютную ошибку. Относительная ошибка (ошибка, отнесенная к величине решения) может и не вести себя подобным образом. На практике стараются, чтобы именно такая погрешность контролировалась программой (см., например, раздел 5.1), однако абсолютную ошибку проще исследовать, вот почему наше внимание сконцентрировано именно на ней.

сти уравнения. Число f_y называется *якобианом* уравнения и обозначается через J. Устойчивым уравнениям соответствуют отрицательные значения J. Пример тому вы можете увидеть на рис. 8.1. В заключение скажем, что не следует рассчитывать на высокую точность при численном решении неустойчивого ОДУ.

Пример 8.2. Устойчивое и неустойчивое уравнения.
Уравнение

$$y' = -2\alpha(t-1)y$$

обладает семейством решений

$$y = c\exp(-\alpha(t-1)^2).$$

Некоторые кривые из этого семейства для случая $\alpha = 5$ и интервала $[0,2]$ изображены на рис. 8.3. Мы видим, что эти кривые вначале удаляются друг от друга, а затем сближаются. Это согласуется с тем фактом, что якобиан $J = -2\alpha(t-1)$ положителен при $t < 1$ и отрицателен при $t > 1$. Это области умеренной неустойчивости и устойчивости соответственно. Изменение начального условия при $t = 0$ с 0.01 до 0.1 приводит к росту решения в точке $t = 1$ с 1.0 примерно до 15.0. □

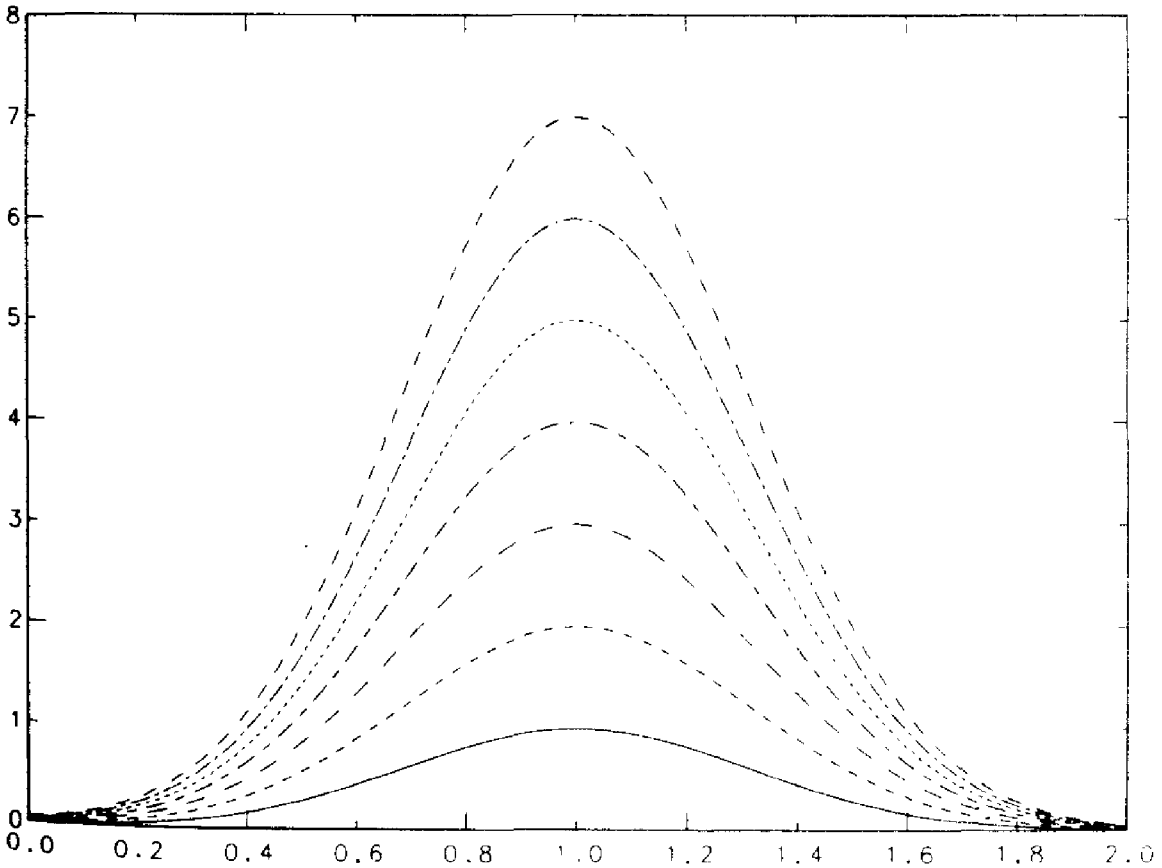


Рис. 8.3. Уравнение с областями устойчивости и неустойчивости.

*8.2.1. Устойчивость системы

Понятие устойчивости применимо также и к системам уравнений. Единственная новая идея связана с *матрицей Якоби* (*якобианом*) J, введенной в § 5 гл. 7. Напомним, что J — это матрица, у которой элемент (i, j) равен $J_{i,j} = \partial f_i / \partial y_j$. В задаче о маятнике J представляет собой матрицу размера 2×2

$$J = \begin{pmatrix} 0 & 1 \\ 3(g - A\omega^2 \sin t)(\cos y_1)/(2L) & 0 \end{pmatrix}.$$

В задаче «лисы — кролики» J также является матрицей размера 2×2:

$$J = \begin{pmatrix} 2 - \alpha y_2 & -\alpha y_1 \\ \alpha y_2 & -1 + \alpha y_1 \end{pmatrix}.$$

Для системы из n уравнений J является матрицей $n \times n$. В случае одного уравнения J представляет собой матрицу 1×1, единственный элемент которой — просто число, равное частной производной f_y. Вот почему J называют якобианом даже в том случае, когда рассматривается одно уравнение. Как мы увидим далее, якобиан можно обсуждать в разных контекстах. Некоторые программы требуют, чтобы пользователь написал подпрограмму, вычисляющую элементы якобиана.

Устойчивость систем непосредственно связана с собственными значениями матрицы J. Напомним, что собственные значения — это n чисел $\lambda_1, \lambda_2, \ldots, \lambda_n$, для которых матрица $J - \lambda_i I$ будет вырожденной, т. е. определитель этой матрицы обратится в нуль. В случае одного уравнения $\lambda_1 = J$. Для задачи о маятнике собственными значениями являются корни квадратного уравнения

$$\lambda^2 - \frac{3}{2L}(g - A\omega^2 \sin t)\cos y_1 = 0.$$

Вообще говоря, собственные значения — это комплексные числа, которые отнюдь не постоянны, а зависят от t и решения $\mathbf{y}$. Можно показать, что устойчивостью управляет собственное значение с наибольшей вещественной частью. Положительные вещественные части обычно соответствуют областям с расходящимися (неустойчивыми) решениями. Для большинства задач, представляющих практический интерес, наличие у всех собственных значений отрицательных вещественных частей влечет за собой устойчивость решений. При каждом фиксированном t могут встречаться собственные значения как с положительными, так и с отрицательными вещественными частями.

Утверждение, что устойчивость определяется собственными значениями, становится более очевидным, если исследовать линейную задачу в расчете на то, что полученные результаты дадут информацию, полезную и в более общей ситуации. Для этого рассмотрим тело массы $m = 1/2$, лежащее на гладкой поверхности и прикрепленное к близлежа-

щей стенке посредством пружины и буфера (амортизатора) [1]). Пружина и буфер предполагаются линейными в том смысле, что упругая сила сопротивления составляет $-x/2$, а сила демпфирования равна $-dx'$, где x – расстояние от стенки до тела. Если игнорировать силу тяжести, то результирующая сила mx'', действующая на тело, равна сумме усилий пружины и буфера:

$$x'' = -2dx' - x.$$

Попытаемся найти решение вида $x(t) = \exp(\lambda t)$. Подстановка в уравнение показывает, что такое $x(t)$ будет решением, если λ удовлетворяет квадратному уравнению

$$\lambda^2 + 2d\lambda + 1 = 0.$$

Следовательно,

$$\lambda_i = -d \pm \sqrt{d^2 - 1}, \quad i = 1, 2.$$

Из общей теории известно, что каждое решение нашего дифференциального уравнения представимо в виде следующей линейной комбинации:

$$x(t) = A\exp(\lambda_1 t) + B\exp(\lambda_2 t).$$

(Для этой задачи характерные времена равны $1/\lambda_1$ и $1/\lambda_2$.) Поскольку оба числа λ отрицательны, всякое решение этого уравнения является суммой двух убывающих экспонент и кривые любых двух решений сближаются, т. е. уравнение устойчиво в рассматриваемом здесь смысле. Наш опыт подсказывает, что задача устойчива и в физическом смысле: колебания тела со временем затухают, и можно ожидать, что и ОДУ будет обладать подобным свойством. Наконец, если мы запишем ОДУ второго порядка в виде системы двух уравнений первого порядка

$$\begin{aligned} y_1' &= y_2, \\ y_2' &= -2dy_2 - y_1, \end{aligned} \qquad J = \begin{pmatrix} 0 & 1 \\ -1 & -2d \end{pmatrix},$$

то λ_1 и λ_2 окажутся собственными значениями якобиана – матрицы J размера 2×2.

Теперь посмотрим, что будет происходить в случае жесткого буфера ($d \gg 1$). В этом случае

$$\lambda_i = d(-1 \pm (1 - 1/d^2)^{1/2}) \approx d(-1 \pm (1 - 1/(2d^2))),$$

$$\lambda_1 \approx -2d, \quad \lambda_2 \approx -\frac{1}{2d}.$$

Жесткость рассматриваемой механической системы связана с наличием большого (по модулю) отрицательного собственного значения.

Начальная задача для системы ОДУ является *жесткой*, если хотя бы

[1]) Этот пример взят (с разрешения автора) из работы Хиндмарша 1974 г.

у одного из собственных чисел якобиана J вещественная часть отрицательна и велика (по модулю), а решение на большей части отрезка интегрирования меняется медленно.

8.3. Жесткие дифференциальные уравнения

Обыкновенное дифференциальное уравнение называют *сверхустойчивым*, если $J \ll 0$. Мы вправе рассчитывать, что численное решение начальных задач для таких уравнений окажется значительно проще, чем для неустойчивых уравнений, поскольку здесь соскальзывание с правильной кривой решения не будет приводить к чересчур разрушительным последствиям. Мы увидим, однако, что это будет зависеть от численного метода. Обычно сверхустойчивые уравнения называют жесткими, хотя этот термин больше подходит для систем уравнений, а не для одного уравнения. Задачи, возникающие на практике, обычно бывают устойчивыми, жесткими или лишь умеренно неустойчивыми.

Жесткость — это довольно сложное понятие, которому мы не дали строгого определения. Рабочее описание жесткой задачи таково: это задача, моделирующая физический процесс, компоненты которого обладают несоизмеримыми характерными временами, или же процесс, характерное время которого намного меньше отрезка интегрирования. (Более удовлетворительное описание, указывающее на роль величины собственных значений якобиана J, приведено в разделе 2.1.) Многие практики отождествляют жесткость и «трудность», однако это неверно, поскольку чем жестче система, тем лучше ее решает хорошая «жесткая» программа. На самом деле этот термин следует связывать со всем комплексом, включающим задачу, отрезок интегрирования, применяемый численный метод и даже малозначительные на поверхностный взгляд детали реализации алгоритма. Читатель, желающий ознакомиться с этой темой глубже, может найти много полезных вводных статей и ссылок в книге [Aiken, 1985] или в обзорной статье [Byrne, Hindvarsh, 1987]. Жесткие задачи возникают почти во всех прикладных областях, но особенно характерны для химической кинетики.

Пример 8.3. Жесткое уравнение.

Рассмотрим начальную задачу $y' = -\alpha(y - \sin t) + \cos t$, $y(0) = 1$. Ее решение имеет вид

$$y(t) = e^{-\alpha t} + \sin t.$$

Если мы хотим решить эту задачу на отрезке $0 \leqslant t \leqslant 1$ и α велико, например $\alpha \approx 1000$, то $J = -1000$ и задача является жесткой. (На ином отрезке интегрирования, скажем при $t < 0.002$, эта задача не будет считаться жесткой.) Для t, близких к нулю, решение y быстро уменьшается и падает почти до 0. Потом оно практически не отличается от $\sin t$. Решение вблизи $t = 0$ качественно отличается от решения при больших t. Начальный отрезок решения называется *пограничным слоем*, остальная

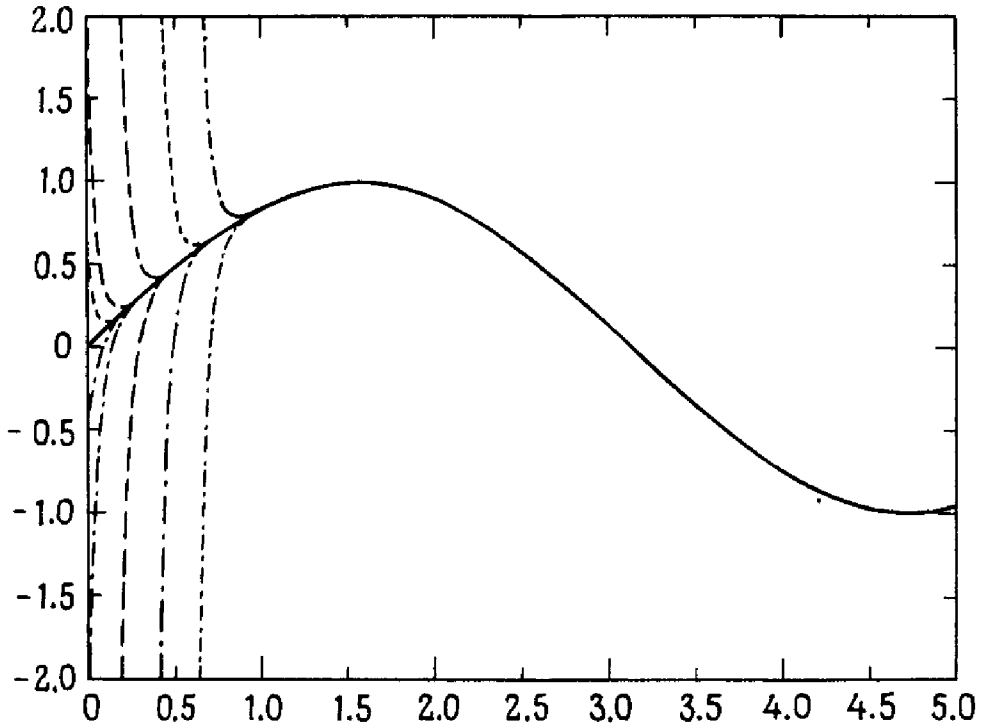


Рис. 8.4. Жесткое уравнение.

его часть характеризуется *режимом медленного (плавного) изменения*. На рис. 8.4 проиллюстрировано семейство решений этой задачи. Важно отметить, что всякое решение, исходящее из какой-либо точки плоскости, расположенной выше или ниже плавно меняющейся кривой $\sin t$, очень быстро стремится к этой кривой. На приведенном рисунке для лучшей визуализации графиков мы заменили константу 1000 на 20; у новой задачи жесткость гораздо меньше. □

На практике жесткие уравнения часто характеризуются обоими типами поведения, поскольку моделируют физические процессы, протекающие с различными характерными временами. Часто пограничный слой возникает не в начале интегрирования, а лишь тогда, когда некоторый «управляющий» параметр резко изменит свое значение. Позже мы увидим пример такого поведения, когда будем рассматривать модель количества озона в атмосфере. В некоторых приложениях важен пограничный слой; в других — лишь предельное поведение (плавное изменение) при больших временах интегрирования. Об этом различии важно помнить, продумывая способ решения задачи. Можно ожидать, что отследить пограничный слой окажется довольно трудно, а медленное изменение — легче. Мы увидим, что при численном решении такого рода задач наши ожидания подтверждаются не всегда, если только мы не выберем численный метод очень тщательно.

8.4. Метод Эйлера

Теорема Тейлора служит основным инструментом вывода большинства численных методов решения ОДУ. Конечно, цель настоящей главы не в том, чтобы конструировать методы, а в том, чтобы помочь читателю понять, как устроена и как работает некоторая конкретная программа. Однако ради этого нам придется вывести несколько простых методов, в том числе метод Эйлера, обратный метод Эйлера и метод трапеций. Мы сделаем это, чтобы проиллюстрировать такие понятия, как порядок, точность и устойчивость.

Предположим, что требуется решить задачу $y' = f(t, y)$, $y(t_0) = y_0$ на интервале $[t_0, T]$. Допустим, что решение $y(t)$ имеет по крайней мере непрерывную вторую производную. Тогда

$$y(t + h) = y(t) + hy'(t) + h^2y''(\xi)/2 = y(t) + hf(t, y(t)) + h^2y''(\xi)/2.$$

Если известно приближенное решение y_k в узле t_k и выбран новый узел t_{k+1}, то для *метода Эйлера* – одношагового метода – значение y_{k+1}^{EM} (где индекс EM означает метод Эйлера – the Euler Method) определяется по выписанной выше формуле, в которой отброшен остаточный член:

$$y_{k+1}^{EM} \equiv y_k + (t_{k+1} - t_k)f(t_k, y_k) = y_k + h_k y'_k.$$

Пример 8.4. Применение метода Эйлера.

Решим методом Эйлера задачу $y' = -10(t - 1)y$, $y(0) = \exp(-5)$ на интервале $[0, 2]$. Выберем априори равномерную сетку с 21 узлом, а именно $t_0 = 0$, $t_1 = 0.1, \ldots, t_{20} = 2.0$. Приступим к вычислениям:

$$y_0 = \exp(-5),$$
$$y_1 = y_0 + hy'_0 = y_0 + (.1)[-10(t_0 - 1)y_0],$$
$$y_2 = y_1 + (.1)[-10(t_1 - 1)y_1].$$

Таблица 8.1

t_n	y_n	Ошибка	t_n	y_n	Ошибка
0.1	0.0134	0.004	1.1	0.452	0.499
0.2	0.0256	0.015	1.2	0.406	0.412
0.3	0.0461	0.040	1.3	0.325	0.312
0.4	0.0784	0.087	1.4	0.223	0.222
0.5	0.1254	0.161	1.5	0.137	0.150
0.6	0.1880	0.261	1.6	0.068	0.097
0.7	0.263	0.374	1.7	0.003	0.059
0.8	0.342	0.477	1.8	0.0082	0.033
0.9	0.411	0.541	1.9	0.0016	0.016
1.0	0.452	0.548	2.0	0.0002	0.007

В таблице приведены вычисленные значения y_k и ошибки для всех узлов сетки t_k. Точным решением будет $y(t) = \exp(-5(t-1)^2)$.

Из примера 8.2 мы уже знаем, что это уравнение неустойчиво на [0,1], поэтому неудивительно, что ошибка при $t = 1$ столь велика. Если же мы повторим вычисления с меньшим шагом h, то ошибка в точке $t = 1$ будет уменьшаться. В следующей таблице приведены величины ошибок в точке $t = 1$ при различных значениях h.

Таблица 8.2 Ошибка в точке $t = 1$ в зависимости от шага h

h	Ошибка
0.1	0.548
0.05	0.375
0.025	0.229
0.0125	0.128
0.01	0.105
0.001	0.01154
0.0001	0.001165
0.00001	0.0001167
0.000001	0.00001167

8.5. Точность и устойчивость численных методов

При численном решении начальной задачи методом Эйлера единственным параметром, которым можно управлять, является шаг сетки h. Типичные программы численного интегрирования, подобные программе, описанной в данной главе, требуют, чтобы пользователь задал допуск ε. Эта величина используется при выборе шага интегрирования. Выбрав шаг h достаточно малым, мы надеемся уменьшить ошибку до величины, не превосходящей ε. Чтобы сделать это, нам надо уметь оценивать точность вычислений. В данном разделе объясняется, как это делается.

Вернемся к формуле Тейлора, положив $t = t_k$ и $h = h_k$:

$$y(t_{k+1}) = y(t_k) + h_k f(t_k, y(t_k)) + h_k^2 y''(\xi_k)/2, \quad t_k < \xi_k < t_{k+1}.$$

Как правило, $y_k \neq y(t_k)$. Единственное исключение составляет узел t_0, где $y_0 = y(t_0)$. Если из выписанного выражения вычесть формулу метода Эйлера, получим

$$y(t_{k+1}) - y_{k+1}^{\text{EM}} = y(t_k) - y_k^{\text{EM}} + h_k[f(t_k, y(t_k)) - f(t_k, y_k^{\text{EM}})] + h_k^2 y''(\xi_k)/2.$$

В левой части записана ошибка в узле t_{k+1}. Она называется глобальной ошибкой; именно эту величину мы хотим контролировать. Глобальная

ошибка является разностью между вычисленным и «теоретическим» решениями. Мы видим, что она имеет два источника.

(1) Локальная ошибка. Если по счастливой случайности $y_k^{\text{EM}} = y(t_k)$, то в правой части нашего выражения две разности обращаются в нуль. Тем не менее глобальная ошибка нулю не равна. Оставшийся член называется *локальной ошибкой* или *ошибкой усечения*. Он в точности совпадает с остаточным членом $h^2 y''(\xi)/2$ в формуле Тейлора. Локальная ошибка — это ошибка, вносимая при переходе от t_k к t_{k+1} в предположении, что все величины при t_k (и при $t < t_k$) вычислены точно; в этом смысле она представляет собой искусственную характеристику.

(2) Распространяемая ошибка. Согласно теореме о среднем, второе слагаемое в правой части можно переписать в виде

$$h_k [f(t_k, y(t_k)) - f(t_k, y_k^{\text{EM}})] = h_k J(\xi)(y(t_k) - y_k^{\text{EM}}).$$

Запись $J(\xi)$ означает, что якобиан берется в некоторой заранее неизвестной точке. Итак, (1) и (2) можно суммировать следующим образом:

глобальная ошибка при $t_{k+1} = (1 + h_k J)$ (глобальная ошибка при t_k) + локальная ошибка при t_{k+1}.

Глобальная ошибка в узле t_k, $y(t_k) - y_k^{\text{EM}}$, умножается на величину $(1 - h_k J)$, называемую *множителем перехода* (авторы используют термин "amplification factor", дословно — «коэффициент усиления». — *Перев.*). Мы видим, что якобиан J, который раньше встречался при описании устойчивости ОДУ, появляется и здесь, при исследовании численных алгоритмов, где он входит в множитель перехода. Если $|1 + h_k J| < 1$, то ошибки в методе Эйлера не возрастают, если же этот модуль превосходит единицу, то ошибки будут возрастать. В последнем случае говорят, что метод Эйлера *неустойчив*. Итак, понятие устойчивости может использоваться в двух контекстах, связанных с уравнением или с численным методом.

Неравенство $|1 + hJ| < 1$ эквивалентно неравенствам $-2 < hJ < 0$, которые задают так называемый *интервал устойчивости* метода Эйлера. Если ОДУ неустойчиво, то эти неравенства никогда не выполняются и неустойчивость метода Эйлера всегда проявляется. Если же ОДУ устойчиво ($J < 0$), то эти два неравенства приводят к условию $h < -2/J$. Однако даже для устойчивой задачи метод Эйлера может оказаться неустойчивым, если только шаг интегрирования h не выбран меньшим чем $|2/J|$ [1]).

Пример 8.5. Устойчивость метода Эйлера.

Вернемся снова к задачам из примеров 8.2 и 8.4. При $t < 1$ уравнение

[1] Если вас это удивляет, то не огорчайтесь: вы не одиноки. До начала 50-х годов большинство ученых не задумывалось над проблемой устойчивости. Лишь в 1953 году В. Милн провел анализ популярного в то время класса методов интегрирования и показал, что один из методов является неустойчивым при любом положительном шаге h.

неустойчиво, что ведет к росту ошибки метода Эйлера, поскольку $|1 + hJ| > 1$ при любом h. При $t > 1$ уравнение является устойчивым. Однако, несмотря на устойчивость уравнения, метод Эйлера будет неустойчивым, если не выполнено ограничение на шаг $h < 1/[\alpha(t-1)]$. Положив в примере 8.4 $\alpha = 5$ и $h = 0.1$, мы должны ожидать роста ошибки при $t > 3$. На рис. 8.5(а) точки (t_k, y_k) соединены ломаными. Неустойчивость сначала «берет разгон» и уже с точки $t = 4$ проявляется очень резко. Для данной задачи множитель перехода отрицателен; вот почему глобальная ошибка меняет знак от узла к узлу. Подобный эффект, т.е. растущие осцилляции (что бы ни было их конкретной причиной), почти всегда является следствием неустойчивости численного

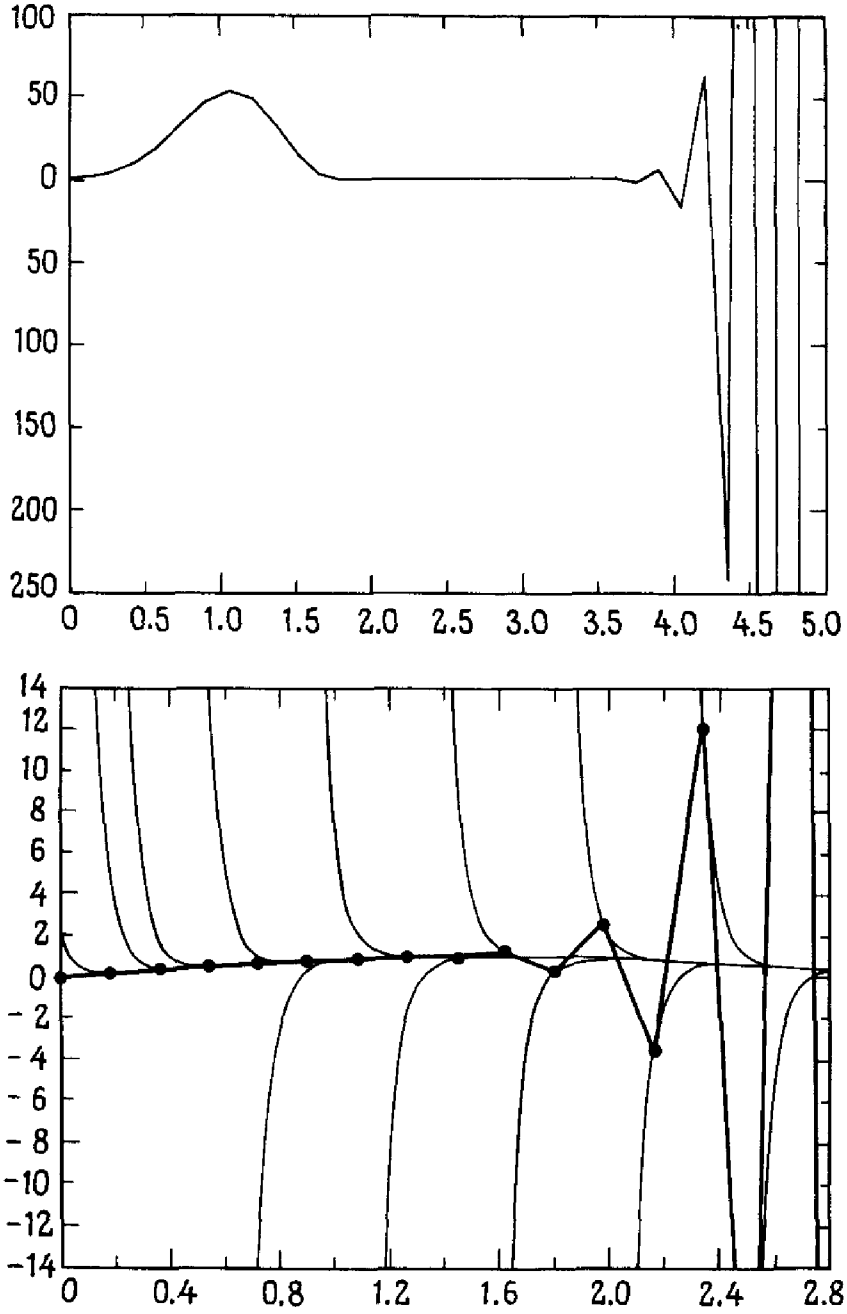


Рис. 8.5. Метод Эйлера: (а) неустойчивость; (б) поведение при решении жесткой задачи.

метода. В таких случаях читателю следует быть настороже и никогда не связывать полученное численное решение с исследуемым физическим процессом. □

Очень важно осознать, что численный метод может быть неустойчивым, даже если ОДУ устойчиво. В примере 8.3 мы рассмотрели задачу, решение которой меняется слабо почти для всех $t > 0$. Если $\alpha = 1000$, то для метода Эйлера ограничение устойчивости на шаг сетки имеет вид $h < 2 \cdot 10^{-3}$. Если нас интересует решение в пограничном слое, то такой малый шаг интегрирования имеет физический смысл, но в области плавного изменения решения выбор такого шага смысла не имеет. В первом случае говорят, что шаг интегрирования для численного метода определяется из условия точности, а во втором случае — из условия устойчивости. Вы сможете увидеть, какие трудности возникают при использовании метода Эйлера для решения жестких задач, если изучите рис. 8.5(б). Чем дальше мы отклоняемся от решения, тем больше «вычисленный» наклон отличается от правильного значения. Для решения жестких задач метод Эйлера используется редко, если только нас не интересует в первую очередь пограничный слой.

Шаг сетки h входит как в локальную ошибку h^2y'', так и в множитель перехода $1 + hJ$. Выбрав h достаточно малым, мы можем сделать локальную ошибку сколь угодно малой и в случае устойчивой задачи добиться, чтобы множитель перехода был меньше единицы. Однако ни y'', ни J в явном виде не известны. Большинство программ пытаются оценить y'', но игнорируют J. Иногда говорят, что такие программы осуществляют только контроль локальной ошибки.

Процедура контроля ошибки пытается выбрать шаг примерно обратно пропорциональным квадратному корню из величины второй производной; следовательно, она оперирует с локальной ошибкой. Однако, как мы уже объясняли, локальная ошибка — это лишь часть общей картины; на величину глобальной ошибки существенно влияет и множитель перехода. Итак, что произойдет, если мы воспользуемся таким алгоритмом выбора шага? Для примеров 8.2 и 8.3 мы знаем точное решение и, следовательно, можем вычислить $y''(t)$ непосредственно, получая (приближенно при больших t) соответственно $t^2\exp(-\alpha t)^2$ и $-\sin t$. Первая из этих функций убывает по t, а вторая никогда не превосходит 1. Таким образом, стратегия выбора шага, оперирующая с этими функциями, приведет к последовательно возрастающему h для примера 8.2 и к более или менее постоянному h для примера 8.3. В обоих случаях возникнет противоречие с условием устойчивости для метода Эйлера, результатом чего будет быстрая и отчетливая неустойчивость. Но если мы обратимся к алгоритму выбора шага, подобному алгоритму из раздела 5.1, который не использует вторую производную явно, а вычисляет только ее оценку, то убедимся, что алгоритм такого рода не порождает «взрывов» и гигантских ошибок. Напротив, h поддерживается примерно пропорциональным $1/|J|$ для обоих примеров, а ошибки

контролируются. Отметим, что шаг будет малым даже в области плавного изменения решения жесткой задачи. Почему это так? Ответ вновь зависит от свойств семейства решений.

Оценка второй производной основана на вычислении разности первых производных y'_{n-1} и y'_n. Если мы находимся на кривой решения, то эти производные точны и оценка второй производной мала, поскольку все величины $y'(t)$ малы. Однако $y'_{n-1} = f(t_{n-1}, y_{n-1})$ и $y'_n = f(t_n, y_n)$. Едва ли y_{n-1} и y_n расположены на одной кривой решения. Таким образом, эти два числа следует рассматривать как точные значения производных, но от разных решений. Однако в случае жесткой задачи небольшое возмущение кривой решения может породить сильное изменение производной y' (поскольку J велико). Итак, похоже, что разность между значениями должна быть велика, а это ведет к малому шагу интегрирования. Пример 8.2 демонстрирует жесткость при больших t и подтверждает наши рассуждения.

Пример 8.6. Переменный шаг метода Эйлера для жесткой задачи. Рассмотрим жесткую задачу

$$y' = -500(y - \sin t) + \cos t, \quad 0.5 \leqslant t \leqslant 1.$$

Исходя из точного начального значения при $t = 0.5$, попробуем проинтегрировать уравнение до $t = 1.0$ методом Эйлера с простой стратегией выбора шага, описываемой в разделе 5.1. Возьмем начальный шаг равным 0.05 и зададим допуск на погрешность $\varepsilon = 0.001$. Если шаг от t к $t + h$ оказался неудачным, т.е. оценка ошибки была велика, то попробуем еще раз с шагом $h/2$. Этот алгоритм легко запрограммировать. Мы обнаружим, что 669 вычислений функции $f(t, y)$ достаточно, чтобы достичь точки $t = 1$, и относительная ошибка в конечной точке равна 0.005. Итак, средний шаг равен $1/(2 \cdot 669)$, или примерно 0.0007; иными словами, метод Эйлера с постоянным шагом $h = 0.0007$ потребовал бы примерно такого же количества вычислений функции f. Данный шаг гораздо меньше того, какой, согласно интуитивному ощущению, нужен, чтобы отследить столь медленно меняющуюся функцию, и это объясняется жесткостью уравнения. □

Понятие устойчивости численного метода играет фундаментальную роль в наших исследованиях. Программы с выбором постоянного шага могут порождать неустойчивость, как это показано в примере 8.5. При выборе переменного шага неустойчивость возникает сравнительно редко, однако такие программы могут быть неэффективными, на что указывает приведенный выше пример.

Резюме: ОДУ является устойчивым или неустойчивым вне зависимости от того, каким образом оно решается. Численный метод интегрирования может быть устойчивым или неустойчивым для конкретной задачи в зависимости от размера шага.

*8.5.1. Простая стратегия выбора шага

Многие из программ, обеспечивающих автоматический выбор шага, реализуют следующий общий алгоритм. Исходя из приближенного решения y_n в точке t_n (а также другой информации):

(1) Оценить следующий шаг интегрирования h_n по известной информации о текущей локальной ошибке метода Эйлера. Существует несколько способов оценки y''. Вот один из них. Предположим, что уже вычислены значения y_{n-1}, y_n и соответственно y'_{n-1}, y'_n и мы намерены перейти к новому узлу t_{n+1} так, чтобы локальная ошибка осталась меньшей, чем заданное ε. Оценим y'' как

$$y'' \simeq \frac{y'_n - y'_{n-1}}{t_n - t_{n-1}} \equiv y''_{n-1,n}.$$

Чтобы при переходе от t_n к t_{n+1} удержать локальную ошибку в пределах, заданных допуском ε, выберем шаг $h = t_{n+1} - t_n$ следующим образом:

$$|y''|h^2/2 \simeq |y''_{n-1,n}|h^2/2 \leqslant \varepsilon, \text{ или } h \leqslant \sqrt{\frac{2\varepsilon}{|y''_{n-1,n}|}}.$$

На практике шаг h выбирают несколько меньшим, скажем равным 0.9 от указанной величины, и никогда не допускают слишком большого шага, даже если величина $y''_{n-1,n}$ мала. Наконец, большинство пользователей стремится получить решение с определенным числом верных знаков, обычно от трех до пяти, вне зависимости от величины самого решения. Поэтому желательно контролировать относительную ошибку. С этой целью мы заменяем ε на $\varepsilon|y_n|$, т.е. выбираем

$$h \leqslant \min\left(0.9\sqrt{\frac{2\varepsilon|y_n|}{|y''_{n-1,n}|}},\ h_{\max}\right).$$

Это типичная стратегия выбора шага. При ее реализации можно допускать смену h на каждом шаге интегрирования, однако имеются теоретические свидетельства, что это не лучшее предложение. Большинство программ построено так, чтобы сделать несколько равных шагов, прежде чем разрешить смену шага. Отметим, что мы не давали рекомендаций по выбору начального шага $t_1 - t_0$. Далее, если y_n близко к нулю, то приведенная выше формула даст очень малый шаг. В большинство коммерческих программ заложено ограничение на минимальную величину шага интегрирования.

(2) Вычислить значение y_{n+1} в точке t_{n+1} по формуле метода Эйлера или иной формуле интегрирования.

(3) По полученной для точки t_{n+1} информации проверить, не является ли шаг слишком большим. Если все благополучно, продолжить процесс; иначе вернуться к (1) для новой попытки. Оценка для y'' использует информацию слева от точки t_{n+1}, чтобы выбрать саму точку t_{n+1}. Как только шаг выбран, мы методом Эйлера вычисляем значение

y_{n+1} и по нему находим y''_{n+1}. Между t_n и t_{n+1} могла скрываться какая-то особенность, которая проявится в неожиданном значении y'_{n+1}. В таком случае выбранный шаг, возможно, слишком велик. Чтобы застраховаться от превышения разумной длины шага, снова оценим y'', но уже с помощью величин, вычисленных в точках t_{n+1} и t_n. Положив

$$y''_{n,n+1} \equiv \frac{y'_{n+1} - y'_n}{t_{n+1} - t_n},$$

проверим, является ли величина $|y''_{n,n+1}|h^2/2$ достаточно малой. Если это так, то примем данный шаг и перейдем к интегрированию из точки t_{n+1}. В противном случае изменения в y'' или y достаточно велики, чтобы поставить под сомнение наш выбор длины шага, поэтому нужно вернуться в точку t_n и взять меньший шаг.

В заключение напомним, что большинство стратегий выбора шага лишь отчасти базируется на строгих теоретических обоснованиях. Гораздо чаще такие стратегии синтезируют теорию, эксперимент и интуицию.

*8.5.2. Анализ устойчивости для систем ОДУ

В случае систем исследование устойчивости можно провести по образцу анализа из § 5. Результаты, по существу, окажутся теми же: метод Эйлера будет неустойчивым для устойчивой задачи, если не выполнено условие $\|I + hJ\| < 1$, где I – единичная матрица, а $\|\cdot\|$ – некоторая матричная норма (см. § 7 гл. 3). Это условие будет выполнено, если $|1 + h\lambda| < 1$, где λ – любое собственное значение матрицы J. В случае одного вещественного уравнения это означает, что hJ должно лежать в интервале $(-2, 0)$; но собственные значения матриц, как правило, комплексны, вот почему неравенство $|1 + h\lambda| < 1$ уже не описывает интервал. Теперь $h\lambda$ должно отстоять от точки -1 на расстояние, меньшее 1 (отметим, что $|1 + h\lambda| = |1 - (-h\lambda)|$). Множеством точек, удовлетворяющих данному условию, является внутренность круга радиуса 1 с центром $z = -1$. Эта область называется *областью устойчивости*; если произведения $h\lambda$ лежат внутри нее, то метод Эйлера устойчив, в противном случае неустойчив. Отметим, что область устойчивости содержит интервал $(-2, 0)$, так что случай одного вещественного уравнения также покрывается нашим анализом. Величину $I + hJ$ следовало бы назвать матрицей перехода, но мы по-прежнему будем пользоваться термином «множитель перехода».

8.5.3. Историческая справка: Эйлер

Леонард Эйлер (1707–1783) родился и воспитывался в Базеле (Швейцария). Одним из его учителей был Йоганн Бернулли. Годы спустя Бернулли отзывался об Эйлере как о «знаменитейшем и мудрейшем

математике». Эйлер писал и публиковался с поразительной продуктивностью: почти 600 книг и статей в течение жизни. Его влияние было столь велико, что, по крайней мере в математике, восемнадцатое столетие вполне можно назвать эпохой Эйлера. Эйлер внес вклад не только в математику, но и в физику, астрономию, гидродинамику, оптику, в теорию электричества и магнетизма. В 1768 году он предложил метод решения начальных задач, который мы описали в § 4. Он также ввел многие из привычных сегодня математических обозначений, в том числе символ e для основания натурального логарифма, i для $\sqrt{-1}$, Δy для конечной разности, Σ для суммы, предложил, среди прочего, использовать символ f со скобками для обозначения функции, а также придумал названия для тригонометрических функций, используемые и сейчас.

Эйлер 14 лет работал в Российской академии наук, в Санкт-Петербурге. Затем он переехал в Берлин и работал в Берлинской академии до тех пор, пока разногласия с королем Фридрихом Великим не заставили его вернуться в Россию, где он и провел оставшиеся годы своей жизни. Он никогда не возвращался в Базель, покинутый им в 1727 году.

Талантливые и удачливые люди очень часто берутся за множество проблем; Эйлер не был исключением. Он работал в области судостроения, входил в разнообразные технические комитеты, занимался проверкой устройств для взвешивания и пожарных насосов, руководил составлением календарей и географических карт, наблюдал за ботаническими садами. Он довольно рано отказался от первоначального намерения стать министром, однако его интерес к религии и гуманитарной деятельности давал ему достаточную перспективу в отношении иных сторон жизни. У Эйлера была потрясающая память. В 1771 г. в результате болезни он ослеп, однако почти половина его научных работ написана после этой даты, и в возрасте семидесяти лет он был способен точно воспроизвести первые и последние строки каждой страницы вергилиевской «Энеиды», выученной им в молодости наизусть. После кончины Эйлера на его могиле на лютеранском кладбище Санкт-Петербурга был установлен массивный монумент, однако в 1956 г. его останки и памятник были перенесены в городской некрополь.

8.6. Порядок метода интегрирования

Метод Эйлера – это не единственный одношаговый метод. Важные примеры других методов приводятся в §§ 8 и 19. Для всех этих методов понятия локальной и глобальной ошибок, порядка и области устойчивости также имеют смысл (см. книгу [Ortega, Poole, 1981]). Для наших целей будет достаточно знать, что, как правило, удается установить соотношение вида

$$y(t_{k+1}) - y_{k+1} = a_k(y(t_k) - y_k) + L_k,$$

подобное формуле для глобальной ошибки из § 5. Множителем перехода

служит a_k; для метода Эйлера это $1 + hJ$. Величина L_k называется *локальной ошибкой*; для метода Эйлера это $h^2y''/2$. Как и выше, локальная ошибка представляет собой ошибку в точке t_{k+1} в предположении, что все величины, по которым вычисляется y_{k+1}, заданы точно. Используя обозначение O (см. § 4 гл. 2), можно сказать, что локальная ошибка метода Эйлера составляет $O(h^2)$. Вообще, если локальная ошибка может быть записана в виде $O(h^{p+1})$, то говорят, что это *метод p-го порядка*. (Не путайте это понятие с порядком дифференциального уравнения.) Метод Эйлера является методом первого порядка. Если локальная ошибка содержит $(p + 1)$-ю степень, то почему бы не воспользоваться названием «метод $(p + 1)$-го порядка»? Дело в том, что, применяя метод p-го порядка, мы при достаточно малом шаге часто получаем глобальную ошибку, пропорциональную h^p.

Ныне во многих программах не используется постоянный шаг, однако понятия локальной ошибки и порядка метода сохраняют свое значение.

(1) Эти понятия показывают, что при том же шаге метод более высокого порядка обычно обеспечивает бóльшую точность по сравнению с методом меньшего порядка.

(2) Повторное вычисление от точки t_0 к T с новым шагом $h' < h$ должно сократить ошибку в точке T примерно в $(h/h')^p$ раз. Это можно взять за основу эффективной оценки ошибки и стратегии выбора шага.

Пример 8.7. Метод Эйлера является методом первого порядка.

Рассмотрим таблицу из примера 8.4, в которой приведена ошибка метода Эйлера в точке $t = 1$ при различных шагах сетки. Видно, что в интервале $h \leqslant 0.001$ каждое десятикратное сокращение шага приводит к соответствующему десятикратному сокращению глобальной ошибки. □

8.7. Подпрограмма SDRIV2

SDRIV – это родовое имя семейства программ Каханера и Сазерленда, описанных в статье [Boisvert et al., 1984] и предназначенных для решения широкого круга начальных задач для обыкновенных дифференциальных уравнений. Существует три родственных пакета программ: SDRIV, DDRIV и CDRIV, рассчитанных на одинарную и двойную точность, а также на комплексную арифметику. Здесь мы представим только одну программу SDRIV2 из первого пакета. Эта программа способна решать уравнения вида $y' = f(t, y)$ многозначными методами, которые будут описаны ниже. Она специально разработана для решения жестких задач, но подходит и для нежестких задач. В рамках пакета подпрограмма SDRIV2 представляется средней по сложности и гибкости интерфейса. Чтобы воспользоваться этой подпрограммой, пользователю нужно написать вызывающую программу и подпрограмму, вычисляющую значения правых частей уравнения. Для последней здесь используется буква F, хотя можно выбрать и любое другое имя.

Некоторые из параметров подпрограммы SDRIV2 (такие, как WORK, LENW) и способ использоваия оператора EXTERNAL уже знакомы читателю по предыдущим главам. Однако на некоторые другие входные параметры следует обратить особое внимание.

MINT – указатель метода. В § 16 мы опишем реализованные методы. Задавайте MINT = 1 в случае нежесткой и MINT = 2 в случае жесткой задачи. Если вы затрудняетесь классифицировать свою задачу, положите MINT = 2.

EPS, EWT – точность и нуль задачи. Первый параметр задает порог погрешности при интегрировании. Задав EWT > 0, пользователь сообщает, что нет необходимости отслеживать с высокой точностью компоненты решения, если их величины не превосходят EWT. Мы называем это «нулем задачи». Поскольку требование, чтобы подпрограмма SDRIV2 отслеживала компоненты решения, не превосходящие некоторый физически обоснованный уровень, может обойтись дорого, пользователю имеет смысл обдумать, какую величину задать в качестве EWT. Значение EWT = 0.0 указывает на требование достижения относительной точности EPS для всех компонент решения.

NROOT, G – количество корней и корневая функция. Подпрограмма SDRIV2 требует, чтобы пользователь задал интервал [T, TOUT], на котором следует решить задачу, и возвращает решение в точке TOUT. В определенных случаях требуется знать решение в некоторой специальной точке $t <$ TOUT, которая заранее неизвестна. Даже более того, нахождение самой такой точки может явиться основной задачей. На этот случай пользователю SDRIV2 предоставлена возможность «найти корень», т. е. завершить процесс интегрирования еще до точки TOUT, как только написанная им функция, названная здесь G, сменит знак. Имя этой подпрограммы, выбранное пользователем, указывается в конце списка параметров подпрограммы SDRIV2. Если такая возможность не используется (NROOT = 0), то в вашем операторе вызова SDRIV2 последним параметром можно указать имя подпрограммы, вычисляющей правые части ОДУ. Мы назвали эту подпрограмму F.

Во многих практических задачах уравнения содержат какие-либо параметры. Если параметры известны заранее, то их можно учесть в F посредством оператора присваивания, DATA или PARAMETER. Если они меняются от задачи к задаче, то в рамках одного расчета их значения нужно передавать в F из вызывающей программы. Всегда есть возможность ввести фортрановский оператор COMMON, но многие пакеты предоставляют и некоторые другие возможности. Подпрограмма SDRIV2 реализует следующий вариант. Как правило, массив Y инициализируется в вызывающей программе, в него помещают N компонент начальных значений для решаемой системы, и этот массив через SDRIV2 передается в F. Если размер массива Y описан большим, чем N,

то такая вызывающая программа может разместить в Y после его N-го элемента те параметры, которые должны использоваться в F. Один из недостатков такого подхода заключается в том, что для ссылок на параметры нужно использовать переменную Y с индексом, например Y(N + 1), а не более подходящие имена.

В качестве иллюстрации рассмотрим движение тела массы m, падающего с высоты $H > 0$. ОДУ, описывающее высоту как функцию времени, записывается в виде

$$y'' = -32 + \frac{a(t)}{m}, \quad 0 \leqslant t \leqslant H, \quad y(0) = H, \quad y'(0) = 0.$$

Функция $a(t)$ соответствует сопротивлению воздуха. Например, мы можем предположить, что $a(t) = -y'(t)$, т. е. сопротивление равно скорости, и взять $H = 10$. Сначала преобразуем уравнение в систему с $y \equiv y_1$:

$$\begin{aligned} y_1' &= y_2, & y_1(0) &= H, \\ y_2' &= -32 - y_2/m, & y_2(0) &= 0, \end{aligned} \qquad 0 \leqslant t \leqslant H.$$

Задача состоит в том, чтобы определить время, через которое тело достигнет земли. Воспользуемся «корневой функцией» $g(t, y_1, y_2) = y_1$; интегрирование завершится, как только эта функция, вначале равная 10, обратится в нуль.

```
C Пример использования подпрограммы SDRIV2
C
C Обратите внимание, что NROOT равняется 1
C
      PARAMETER (N = 2, H = 10.0, NROOT = 1, MINT = 2,
     *           LW = N*N + 10*N + 2*NROOT + 204, LIW = 23)
      REAL       Y(N + 1), W(LW), MASS
      INTEGER    IW(LIW)
      EXTERNAL   FSUB, GFUN
      DATA       MASS/0.125/
C
      EPS = 1.E-5
C Задать начальную точку
      T = 0.
      TOUT = H.
C Получить одинаковую относительную точность для всех
C компонент
      EWT = 0.0
C Задать начальные условия
      Y(1) = H
      Y(2) = 0.0
C Задать параметр
      Y(3) = MASS
```

```
      MS = 1
      WRITE(*, *)'T,       Y(1),       Y(2),       MS'
C
  10  CALL SDRIV2
     * (N,T,Y,FSUB,TOUT,MS,NROOT,EPS,EWT,MINT,W,LW,IW,
     *  LIW,GFUN)
      TOUT = TOUT + 0.1
      IF (MS .EQ. 5) THEN
        WRITE(*,'(3F11.5,I4,A,F11.5)')
     *  T,Y(1),Y(2),MS,' <---Y = 0 ПРИ T = ', T
        STOP
      ELSE
        WRITE(*,'(3F11.5,14)') T,Y(1), Y(2), MS
C Стоп, если код выхода не равен 1 или 2
        IF(MS.GT.2)STOP
      END IF
      GO TO 10
      END
C
      SUBROUTINE FSUB (N,T,Y,YDOT)
C Подпрограмма вычисления правых частей уравнения
        REAL T, Y(*), YDOT(*), MASS, G
        DATA G/32.0/
C Возвратить параметр
        MASS = Y(3)
C
        YDOT(1) = Y(2)
        YDOT(2) = -(G + 1.0/MASS*Y(2))
        RETURN
      END
C
      REAL FUNCTION GFUN(N,T,Y,IROOT)
C Подпрограмма-функция для поиска корня
C Интегрирование прекращается, как только GFUN сменит знак
        REAL Y(*)
        GFUN = Y(1)
        RETURN
      END
C
C     Типичные результаты
C     T,          Y(1),        Y(2),      MS
C     0.00000   10.00000      0.00000    1
C     0.10000    9.87534    -2.20270    2
C                   .
C                   .
C                   .
```

```
C
C     2.60000     0.10000    -4.00000    2 (конечная скорость)
C     2.62500     0.00000    -4.00000    5 < - - Y = 0 ПРИ T = 2.62500
```

8.8. Неявные методы

Напомним (см. § 5), что для метода Эйлера область устойчивости представляет собой интервал $(-2,0)$. Этого явно недостаточно, чтобы интегрировать жесткие задачи с приемлемым для практики шагом. Подобные ограничения существуют и для многих других методов. Чтобы показать, как можно обойти такие трудности, мы введем *неявные* методы, для которых области устойчивости могут быть неограниченными.

Альтернативный подход к выводу метода Эйлера заключается в формальном интегрировании уравнения $y' = f(t, y)$, которое дает

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(t, y(t))\,dt,$$

и аппроксимации интеграла выражением $(t_{n+1} - t_n)f(t_n, y(t_n))$. Можно применять и другие квадратурные формулы, например можно взять правый узел или провести усреднение по двум узлам. Это приводит к следующим формулам интегрирования:

$$y_{n+1}^{\mathrm{BE}} \equiv y_n + h_n f(t_{n+1}, y_{n+1}^{\mathrm{BE}}),$$

$$y_{n+1}^{\mathrm{TR}} \equiv y_n + \frac{h_n}{2}[f(t_n, y_n) + f(t_{n+1}, y_{n+1}^{\mathrm{TR}})],$$

или, что эквивалентно,

$$y_{n+1}^{\mathrm{BE}} \equiv y_n + h_n y'_{n+1}, \quad y_{n+1}^{\mathrm{TR}} \equiv y_n + \frac{h_n}{2}[y'_n + y'_{n+1}],$$

которые описывают соответственно *неявный метод Эйлера* (BE – backward Euler method) и метод трапеций (TR – trapezoidal rule). Возможен и другой подход, более близкий к геометрическому. Можно записать формулу для прямой $l(t)$ с наклоном m, проходящей через точку $(t_n, y(t_n))$:

$$l(t) - y(t_n) = m(t - t_n).$$

Полагая $t = t_{n+1}$ и подставляя y_{n+1} вместо $l(t_{n+1})$, получаем

$$y_{n+1} = y(t_n) + mh.$$

Определяя m разными способами, приходим к различным формулам для y_{n+1}. В качестве m можно использовать $y'(t_n)$ для метода Эйлера, $y'(t_{n+1})$ для неявного метода Эйлера, а также среднее этих значений для метода трапеций. Рис. 8.6 иллюстрирует эти подходы.

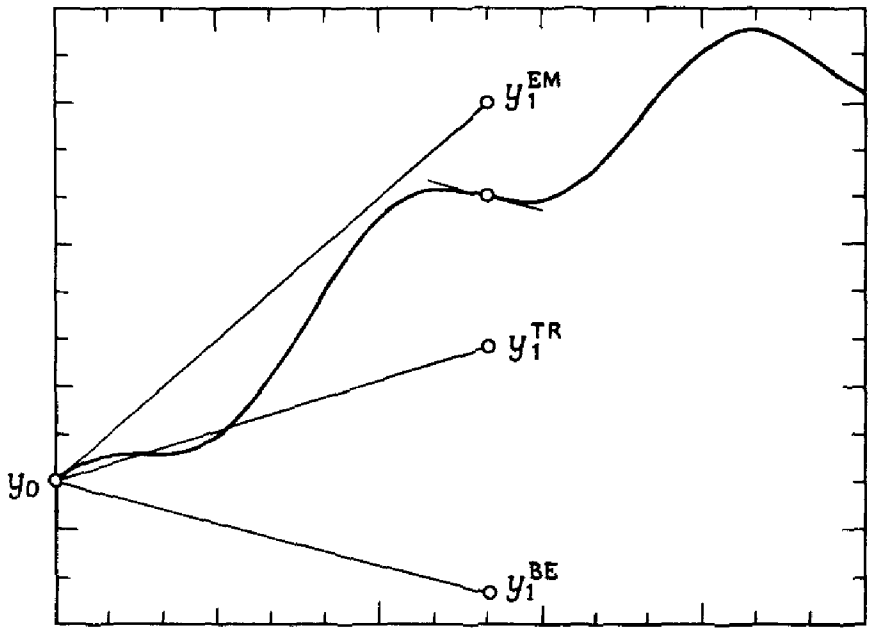


Рис. 8.7. Неявный метод Эйлера при решении жесткой задачи.

Пример 8.8. Неявный метод Эйлера.

В примере 8.4 воспользуемся неявным методом Эйлера. Имеем

$$y_{n+1}^{BE} = y_n + h_n y'^{BE}_{n+1} = y_n + h_n[-10(t_{n+1} - 1)y_{n+1}^{BE}],$$

или

$$y_{n+1}^{BE} = \frac{y_n}{1 + 10h_n(t_{n+1} - 1)}.$$

В следующей таблице приводятся соответствующие ошибки, полученные при расчете с тем же, что и ранее, шагом 0.1.

Обращаясь к рис. 8.3, мы видим, что при $t \leqslant 1$ пребывание в области над

Таблица 8.3

t_n	Ошибка	t_n	Ошибка
0.1	0.050	1.1	15.93
0.2	0.337	1.2	13.25
0.3	1.123	1.3	10.18
0.4	2.642	1.4	7.28
0.5	5.328	1.5	4.87
0.6	8.909	1.6	3.06
0.7	12.731	1.7	1.81
0.8	15.899	1.8	1.01
0.9	17.617	1.9	0.54
1.0	17.570	2.0	0.27

правильным решением ведет к бо́льшим ошибкам, чем в области ниже решения. Исследуя рис. 8.6, убедитесь, что метод BE помещает y_1 выше $y(t_1)$, тогда как метод EM – наоборот. Вот почему для данного примера следует ожидать, что неявный метод Эйлера окажется менее точным, чем явный метод. Наш прогноз подтверждается погрешностями, приведенными в таблице. □

Численный метод, подобный BE или TR, в котором вычисляемое значение y_{n+1} входит в правую часть f, называется *неявным*; если метод не обладает этим свойством, он называется *явным*. В примере 8.8 нам удалось выразить y_{n+1} в явном виде. Но, как правило, это не удается. Например, пусть мы хотим решить неявным методом Эйлера задачу $y' = \sin y$, $y(0) = 1$. Чтобы определить значение y_1^{BE}, потребуется решить уравнение

$$y_1^{\mathrm{BE}} = 1 + h \sin y_1^{\mathrm{BE}} = y_0 + h \sin y_1^{\mathrm{BE}}.$$

В случае метода трапеций мы придем к уравнению

$$y_1^{\mathrm{TR}} = y_0 + \frac{h}{2}(\sin y_1^{\mathrm{TR}} + \sin y_0).$$

В обоих случаях уравнение можно записать в общей форме

$$y_{n+1} = y_n + \beta h f(t_{n+1}, y_{n+1}) + g,$$

причем последнее слагаемое g не зависит от y_{n+1}. Это нелинейное уравнение; выразить y_{n+1} в явном виде не удается, и для его отыскания требуются специальные методы, которым была посвящена гл. 7. Поэтому, в сравнении с явным методом Эйлера, неявность влечет за собой более сложный и дорогой алгоритм перехода от t_n к t_{n+1}. Так к чему же эти усложнения?

Ответ в том, что неявные методы, как правило, более устойчивы, чем явные. Под этим мы подразумеваем, что область устойчивости метода может быть велика, в частности, она может быть неограниченной. Так, для методов BE и TR интервалом устойчивости является полуось $(-\infty, 0)$. Это означает, что у этих методов множитель перехода будет меньше единицы при любом положительном шаге h (в случае устойчивой задачи) и, следовательно, ошибки не будут нарастать. Такие методы называются *абсолютно устойчивыми*. Однако это не значит, что ошибки будут отсутствовать или что шаг можно выбрать сколь угодно большим. Подобно методу Эйлера, неявные методы обладают глобальной ошибкой, представимой в виде суммы двух членов, а именно локальной ошибки и переносимой со множителем перехода старой глобальной ошибки. Например, глобальную ошибку для неявного метода Эйлера можно записать в виде

$$y_{n+1}^{\mathrm{BE}} - y(t_{n+1}) = (1 - hJ)^{-1}([y_n - y(t_n)] + \frac{h^2}{2} y''(\xi_n)) =$$
$$= (1 - hJ)^{-1}[y_n - y(t_n)] + O(h^2).$$

Множитель перехода $(1 - hJ)^{-1}$ по модулю меньше единицы, если $J < 0$. Выбор шага h влияет также на величину локальной ошибки. Устойчивость лишь означает, что ошибки не возрастают. Иногда говорят, что для устойчивого метода выбор шага определяется соображениями, связанными целиком с локальной ошибкой.

Исследование формулы для глобальной ошибки неявного метода Эйлера показывает, что это метод первого порядка, как и явный метод Эйлера.

Пример 8.9. Неявный метод Эйлера имеет первый порядок и является устойчивым.

В следующей таблице приводятся ошибки в точках $t = 2$ и $t = 6$, полученные при решении неявным методом Эйлера задачи из примера 8.4. Эти результаты, вычисленные при различных шагах сетки, подтверждают заключение, что неявный метод Эйлера имеет первый порядок.

Таблица 8.4

h	$\lvert y_n - y(2)\rvert$	$\lvert y_n - y(6)\rvert$
10^{-1}	$2.7 \cdot 10^{-1}$	$8 \cdot 10^{-25}$
10^{-2}	$1.8 \cdot 10^{-3}$	$9 \cdot 10^{-48}$
10^{-3}	$1.6 \cdot 10^{-4}$	$3 \cdot 10^{-54}$
10^{-4}	$1.6 \cdot 10^{-5}$	$1 \cdot 10^{-55}$
10^{-5}	$1.6 \cdot 10^{-6}$	$1 \cdot 10^{-56}$
10^{-6}	$1.6 \cdot 10^{-7}$	$1 \cdot 10^{-57}$

■

Можно показать, что глобальная ошибка метода трапеций представима в виде

$$y_{n+1}^{\mathrm{TR}} - y(t_{n+1}) = (1 - 0.5hJ)^{-1}(1 + 0.5hJ)(y_n - y(t_n)) + O(h^3).$$

Множитель перехода

$$(1 - 0.5hJ)^{-1}(1 + 0.5hJ)$$

будет меньше единицы при $J < 0$. Итак, метод TR также абсолютно устойчив. Если $y_n = y(t_n)$, то $y_{n+1}^{\mathrm{TR}} - y(t_{n+1}) = O(h^3)$, и, таким образом, метод трапеций имеет второй порядок.

В § 6 мы говорили о том, что более высокий порядок предпочтительнее *в случае достаточно малого шага сетки.* Отсюда следует, что для решения нежестких задач лучше выбрать метод трапеций, а не методы Эйлера. Вы сможете убедиться в этом сами, решив задачу 8.10. Для жестких задач явный метод Эйлера при разумных шагах неприменим, так как он неустойчив. Оба неявных метода TR и BE устойчивы, т. е. ошибки не нарастают, но в этом случае порядок метода не может служить хорошим показателем того, какой метод точнее. Так, у метода

TR порядок выше, но это имеет значение лишь при столь малых шагах, что они уже не представляют интереса для решения жестких задач. Более важно, что в случае больших и отрицательных J множитель перехода для метода TR близок к -1. Вот почему погрешности на соседних шагах меняют знак и уменьшаются лишь слегка. Напротив, у неявного метода Эйлера множитель перехода гораздо меньше 1 и положителен, поэтому ошибки быстро убывают и не осциллируют. Можно заключить, что, хотя метод TR имеет второй порядок и устойчив, этот метод недостаточно точен при интегрировании жестких задач.

Рисунок 8.7 иллюстрирует применение неявного метода Эйлера к решению жесткой задачи. Отметим, что наклон ломаной вычисляется в последующий момент времени, когда он меньше, чем в данный момент, и это помогает методу «приблизиться» к кривой искомого решения. Рассмотрев также рисунок для явного метода Эйлера (рис. 8.5 b), вы увидите, почему метод TR менее полезен для решения жестких задач.

Печально то, что для всех известных явных методов справедливо ограничение на шаг сетки вида $|hJ| <$ малой константы, в то время как для многих неявных методов такого ограничения нет. Это оправдывает особый интерес к последним методам. Однако не следует думать, что все неявные методы обладают хорошими характеристиками по устойчивости. Неявность необходима, но определенно не является достаточной. Верно и обратное: устойчивый метод не всегда точен, что показывает обсуждение методов BE и TR.

Вернемся к нелинейному уравнению, которое нужно решить, чтобы найти y_{n+1}^{BE} или y_{n+1}^{TR}, и, пользуясь обозначениями из гл. 7, запишем

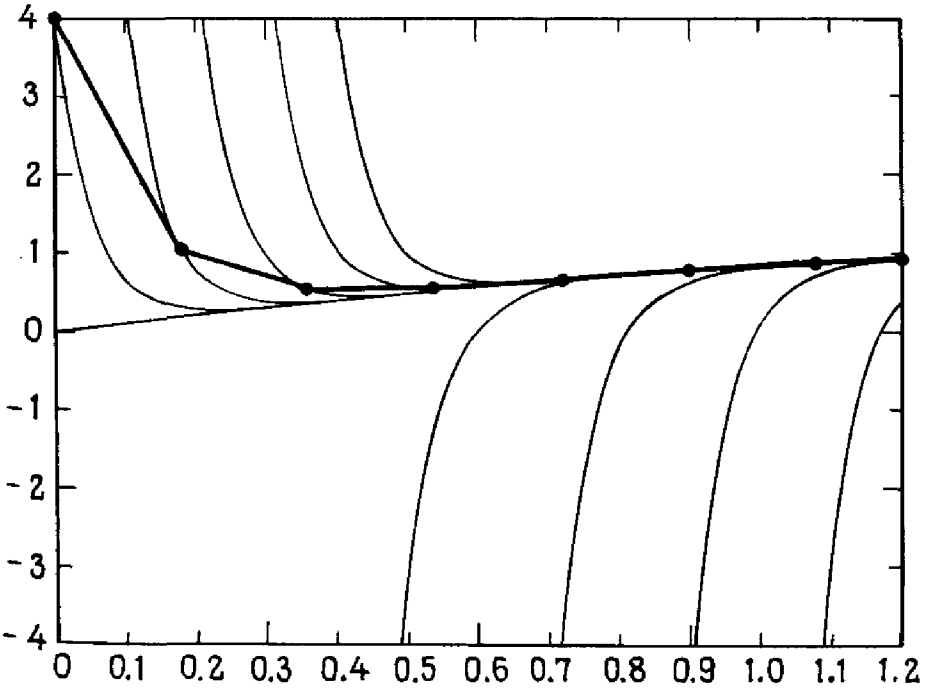


Рис. 8.6. Графическая интерпретация методов EM, BE и TR.

$$0 = F(y) \equiv y - y_n - \beta h f(t_{n+1}, y) - g.$$

Как мы уже знаем, подобные уравнения решаются с помощью какого-либо итерационного процесса, например бисекции, метода Ньютона и т. п. Следует выбрать начальное приближение к y_{n+1}, которое мы обозначим через $y_{n+1}^{(0)}$, а затем последовательно вычислять итерационные приближения $y_{n+1}^{(1)}, \ldots, y_{n+1}^{(m)}$. Мы надеемся, что m не окажется слишком большим, но это, очевидно, определяется самой задачей и начальным приближением.

Среди универсальных методов, описаных в гл. 7, метод Ньютона является одним из наиболее эффективных. Однако в данном случае $F(y)$ – это не просто произвольная функция. Она обладает той особенностью, что неизвестное y не только входит в f, но и входит туда как отдельное слагаемое. Это приводит к методу, называемому *функциональной итерацией* (или *методом последовательных приближений.* – *Перев.*):

$$y_{n+1}^{(k+1)} = y_n + \beta h f(t_{n+1}, y_{n+1}^{(k)}) + g.$$

Метод Ньютона и функциональная итерация анализировались на предмет скорости их сходимости и тех трудностей, что возникают при их алгоритмической реализации. Некоторые из этих вопросов освещены в § 12.

*8.8.1. Устойчивость неявного метода Эйлера и метода трапеций для систем ОДУ

Все результаты по устойчивости, приведенные в предыдущем разделе, распространяются на случай системы уравнений, подобно обобщениям метода Эйлера из разд. 5.2. Например, чтобы неявный метод Эйлера был устойчив, должно выполняться неравенство $|1 - h\lambda|^{-1} < 1$, где λ – произвольное (комплексное) собственное значение якобиана. До тех пор пока комплексное число $h\lambda$ располагается вне единичного круга с центром в точке $z = 1$, множитель перехода будет меньше единицы. Поэтому областью устойчивости для метода BE является вся комплексная плоскость, за исключением круга радиуса 1 с центром в $z = 1$. В частности, для устойчивой задачи (собственные значения расположены в левой полуплоскости) неявный метод Эйлера устойчив.

Метод трапеций будет устойчивым, когда множитель перехода меньше единицы, т. е.

$$\left| \frac{1 + 0.5h\lambda}{1 - 0.5h\lambda} \right| < 1.$$

Областью устойчивости для метода TR является вся левая полуплоскость. Это можно показать геометрически, интерпретируя множитель перехода как отношение двух расстояний; в числителе фигурирует расстояние от точки $z = 0.5h\lambda$ до -1, а в знаменателе – расстояние от той же точки z до $+1$. Когда точка z расположена ближе к -1, чем к $+1$, отношение меньше единицы.

8.9. Многошаговые методы

В § 2 мы уже пояснили, что в многошаговом методе для перехода от t_n к t_{n+1} используются значения решения (а возможно, и производной), вычисленные в нескольких предшествующих узлах. В типичном трехшаговом методе для вычисления y_{n+1} используются решение и его производная в узлах t_n, t_{n-1} и t_{n-2}. Этот метод записывается в виде

$$y_{n+1} = \alpha_1 y_n + \alpha_2 y_{n-1} + \alpha_3 y_{n-2} + \\ + h[\beta_0 f(t_{n+1}, y_{n+1}) + \beta_1 f(t_n, y_n) + \beta_2 f(t_{n+1}, y_{n-1}) + \beta_3 f(t_{n-2}, y_{n-2})].$$

Константы α_i и β_i подбираются априори. Если $\beta_0 = 0$, то метод будет явным; ненулевое β_0 приводит к неявному методу. Аналогичным образом описываются k-шаговые методы общего вида. Коэффициенты для двух трехшаговых методов приводятся в примере 8.10.

В явном многошаговом методе для получения приближенного решения в точке t_{n+1}, вне зависимости от числа шагов в формуле, требуется вычислить только одно значение правой части, а именно значение $f(t_n, y_n)$. Неявный метод требует одного вычисления на каждой итерации, но при наличии быстрой сходимости понадобится немного вычислений. В сравнении с одношаговыми методами, многошаговые алгоритмы могут быть более эффективными при той же точности или более точными при том же объеме вычислений.

В большинстве неявных многошаговых алгоритмов реализованы или метод функциональной итерации, или метод Ньютона. На каждом шаге в качестве начального берется приближение, получаемое по явной многошаговой формуле, которое затем уточняется в результате итераций и принимается за решение в точке t_{n+1}. По соображениям точности и простоты реализации оба алгоритма должны быть совместимы, т. е. использовать одни и те же данные и быть сравнимыми по точности.

Уже многие годы многошаговые методы являются стандартным средством решения начальных задач. Однако при их реализации приходится сталкиваться с серьезными трудностями.

(1) Для трехшагового метода необходимы решения в двух предшествующих точках. На первом шаге интегрирования известно лишь y_0. Один из возможных рецептов – воспользоваться одношаговым методом, чтобы вычислить y_1 и y_2, а затем переключиться на трехшаговый метод.

(2) Многошаговые методы в том виде, как они были описаны, могут применяться только с постоянным шагом h при интегрировании от t_0 к T; одношаговый метод допускает переменный шаг.

Достоинства, внутренне присущие многошаговым методам, позволили создать программы, обходящие трудности (1) и (2); в некоторых из этих программ используются чисто эвристические приемы. Раздел, посвященный этим методам, есть почти в каждом учебнике по численным методам. В 1962 г. Нордсик предложил другой подход, получивший

название *многозначного метода* (его также часто называют методом Нордсика. – *Перев.*), в котором элегантно преодолены обе проблемы. Важное значение имела и связанная с этим работа Крога [Krogh, 1970]. Ныне большинство коммерческих программ, таких, как SDRIV, реализуют подход Нордсика или его вариации. Однако было показано, что эти два подхода, хотя и кажутся разными, по существу связаны с одним и тем же процессом (см., например, [Osborne, 1966] или [Skeel, 1979]). Обзор некоторых результатов на эту тему приводится в §§ 14–19. Если вас интересует только использование SDRIV2 или похожей программы, то эти параграфы можно при чтении пропустить.

*8.10. Порядок и погрешность многошаговых методов

Вспомним, что в § 6 мы обсуждали понятия порядка, устойчивости и ошибки для одношаговых методов. Эти понятия можно распространить и на многошаговые методы. Многошаговый метод имеет p-й порядок, если он точен для задачи, решением которой является полином степени p. Как и следовало ожидать, увеличение числа шагов в формуле позволяет увеличить порядок метода. Если же метод p-го порядка применяется к задаче, решение которой не является полиномом, то возникает ошибка. Если даже все предыдущие значения решения были вычислены точно, появляется локальная ошибка, называемая так по аналогии с одношаговым методом. Для метода p-го порядка локальная ошибка записывается в виде

$$L_k = ch^{p+1}y^{(p+1)}(\xi_k) = O(h^{p+1}).$$

Естественно, глобальная ошибка складывается из локальной ошибки и предшествующих ошибок, взятых со множителями перехода.

Определение порядка метода позволяет указать способ выбора коэффициентов. Например, если $y(t) \equiv 1$, т.е. интегрируется задача $y' = 0$ и $y(0) = 1$, то для метода p-го порядка ($p > 0$) ошибка должна равняться нулю. Для трехшагового метода из § 9 это означает, что

$$\alpha_1 + \alpha_2 + \alpha_3 = 1.$$

Аналогично, при $y'(t) = (t^k)' = kt^{k-1}$, $k = 1, 2, \ldots, p-1$, ошибка также должна обращаться в нуль. Это приводит к системе из p линейных уравнений относительно констант α_i, β_j, и если выбрать p равным числу этих констант, то можно надеяться найти единственное решение данной системы. Очень часто некоторые из этих констант задаются заранее, а остальные подлежат вычислению.

Пример 8.10. Вывод двух многошаговых методов.

Выведем трехшаговый метод наивысшего порядка при условиях (а) $\alpha_2 = \alpha_3 = 0$ и (б) $\beta_1 = \beta_2 = \beta_3 = 0$.

В случае (а) имеем пять неизвестных констант. Они определяются исходя из требования точного интегрирования пяти уравнений $y' = kt^{k-1}$,

$k = 0, 1, \ldots, 4$. При $n = 0$ получим следующую систему из пяти уравнений:

$$\begin{aligned}
&k = 0, \quad y(t) = 1, \quad && 1 = \alpha_1,\\
&k = 1, \quad y(t) = t, \quad && 1 = \beta_0 + \beta_1 + \beta_2 + \beta_3,\\
&k = 2, \quad y(t) = t^2, \quad && 1 = 2[\beta_0 + 0 - \beta_2 - 2\beta_3],\\
&k = 3, \quad y(t) = t^3, \quad && 1 = 3[\beta_0 + 0 + \beta_2 - 4\beta_3],\\
&k = 4, \quad y(t) = t^4, \quad && 1 = 4[\beta_0 + 0 - \beta_2 - 8\beta_3].
\end{aligned}$$

Ее решение дает

$$\alpha_1 = 1, \quad \beta_0 = \frac{9}{24}, \quad \beta_1 = \frac{19}{24}, \quad \beta_2 = -\frac{5}{24}, \quad \beta_3 = \frac{1}{24}.$$

Мы получили неявный метод четвертного порядка. Аналогично, если положить $\beta_0 = 0$, то можно найти константы, определяющие явный трехшаговый метод третьего порядка.

В случае (б) мы имеем четыре неизвестных и можем надеяться построить неявный трехшаговый метод третьего порядка. Приведем результат:

$$\beta_0 = \frac{6}{11}, \quad \alpha_1 = \frac{18}{11}, \quad \alpha_2 = -\frac{9}{11}, \quad \alpha_3 = \frac{2}{11}. \ \square$$

Зная порядок многошагового метода, можно сделать некоторые выводы о его локальной ошибке. В частности, мы знаем, что метод, описанный в примере 8.10 (а), имеет четвертый порядок. Следовательно, этим методом нельзя точно проинтегрировать уравнение $y' = 5t^4$. Используя этот факт, можно показать, что для данного метода

$$L_k = -\frac{19}{720} h^5 y^{(5)}(\xi_n).$$

8.11. Устойчивость многошаговых методов

Многошаговые методы весьма привлекательны, поскольку среди них можно найти методы сколь угодно высокого порядка. Однако, реализовав явный многошаговый метод, мы убедимся в его неустойчивости для многих устойчивых задач, если только шаг сетки не слишком мал. Этому не следует удивляться, особенно в свете замечения из § 8 о том, что явные методы не всегда устойчивы. Но удивительно то, что подобное поведение наблюдается и у неявных методов. В 1963 году Дальквист доказал, что никакой многошаговый метод не может быть абсолютно устойчивым (см. § 8), если его порядок выше второго. Здесь даже неявность не способна помочь.

Хотя это действительно разочаровывает, не стоит отчаиваться. К счастью, у большинства реальных задач характерные времена, т. е. величины J, не могут быть произвольными. Напомним, что эти величины определяют, насколько устойчива система. Для жестких задач

$J \ll 0$, а у нежестких задач значения J более умеренны. Вот почему представляется практичным искать методы, устойчивые не для всех, а лишь для определенных классов задач. В этом направлении были приложены гигантские усилия, и одним из наиболее плодотворных результатов явилось семейство *жестко-устойчивых* методов (формул) Гира [1]). Мы приведем одну из этих формул в § 16. Формулы Гира оказываются эффективными при интегрировании жестких задач, если только их решения не являются осциллирующими функциями. Для задач с сильно осциллирующими компонентами решения формулы Гира тоже можно использовать, но с меньшим шагом интегрирования.

*8.12. Метод функциональной итерации и метод Ньютона для решения неявных уравнений

Неявные методы решения ОДУ (такие, как BE или TR) приводят к уравнениям вида

$$y_{n+1} = y_n + \beta h f(t_{n+1}, y_{n+1}) + g, \text{ или } 0 = F(y) \equiv y - y_n - \beta h f(t_{n+1}, y) - g.$$

Двумя наиболее популярными алгоритмами их решения являются метод функциональной итерации

$$y_{n+1}^{(k+1)} = y_n + \beta h f(t_{n+1}, y_{n+1}^{(k)}) + g$$

и метод Ньютона

$$y_{n+1}^{(k+1)} = y_{n+1}^{(k)} - [F'(y_{n+1}^{(k)})]^{-1} F(y_{n+1}^{(k)}), \quad F'(y) = I - \beta h \frac{\partial f(t_{n+1}, y)}{\partial y}.$$

Обсудим важнейшие свойства каждого из этих методов.

*8.12.1. Функциональная итерация

На каждой итерации вычисляется одно значение правой части f. В ранних стратегиях процесс продолжали до совпадения соседних итерационных приближений с машинной точностью; в таком случае

[1]) Чарльз Вильям (Билл) Гир (1935) получил математическое образование в Кеймбридже (ныне работает в Иллинойсском университете). Его имя стало весьма популярным среди инженеров после того, как в 1971 году появилась его книга, в которой не только был описан новый класс методов решения жестких задач, но и были представлены тексты программ этих методов. Многие из программ, используемых ныне для серийных расчетов (включая и SDRIV), восходят к программам Гира. Очень скоро после выхода книги Гира задачи, прежде не поддававшиеся решению, стали обрабатываться рутинным образом. Эта работа Гира привлекла столь большое внимание, что затмила важный вклад, внесенный им в создание систем автоматического решения задач, моделирование и сетевой анализ. Впрочем, в 1986 году сообщество ученых-вычислителей признало большие достижения Гира, избрав его президентом своей наиболее влиятельной организации – Общества промышленной и прикладной математики (Society for Industrial and Applied Mathematics - SIAM).

говорили, что сходимость достигнута. Но даже в случае полной сходимости ($m \approx \infty$) в результате получается y_{n+1}, а не $y(t_{n+1})$. Поскольку y_{n+1} едва ли будет точным решением, нет смысла вычислять это значение более точно, чем необходимо. Для нежестких задач подход, принятый в настоящее время, предусматривает выполнение фиксированного числа итераций, обычно одной или двух, и завершение процесса, независимо от того, насколько далеко до сходимости. Приближение $y_{n+1}^{(0)}$ называют *предикцией* (предсказанием) и обозначают символом P, вычисление f обозначают через E, а вычисление $y_{n+1}^{(1)}$ называют *коррекцией* и обозначают через C. Как правило, после заключительной коррекции снова вычисляется правая часть, чтобы оценить производную $y'(t_{n+1}) \approx f(t_{n+1}, y_{n+1}^{(m)})$. Таким образом, одна итерация кодируется аббревиатурой PECE, двум итерациям соответствует код PECECE и т. д. Описанный метод носит название *предиктор-корректор*.

Пример 8.11. Неявный метод Эйлера с функциональной итерацией. Воспользуемся методом Эйлера как предиктором на каждой итерации, т. е. пусть $y_{n+1}^{(0)} \equiv y_n + hf(t_n, y_n)$, и применим неявный метод Эйлера с функциональной итерацией для решения следующей задачи:

$$y' = \sin(y), \quad y(0) = 1, \quad 0 \leqslant t \leqslant 2.$$

Для нашего уравнения метод BE записывается в виде

$$y_{n+1} = y_n + h \sin y_{n+1}.$$

Возьмем шаг $h = 0.5$. В таблице представлены результаты для каждого узла. Процесс коррекции продолжался до тех пор, пока не выполнялось неравенство

$$|y_{n+1}^{(k+1)} - y_{n+1}^{(k)}| \leqslant 10^{-6} |y_{n+1}^{(k+1)}|.$$

В первых строках указаны предикции, а далее все коррекции, проделанные вплоть до достижения сходимости алгоритма функциональной итерации.

Таблица 8.5

Точка предикции	Итерационные приближения	Точка предикции	Итерационные приближения
$t = 0.0$	1.420735492	$t = 0.5$	1.997402250
	1.494380992		1.953888825
	1.498540884		1.962457688
	1.498695355		1.960839202
	1.498700925		1.961147505
	1.498701126		1.961088870
			1.961100025
			1.961097903
			1.961098306

Точка предикции	Итерационные приближения	Точка предикции	Итерационные приближения
$t = 1.0$	2.423495411	$t = 1.5$	2.689656949
	2.290074794		2.543731664
	2.337238957		2.606815696
	2.321289573		2.580202380
	2.326774589		2.591559605
	2.324898708		2.586735507
	2.325541495		2.588788747
	2.325321383		2.587915590
	2.325396774		2.588287043
	2.325370953		2.588129046
	2.325379797		2.588196254
	2.325376768		2.588167666
	2.325377805		2.588179827
			2.588174654
			2.588176854

Методы типа предиктор-корректор были описаны Милном в 1953 г. и широко применялись начиная с середины 60-х годов [1]). В середине 70-х итерации до достижения сходимости были вытеснены фиксированным числом итераций PECE. Такие итерации эффективны при решении большинства задач, но обладают одним крупным недостатком. Метод PECE с фиксированным числом итераций является явным методом! Если число итераций m фиксировано заранее, то наилучшее приближение

[1]) Одним из первых пропагандистов методов типа предиктор-корректор был Ричард Хэмминг (1915) – математик, сочетающий быстрый творческий ум со взглядами, зачастую неортодоксальными и непочтительными по отношению к устоявшимся представлениям. В 44–45 годах Хэмминг участвовал в осуществлении проекта «Манхэттен» в Лос-Аламосе. В дальнейшем его научная работа была связана главным образом с компанией Bell Telephone, где он в итоге возглавил отделения численных методов и компьютерных наук. Он внес большой вклад в решение задач, важных для индустрии связи. Можно упомянуть окно Хэмминга в цифровой фильтрации, код Хэмминга для обнаружения/исправления ошибок при передаче данных и метод Хэмминга типа предиктор-корректор. Он был активным сторонником использования тригонометрических приближений в численных задачах. В предисловии к своему учебнику 1962 года «Численные методы для научных работников и инженеров» он прямо сформулировал свое кредо: «Целью вычислений является проникновение в природу вещей, а не числа». Задолго до 1970 года он изумлял пользователей – пожирателей машинного времени в научных лабораториях, заявляя, что еще до 1980 года одним из наиболее значительных применений компьютеров станут домашние развлечения. Впрочем, нам придется еще подождать, пока сбудется другое его предсказание: о создании компьютера, реагирующего на факторы окружающей среды, такие, как погода или время суток.

к y_{n+1}, обозначаемое через $y_{n+1}^{(m)}$, можно выразить по явной (хотя и громоздкой) формуле через известные величины. Например, одна итерация PECE, составленная из методов EM и BE, описывается формулой

$$y_{n+1} \approx y_{n+1}^{(1)} = y_n + hf(t_{n+1}, y_n + hf(t_n, y_n)).$$

Свойство абсолютной устойчивости уже не распространяется на $y_{n+1}^{(1)}$. Все методы PECE с фиксированным числом итераций характеризуются условием устойчивости типа

$$|hJ| < \text{const},$$

и их не следует применять для решения жестких задач. Для жестких задач фиксированного числа итераций недостаточно. Итерационный процесс должен продолжаться вплоть до достижения сходимости или, по крайней мере, достаточно долго, чтобы полностью реализовать преимущества неявного метода.

Однако мы знаем, что итерационные процессы не всегда сходятся и, к сожалению, функциональные итерации не сходятся достаточно часто. Действительно, можно показать, что если не выполнено условие

$$|\beta hJ| < 1,$$

то функциональная итерация не будет сходиться, независимо от того, насколько удачно выбрано начальное приближение. Хотя β зависит от метода, обычно это число, близкое к единице (для метода BE $\beta = 1$), и, таким образом, функциональная итерация сходится, если $|hJ|$ не превосходит малой константы, и расходится во всех других случаях. Для жестких задач это приводит к очень обременительным ограничениям на шаг сетки. Именно с таким ограничением мы столкнулись, обсуждая устойчивость явных методов, и именно такое ограничение на шаг сетки побудило нас заинтересоваться неявными методами. Однако, чтобы применить неявный метод, нужно решить нелинейные уравнения, а решая их методом функциональной итерации, мы опять сталкиваемся с подобным ограничением на шаг сетки. Выход один: следует взять другой итерационный процесс.

*8.12.2. Метод Ньютона

На каждой итерации метода Ньютона, применяемого для решения задачи $F(y) = 0$, требуется решать систему линейных уравнений с матрицей $F'(y)$. Конечно, метод Ньютона сходится не всегда. Однако можно показать, что при произвольно заданном шаге h метод Ньютона сойдется, если начальное приближение $y_{n+1}^{(0)}$ выбрано достаточно хорошо (сравните это с функциональной итерацией, для которой сходимость зависит не от качества начального приближения, а только от длины шага). Шаг сетки, очевидно, влияет на точность предсказания. Для

функциональной итерации он также определяет, будет ли иметь место сходимость, но для метода Ньютона это не так. Интуитивно мы хотели бы распоряжаться шагом сетки для контроля ошибок, а не для того, чтобы обеспечивать правильное функционирование других частей вычислительного процесса. Мы можем выбрать h так, чтобы предсказание было достаточно точным, и тогда для достижения сходимости потребуется, как правило, лишь несколько ньютоновых итераций; правда, число их заранее неизвестно. Это объясняет, почему метод Ньютона или какие-либо варианты этого метода всегда используются в качестве итерационных процессов для решения нелинейных задач, порождаемых неявными методами. Для этого требуется гораздо больше усилий, чем при использовании функциональной итерации. Мы уже обсуждали роль якобиана, но до сих пор не использовали его значений в численных алгоритмах. Теперь, чтобы реализовать метод Ньютона, нам потребуется матрица частных производных. Она должна или задаваться пользователем – тогда ему придется дифференцировать систему на бумаге, – или аппроксимироваться самой программой. (Некоторые новейшие программы пытаются анализировать фортранную подпрограмму, написанную пользователем для f, а именно правые части системы, для автоматической генерации подпрограммы, вычисляющей якобиан. Однако эти идеи еще не реализованы в серийном программном обеспечении.) В обоих случаях под эту матрицу следует отвести память. На решение системы линейных уравнений также потребуется время; и все эти действия по мере необходимости повторяются вплоть до достижения сходимости итерационного процесса.

Отметим, что $F'(y_{n+1}^{(k)})$ зависит как от h, так и от текущего приближения $y_{n+1}^{(k)}$. Эта матрица должна меняться от итерации к итерации. Пакеты, подобные SDRIV, не пересчитывают F' автоматически, если в этом нет необходимости. Опыт показал, что это не уменьшает надежности программы, но может существенно повысить ее эффективность. Не входя в детали разумной программной реализации, отметим все же, что пользование неявным методом требует значительно больше вычислений по сравнению с явными методами, но, несмотря на это, неявные методы эффективны, и в настоящее время лучшей альтернативы им нет. Почти все программы решения жестких уравнений реализуют именно неявные методы.

Пример 8.12. Неявный метод Эйлера с алгоритмом Ньютона.

Воспроизведем пример 8.11, используя метод Ньютона. Предшествующий анализ показал, что функциональная итерация будет сходиться при условии

$$|hJ| = |h \cos y| < 1 .$$

Такое условие, конечно, выполняется при $h < 1$ – здесь нет необходимости в применении метода Ньютона. Однако ради иллюстрации проведем

вычисления. Итерационная схема записывается в виде

$$y_{n+1}^{(k+1)} = y_{n+1}^{(k)} - \frac{y_{n+1}^{(k)} - y_n - h \sin y_{n+1}^{(k)}}{1 - h \cos y_{n+1}^{(k)}}.$$

В таблице для каждого шага по времени от $t = 0$ до $t = 4.5$ приведены предикции и коррекции по Ньютону. Последним приводится решение в точке $t = 5.0$. Отметим, что для метода Ньютона потребовалось меньше итераций, чем для функциональной итерации (пример 8.11).

Таблица 8.6

Точка предикции	Итерационные приближения	Точка предикции	Итерационные приближения
$t = 0.0$	1.420735492404	$t = 0.5$	1.997402267036
	1.500330663274		1.961348131480
	1.498701819848		1.961098260881
	1.498701133518		1.961098248754
$t = 1.0$	2.423495363990	$t = 1.5$	2.689656746824
	2.326570008884		2.589004843113
	2.325377690520		2.588176045381
	2.325377497789		2.588175982114
$t = 2.0$	2.850974466440	$t = 2.5$	2.904679600000
	2.770160735513		2.893007237193
	2.769813021941		2.892887564025
	2.769813014455		2.892887563431
$t = 3.0$	3.015962112407	$t = 3.5$	3.058182875049
	2.975573085881		3.030823764432
	2.975535219280		3.030812214168
	2.975535219240		3.030812214166
$t = 4.0$	3.086089209091	$t = 4.5$	3.104621053709
	3.067720100573		3.092336368097
	3.067716633938		3.092335335059
	3.067716633937		

Резюме.

(1) Все явные методы характеризуются ограничением вида $h \approx |J|^{-1}$ на шаг сетки h.

(2) В неявных методах следует проводить итерации вплоть до

достижения сходимости, иначе и для этих методов будет справедливо такое же ограничение.

(3) В неявных методах могут применяться различные итерационные процессы; двумя наиболее популярными являются функциональная итерация и метод Ньютона.

(4) Применение функциональной итерации приводит к ограничению на шаг сетки $h \approx |J|^{-1}$, обеспечивающему сходимость; для жестких задач это неприемлемо.

(5) Метод Ньютона не накладывает никаких ограничений на шаг сетки, но требует задания начального приближения, достаточно близкого к решению.

*8.13. Озон в атмосфере — жесткая система

Не существует «типичных» жестких начальных задач. Однако именно при исследовании задач химической кинетики был предложен целый ряд важных численных методов. В данном параграфе описывается одна из подобных задач, моделирующая концентрацию озона в атмосфере. Высотный слой озона защищает нас от вредоносного солнечного ультрафиолетового излучения, и некоторые ученые полагают, что увеличение потребления топлива и сопутствующих выбросов в атмосферу приводит к постоянным изменениям количества озона. Для нас же рассматриваемая модель иллюстрирует целый класс жестких начальных задач, которые современными программами решаются как типовые. Превосходное описание как самой задачи, так и возникающих при ее решении проблем можно найти в статье [Hindmarsh, 1980].

Мы, как правило, думаем, что при взаимодействии веществ, например при образовании воды из кислорода и водорода, химические реакции протекают мгновенно. На самом деле, конечно, вещества реагируют с различными скоростями, что зависит от ряда условий, таких, как тип реагентов, температура, давление, а также от присутствия некоторых других веществ. В сложных системах количество или концентрация каждого отдельного реагента может возрастать вследствие одних реакций и убывать вследствие других. Каждая реакция характеризуется своей «константой скорости», обозначаемой через k_i, которая определяется эмпирически. Иногда константы скорости могут зависеть от времени или каких-либо других переменных. Важная задача химической кинетики заключается в том, чтобы описать эволюцию во времени концентрацией набора реагентов, исходя из их начальных значений и констант скоростей реакций.

Бимолекулярная реакция записывается в виде

$$A + B \xrightarrow{k} C + D,$$

который означает, что реагенты A и B порождают продукты реакции

C и D со скоростью k в единицу времени. По-видимому, реальный физический процесс включает в себя и другие реакции, в которых участвуют данные реагенты. Если взаимодействуют N реагентов, то это определяет N обыкновенных дифференциальных уравнений независимо от числа реакций для каждого реагента. Отсюда мы делаем вывод, что для той единственной реакции, что записана выше, следует написать по крайней мере четыре обыкновенных дифференциальных уравнения, по одному для A, B, C и D. Формула реакции подсказывает нам, что уравнения для реагентов A и B должны включать «стоки», поскольку в процессе концентрация этих реагентов убывает. Аналогично, уравнения для продуктов C и D должны включать «прибыль», или «источники». Потеря или прибыль находится как произведение концентраций двух (истощающихся) реагентов и константы скорости реакции. Если через $[A]$ мы обозначим концентрацию вещества A и т. д., то одна наша реакция приводит к следующим четырем обыкновенным дифференциальным уравнениям:

$$\begin{aligned}
[A]' &= \ldots - k[A][B] + \ldots, \\
[B]' &= \ldots - k[A][B] + \ldots, \\
[C]' &= \ldots + k[A][B] + \ldots, \\
[D]' &= \ldots + k[A][B] + \ldots .
\end{aligned}$$

Если других реакций нет, то многоточия следует опустить. Таким образом, процесс генерации уравнений по реакциям можно автоматизировать, а программы, используемые для моделирования сложных процессов, могут потребовать решения нескольких сотен уравнений. Эти уравнения могут оказаться чрезвычайно жесткими, поскольку константы скорости могут различаться на много порядков.

В качестве примера исследуем простую модель озона в атмосфере. Предположим, что атмосфера Земли представляет собой замкнутую систему с неизменными температурой и объемом, и рассмотрим одновременное взаимодействие трех реагентов: свободного кислорода O, озона O_3 и молекулярного кислорода O_2. Механизм реакций этих веществ таков:

$$O + O_2 \xrightarrow{k_1} O_3,$$

$$O + O_3 \xrightarrow{k_2} 2O_2,$$

$$O_2 \xrightarrow{k_3(t)} 2O,$$

$$O_3 \xrightarrow{k_4(t)} O + O_2.$$

Запись $k_3(t)$ и $k_4(t)$ означает, что у этих двух «констант» скорости значения меняются со временем. Это вызвано тем, что две последние

реакции описывают влияние солнечного света, под воздействием которого молекулярный кислород и озон «фотодиссоциируют».

Эта модель основана на весьма спорных предположениях, в частности на допущении, что концентрации не зависят от высоты. Как бы то ни было, описанный выше процесс позволяет выписать дифференциальные уравнения

$$[O]' = -k_1[O][O_2] - k_2[O][O_3] + 2k_3(t)[O_2] + k_4(t)[O_3],$$
$$[O_3]' = k_1[O][O_2] - k_2[O][O_3] - k_4(t)[O_3].$$

Поскольку значения $[O_2]$ на много порядков больше концентраций $[O]$ и $[O_3]$, мы можем допустить, что на концентрацию молекулярного кислорода два других реагента существенного влияния не оказывают и ее можно считать постоянной во времени. Вот почему мы игнорируем соответствующее дифференциальное уравнение. Значения констант k_1 и k_2 известны:

$$k_1 = 1.63 \cdot 10^{-16}, \quad k_2 = 4.66 \cdot 10^{-16}.$$

Две другие константы скоростей меняются дважды в сутки; они описываются формулами

$$k_i(t) = \begin{cases} \exp(-c_i/\sin\omega t), & \sin\omega t > 0, \\ 0, & \sin\omega t \leqslant 0, \end{cases} \quad i = 3, 4,$$

в которых $\omega = \pi/43\,200$ с$^{-1}$ $(= \pi/12$ ч$^{-1})$, $c_3 = 22.62$, $c_4 = 7.601$. Значения k_3 и k_4 резко возрастают «на рассвете» $(t = 0)$, достигают максимума «в полдень» $(t = 6 \times 3600$ с) и падают до нуля «на закате солнца» $(t = 12 \times \times 3600$ с). Время t измеряется в секундах. Разумно взять следующие начальные значения:

$$[O](0) = 10^6 \text{ см}^{-3}, \quad [O_3](0) = 10^{12} \text{ см}^{-3}, \quad [O_2](0) = 3.7 \cdot 10^{16} \text{ см}^{-3}.$$

Когда t попадает в интервалы, соответствующие светлому времени суток, решение претерпевает резкие изменения. Например, концентрация $[O]$ изменяется от начального значения (10^6) почти до нуля (10^{-30}), когда t достигает 60 секунд, к полудню возрастает почти до 10^{18} и к ночи снова падает. Ночью обе концентрации $[O]$ и $[O_3]$, по существу, постоянны; процесс повторяется на следующее утро, когда встает солнце. На рис. 8.8 приведен график изменения концентрации $[O]$ для полутора суток. Убывание в начале процесса столь стремительно, что его трудно увидеть на графике. Отметим, что вертикальная шкала взята логарифмической, а время дано в секундах.

Эта система является жесткой на интервалах, соответствующих ночи, например при $12 \leqslant t/3600 \leqslant 24$ (ч). При интегрировании на интервалах дневного времени для получения хорошего разрешения требуется малый шаг интегрирования; здесь уравнения можно решать каким-либо обычным нежестким методом. Тем не менее уравнения оказываются все еще умеренно жесткими, поэтому бóльшую эффективность дает применение

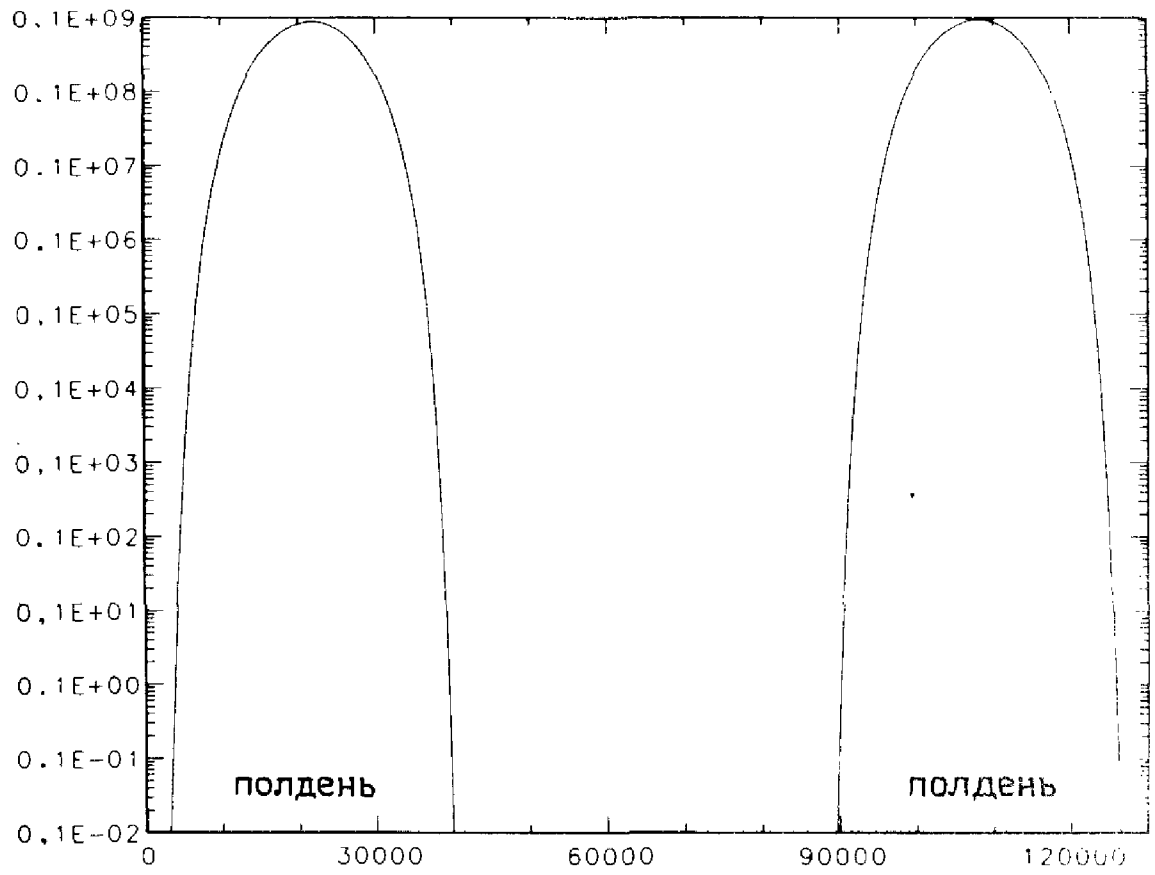


Рис. 8.8. Атомарный кислород в простой модели концентрации озона.

программы жесткого интегрирования. Рассматриваемая задача характеризуется двумя собственными значениями: $\lambda_1 \approx -6.03$ (всегда) и λ_2, которое определяет скорость изменения системы. Модуль λ_2 возрастает к полудню, когда ускоряется ход реакций, а к закату солнца – периоду ускорения – убывает. Ночью λ_2 равняется машинному нулю, но в течение дня становится отрицательным и достигает значения $-1.6 \cdot 10^{-7}$. Хотя эта задача является только простой моделью, но и ее было невозможно решить, пока не появились программы жестких методов.

8.14. Многозначные методы

Ключевой проблемой реализации эффективного многошагового метода является вопрос, каким образом продолжить процесс с шагом h', отличным от шага h, использованного в предыдущих узлах. Путь к решению проблемы заключается в том, чтобы представить число y_{n+1}, т. е. результат, вычисляемый (например, по трехшаговой формуле из §§ 9, 10) в узле t_{n+1}, двумя способами:

(1) как полученное собственно многошаговым методом, т. е. описанным выше способом;

(2) как значение некоторого полинома $P(t)$ в точке t_{n+1}.

Преимущество второго подхода состоит в том, что, зная полином $P(t)$,

мы можем вычислить его значение в любой нужной точке, а не только в точке t_{n+1}. Например, проинтегрируем уравнение $y' = f(t, y)$ от t_n до t:

$$y(t) = y(t_n) + \int_{t_n}^{t} f(\tau, y(\tau))\, d\tau,$$

и аппроксимируем f кубическим полиномом P_3. Интеграл от P_3 можно вычислить по любому отрезку. В качестве P_3 мы берем тот единственный полином, который интерполирует f по четырем точкам t_{n-2}, t_{n-1}, t_n и t_{n+1}. Обозначив соответствующие ординаты через f_{n-2}, f_{n-1}, f_n, f_{n+1} и применяя любое из представлений, описанных в гл. 4, мы можем выписать P_3 в явном виде. Далее можно вычислить интеграл, что, как нетрудно показать, приводит к выражению

$$\int_{t_n}^{t_{n+1}} P_3(t)\, dt = \frac{h}{24}[f_{n-2} - 5f_{n-1} + 19f_n + 9f_{n+1}].$$

Добавив сюда y_n, получим в точности ту же формулу для y_{n+1}, что была выведена в примере 8.10 для неявного метода 4-го порядка. Стоит повторить, что формулу многошагового метода можно рассматривать как частный случай более общей формулы, отвечающий специальному выбору t_{n+1}. Все другие формулы многошаговых методов тоже можно вывести подобным образом, и такой подход дает нам возможность вычислить приближенное решение в совершенно произвольной точке.

Завершающим шагом этого подхода является выбор способа представления полинома $P_3(t)$. В многошаговых методах используется представление полиномами Лагранже (см. § 3 гл. 4). Это значит, что полином $P_3(t)$ строится по вычисленным в узлах значениям y (и y'). В процессе интегрирования узлы интерполяции сдвигаются вправо, как если бы мы перемещали лекало. Однако мы знаем, что есть и другие представления, которые математически эквивалентны, но записываются по-разному. Интеграл полинома P_3, взятый от t_n до произвольной точки t, не может зависеть от формы представления, однако для некоторых форм он вычисляется проще, чем для других. Если записать P_3 в виде

$$P_3(t) = a_0 + a_1(t - t_n) + a_2(t - t_n)^2 + a_3(t - t_n)^3,$$

то интеграл вычисляется мгновенно. Значение a_0 очевидно, потому что $P_3(t_n) = f_n = a_0$. Поскольку $P_3(t) \approx f(t, y) \equiv y'$, то a_i должны аппроксимировать производные:

$$a_i = P_3^{(i)}(t_n)/i! \approx y^{(i+1)}(t_n)/(i+1)!.$$

Итак, a_i выражаются через приближения к производным от решения в точке t_n. В этом суть многозначного подхода. До сих пор мы рассматривали данные, предоставляемые для интегрирования, как вычисленное в некоторых точках решение (и его производную), в то время как в многозначных методах данные трактуются как первые последовательные коэффициенты ряда Тейлора, записанного относительно теку-

щего узла. Теперь цель интегрирования заключается в том, чтобы получить те же коэффициенты ряда Тейлора, но с центром в следующем узле сетки. Чтобы объяснить это более подробно, проделаем весь вывод для конкретного примера.

Предположим, что в узле t_n мы знаем точное решение $y(t_n)$ и первые три производные. Мы хотим перейти к узлу $t_n + h \equiv t_{n+1}$, т. е. вычислить $y(t_{n+1})$, $y'(t_{n+1})$, $y''(t_{n+1})$ и $y'''(t_{n+1})$. Запишем это символически следующим образом:

$$\tilde{\mathbf{y}}(t_n) \equiv \begin{bmatrix} y(t_n) \\ hy'(t_n) \\ \frac{h^2}{2}y''(t_n) \\ \frac{h^3}{6}y'''(t_n) \end{bmatrix} \to \tilde{\mathbf{y}}(t_{n+1}) \equiv \begin{bmatrix} y(t_{n+1}) \\ hy'(t_{n+1}) \\ \frac{h^2}{2}y''(t_{n+1}) \\ \frac{h^3}{6}y'''(t_{n+1}) \end{bmatrix}.$$

Назовем метод, пытающийся вычислить $\tilde{\mathbf{y}}(t_{n+1})$, *четырехзначным* методом. На практике мы имеем дело не с точными данными, а с результатами предшествующих вычислений. Обозначим через $\tilde{y}_n$ и $\tilde{\mathbf{y}}_{n+1}$ соответствующие массивы приближенных величин. В них входят множители $h^2/2$ и т. п., которые введены лишь для масштабирования; они всегда возникают при реализации алгоритмов, и мы отражаем это в своей записи.

Отметим, что компоненты в $\tilde{\mathbf{y}}(t_n)$ не могут быть независимыми; например, зная первую компоненту, мы можем вычислить вторую с помощью подстановки в дифференциальное уравнение. Следует потребовать, чтобы для аппроксимации $\tilde{\mathbf{y}}_n$ сохранялось то же свойство.

Будем осуществлять переход от t_n к t_{n+1} по следующей схеме: разложим решение $y(t)$ в ряд Тейлора относительно точки t_n, продифференцируем три раза и выпишем результаты для точки t_{n+1}. Получим

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + \frac{h^3}{6}y'''(t_n) + \frac{h^4}{24}y''''(t_n) + \ldots,$$

$$hy'(t_{n+1}) = hy'(t_n) + h^2y''(t_n) + \frac{h^3}{2}y'''(t_n) + 4\frac{h^4}{24}y''''(t_n) + \ldots,$$

$$\frac{h^2}{2}y''(t_{n+1}) = \frac{h^2}{2}y''(t_n) + \frac{h^3}{2}y'''(t_n) + 6\frac{h^4}{24}y''''(t_n) + \ldots,$$

$$\frac{h^3}{6}y'''(t_{n+1}) = \frac{h^3}{6}y'''(t_n) + 4\frac{h^4}{24}y''''(t_n) + \ldots.$$

Отбрасывая в этих выражениях все слагаемые, содержащие четвертую и более высокие производные, можно записать полученные соотношения в виде

$$\tilde{\mathbf{y}}^{P}_{n+1} \equiv \begin{bmatrix} y^{P}_{n+1} \\ hy^{P'}_{n+1} \\ \frac{h^2}{2} y^{P''}_{n+1} \\ \frac{h^3}{6} y^{P'''}_{n+1} \end{bmatrix} \equiv B\tilde{\mathbf{y}}(t_n), \qquad B \equiv \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Мы можем рассматривать $\tilde{\mathbf{y}}^{P}_{n+1}$ как полиномиальную предикцию для решения в узле t_{n+1}. Отметим, что предикция записывается в виде произведения $B\tilde{\mathbf{y}}(t_n)$ и никак не использует дифференциальное уравнение. Получаем

$$\tilde{\mathbf{y}}(t_{n+1}) = \tilde{\mathbf{y}}^{P}_{n+1} + \begin{bmatrix} \frac{h^4}{24} y^{(4)}(t_n) + \dots \\ 4\frac{h^4}{24} y^{(4)}(t_n) + \dots \\ 6\frac{h^4}{24} y^{(4)}(t_n) + \dots \\ 4\frac{h^4}{24} y^{(4)}(t_n) + \dots \end{bmatrix}.$$

Это точные уравнения. Теперь мы откажемся от точных равенств, игнорируя те члены в скобках, которые не выписаны явно. Тогда правая часть примет вид

$$B\tilde{\mathbf{y}}(t_n) + \begin{bmatrix} 1 \\ 4 \\ 6 \\ 4 \end{bmatrix} y^{(4)}(t_n) h^4/24.$$

Это выражение уже не равняется $\tilde{\mathbf{y}}(t_{n+1})$, а может лишь служить приближением $\tilde{\mathbf{y}}_{n+1}$ к решению. Это приближение имеет два недостатка:

(1) Величина $y^{(4)}$ неизвестна. Напомним, что наши данные для точки t_n включают лишь первые три производные.

(2) Если бы мы знали точное значение $y^{(4)}(t_n)$, то смогли бы найти $\tilde{\mathbf{y}}_{n+1}$, не прибегая к дифференциальному уравнению. Но на такую аппроксимацию не следует полагаться; например, если через y_{n+1} обозначена первая компонента в $\tilde{\mathbf{y}}_{n+1}$, то нет никаких оснований надеяться, что число $hf(t_{n+1}, y_{n+1})$ будет близко ко второй компоненте. Эти две компоненты не будут согласованы.

Из двух указанных проблем вторая является более серьезной; мы обязаны воспользоваться дифференциальным уравнением. Эту проблему можно обойти, опустив в правой части производную $y^{(4)}(t_n)$ и заменив ее некоторой аппроксимацией a_n, которая выбирается так, чтобы она

в определенной мере удовлетворяла уравнению. Воспользуемся таким критерием: первые две компоненты вектора $\tilde{\mathbf{y}}_{n+1}$ должны удовлетворять уравнению $y' = f(t, y)$. Запишем правую часть второго уравнения в виде

$$hy'^{P}_{n+1} + 4a_n.$$

Если определить

$$4a_n \equiv hf(t_{n+1}, y_{n+1}) - hy'^{P}_{n+1}, \quad \tilde{\mathbf{y}}_{n+1} \equiv \begin{pmatrix} y_{n+1} \\ hy'_{n+1} \\ \frac{h^2}{2} y''_{n+1} \\ \frac{h^3}{6} y'''_{n+1} \end{pmatrix},$$

то можно быть уверенным, что равенство $y'_{n+1} = f(t_{n+1}, y_{n+1})$ выполнено.

Таким образом, уравнения записываются в виде

$$\tilde{\mathbf{y}}_{n+1} = B\tilde{\mathbf{y}}_n + \mathbf{r}[hf(t_{n+1}, y_{n+1}) - hy'^{P}_{n+1}], \quad \mathbf{r} = \begin{pmatrix} 1/4 \\ 1 \\ 3/2 \\ 1 \end{pmatrix},$$

поэтому процесс в целом определяется матрицей B и вектором $\mathbf{r}$.

Напомним, что компоненты массива $\tilde{\mathbf{y}}_{n+1}$ являются аппроксимациями решения и его производных. Метод был построен таким образом, чтобы первые две компоненты удовлетворяли условию $y'_{n+1} = f(t_{n+1}, y_{n+1})$, но последние две компоненты подобными условиями никак не связаны; сопутствующие им двойные и тройные штрихи условны – еще требуется доказать, что эти компоненты аппроксимируют y'' и y'''. Отметим также, что a_n с точностью до коэффициента 4 совпадает с разностью между вторыми компонентами окончательного результата $\tilde{\mathbf{y}}_{n+1}$ и предикции $\tilde{\mathbf{y}}^{P}_{n+1}$. Это аппроксимация величины $h^4 y^{(4)}/24$, и программы многозначных алгоритмов могут использовать этот факт.

8.15. Пример многозначного метода

Воспользуемся многозначным методом для решения задачи

$$y' = -10(t-1)y, \quad y(0) = 1.$$

Возьмем шаг $h = 0.1$. Чтобы начать вычисления, нам необходим массив

$$\tilde{\mathbf{y}}(0) = \begin{pmatrix} y(0) \\ hy'(0) \\ (h^2/2)y''(0) \\ (h^3/6)y'''(0) \end{pmatrix}.$$

Первая компонента массива известна из начального условия, вторую компоненту можно найти с помощью дифференциального уравнения $hy'(0) = y(0)$. Остальные компоненты можно получить, только дифференцируя уравнение. Современные программы решают эту проблему, но в данном примере мы предположим, что требуемые значения заданы. Итак,

$$\tilde{\mathbf{y}}(0) = \begin{bmatrix} 1 \\ 1 \\ 0.45 \\ 7/60 \end{bmatrix}.$$

Первая предикция дает

$$\tilde{\mathbf{y}}_1^p = \begin{bmatrix} 2.56667 \\ 2.25 \\ 0.8 \\ 0.116667 \end{bmatrix},$$

далее,

$$\tilde{\mathbf{y}}_1 = \begin{bmatrix} y_1 \\ 0.1y_1' \\ (0.01/2)y_1'' \\ (0.001/6)y_1''' \end{bmatrix} = \tilde{\mathbf{y}}_1^p + \mathbf{r}[0.9y_1 - 2.25].$$

Здесь фигурируют четыре неизвестных, но только первая из них, y_1, входит в обе части системы. Таким образом, многозначный метод является неявным, но только по этой компоненте. Как только y_1 найдено, остальные компоненты вычисляются непосредственно. Первое уравнение приводится к виду

$$y_1 = 2.56667 + 0.25[0.9y_1 - 2.25].$$

Если бы исходное уравнение было нелинейным, то в правой части величину $0.9y_1$ следовало бы заменить на $0.1f(0.1, y_1)$ и нам пришлось бы решать нелинейное уравнение итерационным методом. В нашем примере уравнение является линейным и y_1 можно найти в явном виде:

$$y_1 = \frac{1}{1 - 0.25 \cdot 0.9}[2.56667 - 0.25 \cdot 2.25] = 2.58603.$$

Тогда

$$\tilde{\mathbf{y}}_1 = \begin{bmatrix} 2.5860 \\ 2.3274 \\ 0.9161 \\ 0.1941 \end{bmatrix}.$$

Напомним, что массив $\tilde{\mathbf{y}}_1$ должен аппроксимировать массив

$$\tilde{\mathbf{y}}(0.1) = \begin{bmatrix} 2.5857 \\ 2.3272 \\ 0.9179 \\ 0.1978 \end{bmatrix}.$$

Массивы $\tilde{\mathbf{y}}_2, \tilde{\mathbf{y}}_3, \ldots, \tilde{\mathbf{y}}_{14} \approx \tilde{\mathbf{y}}(1.4)$ вычисляются аналогичным образом. В приведенной ниже таблице указаны вычисленные и точные решения в узлах $t = 0.1, \ldots, 1.4$. Мы видим, что вначале ошибки уменьшаются, но затем быстро растут.

t	MV	
0.1	2.58062	2.58571
0.2	6.04772	6.04965
0.3	12.81248	12.80710
0.4	24.52068	24.53253
0.5	42.57806	42.52108
0.6	66.54054	66.68633
0.7	95.19001	94.63241

t	MV	
0.8	119.66133	121.51042
0.9	147.82166	141.17496
1.0	124.31296	148.41316
1.1	231.04534	141.17496
1.2	−221.15943	121.51042
1.3	1431.65346	94.63241
1.4	−5266.04169	66.68633

Результаты, полученные многозначным методом в начальных узлах, приемлемы, однако в дальнейшем погрешность становится значительной и, более того, за точкой $t = 1$ начинаются осцилляции. Ничего не подозревающий пользователь может подумать, будто причина в том, что на интервале [0,1] задача неустойчива. Однако, повторив вычисления с меньшим шагом, он получит еще худший эффект. Как отмечалось в примере 8.5, не имеющие физического смысла растущие осцилляции в численном решении, которые усиливаются по мере уменьшения шага, почти всегда указывают на неустойчивость метода. В следующем параграфе мы увидим, что дело заключается именно в этом.

8.16. Некоторые другие многозначные методы

Неудача многозначного метода в нашей простой задаче вызывает разочарование, и на первый взгляд может показаться, что вся гибкость в выборе метода сводится к выбору числа хранимых значений, т.е. длины массива $\tilde{\mathbf{y}}_n$. Рассмотрим этот вопрос более подробно. Если истинное решение дифференциального уравнения является полиномом степени не выше третьей и $\tilde{\mathbf{y}}_0 = \tilde{\mathbf{y}}(t_0)$, т.е. начальные значения заданы точно, то все последующие $\tilde{\mathbf{y}}_n$ будут совпадать с $\tilde{\mathbf{y}}(t_n)$ при любом $\mathbf{r}$. Это наводит на мысль, что и другие значения $\mathbf{r}$, отличные от $\mathbf{r} = (1/4, 1, 3/2, 1)^T$, также могут оказаться полезными. Необходимо лишь потребовать выполнения условия $r_2 = 1$, чтобы обеспечить согласованность первых

двух компонент с дифференциальным уравнением. Можно исследовать многозначные методы на устойчивость и показать, что при использованном выше **r** метод никогда не будет устойчивым и численные результаты, приведенные в предыдущем параграфе, типичны. К счастью, есть и другие многозначные методы, свойства которых значительно лучше. Два наиболее известных метода соответствуют $\mathbf{r} = (3/8, 1, 3/4, 1/6)^T$ – метод Адамса – Мултона – и $\mathbf{r} = (6/11, 1, 6/11, 1/11)^T$ – метод Гира. Метод Адамса – Мултона имеет ограниченную область устойчивости и больше подходит для решения нежестких задач. У метода Гира область устойчивости не ограничена, хотя метод не является абсолютно устойчивым. Он подходит для интегрирования как жестких, так и нежестких задач. Его точность несколько ниже, чем у метода Адамса – Мултона, и он не очень эффективен для задач с осциллирующим решением. Подпрограмма SDRIV2 предоставляет пользователям на выбор методы Гира (MINT = = 2) и Адамса – Мултона (MINT = 1).

Пример 8.13. Многозначный метод Адамса – Мултона.

Повторим расчеты из § 15, применяя метод Адамса – Мултона. Первый шаг к $t = 0.1$ дает

$$\tilde{\mathbf{y}}_1 = \begin{bmatrix} 2.60063 \\ 2.34057 \\ 0.86792 \\ 0.13176 \end{bmatrix}.$$

В представленной ниже таблице приводятся относительные ошибки в точке $t = 1$; их можно сравнить с результатами из таблицы в предыдущем параграфе. Теперь мы видим, что при уменьшении шага h ошибки вначале растут, но, как только шаг становится достаточно малым, они начинают уменьшаться. Наибольшим значением hJ является $10h$, и при $h \leqslant \approx 0.25$ метод устойчив.

Таблица 8.8. Ошибка в точке $t = 1$ при решении задачи $y' = -10\,(t-1)\,y$, $y(0) = 1$ методом Адамса-Мултона

Ошибка $\cdot e^5$	Шаг сетки h
0.026390	1.0000
0.093329	0.5000
0.190202	0.3333
0.094691	0.2500
0.048559	0.2000
0.004978	0.1000
0.000425	0.0500
0.000032	0.0250
0.000002	0.0125

Пример 8.14. Многозначный метод Гира.

Повторим расчеты, пользуясь методом Гира. Первый шаг к точке $t = 0.1$ дает

$$\tilde{\mathbf{y}}_1 = \begin{bmatrix} 2.63095 \\ 2.36786 \\ 0.86429 \\ 0.12738 \end{bmatrix}.$$

Приведенная ниже таблица содержит относительные ошибки в точке $t = 1$; их можно сравнить с результатами из предыдущей таблицы. Как уже отмечалось, метод устойчив, однако не столь точен, как метод Адамса – Мултона.

Таблица 8.9. Ошибка в точке $t = 1$ при решении задачи $y' = -10\,(t-1)\,y$, $y\,(0) = 1$ методом Гира

Ошибка $\cdot\, e^5$	Шаг сетки h
0.028380	0.1000
0.005067	0.0500
0.000841	0.0250
0.000124	0.0125

Резюме.

(1) Многозначные методы определяются выбором массива **r**.

(2) Наиболее «точный» многозначный метод неустойчив.

(3) Многозначные методы Адамса – Мултона дают хорошую точность и подходят для решения нежестких задач.

(4) Метод Гира при решении нежестких задач менее точен, чем метод Адамса – Мултона, но хорош для решения жестких задач.

8.17. Связь многошаговых и многозначных методов

Многошаговые методы легко понять, но трудно реализовать эффективно; с многозначными методами ситуация обратная. В 1968 году Скил показал, что эти методы, по существу, эквивалентны. Под эквивалентностью мы подразумеваем тот факт, что первая компонента вектора $\tilde{\mathbf{y}}_{n+1}$, а именно приближение y_{n+1} к $y(t_{n+1})$, идентична приближенному значению, вычисляемому многошаговым методом с соответствующими коэффициентами α и β. Это важный результат, поскольку он позволяет нам принять ту из точек зрения, которая лучше соответствует данному контексту. На практике метод почти всегда реализуется как многозначный, но рассматривается как особая форма многошагового, устроенного так, чтобы нести ту же информацию. Дадим несколько иллюстраций эквивалентности многошаговых и многозначных методов.

Пример 8.15. Эквивалентность многошаговых и многозначных методов.

(а) Четырехзначный метод Адамса–Мултона с $r_1 = 3/8$, $r_3 = 3/4$, $r_4 = 1/6$ эквивалентен методу

$$y_{n+1} = y_n + \frac{h}{24}(9y'_{n+1} + 19y'_n - 5y'_{n-1} + y'_{n-2}).$$

Этот метод называется трехшаговым методом Адамса–Мултона и описывается в примере 8.10, где показано, что он имеет четвертый порядок.

(б) Четырехзначный метод с $r_1 = 5/12$, $r_3 = 3/4$, $r_4 = 1/6$ эквивалентен методу

$$y_{n+1} = y_n + \frac{h}{12}(5y'_{n+1} + 8y'_n - y'_{n-1}),$$

называемому двухшаговым методом Адамса–Мултона.

(в) Четырехзначный метод с $r_1 = 0$, $r_3 = 3/4$, $r_4 = 1/6$ является явным (поскольку $r_1 = 0$) и эквивалентен методу

$$y_{n+1} = y_n + \frac{h}{12}(23y'_n - 16y'_{n-1} + 5y'_{n-2}).$$

Многошаговые формулы с $\alpha_1 = 1$ и прочими α, равными нулю (см. § 9), называются формулами Адамса; те из них, которые являются неявными, называются формулами Адамса–Мултона. Явные формулы, подобные приведенной выше, называются формулами Адамса–Бэшфорта.

(г) Четырехзначный метод Гира с $r_1 = 6/11$, $r_3 = 6/11$, $r_4 = 1/11$ эквивалентен трехшаговому методу Гира

$$y_{n+1} = \frac{1}{11}(18y_n - 9y_{n-1} + 2y_{n-2}) + h\frac{6}{11}y'_{n+1}.$$

Отметим, что в последней формуле все коэффициенты при производной, исключая y'_{n+1}, равны нулю (что характеризует эти методы). Если мы выразим y'_{n+1} из этого равенства, то полученное соотношение можно интерпретировать как разностную формулу для производной в новой точке, аппроксимирующую ее по значениям функции в новой, текущей и двух предшествующих точках. Вот почему формулы Гира иногда называют *формулами дифференцирования назад*. □

*8.18. Привлекательные черты многозначных методов: смена шага и порядка

В многошаговых методах используются ранее вычисленные значения решения и его производной. Для многозначных методов требуются текущие значения решения и нескольких его производных – первые члены разложения в ряд Тейлора. Со строкой ряда Тейлора легко работать,

поэтому неудивительно, что многозначные алгоритмы позволяют легко менять шаг. В этом параграфе обсуждается ряд деталей реализации подобных алгоритмов.

*8.18.1. Смена шага

Предположим, что мы проводили интегрирование с шагом h и, дойдя до узла t_n, захотели получить решение в точке $t_n + h'$. Другими словами, мы хотим изменить длину шага с величины h на h'. Многозначные методы позволяют выполнить эту операцию весьма просто. Напомним, что массив $\tilde{\mathbf{y}}_n$ уже вычислен; значение h уже «вошло» в его компоненты. Перед началом движения из точки t_n нам потребуется лишь домножить компоненты массива $\tilde{\mathbf{y}}_n$ на соответствующие степени отношения h'/h. Пусть, например (обратимся снова к § 15, 16), дойдя до $t_1 = 0.1$, мы хотим перейти к точке 0.15, а не к точке 0.20, т. е. сделать шаг 0.05 вместо 0.10. Для этого мы просто вычислим

$$\text{новое } \tilde{\mathbf{y}}_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.25 & 0 \\ 0 & 0 & 0 & 0.125 \end{pmatrix} \cdot \tilde{\mathbf{y}}_1 = \begin{pmatrix} 2.5860 \\ 1.1637 \\ 0.2129 \\ 0.0210 \end{pmatrix}.$$

От узла t_1 вычисления продолжаем так же, как и ранее, с единственным отличием: теперь используются новые $\tilde{\mathbf{y}}_1$ и h'.

Смена шага требует небольших накладных расходов, но практически в многозначном методе она столь же легка, как и в одношаговом методе. Такой же подход используется и при вычислении решения в точке, не совпадающей с конечным узлом интегрирования. Мы знаем, что компоненты массива $\tilde{\mathbf{y}}_n$ представляют собой нормированные приближения к первым членам ряда Тейлора для $y(t)$ относительно точки t_n. Поэтому хорошим приближением к решению в точке $t_n + h'$ будет служить следующее скалярное произведение:

$$\left(1, \frac{h'}{h}, \left(\frac{h'}{h}\right)^2, \left(\frac{h'}{h}\right)^3\right) \cdot \tilde{\mathbf{y}}_n.$$

В таком случае говорят, что в новой точке решение *интерполировано*. SDRIV и другие аналогичные программы работают именно таким образом. Пользователь задает конечную точку интегрирования, однако SDRIV использует свой внутренний шаг до тех пор, пока не перейдет через эту конечную точку, а затем интерполирует решение «назад». В таком случае шаг h' становится отрицательным, но процедура выглядит так же. Следовательно, конечные точки, которые заданы пользователем, и те, которые получаются в программе, не связаны между собой. Действительно, если решение достаточно гладкое, то программа может проскочить далеко за точку, заданную пользователем.

*8.18.2. Оценка ошибок

Мы уже описали несколько подходов к оценке ошибок методов численного интегрирования. Напомним, что мы можем работать только с локальной ошибкой (или ошибкой на шаге). Все эти подходы основаны на том факте, что локальная ошибка пропорциональна произведению известной степени длины шага на производную решения, т. е.

$$\text{локальная ошибка} = ch^{p+1}y^{(p+1)}(\xi).$$

Значение p определяет порядок метода, где p – наибольшая степень решения вида t^p, которое данный метод позволяет вычислить точно.

Как мы видели, многозначные и многошаговые методы приводят к одинаковым результатам при соответствующем выборе вектора $\mathbf{r}$. Это свойство весьма полезно при отыскании констант, входящих в ошибку многошаговых формул. Так, если $\mathbf{r} = (3/8,\ 1,\ 3/4,\ 1/6)^T$, то в силу замечания к примеру 8.10 локальная ошибка записывается в виде

$$y_{n+1} - y(t_{n+1}) = -\frac{19}{720}h^{(5)}y^{(5)}(\xi).$$

Если бы мы оценили $y^{(5)}$, то смогли бы построить стратегию выбора шага. Такую оценку можно получить, численно дифференцируя доступные нам приближения к $y^{(4)}$. Как отмечалось в конце § 14, число a_n аппроксимирует величину $y^{(4)}(t_n)h^4/4!$, и его можно вычислить, вычитая вторую компоненту предикции из второй компоненты итогового приближения, или, что то же самое, a представляет собой приращение к предикции. Большинство программ многозначных методов добавляет это число пятой компонентой в массив $\tilde{\mathbf{y}}_{n+1}$, которая пересчитывается вместе с четырьмя предшествующими. Разность двух последовательных оценок составляет

$$\frac{h^4y_{n+1}^{(4)}}{4!} - \frac{h^4y_n^{(4)}}{4!} \approx \frac{h^5y^{(5)}}{4!} \equiv e_5,$$

поэтому для ошибки получаем оценку

$$|y_{n+1} - y(t_{n+1})| \approx \left|\frac{19}{720}\cdot 4!e_5\right|,$$

которую можно сверить с заданным допуском ε, чтобы проверить, приемлема ли данная длина шага интегрирования.

Длина следующего шага интегрирования или уменьшенная длина шага (в случае неудачи последнего теста) берется в виде αh, где α подбирается так, чтобы локальная ошибка была меньше допуска ε, а именно

$$\varepsilon \geqslant \left|\frac{19}{720}(\alpha h)^5y^{(5)}(\xi)\right| \approx \alpha^5\left|\frac{19}{720}4!e_5\right|.$$

Для страховки величину α ограничивают приблизительно 80% указанного значения:

$$\alpha = \frac{1}{1.2}\left[\frac{\varepsilon}{19/720 \cdot 4!\, e_5}\right]^{1/5}.$$

Опыт и анализ ряда программ, таких, как SDRIV, показывает, что не следует увеличивать шаг в каждой точке, даже если на первый взгляд свойства уравнения это оправдывают; если применяется метод p-го порядка, то увеличение шага допускают не чаще одного раза за каждые $p + 1$ узлов, чтобы оценки ошибки могли «стабилизироваться».

* 8.18.3. Смена порядка

Наша конечная цель – перейти от точки t_0 к точке T, выполнив как можно меньше вычислений и удовлетворив требованию точности. Общий объем вычислительной работы является сложной функцией от шага интегрирования, поскольку от последнего зависит, сколько итераций потребуется для нахождения y_{n+1}, как часто понадобится обновлять якобиан при ньютоновых итерациях (если они используются) и т. п. Для конкретного метода, например для четырехзначного метода Адамса, описанный выше алгоритм выбора шага будет работать вполне удовлетворительно. Однако то же самое верно по отношению ко многим методам, и программы типа SDRIV способны динамически выбирать метод интегрирования из заданного семейства методов.

Предположим, что мы вычислили $\tilde{\mathbf{y}}_n$ и оценили, как описано выше, величину αh. Можно поставить вопрос, не приведет ли выбор другого метода (другого вектора $\mathbf{r}$) для следующего шага к большему значению α. Мы уже знаем, что, исходя из массива $\tilde{\mathbf{y}}_n$, можно продолжить интегрирование несколькими методами. Взглянув на приведенную выше формулу для α, мы увидим, что применение метода того же четвертого порядка, но с другими коэффициентами приведет к выбору другой длины шага. При аналогичном рассмотрении формулы другого порядка изменится и порядок производной. Например, если мы хотим попробовать метод третьего порядка, то в точке t_n его локальная ошибка будет иметь вид

$$c_3 (\alpha h)^4 y^{(4)}(\xi).$$

У нас уже есть оценка для $h^4 y^{(4)}/24$; эту дополнительную компоненту мы разместили в массиве $\tilde{\mathbf{y}}_n$. Если бы мы знали константу c_3, то могли бы вычислить предполагаемый шаг αh в точке t_n и, если он окажется больше первоначального значения αh, взять именно его. Конечно, это потребует использования вектора $\mathbf{r}$, соответствующего новому методу.

Таким же образом можно оценить шаг метода более высокого порядка, т. е. метода пятого порядка. Для этого потребуется производная $y^{(6)}(t_n)$, которую мы оценим, обратившись к пятой компоненте

разности $\tilde{\mathbf{y}}_n - \tilde{\mathbf{y}}_{n-1}$. Конечно, здесь много величин, выбираемых по умолчанию, и «чувствительных» параметров, для настройки которых требуется опыт и интуиция. Но, по существу, алгоритм выбирает шаг или по текущей формуле, или по формуле, порядок которой на единицу выше или ниже, и выбор определяется той оценкой, которая дает наибольшую длину шага. Все формулы принадлежат одному семейству, например семейству многозначных методов Гира или многозначных методов Адамса. Определенные успехи были достигнуты и для формул смешанного типа. Программа SDRIV2 может выбирать между методами Адамса и Гира (MINT = 3), что для некоторых задач дает возможность более быстрого интегрирования.

Один из наиболее важных аспектов динамического выбора порядка состоит в том, что соответствующие программы становятся «самостартующими» – для начала процесса интегрирования никаких специальных приемов уже не требуется. Используя начальное значение и дифференциальное уравнение, можно вычислить значение hy'_0. Этого достаточно, чтобы запустить метод первого порядка (например, неявный метод Эйлера), а уже затем блок программы, отвечающий за управление порядком, сможет увеличить порядок метода до подходящей величины.

Более того, можно даже оценить начальный шаг. Первая предикция вычисляется по методу Эйлера: $y_1^P = y_0 + hy'_0$; следовательно, ее ошибка составляет $y''h^2/2$. Таким образом,

$$\frac{h^2}{2}y'' = y_1^P - y(t_1) = y(t_0) + hy'(t_0) - y(t_1) \approx$$
$$\approx y(t_0) + hy'(t_0) - y(t_0) = hy'(t_0).$$

Если $y'(t_0) \neq 0$, то в качестве грубой оценки начального шага можно взять число $\varepsilon/|y'(t_0)|$.

*8.19. Методы рядов Тейлора и методы Рунге – Кутты

Одношаговые методы сохраняют популярность ввиду своей простоты. Однако у метода Эйлера порядок слишком мал, чтобы метод мог быть полезным. В этом параграфе мы опишем несколько подходов к построению одношаговых методов более высокого порядка.

Наш вывод метода Эйлера был основан на разложении решения $y(t)$ в ряд Тейлора. Взяв в разложении дополнительно еще один член

$$y(t_{k+1}) = y(t_k) + y'(t_k)h_k + y''(t_k)h_k^2/2 + y'''(\xi)h_k^3/6,$$

мы придем к следующей численной схеме:

$$y_{k+1} = y_k + h_k y'_k + y''_k h_k^2/2.$$

Она характеризуется локальной ошибкой $y'''(\xi)h_k^3/6$ и поэтому представляет собой метод второго порядка. Можно рассматривать формулы Тейлора с произвольным числом членов в разложении.

Как следует поступить с производной y_k'', которая требуется при вычислении решения y_{k+1}? В разд. 5.1 мы пользовались разностной аппроксимацией

$$y_k'' \approx \frac{y_k' - y_{k-1}'}{t_k - t_{k-1}},$$

но это приводит к двухшаговому методу. С другой стороны, дифференцируя выражение $y' = f(t, y)$ как сложную функцию, найдем

$$y''(t) = f_t(t, y) + f(t, y) f_y(t, y), \text{ или } y_k'' = f_t(t_k, y_k) + y_k' f_y(t_k, y_k).$$

Это дает следующий одношаговый метод:

$$y_{k+1} = y_k + h_k y_k' + \frac{h_k^2}{2}(f_t(t_k, y_k) + y_k' f_y(t_k, y_k)).$$

Если функция $f(t, y)$ не слишком сложна, то частные производные можно вычислить алгебраически. Например, для уравнений $y' = -10(t-1)y$ имеем $f_t = -10y$ и $f_y = -10(t-1)$.

Пример 8.16. Метод рядов Тейлора второго порядка.

Воспользуемся одношаговым методом Тейлора второго порядка для решения задачи из примера 8.4. При том же шаге интегрирования 0.1 теперь будут получены ошибки, приведенные в следующей таблице.

Таблица 8.10

t_n	Ошибка	t_n	Ошибка
0.1	0.0022283	1.1	0.6497108
0.2	0.0093105	1.2	0.5865614
0.3	0.0266926	1.3	0.4762708
0.4	0.0621891	1.4	0.3476736
0.5	0.1241069	1.5	0.2280530
0.6	0.2171000	1.6	0.1343194
0.7	0.3368916	1.7	0.0709585
0.8	0.4668690	1.8	0.0335548
0.9	0.5800153	1.9	0.0141430
1.0	0.6473466	2.0	0.0052622

На первых нескольких шагах интегрирования локальная ошибка доминирует над глобальной. Сравнение этих результатов с соответствующими ошибками из примера 8.4 показывает, что данный метод второго порядка характеризуется гораздо меньшими локальными ошибками, чем метод Эйлера. При дальнейшем интегрировании большой разницы между этими двумя методами уже не наблюдается. Мы знаем,

что данное уравнение неустойчиво на [0, 1]; вот почему рост ошибок имеет гораздо большее значением, чем различие локальных ошибок. □

Для дифференциальных уравнений общего вида предпринимались попытки привлечь «символическое дифференцирование» для вычисления частных производных, которые требуются в методах Тейлора второго и более высоких порядков. Однако методы Рунге–Кутты, многошаговые и многозначные методы сохраняют большинство хороших свойств методов Тейлора и не столь сложны в реализации. В итоге методы Тейлора оказались непрактичными и в настоящее время применяются мало.

Методы Рунге–Кутты являются одношаговыми методами и, позволяя избежать проблем, связанных с символическим дифференцированием, обеспечивают ту же точность, что и методы Тейлора высокого порядка. Эти методы пользуются популярностью и часто описываются в учебниках для инженеров. Методы Рунге–Кутты легко программировать, поскольку они одношаговые. Чтобы вычислить значение y_{n+1}, требуется лишь y_n; предшествующие приближения $y_{n-1}, \ldots$ не используются. Это особенно важно в начале процесса интегрирования, когда известно только начальное значение решения y_0.

Вывод методов Рунге–Кутты достаточно утомителен, требуется аккуратность при разложении в ряды Тейлора двух переменных. Методы из этого класса характеризуются количеством *этапов*. Например, *двухэтапный* метод Рунге–Кутты записывается таким образом:

$$y_{n+1} = y_n + \frac{1}{2}k_1 + \frac{1}{2}k_2, \quad \text{где} \quad k_1 = hf(t_n,\ y_n), k_2 = hf(t_n + h,\ y_n + k_1).$$

Отметим, что по известным t_n, y_n и h сначала вычисляется k_1, затем k_2 и, наконец, y_{n+1}. Таким образом, данный метод Рунге–Кутты требует двух вычислений правой части уравнения, вот откуда термин «двухэтапный», однако это одношаговый метод.

Реализовать этот метод гораздо легче, чем метод Тейлора второго порядка, поскольку требуются лишь значения правых частей f. Но можно показать, что этот метод, как и рассмотренный выше метод Тейлора, имеет второй порядок.

Пример 8.17. Двухэтапный метод Рунге–Кутты второго порядка. Воспользуемся двухэтапным методом Рунге–Кутты для решения задачи из примера 8.4. Ошибки, полученные при том же шаге 0.1, приведены в таблице. Сравнение этих результатов с ошибками метода Тейлора второго порядка (см. пример 8.16) показывает, что данный метод Рунге–Кутты приводит примерно к таким же или даже чуть меньшим ошибкам.

Таблица 8.11

t_n	Ошибка	t_n	Ошибка
0.1	0.0012513	1.1	0.2086880
0.2	0.0050241	1.2	0.1801451
0.3	0.0137453	1.3	0.1395314
0.4	0.0303591	1.4	0.0956802
0.5	0.0571071	1.5	0.0566331
0.6	0.0937626	1.6	0.0273759
0.7	0.1362795	1.7	0.0090567
0.8	0.1770045	1.8	0.0001733
0.9	0.2068270	1.9	0.0034548
1.0	0.2183775	2.0	0.0037006

Между прочим, если продолжить интегрирование для больших значений t, то и метод Рунге–Кутты, и метод Тейлора второго порядка точно так же, как и метод Эйлера, станут неустойчивыми. Неустойчивость проявляется уже вблизи $t = 4$. Как и для метода Эйлера, проявление неустойчивости можно отсрочить, используя меньший шаг. Если нужно интегрировать до некоторого априори известного значения T, то можно выбрать шаг h столь малым, что в этой точке удастся получить хорошую точность, хотя, возможно, понадобится больше шагов, чем хотелось бы. □

Метод Рунге–Кутты второго порядка не находит широкого применения. Большинство программ реализуют схемы более высоких порядков. Наиболее известным, или «классическим»[1]), методом Рунге–Кутты является четырехэтапный метод четвертого порядка

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4),$$
$$k_1 = hf(t_n, y_n),$$
$$k_2 = hf(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1),$$
$$k_3 = hf(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2),$$
$$k_4 = hf(t_n + h, y_n + k_3).$$

Для перехода от точки t_n к t_{n+1} требуется четыре раза вычислить значения правых частей, и пока у нас нет способа оценки локальной

[1]) Рунге (см. § 6 гл. 5) опубликовал в 1895 году несколько отдельных методов. Вильгельм Кутта независимо от него предложил в 1901 году общую схему и вывел рассматриваемый метод.

ошибки и изменения шага. Обычно используется следующий прием: выбирается шаг h и интегрирование от t_n к t_{n+1} проводится дважды: первый раз с шагом h, чтобы вычислить y_{n+1}, а затем с использованием двух шагов длины $h/2$. Зная из теории, что классический метод Рунге–Кутты имеет четвертый порядок, можно сравнить две оценки, полученные для y_{n+1}, и с помощью их разности оценить локальную ошибку.

Такой процесс хорошо работает и подробно описан во многих работах, однако он дорог. Даже успешно выполненный шаг требует 11 вычислений правых частей. В 60-х годах Фельберг предложил метод Рунге–Кутты, который столь же точен, как и классический, но позволяет интегрировать и оценивать локальную ошибку на основе всего 6 вычислений правых частей. Фельберг исходил из шестиэтапного метода Рунге–Кутты, записываемого в следующем виде:

$$\begin{aligned} y_{n+1} &= y_n + \gamma_1 k_1 + \gamma_2 k_2 + \ldots + \gamma_6 k_6, \\ k_1 &= hf(t_n,\ y_n), \\ k_2 &= hf(t_n + \alpha_2 h, y_n + \beta_{21} k_1), \\ &\vdots \\ k_6 &= hf(t_n + \alpha_6 h,\ y_n + \beta_{61} k_1 + \ldots + \beta_{65} k_5). \end{aligned}$$

Всего здесь 26 коэффициентов α, β и γ. Фельберг подобрал их таким образом, чтобы метод имел пятый порядок. Может быть, вас это удивит, но можно показать, что для метода пятого порядка всегда необходимо шесть вычислений правых частей, в то время как для метода четвертого порядка достаточно четырех. Однако «лишние» вычисления не пропадают даром. Фельберг также показал, что по четырем значениям k, т. е. четырем вычисленным значениям правых частей, можно, скомбинировав их с другими весами γ, построить четырехэтапный метод Рунге–Кутты, не совпадающий с классическим. Таким образом, шесть вычислений правых частей f позволяют получить две оценки решения в точке t_{n+1}; их разность можно использовать для оценки локальной ошибки и длины следующего шага интегрирования.

Как и в случае квадратур, одним из показателей эффективности является количество вычислений правых частей, требуемых для интегрирования. Суммарное количество вычислений правых частей f при интегрировании от t_0 до T зависит как от числа вычислений на шаге, так и от числа самих шагов. Эти величины не являются независимыми.

Мы уже отмечали в данном параграфе, что формулы классического метода Рунге–Кутты весьма просты, и это вдохновило целое поколение научных работников и инженеров на то, чтобы писать свои собственные «быстрые и грязные» программы. Неудивительно, что разработка эффективной и надежной программы, реализующей метод Рунге–Кутты с контролем локальной ошибки, выбором шага и интерфейсом, является делом не простым. Превосходная реализация – программа RKF45, в которую включен и метод Фельберга, – приведена в учебнике [Forsythe,

Malcolm, Moler, 1977]. Программы, подобные RKF45, сослужили хорошую службу росту уже и без того большой популярности методов Рунге–Кутты. Далее, множество экспериментальных исследований показало, что для многих нежестких задач при обычных требованиях к точности ($\approx 10^{-5}$) эти программы не менее эффективны, чем любая из имеющихся программ. Если требуется высокая точность, они уже не являются столь эффективными, но такие ситуации встречаются не часто. В практических задачах моделирования редко требуется более пяти или шести верных знаков решения. С другой стороны, в случае нежестких задач различие между наилучшими программами любого типа часто составляет не более 50%, а обычно и того меньше. Потому при выборе среди нескольких программ более важными становятся такие соображения, как надежность и применимость.

Основной недостаток современных программ метода Рунге–Кутты связан с их неспособностью эффективно решать жесткие задачи, которые все чаще встречаются на практике. Еще одна проблема связана с выводом решения. Иногда автоматический выбор шага оказывается столь эффективным, что порождает шаги интегрирования большей длины, чем того желает пользователь. Это особенно существенно в том случае, когда решение выводится на графопостроитель; для гладкого графика требуется много точек, однако программа может делать гораздо бóльшие шаги, не нарушая при этом допуск на ошибку. Запрос на близко расположенные точки выхода для программ типа RKF45 приводит к такому эффекту, будто программу принуждают выбирать шаг меньшим, чем тот, который диктуется требованием к ошибкам. Поскольку каждый шаг обходится в шесть вычислений правых частей, это может оказаться неэффективным; удвоение числа выходных точек на том же самом отрезке интегрирования $[t_0, T]$ заставит программу работать вдвое дольше. Программы многозначных методов не связаны подобным ограничением. Для них нет особой взаимосвязи между внутренними шагами интегрирования и точками выхода. Удвоение числа последних может лишь незначительно – на несколько процентов – увеличить время счета многозначных программ. Многошаговые и многозначные методы имеют еще и то преимущество, что в принципе им требуется только одно вычисление правых частей на каждом внутреннем шаге интегрирования. Однако по методам Рунге–Кутты по-прежнему проводятся активные исследования, и, возможно, через несколько лет программы Рунге–Кутты нового поколения снова окажутся наилучшими из всех имеющихся.

*8.20. Некоторые из опущенных тем

В этом параграфе мы кратко обсудим несколько тем, которые были опущены по причине их сложности или из-за недостатка места.

Обыкновенное дифференциальное уравнение называется *линейным*, если его правая часть представима в виде $f(t, y) = \alpha(t)y + \beta(t)$; в любом

другом случае оно является нелинейным. Так, уравнение

$$y' = t^2 e^{-t^3} y + \sin t$$

линейно, а уравнения

$$y' = \sin y, \quad y' = y^2, \quad y' = \frac{1}{t - y}$$

нелинейны.

Линейные уравнения и их более узкий подкласс – линейные уравнения с постоянными коэффициентами (когда α и β не зависят от времени) – составляют важную часть теории, необходимую для понимания общего случая. Они подсказывают, как нужно строить численные алгоритмы; однако на практике такие уравнения встречаются редко.

Линейные дифференциальные уравнения (которые часто можно решить аналитически) иногда используются для получения приближенных решений. Такой подход был популярен до появления надежных численных методов. Сегодня большинство исследователей предпочитают решать нелинейные уравнения численно. Аналитические методы ныне используются главным образом для получения качественной информации о решении или асимптотических оценок при больших временах, т. е. при $t \gg t_0$.

У *смешанных* (или *гибридных*) *дифференциально-алгебраических* систем одна или несколько левых частей являются нулевыми. Система по-прежнему содержит n неизвестных функций и n уравнений, но лишь m из них, $m < n$, являются дифференциальными уравнениями. В качестве примера заменим последнее уравнение системы ОДУ уравнением вида

$$0 = f_n(t, y_1, y_2, \ldots, y_n).$$

Функция y_n по-прежнему зависит от времени, $y_n = y_n(t)$, и ее значения требуются для вычисления всех функций f_i. В простых ситуациях удается разрешить алгебраическое уравнение относительно y_n и подставить полученное выражение в предшествующие уравнения. Тем самым функция y_n будет исключена и можно будет решать редуцированную систему из $n-1$ уравнений. Однако, как правило, получить выражение для y_n не удается, так что алгебраическое и дифференциальные уравнения должны решаться одновременно. Весьма часто алгебраические уравнения включаются в систему ради того, чтобы сохранялось определенное равновесие, выполнялся принцип консервативности или соблюдались определенные физические ограничения.

Полулинейная неявная система записывается в виде

$$A\,\mathbf{y}' = \mathbf{f}(t, \mathbf{y}),$$

где A – матрица размера $n \times n$, элементы которой могут зависеть от t и y_i. Если A не вырождена (что встречается часто, но не всегда), то умножением на обратную матрицу A^{-1} можно свести систему к обычному виду, но такой подход не всегда рационален. Например, при решении уравнений в частных производных методом конечных элемен-

тов возникают неявные системы большого размера. Матрица A может быть трехдиагональной; в таком случае целесообразно хранить только ненулевые элементы матрицы. Напротив, A^{-1} обычно является полной матрицей, слишком большой для хранения.

В наиболее общей форме ОДУ первого порядка записывается так:

$$f(t, y, y') = 0.$$

Это уравнение наиболее сложно исследовать как теоретически, так и численно.

Некоторые задачи записываются в виде $\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$, $\mathbf{y}(t_0) = \mathbf{A}$, но не все значения A_i задаются в одной и той же точке t_0. Другими словами, нам задано

$$y_i(t_i) = A_i, \quad i = 1, \ldots, n.$$

Такие задачи не являются начальными; они называются *краевыми* задачами. Теория и численные методы решения краевых задач сложнее, чем для начальных задач. Начальные задачи почти всегда имеют решение, но с краевыми задачами дело обстоит иначе. Это, естественно, усложняет ситуацию. Некоторые из методов, разработанных для начальных задач, можно приспособить и для решения краевых задач. Если вас интересуют другие подходы, обратитесь к списку литературы в гл. 1. Мы также не будем обсуждать дифференциальные уравнения с частными производными, решения которых представляют собой функции двух или более переменных. Опять-таки, хотя это очень важный класс задач, его обсуждение выходит за рамки нашей книги.

8.21. Задачи

8.1. Функцию ошибок обычно определяют как интеграл

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt,$$

но ее можно определить и как решение начальной задачи

$$y' = \frac{2}{\sqrt{\pi}} e^{-x^2}, \quad y(0) = 0.$$

(а) Используя подпрограмму SDRIV2, напишите программу, печатающую таблицу значений $\operatorname{erf}(x)$ для $x = 0.0, 0.1, 0.2, \ldots, 1.9, 2.0$. Сравните вашу таблицу с таблицей, полученной в задаче 5.1. Если у вас есть доступ к счетчику (таймеру), определяющему время исполнения программы, то измерьте время генерации таблиц этими двумя методами.

(б) В подпрограмму, вычисляющую y' (вы должны ее написать), вставьте оператор, печатающий входные значения x и y, а также

вычисленное значение y'. Почему ваш листинг свидетельствует о том, что при вычислении квадратур SDRIV2 использует примерно вдвое больше вычислений правой части, чем необходимо? В чем причина такой низкой эффективности?

(в) С помощью подпрограммы SDRIV2 попробуйте вычислить $y(2) = 5.3141\ldots$, где

$$y' = \frac{1}{\sin\sqrt{|t|}}, \quad y(-1) = 0.$$

Затем воспользуйтесь для этой же задачи программой Q1DA. Прокомментируйте трудности, которые вам встретятся. *Указание.* Проследите, чтобы написанная вами подпрограмма для SDRIV2 не называлась F. Вы сможете получить лучшие результаты, если разделите весь интервал точкой 0.

8.2. С помощью SDRIV2 решите жесткую начальную задачу $y' = -1000(y - \sin t) + \cos t$, $y(0) = 1$, $0 \leqslant t \leqslant 1$. Возьмите $EPS = 0.001$. Проведите счет дважды: сначала с MINT = 2, а затем с MINT = 1. В каждом случае напечатайте решение в точках 0.1, 0.2, ..., 1.0. Также напечатайте потребовавшееся число шагов интегрирования IWORK(3). Какой из методов потребовал меньше шагов?

8.3. Парашютист массой 200 фунтов выпрыгнул из самолета на высоте 1000 футов. Через 5 с парашют раскрылся. Допустим, что сопротивление падения равно $a(t) = k_1 y'^2(t)$ при раскрытом куполе, где $k_1 = 1/150$ и $k_2 = 4/150$. На какой высоте был раскрыт парашют, сколько времени прошло до приземления и с какой скоростью приземлился парашютист? *Указание.* Масса = 200/32 = 6.25 слагов (слаг (slug) – единица массы в системе фунт-сила, фут, секунда. – *Перев.*).

8.4. Пушечное ядро старинного образца или ракета, запускаемая под малым углом возвышения, стартует с начальной скоростью $v(0) = v_0$ под углом возвышения $\theta(0) = \theta_0$. Требуется определить траекторию полета в прямоугольной декартовой системе координат (с центром в точке старта, горизонтальной осью x и вертикальной осью y). На снаряд воздействуют только следующие силы: сила тяжести mg в вертикальном направлении, реактивная тяга $T(t)$ в направлении вектора скорости ($T = 0$ в случае пушечного ядра), аэродинамическое сопротивление, направленное противоположно вектору скорости, и сила ветра $W(t)$, действующая, по предположению, только в направлении оси x. Уравнения, описывающие полет снаряда, таковы (см., например, книгу [Ortega, Poole, 1981]):

$$x' = v\cos\theta + W, \quad y' = v\sin\theta,$$

$$\theta' = -\frac{g}{v}\cos\theta, \quad mv' = T - D - mg\sin\theta - m'v.$$

Для нашей задачи можно взять $D(t) = c\rho sv^2/2$, где $c = 0.2$ – коэффициент сопротивления, $\rho = 1.29$ кг/м^3 – плотность воздуха, $s = 0.25$ м^2 – пло-

щадь поперечного сечения снаряда, $g = 9.81$ м/с^2 – ускорение свободного падения, и $v_0 = 50$ м/с.

(а) Пусть $T = 0$, $m = 15$ кг, $m' = 0$ и $W(t) = 0$. Для углов возвышения $0.3 \leqslant \theta_0 \leqslant 1.5$ с шагом в одну десятую радиана получите таблицу, содержащую дальность полета, конечную скорость и время полета. Напечатайте также данные об объеме работы, измеряемом числом обращений к подпрограмме F. Исходя из вашей таблицы, оцените угол возвышения, при котором дальность полета максимальна.

(б) Пусть теперь $W(t) = 10$ м/с, $1 \leqslant t \leqslant 2$. Повторите вычисления пункта (а). Поскольку W действует только в горизонтальном направлении, время полета не должно измениться, однако точка приземления отодвинется примерно на 10 м. Окажется ли эта задача более трудной для SDRIV2? Почему?

(в) Снаряд запускается при порывистом ветре. Повторите вычисления пункта (а), полагая $W(t) = 10 \times$ RNOR( ), $1 \leqslant t \leqslant 2$, где RNOR( ) – нормально распределенная случайная величина с нулевым средним и единичной дисперсией (см. гл. 10). Почему эта задача оказывается еще более трудной для SDRIV2?

8.5. Рассмотрим простую экосистему, состоящую из кроликов, для которых имеется неограниченный запас пищи, и лис, которые для пропитания охотятся за кроликами. Классическая математическая модель, принадлежащая Вольтерра, описывает эту систему двумя нелинейными уравнениями первого порядка:

$$dr/dt = 2r - \alpha rf, \quad r(0) = r_0,$$
$$df/dt = -f + \alpha rf, \quad f(0) = f_0.$$

Здесь t – время, $r = r(t)$ – число кроликов, $f = f(t)$ – число лис и α – положительная константа. При $\alpha = 0$ две популяции не взаимодействуют, и кролики делают то, что у кроликов получается лучше всего, а лисы вымирают от голода. При $\alpha > 0$ лисы встречают кроликов с вероятностью, пропорциональной произведению числа тех и других. В результате таких встреч число кроликов убывает, а число лис (по менее очевидным причинам) возрастает.

Исследуйте поведение этой системы для $\alpha = 0.01$ и различных значений r_0 и f_0, простирающихся от 2 или 3 до нескольких тысяч. Нарисуйте графики наиболее интересных решений. Начертите также график с осями r и f. Поскольку мы умалчиваем о единицах измерения, нет причин ограничивать r и f только целыми значениями.

(а) Вычислите решение для $r_0 = 300$ и $f_0 = 150$. Вы должны обнаружить, что поведение системы периодично с периодом, очень близким к 5 единицам времени. Иными словами, $r(0) \approx r(5)$ и $f(0) \approx f(5)$.

(б) Вычислите решение для $r_0 = 15$ и $f_0 = 22$. Вы должны получить, что число кроликов в конечном счете становится меньше 1. Это можно интерпретировать так, что кролики вымирают. Найдите

начальные условия, которые обрекают на вымирание лис. Найдите начальные условия с $r_0 = f_0$, при которых вымирают оба вида.

(в) Может ли какая-либо компонента точного решения стать отрицательной? Может ли стать отрицательным численное решение? Что произойдет в этом случае? (На практике ответы на последние два вопроса могут зависеть от заданных вами границ погрешностей.)

(г) Было предложено много модификаций этой простой модели, чтобы более полно отразить то, что происходит в природе. Например, можно воспрепятствовать неограниченному возрастанию числа кроликов, заменив первое уравнение на

$$\frac{dr}{dt} = 2\left(1 - \frac{r}{R}\right)r - \alpha rf.$$

Теперь даже при $\alpha = 0$ число кроликов никогда не может превысить R. Выберите какое-либо разумное значение для R и вновь рассмотрите некоторые из поставленных вопросов. В частности, что произойдет с периодичностью решений?

Данную модель интенсивно изучали и математики, и биологи. Проводилось сравнение с реальными наблюдениями за популяциями рысей и зайцев в регионе Гудзонова залива. Много интересного можно найти в увлекательной книге [Haberman, 1977].

8.6. Приводимые ниже дифференциальные уравнения описывают движение тела по орбите около двух значительно более массивных тел. Примером может служить капсула корабля «Аполлон» на орбите Земля–Луна. Координатная система здесь несколько необычная. Три тела определяют в пространстве плоскость и двумерную декартову систему координат в этой плоскости. Начало находится в центре масс двух тяжелых тел; за ось x берется прямая, проходящая через эти тела, а расстояние между ними принимается за единицу. Итак, если μ – отношение массы Луны к массе Земли, то Луна и Земля локализуются в точках с координатами $(1 - \mu, 0)$ и $(-\mu, 0)$ соответственно; координатная система перемещается при обращении Луны вокруг Земли. По предположению масса третьего тела, «Аполлона», пренебрежимо мала в сравнении с двумя другими; его положение как функция времени есть $(x(t), y(t))$. Уравнения выводятся из ньютонова уравнения движения и гравитационного закона обратных квадратов. Первые производные в уравнениях возникают вследствие вращения системы координат и трения, которое полагается пропорциональным скорости с константой пропорциональности f:

$$x'' = 2y' + x - \frac{\tilde{\mu}(x + \mu)}{r_1^3} - \frac{\mu(x - \tilde{\mu})}{r_2^3} - fx',$$

$$y'' = -2x' + y - \frac{\tilde{\mu}y}{r_1^3} - \frac{\mu y}{r_2^3} - fy',$$

где

$$\mu = \frac{1}{82.45}, \quad \tilde{\mu} = 1 - \mu, \quad r_1^2 = (x + \mu)^2 + y^2, \quad r_2^2 = (x - \tilde{\mu})^2 + y^2.$$

Хотя об этих уравнениях известно многое, найти их решения в замкнутом виде не удается. Один интересный круг вопросов связан с изучением периодических решений в отсутствие трения. Известно, что начальные условия

$$x(0) = 1.2, \quad x'(0) = 0, \quad y(0) = 0, \quad y'(0) = -1.04935751$$

приводят к периодическому решению с периодом $T = 6.19216933$, если $f = 0$. Это означает, что «Аполлон» стартует с указанной начальной скоростью, находясь за дальней стороной Луны на высоте, примерно равной 0.2 расстояния Земля - Луна. Получающаяся орбита приводит «Аполлон» очень близко к Земле, затем далеко за противоположную от Луны сторону Земли, потом снова близко к Земле и, наконец, обратно за дальнюю сторону Луны в исходное положение и с исходной скоростью.

(а) Воспользуйтесь SDRIV2 для решения задачи при указанных начальных условиях. Убедитесь, что решение будет периодическим с приведенным выше периодом. Насколько орбита приближает «Аполлон» к Земле? В уравнениях расстояния измеряются от центров планет. Считайте, что Луна находится на расстоянии 238 000 миль от Земли, а Земля является шаром радиусом 4000 миль. Отметим, что начало координат попадает внутрь этого шара, но не совпадает с его центром.

(б) Положив $f = 1$, проинтегрируйте уравнение на отрезке $0 \leqslant t \leqslant 5$ при тех же начальных условиях, что и в (а). Нарисуйте решение на фазовой плоскости, т. е. постройте график в координатах $x(t)$ и $y(t)$. При рассматриваемых условиях «Аполлон» будет «захвачен» Землей и в конце концов упадет на нее.

(в) Положив $f = 0.1$, повторите расчет из пункта (б). Посмотрите на фазовую плоскость. Можете ли вы объяснить, что случилось? Разобраться в ситуации будет легче, если проинтегрировать уравнение на большем интервале, скажем вплоть до $t = 8$.

8.7. Изобретательный пользователь решал систему двух уравнений и пожелал завести собственный счетчик числа обращений к подпрограмме вычисления правой части F. Вместо того чтобы ввести общий блок COMMON, он решил увеличить длину массива Y. Фрагмент его программы выглядел так:

```
REAL Y(3), WORK(500), ...
EXTERNAL F
N = 2
.
.
.
```

```
Y(3) = 0 [Счетчик обращений к F обнулен]
CALL SDRIV2 (N, T, Y, F, ...)
WRITE (*,*) 'Счетчик = ', Y(3)
      .
      .
      .
END
SUBROUTINE F (N, T, Y, YDOT)
   REAL Y(*), YDOT(*)
   .
   .
   .
   Y(3) = Y(3) + 1.
   RETURN
END
```

Его способ сработал, но, когда он проделал то же самое с другой популярной программой численного интегрирования, это привело к резким изменениям в выборе шага. Объясните, какое именно свойство программы SDRIV2 позволяет использовать такой способ. Не рассматривая исходные тексты программ, объясните, каким образом две программы должны различаться в способе обращения к F. Почему такого рода ошибку бывает трудно распознать?

8.8. Мяч брошен с высоты $H_0 = 4$ м. Достигнув земли, он подпрыгивает, т. е. его скорость меняется скачком. Модель, описывающая высоту как функцию от времени (сопротивление воздуха игнорируется), задается системой $y_1' = y_2$, $y_2' = -g$, $y_1(0) = H_0$, $y_2(0) = 0$ при условии, что $y_1(t)$ положительно. Когда $y_1(t)$ обратится в нуль, скажем в момент $t = \tau$, значение $y_2(\tau)$ следует заменить на $-ky_2(\tau)$, $0 < k < 1$.

(а) Воспользуйтесь SDRIV2, чтобы определить моменты первых десяти ударов о землю и высоты первых десяти прыжков, положив $k = 0.7$. Изобразите график $y(t)$ в зависимости от t. *Указание.* Единственная возможность изменить значение одной из компонент решения – это запуск SDRIV2 с новыми начальными условиями.

(б) Повторите вычисления из (а), изменив задачу таким образом: когда y_1 обращается в нуль, то y_2 заменяется на $k|y_2|^{0.9}$.

8.9. Температура стержня, описанного в задаче 7.3, в начальный момент времени, перед включением тока, равняется T_∞. Как функция от времени температура описывается уравнением

$$\frac{dT}{dt} = Q.$$

Сколько потребуется времени, чтобы температура стержня стала отличаться от своего предельного значения не более чем на 2%? (Ответ: примерно 0.58 с.)

8.10. Вычерчивание окружности. Фазовая кривая $(x(t), y(t))$ для начальной задачи

$$x' = -y, \quad y' = x, \quad 0 \leqslant t \leqslant 2\pi, \quad x(0) = 1, \quad y(0) = 0$$

является окружностью. Как метод Эйлера, так и формула трапеций предоставляют экономичный способ построения ряда точек на окружности. Добавим, что если шаг h представляет собой степень двойки, то большинство трансляторов с Фортрана реализует умножение на h посредством сдвига показателя.

(а) Решите выписанную систему обоими методами и сравните объем работы и достигнутую точность. Сравните данный подход с описанным в задаче 4.10.

(б) Проинтегрируйте систему от $t = 0$ до $t = 2\pi$ при различных значениях длины шага h. Сведите в таблицу погрешность в радиусе для точки $t = 2\pi$. Согласуются ли ваши результаты с тем фактом, что метод Эйлера – это метод первого порядка, а формула трапеций – второго?

8.11. Одним из наиболее важных и ранних приложений дифференциальных уравнений в аэродинамике стало уравнение Блазиуса

$$2f''' + ff'' = 0,$$

описывающее профиль скорости несжимаемого газа при обтекании плоской пластины. Для обозначения независимой переменной обычно выбирается буква η. В этой задаче два начальных условия записываются в виде $f(0) = f'(0) = 0$. Также известно, что $f'(\eta) \to 1$ при $\eta \to \infty$.

(а) Поэкспериментируйте с выбором значения для третьего начального условия, используя для решения задачи SDRIV2. Затем нарисуйте график с осями η и $f'(\eta)$ и сравните ваши результаты с опубликованными стандартными профилями (см., например, книгу [Anderson, 1984], с. 531). (Ответ: $f''(0) = 0.332$.) *Предостережение.* По мере того как f' стремится к константе, $f'' \to 0$. Если f'' становится достаточно малым, то возможно исчезновение порядка. В известном отношении такое событие следует рассматривать как удачу, а не как провал; в то же время прерывание программы по этой причине было бы крайне досадным. Некоторые трансляторы позволяют генерировать машинные нули при исчезновении порядка, и в данном случае такой способ подходит.

(б) Толщина потери импульса для несжимаемого течения оказывается пропорциональной величине

$$\theta(u) = \int_0^u f' \cdot (1 - f')\, d\eta.$$

Определите θ (5.0).

8.12. Для решения систем нелинейных уравнений был предложен способ, согласно которому искомое решение рассматривается как решение некоторой системы ОДУ, достигшее положения равновесия. При

этом, чтобы «подобраться» к решению, используются хорошие программы решения ОДУ. Методы этого типа иногда называются *методами продолжения*. Хотя и здесь возникает достаточно много технических сложностей, такой подход часто оказывается вполне практичным, поскольку программное обеспечение для ОДУ весьма робастно. Приведем типичный пример. Пусть требуется решить систему нелинейных уравнений

$$f_i(y_1, \ldots, y_n) = 0, \qquad i = 1, \ldots, n.$$

Пусть $y_1(t), \ldots, y_n(t)$ – неизвестные функции переменной t, $0 \leqslant t \leqslant 1$, а параметры $k_1, \ldots, k_n$ – произвольные, но фиксированные числа. Если бы функции $y_i(t)$ были известны, то, подставив их в левую часть каждого из уравнений, мы получили бы функции переменной t:

$$f_i(y_1(t), \quad y_2(t), \ldots, y_n(t)).$$

Потребуем, чтобы каждая из полученных функций была специальным скалярным кратным функции $1 - t$, а именно

$$f_i(y_1, y_2, \ldots, y_n) = f_i(k_1, k_2, \ldots, k_n)(1 - t).$$

Отметим, что при заданных k_i первый сомножитель правой части будет известен. Если такие функции y_i удастся найти, то их значения при $t = 1$ дадут решение системы нелинейных уравнений. (Почему?) Чтобы получить y_i, продифференцируем по t выписанные выше соотношения:

$$\frac{\partial f_1}{\partial y_1} y_1' + \ldots + \frac{\partial f_1}{\partial y_n} y_n' = -f_1(k_1, \ldots, k_n),$$

$$\vdots$$

$$\frac{\partial f_n}{\partial y_1} y_1' + \ldots + \frac{\partial f_n}{\partial y_n} y_n' = -f_n(k_1, \ldots, k_n),$$

или

$$Jy' = -f, \quad 0 \leqslant t \leqslant 1, \quad y_i(0) = k_1,$$

где J – якобиан системы уравнений. Воспользуйтесь SDRIV2 для решения системы уравнений из задачи 7.7, выбрав $k_1 = k_2 = k_3 = 0$. Вам потребуется программа SGEFS, хотя здесь она неэффективна. Проверьте, совпадут ли ваши результаты с теми, что были ранее получены для задачи 7.7.

8.13. Знаменитой задачей нелинейной механики является задача о *перевернутом маятнике*. Маятник – это жесткий стержень длины L, поддерживаемый на одном конце шарниром без трения. Посредством электромотора вынуждаются быстрые колебания поддерживающего шарнира в вертикальном направлении: $s = A \sin \omega t$. Применение второго закона Ньютона дает уравнение движения $L\theta'' = (g - A\omega^2 \sin \omega t) \sin \theta$, где θ – угол отклонения стержня от вертикального положения (для которого $\theta = 0$), а $g = 386.09$ дюйм/с^2 – ускорение силы тяжести. (При

малых θ, когда $\sin\theta \approx \theta$, это уравнение переходит в известное уравнение Матью.) При $A = 0$ это уравнение называется уравнением колебаний маятника; с ним вы встречались в курсе элементарной физики:

$$\theta'' = \frac{g}{L}\sin\theta.$$

Даже это уравнение не удается разрешить в элементарных функциях. Однако известно, что если $A = 0$ и маятник отпускается из точки покоя, в которой $\theta'(0) = 0$, то период колебаний T находится по формуле

$$T = 4\sqrt{\frac{L}{g}}K(\theta(0)/2),$$

где

$$K(u) = \int_0^{\pi/2} \frac{d\psi}{\sqrt{1 - \cos^2 u \sin^2 \psi}}$$

есть полный эллиптический интеграл первого рода.

Самый интересный аспект этой задачи состоит в том, что существуют области, в которых уравнение движения устойчиво для начальных значений, соответствующих перевернутой конфигурации, и такие случаи были реализованы физически. Напишите программу, которая с помощью SDRIV2 рассчитывает движение $\theta(t)$ для различных значений L, A, ω и начальных значений $\theta(0)$ и $\theta'(0)$. Отладьте вашу программу для случая $L = 10$ дюйм, $A = 0$ дюйм, $\omega = 0$ радиан/с, $\theta(0) = \pi/3$ радиан и $\theta'(0) = 0$ радиан/с. Сравните период, вычисленный с помощью SDRIV2, с приведенным выше значением T. Воспользуйтесь Q1DA, чтобы вычислить K, или возьмите его значение из справочника. Напечатайте значения θ и θ' для двух или трех осцилляций маятника. Когда ваша программа будет работать удовлетворительно, попробуйте рассмотреть более интересные случаи:

Таблица 8.12

L	A	ω	$\theta(0)$	$\theta'(0)$
10	0.5	5.3	3.10	0
10	10.0	100.0	3.10	0
10	10.0	100.0	0.10	0
10	2.0	100.0	0.10	0
10	0.5	200.0	0.05	0

Для каждого случая нарисуйте фазовую плоскость и дайте физическую интерпретацию полученных решений.

8.14. Приведенная ниже модель используется для лабораторного изучения химических веществ, участвующих в цикле истощения атмо-

сферного озона. Она описывает быстрое формирование атомарного хлора из атомов кислорода, водорода и молекулярного хлора:

$$Cl + H_2 \xrightarrow{k_1} HCl + H, \qquad k_1 = 1.6 \times 10^{-14},$$

$$H + Cl_2 \xrightarrow{k_2} HCl + Cl, \qquad k_2 = 2.0 \times 10^{-11},$$

$$H + O_2 \xrightarrow{k_3} HO_2, \qquad k_3 = 3.6 \times 10^{-13},$$

$$Cl + O_2 \xrightarrow{k_4} ClO_2, \qquad k_4 = 1.3 \times 10^{-14},$$

$$Cl + ClO_2 \xrightarrow{k_5} Cl_2 + O_2, \qquad k_5 = 1.4 \times 10^{-10}.$$

(а) Напишите восемь обыкновенных дифференциальных уравнений, описывающих указанные реакции.

(б) С помощью SDRIV2 проинтегрируйте уравнения на промежутке от $t = 0$ до $t = 10^{-4}$ секунд. Задайте MINT = 2 и EPS = 0.001. В качестве начальных значений возьмите $[Cl] = 10^{14}$, $[Cl_2] = 3.25 \times 10^{16}$, $[H_2] = 1.62 \times 10^{18}$, $[O_2] = 4.84 \times 10^{18}$ (молекул на кубический сантиметр). Остальные начальные значения возьмите нулевыми. Напечатайте решение в пятидесяти равноотстоящих точках. Напечатайте также число шагов интегрирования, IWORK(3), и процессорное время решения задачи. В некоторых распечатываемых точках вы можете столкнуться с тем, что шагов интегрирования не потребовалось. Объясните это. Приближаются ли концентрации к своим стационарным значениям? *Указание.* При столь малых концентрациях промежуточные вычисления в SDRIV2 могут сопровождаться исчезновением порядка. Установите на компьютере режим генерирования машинного нуля в случае исчезновения порядка.

(в) Повторите расчет (б) с MINT = 1. Эта задача слабо жесткая. Вы должны убедиться, что для двух первых выводимых на печать точек нежесткий режим более эффективен, поскольку программе приходится отслеживать наиболее быстрые изменения концентраций. Затем концентрации начнут меняться медленно, и нежесткий режим будет ограничивать шаг интегрирования.

8.15. Это задача о распространении волн в среде, в которой скорость распространения колебаний является переменной. В качестве примера можно указать распространение звука под водой, однако данный подход применим и к другим ситуациям. Пусть z обозначает глубину в футах под поверхностью океана (ось направлена вниз), а $c(z)$ – скорость звука на глубине z. Мы игнорируем возможные отклонения в скорости звука, наблюдаемые в горизонтальных направлениях. Можно измерить $c(z)$ для ряда глубин; типичные результаты приведены в таблице. Чтобы

узнать значения $c(z)$ между точками измерений, можно применить интерполяцию кубическими сплайнами. Нам также потребуется вычислять производную $c'(z) = dc(z)/dz$.

Таблица 8.13

z (футы)	$c(z)$ (фут/с)
0	5,042
500	4,995
1,000	4,948
1,500	4,887
2,000	4,868
2,500	4,863
3,000	4,865
3,500	4,869
4,000	4,875
5,000	4,875
6,000	4,887
7,000	4,905
8,000	4,918
9,000	4,933
10,000	4,949
11,000	4,973
12,000	4,991

Поскольку скорость звука изменяется с глубиной, то звуковой «луч» трансформируется в кривую. Этот эффект представляет собой непрерывный аналог хорошо известного явления преломления световых лучей на границе раздела воздух–вода. Основным уравнением модели служит непрерывный аналог закона Снелла. Пусть x обозначает радиальное расстояние (в горизонтальной плоскости) в футах от источника звука, а через $z(x)$ обозначена глубина отдельного луча на расстоянии x. Пусть $\theta = \theta(x)$ – угол между горизонтальной линией и касательной к лучу в точке x, т. е.

$$\operatorname{tg}\theta = \frac{dz}{dx}.$$

Согласно закону Снелла,

$$\frac{\cos\theta}{c(z)} = \text{constant} = A.$$

Фиксированный подводный источник звука испускает лучи во всех направлениях. Мы хотим проследить распространение луча для заданной точки и заданного начального направления. Итак, пусть заданы

$$z(0) = z_0, \quad \frac{dz}{dx}(0) = \operatorname{tg}\theta_0 .$$

Чтобы использовать эти условия, требуется дифференциальное уравнение второго порядка. Такое уравнение можно получить, дифференцируя выписанные выше соотношения и комбинируя их; в результате получается

$$\frac{d^2z}{dx^2} = -\frac{c'(z)}{A^2 c(z)^3}, \text{ где } A^2 = \left(\frac{\cos\theta_0}{c(z_0)}\right)^2 .$$

(а) Воспользуйтесь SDRIV2, чтобы проследить луч, исходящий из точки $z_0 = 2000$ футов под углом $\theta_0 = 5.4$ градуса. (Вам понадобятся программы PCHEZ и PCHEV, чтобы вычислять $c(z)$ и $c'(z)$ во всех требуемых точках.) Проследите за лучом на расстояние 24 морские мили, печатая глубину луча с интервалом в одну милю. Считайте, что в одной морской миле 6076 футов, и не забудьте, что у фортранных тригонометрических функций аргументы измеряются в радианах. Вы должны получить, что на расстоянии 24 мили глубина близка к 3000 футам. Постройте график луча как функции от x.

Теперь предположим, что источник звука находится на глубине 2000 футов, а приемник — на глубине 3000 футов и на расстоянии 24 мили от источника. Предшествующие вычисления показывают, что один из лучей, достигающий приемника, покидает источник под углом, близким к 5.4°. В силу нелинейности уравнения могут быть и другие лучи, покидающие источник под различными углами, но достигающие того же приемника. Пусть $x_f = 24$ морским милям. Если угол θ_0 меняется, то меняется и $z(x_f)$. Мы ищем такие начальные значения θ_0, для которых $z(x_f) = 3000$.

(б) Напишите подпрограмму-функцию F (THETA), которая отслеживает луч с начальным углом THETA и выдает величину $z(x_f) - 3000$. Напечатайте таблицу таких величин для углов THETA из диапазона от -10 до 10°. Поскольку такие вычисления будут весьма дорогими, то шаг таблицы будет зависеть от доступного вам машинного времени и эффективности вашей программы.

(в) Воспользуйтесь программой FZERO и возьмите в качестве стартовых значения, полученные в (б), чтобы найти лучи, проходящие через приемник (или наиболее близко к нему).

(г) Предположим, что дно океана находится на глубине 12000 футов, а поверхности океана приписано значение 0 футов. Если вы

повторите вычисления из (б), но возьмете для THETA интервал от -15 до 15°, то сможете обнаружить, что некоторые из лучей достигают дна или поверхности океана. Модифицируйте вашу программу так, чтобы интегрирование прекращалось, как только луч достигнет дна океана. Кроме того, измените программу так, чтобы луч, достигший поверхности, отражался вниз под отрицательным углом. Постройте графики для 31 луча.

8.16. Решите следующую нелинейную краевую задачу для функции $y(x)$, поставленную на промежутке $0 \leqslant x \leqslant 1$:

$$y'' = y^2 - 1, \quad y(0) = 0, \quad y(1) = 1.$$

(а) Допустим, что нам известно значение производной $z = y'(0)$. Тогда можно воспользоваться программой SDRIV2, чтобы решить задачу

$$y'' = y^2 - 1, \quad y(0) = 0, \quad y'(0) = z.$$

Каждому значению z будет соответствовать определенное значение $y_z(1)$. Один из способов решения краевых задач, называемый *методом стрельбы*, заключается в том, чтобы решить нелинейное уравнение

$$f(z) = y_z(1) - 1 = 0.$$

Воспользуйтесь методом стрельбы и программой FZERO для решения краевой задачи.

(б) Теперь попробуем решить нашу задачу другим способом. Разобьем отрезок на n равных подынтервалов:

$$0 = x_0 < x_1 < x_2 < \ldots < x_{n-1} < x_n = 1.$$

Заменим дифференциальное уравнение разностными уравнениями (о разностных методах см., например, книги [Самарский А. А. Введение в численные методы.– М.: Наука, 1987]; [Самарский А. А. Теория разностных схем.– М.: Наука, 1989].– *Перев.*), содержащими $n - 1$ неизвестных y_i (которые аппроксимируют числа $y(x_i)$):

$$y_{i+1} - 2y_i + y_{i-1} = h^2(y_i^2 - 1), \quad i = 1, 2, \ldots, n - 1,$$

где $y_0 = 0$ и $y_n = 1$. Решите полученную нелинейную грехдиагональную систему, применяя программу SNSQE, например, при $n = 50$.

(в) В качестве третьего подхода попробуйте следующий. Заметим, что уравнение $y'' = y^2 - 1$ можно записать в виде

$$\frac{d}{dx}\left(\frac{(y')^2}{2} - \frac{y^3}{3} + y\right) = 0.$$

Отсюда

$$\frac{(y')^2}{2} - \frac{y^3}{3} + y = c,$$

где c — некоторая константа. Поскольку $y(0) = 0$, то $y'(0) = \sqrt{2c}$. Поэтому, вычислив c, мы сможем свести краевую задачу к начальной задаче. Интегрируя приведенное выше уравнение, получим

$$x = \int_0^y \frac{dy}{\sqrt{2(c + y^3/3 - y)}}.$$

Теперь воспользуйтесь FZERO и Q1DA, чтобы из уравнения

$$1 = \int_0^1 \frac{dy}{\sqrt{2(c + y^3/3 - y)}}$$

найти c, а затем используйте SDRIV2 и получите искомое решение.

8.22. Пролог: SDRIV2

```
      SUBROUTINE SDRIV2 (N, T, Y, F, TOUT, MSTATE, NROOT,
     EPS, EWT, MINT, WORK, LENW, IWORK, LENIW, G)
C*** НАЧАЛО ПРОЛОГА SDRIV2
C*** ДАТА СОЗДАНИЯ 790601 (ГГММДД)
C*** ДАТА ПЕРЕСМОТРА 871105 (ГГММДД)
C*** КЛАССИФИКАЦИЯ NO. I1A2, I1A1B
C*** КЛЮЧЕВЫЕ СЛОВА: Оду, жесткие, обыкновенные дифферен-
C                     циальные уравнения, начальные задачи,
C                     метод Гира, одинарная точность
C*** АВТОРЫ: KAHANER D.K., NATIONAL BUREAU OF STAN-
C            DARDS, SUTHERLAND C.D., LOS ALAMOS NA
C            LABORATORY
C*** НАЗНАЧЕНИЕ: SDRIV2 предназначена для решения системы из
C                N обыкновенных дифференциальных уравнений
C                вида dY(I)/dT = F(Y(I),T) с заданным началь-
C                ным условием Y(I) = YI. В программе реализо-
C                ваны режимы решения как жестких, так и не-
C                жестких дифференциальных уравнений. SDRIV2
C                использует однократную точность.
C*** ОПИСАНИЕ
C     Из книги D. Kahaner, C. Moler, S.Nash
C              "Numerical Methods and Software"
C              Prentice-Hall, 1988
C
C
C
C     SDRIV2 предназначена для решения системы из N обыкновенных
C     дифференциальных уравнений вида dY(I)/dT = F(Y(I),T) с задан-
```

```
C     ным начальным условием Y(I) = YI. В программе реализованы
C     режимы решения как жестких, так и нежестких дифференциальных
C     уравнений. Для каждой конечной точки интегрирования T прог-
C     рамма SDRIV2 вызывается один раз.
C
C     Описание параметров:
C
C     В операторе вызова подпрограммы SDRIV2 пользователь должен
C     в качестве параметров указывать переменные для тех величин,
C     которые могут быть изменены в SDRIV2. Параметры подпрог-
C     раммы имеют следующий смысл.
C
C        N      -(На входе) Число дифференциальных уравнений.
C
C        T      -Независимая переменная. На входе при первом обра-
C               щении T есть начальная точка. На выходе T есть точка,
C               которой соответствует решение.
C
C        Y      -Вектор зависимых переменных. На входе при первом
C               обращении в Y задаются начальные значения. На
C               выходе Y содержит вектор вычисленного решения.
C               Массив Y передается через список параметров в под-
C               программу F и в подпрограмму-функцию G, написан-
C               ные пользователем. Информация, необходимая для
C               F и G, может быть размещена в этом массиве начиная
C               с (N + 1)-й компоненты. (Замечание. Изменения, вно-
C               симые пользователем в первые N компонент, проявятся
C               лишь в случае повторного обращения к программе, т. е.
C               после изменения MSTATE на +1(-1).)
C
C        F      -Подпрограмма, которую должен написать пользовате-
C               ль. Ее имя должно быть описано в операторе EXTER-
C               NAL вызывающей программы пользователя. Это под-
C               программа следующего вида:
C                     SUBROUTINE F (N, T, Y, YDOT)
C                     REAL Y(*), YDOT(*)
C                       .
C                       .
C                     YDOT(1) = ...
C                       .
C                       .
C                     YDOT(N) = ...
C                     END (Пример)
C               Она вычисляет YDOT = F(Y,T)-правые части диф-
C               ференциальных уравнений. Здесь Y-массив длины не
C               меньше N. Фактический размер массива Y описы-
```

```
C           вается пользователем в программе, вызывающей
C           SDRIV2. Таким образом, указание размерности для
C           Y в F, необходимое по правилам языка Фортран,
C           в действительности не приводит ни к какому распреде-
C           лению памяти. При обращении к этой подпрограмме
C           первые N компонент массива Y содержат приближения
C           к компонентам решения. Пользователь не должен из-
C           менять значения этих компонент. В данной подпрог-
C           рамме YDOT – массив длины N. Пользователю сле-
C           дует вычислить YDOT(I) только для I от 1 до N. Как
C           правило, возврат из F снова передает управление
C           подпрограмме SDRIV2. Однако, если пользователь
C           желает прервать вычисления, т.е. передать управление
C           в программу, вызывающую SDRIV2, ему следует прис-
C           воить N – нулевое значение. SDRIV2 сигнализирует об
C           этом, возвратив MSTATE со значением 6 (–6). Смена
C           значения N в F не повлияет на значение параметра N,
C           указанного в списке параметров SDRIV2.
C
C
C     TOUT  – (На входе) Точка, в которой требуется вычислить
C           решение.
C
C     MSTATE – Указатель целого типа, описывающий режим интегри-
C           рования. Пользователь может задать MSTATE равным
C           +1 или –1. Если MSTATE положительно, то подпрог-
C           рамма, миновав при интегрировании точку TOUT,
C           проинтерполирует решение. Это наиболее эффектив-
C           ный режим. Если MSTATE отрицательно, то подпрог-
C           рамма будет подбирать шаги интегрирования, чтобы
C           попасть точно в точку TOUT (режим полезен в случае
C           сингулярности правее точки TOUT). Смысл модулей
C           значений MSTATE таков:
C              1 (На входе) Первое обращение к подпрограмме.
C                Данное значение должно быть задано пользова-
C                телем. При всех последующих обращениях поль-
C                зователю необходимо лишь следить за величиной
C                MSTATE. До тех пор, пока к SDRIV2 не обрати-
C                лись заново, пользователем может быть изменен
C                только знак MSTATE. (Для удобства пользова-
C                теля, который хотел бы отказаться от задания
C                начальных условий, SDRIV2 допускает вызов со
C                значениями параметров MSTATE = +1(–1) и
C                TOUT = T. В таком случае подпрограмма вернет
C                MSTATE без изменений, т.е. MSTATE = +1(–1).)
C              2 (На выходе) Успешное завершение интегрирова-
```

```
C          ния. Если требуется продолжить интегрирование
C          (в том же направлении), то просто увеличьте
C          TOUT и обратитесь снова. Все прочие параметры
C          будут установлены автоматически.
C        3 (На выходе) Неудачное завершение. Выполнено
C          1000 шагов интегрирования, но точка TOUT не
C          достигнута. Пользователь может продолжить
C          интегрирование, просто снова обратившись к
C          SDRIV2. Если это не ошибка в постановке задачи,
C          то наиболее вероятной причиной может быть
C          попытка решить жесткую систему уравнений в
C          режиме нежесткого интегрирования. (См. ниже
C          описание параметра MINT.)
C        4 (На выходе) Неудачное завершение. Затребована
C          слишком высокая точность. EPS было увеличено
C          до уровня, рассматриваемого подпрограммой как
C          разумный. Пользователь может продолжить ин-
C          тегрирование, просто снова обратившись к
C          SDRIV2.
C        5 (На выходе) Корень был найден в точке, лежащей
C          левее точки TOUT. Пользователь может продол-
C          жить интегрирование к точке TOUT, просто
C          снова обратившись к SDRIV2.
C        6 (На выходе) Неудачное завершение. В подпрог-
C          рамме F для N было установлено значение 0.
C        7 (На выходе) Неудачное завершение. В подпрог-
C          рамме-функции G для N было установлено зна-
C          чение 0. (См. ниже описание G.)
C
C   NROOT - (На входе) Число уравнений, для которых ищется
C          корень. Если NROOT есть нуль, то режим поиска
C          корня не активизируется. Этот режим полезен, когда
C          требуется получить результат в точках, неизвестных
C          заранее, но зависящих от решения, например в точках,
C          где некоторая компонента решения принимает опреде-
C          ленное значение. Поиск корня ведется с помощью
C          подпрограммы-функции G, написанной пользователем
C          (см. ниже описание G). SDRIV2 пытается найти значе-
C          ние T, при котором одно из уравнений меняет знак.
C          SDRIV2 может найти в лучшем случае один корень на
C          уравнение на внутреннем шаге интегрирования, а затем
C          возвратит решение или в точке TOUT, или в корне
C          в зависимости от того, какая из этих точек встретится
C          первой в направлении интегрирования. Номер уравне-
C          ния, корень которого был найден, записывается в
C          шестую компоненту массива IWORK.
```

```
C              Замечание. NROOT никогда не изменяется в SDRIV2.
C     EPS      - На входе: требуемая относительная точность во всех
C              компонентах решения. Значение EPS = 0 допускается.
C              На выходе: откорректированная относительная точ-
C              ность, если входное значение оказалось слишком ма-
C              лым. Значение EPS следует задать по возможности - в
C              пределах разумного - большим, поскольку объем вы-
C              числений в SDRIV2 возрастет при уменьшении EPS.
C
C     EWT      - (На входе) Нуль задачи, т.е. наименьшее значение
C              (компоненты) решения, которое может быть допущено
C              по смыслу задачи. Эта величина используется
C              внутри SDRIV2 при вычислении массива YWT(I) =
C              = MAX(ABS(Y(I), EWT)). Оценка погрешности на шаге,
C              деленная на YWT(I), поддерживается меньшей, чем
C              EPS. При EWT, равном нулю, имеем просто-напросто
C              контроль относительной погрешности. Однако, задав
C              EWT слишком малым, можно столкнуться с увеличе-
C              нием времени счета.
C
C     MINT     - (На входе) Указатель метода интегрирования.
C                MINT = 1 - используются методы Адамса; рекомен-
C                       дуется для нежестких задач.
C                MINT = 2 - используются «жесткие» методы Гира (т.е.
C                       формулы дифференцирования назад); реко-
C                       мендуются для решения жестких задач.
C                MINT = 3 - подпрограмма динамически выбирает или
C                       методы Адамса, если задача оказывается не-
C                       жесткой, или методы Гира, когда она явля-
C                       ется жесткой.
C                Значение MINT не может быть переопределено без
C                повторного обращения к подпрограмме, т.е. без за-
C                дания для модуля MSTATE значения 1.
C
C     WORK
C     LENW     - (На входе)
C                WORK - вещественный массив длины LENW, исполь-
C                зуемый внутри в качестве рабочего. Пользователь
C                должен отвести в вызывающей программе под этот
C                массив память, например, посредством описания вида
C                                   REAL WORK(...)
C                Длина массива WORK должна быть не меньшей чем
C                  16 × N + 2 × NROOT + 204,        если MINT = 1,
C
C                  N × N + 10 × N + 2 × NROOT + 204,
```

```
C                                          если MINT = 2,
C          N × N + 17 × N + 2 × NROOT + 204,
C                                          если MINT = 3
C     (установленный размер памяти следует присвоить пар-
C     аметру LENW). Содержимое массива WORK не
C     должно изменятся между обращениями к SDRIV2.
C
C  IWORK
C  LENIW - (На входе)
C     IWORK - целый массив длины LENIW, используемый
C     внутри в качестве рабочего. Пользователь должен
C     отвести в вызывающей программе под этот массив
C     память, например, посредством описания вида
C                        INTEGER IWORK(...)
C     Длина массива IWORK должна быть не меньшей чем
C       21,          если MINT = 1,
C       N + 21,      если MINT = 2 или 3,
C     и установленный размер памяти следует присвоить
C     параметру LENIW. Содержимое массива IWORK не
C     должно изменяться между обращениями к SDRIV2.
C
C  G     - Вещественная фортранная подпрограмма-функция,
C     которую должен написать пользователь в том случае,
C     если NROOT не есть 0. В таком случае ее имя должно
C     быть описано в операторе EXTERNAL ь вызывающей
C     программе. G последовательно вызывается с различ-
C     ными значениями IROOT, чтобы получить значение
C     каждой из функций NROOT, корни которых подлежат
C     вычислению. Имеет следующий вид:
C        REAL FUNCTION G(N, T, Y, IROOT)
C        REAL Y(*)
C        GO TO (10, ...), IROOT
C       10 G = ...
C           .
C           .
C           .
C        END (Пример)
C     Здесь Y - массив длины не меньшей N; его первые
C     N компонент содержат решение в точке T. Пользова-
C     тель не должен изменять эти значения. Фактический
C     размер массива Y определяется по его описанию в
C     программе пользователя, вызывающей SDRIV2. Таким
C     образом, указание размерности для Y в G, необходи-
```

```
C           мое по правилам языка Фортран, в действительности
C           не приводит ни к какому распределению памяти. При
C           нормальном завершении управление передается в
C           SDRIV2. Однако, если пользователь желает прервать
C           вычисления, т.е. передать управление в программу,
C           вызывающую SDRIV2, ему следует присвоить N нуле-
C           вое значение. SDRIV2 сигнализирует об этом, возвра-
C           тив MSTATE со значением, равным 7 (-7). В таком
C           случае номер обработанного в G уравнения будет
C             размещен в шестой компоненте массива IWORK.
C             Смена значения N в G не повлияет на значение
C             параметра N, указанного в списке параметров
C             SDRIV2.
C
C     ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ ДЛЯ ПОЛЬЗОВАТЕЛЯ
C
C     В первые три компоненты массива WORK и первые пять эле-
C     ментов массива IWORK будет помещена следующая статисти-
C     ческая информация:
C       AVGH    - среднее значение использованных шагов,
C       HUSED   - последний (успешный) использованный шаг,
C       AVGORD  - среднее значение использованных порядков,
C       IMXERR  - индекс того элемента вектора решения, на котором
C                 достигается максимальная ошибка при последней
C                 проверке,
C       NQUSED  - последний (успешно) использованный порядок,
C       NSTEP   - число шагов, выполненных после последней ини-
C                 циализации,
C       NFE     - число вычислений правых частей,
C       NJE     - число вычислений матриц-якобианов.
C
C*** Ссылки: Gear C. W., Numerical Initial Value Problems in Ordinary
C            Differential Equations, Prentice-Hall, 1971.
C
C*** ВЫЗЫВАЕМЫЕ ПОДПРОГРАММЫ: SDRIV3, XERROR
C*** КОНЕЦ ПРОЛОГА SDRIV2
```

Глава 9

Оптимизация и нелинейный метод наименьших квадратов

9.1. Введение

Слово *оптимизация* означает минимизацию или максимизацию функции. Задачи оптимизации часто возникают непосредственно в процессе поиска наилучшей конструкции какого-либо объекта. В качестве простого примера предположим, что компания по производству супов проектирует цилиндрическую банку для супа и хочет минимизировать количество требуемого металла. Пусть h – высота банки, а r – радиус ее дна, и пусть банка должна содержать 16 унций грибного супа. Тогда мы получаем следующую задачу оптимизации:

минимизировать площадь металлической поверхности $2\pi r^2 + 2\pi rh$
при условии, что объем $= 2\pi r^2 h = 16$.

Решение этой задачи дает $r \approx 1.08$, $h \approx 2.17$, а минимальная площадь металлической поверхности равна 22.14. Если вы сравните полученное решение с фактическими параметрами суповой банки, вы, вероятно, обнаружите, что форма банки не оптимальна с точки зрения использования металла. Это происходит потому, что при проектировании учитываются и другие факторы, такие, как форма этикетки и способ, которым банки будут укладывать на полках в магазине. Эти соображения приводят к появлению дополнительных ограничений в постановке задачи.

В более общем случае задача оптимизации заключается в определении максимума или минимума (и соответствующих ему аргументов) вещественной функции n вещественных переменных $F(x_1, \ldots, x_n)$ на множестве S n-мерного пространства. Функция F представляет собой цель, а множество S выражает ограничения задачи. Положим в приведенном выше примере с суповой банкой $x_1 = r =$ радиус и $x_2 = h =$ = высота. Тогда задача примет следующий вид:

$$\text{минимизировать } F(x_1, x_2) = 2\pi x_1^2 + 2\pi x_1 x_2$$
$$\text{при условии } x \in S = \{(x_1, x_2) \mid 2\pi x_1^2 x_2 = 16\}.$$

Иногда множество S представляет собой все n-мерное пространство; в этом случае говорят, что решается задача оптимизации *без ограничений* (или задача *безусловной оптимизации*). В противном случае рассматривается задача *с ограничениями* в виде условий, определяющих множество S. Совокупность ограничений на решение зависит от моделируемой

области приложений. Простое, но часто встречающееся ограничение требует, чтобы переменные были неотрицательны, т. е. $S = \{x \mid x \geqslant 0\}$; например, не имеет смысла строить -7 фабрик или покупать -12 стиральных машин. В более общем случае S может быть задано множеством функций, удовлетворяющих условиям равенства или неравенства, например $S = \{x \mid g_i(x) \geqslant 0,\ i = 1, \ldots, m\}$, где g_i – заданные функции от x. Для ограничений неотрицательности $x \geqslant 0$ мы имеем $g_i(x) = x_i$. В примере с банкой для супа можно использовать запись

$$S = \{(x_1, x_2) \mid 2\pi x_1^2 x_2 - 16 = 0\}.$$

Точка x, которая удовлетворяет ограничениям, называется *допустимой.*

Иногда задачи оптимизации возникают косвенным образом, в процессе решения некоторой другой задачи. Так, задача приближения данных с помощью нелинейной модели (см. § 1 гл. 6), например

$$b(t) \approx x_1 e^{x_2 t},$$

приводит к системе уравнений

$$\begin{aligned} b(t_1) - x_1 e^{x_2 t_1} &\approx 0, \\ b(t_2) - x_1 e^{x_2 t_2} &\approx 0, \\ &\vdots \\ b(t_m) - x_1 e^{x_2 t_m} &\approx 0. \end{aligned}$$

В идеале хотелось бы точно удовлетворить всем уравнениям, но это в общем случае невозможно. Если вместо этого использовать метод наименьших квадратов, то возникнет следующая задача оптимизации:

$$\min_{x_1, x_2} F(x_1, x_2) \equiv \sum_{j=1}^{m} [b(t_j) - x_1 e^{x_2 t_j}]^2.$$

Эта задача носит название *нелинейной задачи наименьших квадратов.* Она обсуждается более подробно в § 6. Другие методы приближения данных, такие, как оценивание по методу максимального правдоподобия в статистике, также могут приводить к задачам оптимизации.

Задача оптимизации может состоять в минимизации или максимизации. Методы минимизации легко преобразуются в методы максимизации, поскольку минимизация функции $F(x)$ – то же самое, что максимизация функции $-F(x)$. Хотя обе задачи эквивалентны, в этой главе мы для удобства сконцентрируем внимание на задаче минимизации.

Задачи оптимизации тесно связаны с системами нелинейных уравнений. Если x^* минимизирует $F(x)$ в отсутствие ограничений, то первые

производные целевой функции будут равны нулю, т. е.

$$\frac{\partial}{\partial x_1} F(x_1, x_2, \ldots, x_n)\Big|_{x=x^*} = 0,$$

$$\frac{\partial}{\partial x_2} F(x_1, x_2, \ldots, x_n)\Big|_{x=x^*} = 0,$$

$$\vdots$$

$$\frac{\partial}{\partial x_n} F(x_1, x_2, \ldots, x_n)\Big|_{x=x^*} = 0.$$

Это система нелинейных уравнений относительно переменных x.

Такая точка x, в которой производные функции $F(x)$ равны нулю, называется *критической* точкой. Не все критические точки минимизируют $F(x)$. В действительности критическая точка может быть минимумом, максимумом или (если она не дает оптимума ни в каком смысле) *седловой точкой* функции. Нам нужно определить еще два термина. Будем говорить, что $F(x)$ имеет *глобальный минимум* в точке x^*, если точка x^* допустима и $F(x^*) \leqslant F(x)$ для всех допустимых точек x. Аналогично, $F(x)$ имеет *локальный минимум* в точке x^*, если точка x^* допустима и $F(x^*) \leqslant F(x)$ для всех допустимых точек x, которые «близки» к x^*. Поскольку численные методы, которые мы будем использовать, основаны на рядах Тейлора и, следовательно, позволяют исследовать функцию $F(x)$ в окрестности выделенной точки, мы сможем распознать локальный минимум, но распознать глобальный минимум будет трудно.

Неудивительно, что методы решения нелинейных уравнений (см. гл. 7) аналогичны методам решения задач оптимизации. Действительно, методы гл. 7 можно применить к приведенной выше системе нелинейных уравнений. Зачем же тогда нужна отдельная глава о методах оптимизации? На это существуют три важные причины.

(1) Метод оптимизации гарантирует сходимость к локальному минимуму (точке, где обращаются в нуль первые производные), в то время как метод решения нелинейных уравнений этого не гарантирует (он может не найти такой точки; он также может найти максимум или седловую точку функции).

(2) Метод оптимизации может дать выигрыш по объему памяти и по числу арифметических операций.

(3) Задачи оптимизации часто содержат ограничения на переменные x, которые труднее учесть в алгоритме решения нелинейных уравнений.

Мы ограничимся поиском *приближенных локальных* минимумов или максимумов в задачах оптимизации без ограничений, и вот почему.

Локальный минимум функции найти легче, чем абсолютный, или глобальный, минимум на всей области. Отчасти это объясняется тем, что методы математического анализа, такие, как разложение в ряд

Тейлора, дают информацию о поведении функции вблизи заданной точки, но не дают детальной информации об общем поведении функции на всей области. Иногда имеется дополнительная информация о задаче (скажем, результаты физического эксперимента), с помощью которой можно определить примерное положение глобального минимума. Затем можно использовать методы локальной минимизации, чтобы получить более точное решение. Однако существует несколько общих методов глобальной оптимизации, которые обсуждаются в указываемой нами литературе.

Задачи с ограничениями решать значительно труднее, чем задачи безусловной оптимизации, особенно если функции ограничений нелинейны, как в примере с банкой для супа. Однако, поскольку задачи с ограничениями часто решают с помощью различных вариантов методов безусловной оптимизации, материал этой главы может служить основой для изучения методов оптимизации с ограничениями. Например, в § 9 гл. 6 обсуждается, как задачи с линейными ограничениями преобразуются в задачи без ограничений.

Поскольку между задачами оптимизации и системами нелинейных уравнений существует тесная связь, многие замечания из § 1 гл. 7 применимы и здесь. В частности, у нас нет способов минимизировать функцию общего вида за конечное время, даже при поиске локального минимума. Не существует также формулы решения, за исключением некоторых особых случаев. Поэтому все методы, которые мы получим, будут итеративными и будут находить только приближенное решение задачи.

Однако с точки зрения компьютерной арифметики имеется гарантированный способ определения глобального минимума функции за конечное число шагов, по крайней мере если есть верхние и нижние границы для всех переменных. В качестве иллюстрации рассмотрим минимизацию функции одной переменной $F(x)$, где x удовлетворяет ограничению $1 \leqslant x \leqslant 2$. На компьютере числа представляются конечным числом разрядов; поэтому, если относительная машинная точность задается как $\varepsilon_{\text{маш}} = 10^{-16}$, достаточно рассмотреть лишь порядка 10^{16} возможных значений x. Если $F(x)$ можно вычислить за 10^{-7} секунд и если начать поиск минимума в полночь 1 января 1987 г., то глобальный минимум будет найден к обеду в воскресенье 22 ноября 2303 г. Однако это не очень практично.

9.2. Одномерная оптимизация

Начнем с обсуждения методов оптимизации функций одной переменной. На это есть несколько причин.

(1) Одномерные задачи часто встречаются на практике.

(2) Для одномерного случая разработаны особенно эффективные и надежные методы.

(3) Многие идеи, используемые в многомерных задачах, можно проиллюстрировать в одномерном случае.

(4) Основной шаг нашего алгоритма для n измерений будет состоять в приближенном решении одномерной задачи.

Мы опишем два метода решения одномерных задач. Первый метод использует значения производных, чтобы достичь высокой скорости сходимости. Второй метод использует только значения функции и является надежным, но довольно медленным.

9.2.1. Метод Ньютона

Рассмотрим минимизацию одномерной функции $F(x)$ без всяких ограничений на x. Будем предполагать, что функция $F(x)$ имеет по крайней мере две непрерывные производные и ограничена снизу. Поскольку в наших численных методах используются производные, первое предположение необходимо, чтобы эти методы можно было реализовать. Если же функция не ограничена снизу, она может не иметь локального минимума, и тогда задача может оказаться бессмысленной.

Для функции общего вида не существует формулы для точки, минимизирующей $F(x)$. В основе метода Ньютона лежит идея аппроксимации $F(x)$ более простой функцией, которую мы можем минимизировать, и использования точки минимума более простой функции в качестве новой оценки точки минимума функции $F(x)$. Затем процесс повторяется для этой новой точки. Поскольку линейная функция не имеет конечного минимума, простейшей функцией, которую можно для этого использовать, является квадратическая. Чтобы построить квадратичную аппроксимацию, предположим, что x_k – текущая оценка решения x^*, и рассмотрим разложение функции F в ряд Тейлора относительно точки x_k

$$F(x_k + p) = F(x_k) + pF'(x_k) + \frac{1}{2}p^2 F'(x_k) + \ldots .$$

Исходную задачу минимизации можно аппроксимировать с использованием разложения в ряд Тейлора:

$$\begin{aligned} F(x^*) &= \min_x F(x) = \\ &= \min_p F(x_k + p) = \\ &= \min_p [F(x_k) + pF'(x_k) + \frac{1}{2}p^2 F''(x_k) + \ldots] \approx \\ &\approx \min_p [F(x_k) + pF'(x_k) + \frac{1}{2}p^2 F''(x_k)] . \end{aligned}$$

Теперь, когда члены разложения высокого порядка опущены, последняя задача состоит просто в минимизации квадратичной функции p. Мы надеемся, что минимизация этой квадратичной аппроксимации будет эффективной заменой прямой минимизации $F(x)$. Чтобы минимизировать квадратичную функцию, возьмем производную по p и приравняем ее к нулю, что дает

$$p = -F'(x_k)/F''(x_k).$$

Поскольку p – аппроксимация шага, который привел бы нас от x_k к решению x^* исходной задачи, то $x^* \approx x_k + p$ и алгоритм описывается формулой

$$x_{k+1} = x_k + p = x_k - F'(x_k)/F''(x_k).$$

Этот алгоритм называется *методом Ньютона*. Мы получили в точности ту же формулу, которая получается, если решать нелинейное уравнение

$$F'(x) = 0$$

с помощью метода Ньютона в том виде, который описан в гл. 7.

Метод Ньютона можно описать и графически. Пусть $q(x)$ – квадратичная функция, которая интерполирует $F(x)$ следующим образом:

$$q(x_k) = F(x_k),$$
$$q'(x_k) = F'(x_k),$$
$$q''(x_k) = F''(x_k)$$

(см. § 1 гл. 4). Точка, минимизирующая $q(x)$, – это как раз x_{k+1}, оценка Ньютона, выведенная в предыдущем абзаце. Этот подход иллюстрируется рисунком 9.1.

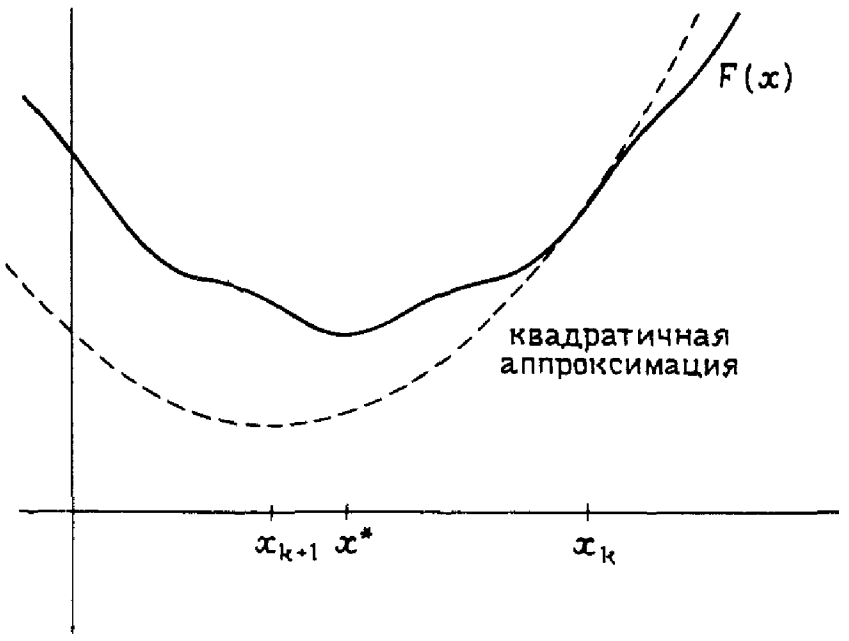


Рис. 9.1. Метод Ньютона.

Главное преимущество метода Ньютона заключается в том, что он имеет высокую (квадратичную) скорость сходимости. То есть если точка x_k достаточно близка к x^* и если $F''(x^*) > 0$, то

$$|x_{k+1} - x^*| \leqslant \beta |x_k - x^*|^2,$$

где β – неотрицательная константа, которая зависит от минимизируемой функции. Грубо говоря, это значит, что число верных значащих цифр в x_k будет удваиваться с каждой итерацией. Работу метода Ньютона демонстрирует пример 9.1.

Пример 9.1. Метод Ньютона.

В качестве иллюстрации применим метод Ньютона к функции $F(x) = \sin(x) - \cos(x)$ с начальным приближением $x_0 = -0.5$. На первой итерации

$$F'(x_0) = \cos(x_0) + \sin(x_0) = \cos(-0.5) + \sin(-0.5) = 0.3982,$$
$$F''(x_0) = -\sin(x_0) + \cos(x_0) = -\sin(-0.5) + \cos(-0.5) = 1.3570,$$
$$p = -F'(x_0)/F''(x_0) = -0.3982/1.3570 = -0.2934,$$
$$x_{k+1} = x_k + p = -0.5 - 0.2934 = -0.7934.$$

Полностью все итерации приведены в табл. 9.1. Как и ожидалось, метод сходится быстро: решение получено всего лишь за 3 итерации.

Таблица 9.1

k	x_k	$\lvert x_k - x^* \rvert$
0	−0.5000000000000000	2.8×10^{-1}
1	−0.7934079930260234	8.0×10^{-3}
2	−0.7853979920965160	1.7×10^{-7}
3	−0.7853981633974483	1.3×10^{-17}

В предположении, что $F''(x^*) > 0$, нет ничего необычного. Если $F''(x^*) < 0$, то x^* максимизирует, а не минимизирует $F(x)$. Если же $F''(x^*) = 0$, то $F'(x)$ имеет кратный нуль в x^*. В этом случае метод Ньютона будет сходиться линейно, т. е. $|x_{k+1} - x^*| \approx c|x_k - x^*|$ с некоторой константой $c \leqslant 1$.

Когда метод Ньютона сходится, он чрезвычайно эффективен для минимизации функции. Однако его сходимость не гарантирована. Метод Ньютона может потерпеть неудачу в нескольких отношениях.

(1) Каждая итерация метода Ньютона основана на приближении функции $F(x)$ первыми тремя членами ряда Тейлора. Если это плохая аппроксимация, нет никаких оснований считать, что x_{k+1} будет ближе к x^*, чем x_k. См. в качестве иллюстрации рис. 9.2.

(2) Хотя шаг $p = -F'(x_k)/F''(x_k)$ определен всегда, когда $F''(x_k) \neq 0$, квадратичная аппроксимация имеет минимум только в том случае, если

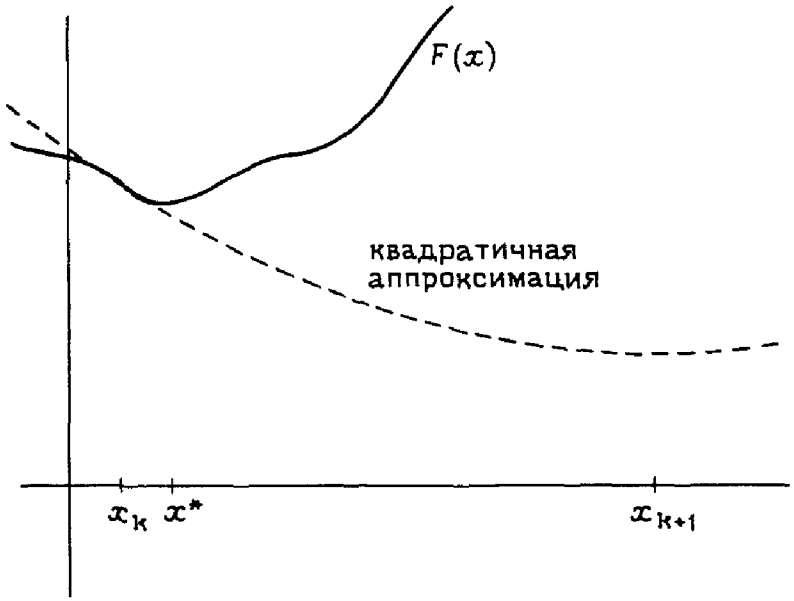


Рис. 9.2. Метод Ньютона не сходится.

$F_k''(x_k) > 0$. Если это не так, метод Ньютона может сходиться к максимуму функции $F(x)$, а не к минимуму. См. в качестве иллюстрации рис. 9.3.

(3) Главный недостаток метода Ньютона заключается в том, что в дополнение к значениям самой функции он требует вычисления ее производных. В сложных реальных задачах это может оказаться трудным или невозможным.

Хотя эти замечания и настораживают, они вовсе не означают, что метод Ньютона бесполезен или неприменим к реальным задачам.

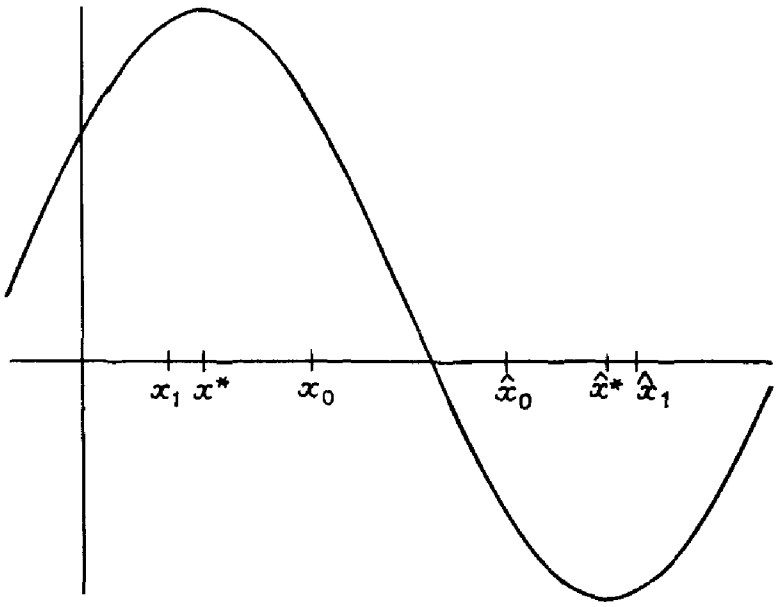


Рис. 9.3. Метод Ньютона не находит минимума.

Некоторые простые модификации превращают его в эффективный общий метод, гарантирующий продвижение к решению на каждой итерации. Наиболее широко используемые и эффективные методы оптимизации основаны именно на методе Ньютона.

Чтобы сделать метод Ньютона более надежным, его модифицируют в двух отношениях. Во-первых, квадратичная аппроксимация всегда должна иметь конечный минимум. Если это не так, для аппроксимации нужно использовать другую квадратичную функцию с разумным (конечным) значением минимума. Это только временная модификация метода; на следующей итерации мы попытаемся вновь использовать стандартный метод Ньютона. Во-вторых, значение функции должно улучшаться с каждой итерацией, т. е. новая точка x_{k+1} должна удовлетворять условию $F(x_{k+1}) < F(x_k)$. Это можно гарантировать выбором $x_{k+1} = x_k + \alpha p$, где $\alpha \in (0,1)$ получается путем приближенной минимизации функции $F(x_k + \alpha p)$ как функции от α. Эти идеи более подробно обсуждаются в параграфе об n-мерной оптимизации. Во многих задачах, в которых $F''(x^*) > 0$, такие модификации будут нужны только далеко от решения; вблизи решения x^* метод будет работать идеальным образом, как описано выше.

Требование вычисления производных можно ослабить за счет добавочных вычислений функции. Как описано в примере 2.8 из гл. 2, производную $F'(x_k)$ можно аппроксимировать разностной формулой

$$F'(x_k) \approx \frac{F(x_k + h) - F(x_k)}{h},$$

где $h \approx \sqrt{\varepsilon_{\text{маш}}}$. Когда число $|F'(x_k)|$ мало по сравнению с $|F(x_k)|$, ошибка оценки конечно-разностного приближения может быть велика. В этом случае вычитание $F(x_k + h) - F(x_k)$ может приводить к большой относительной ошибке. В алгоритме минимизации, поскольку $F'(x^*) = 0$, это будет иметь место при сходимости алгоритма. При малом значении $|F'(x_k)|$ следует использовать более точную центрально-разностную аппроксимацию

$$F'(x_k) \approx \frac{F(x_k + h) - F(x_k - h)}{2h},$$

которая имеет порядок точности $O(h^2)$. Как и в примере 2.8, величина h выбирается так, чтобы сбалансировать ошибку аппроксимации и ошибку округления:

$$h = \sqrt{\frac{3|F(x_k)|\varepsilon_{\text{маш}}}{|F'''(x_k)|}}.$$

Если функция $F(x)$ хорошо масштабирована, можно использовать более простую формулу $h = \sqrt[3]{\varepsilon_{\text{маш}}}$.

Величины конечно-разностных интервалов h были получены в пред-

положении, что функция $F(x)$ может быть вычислена с полной рабочей точностью. Если это не так, $\varepsilon_{\text{маш}}$ следует заменить относительной точностью значений функции. Например, если $F(x)$ можно вычислить с точностью 8 значащих цифр, то относительная точность будет $(1/2) \times 10^{-7}$ и $h \approx 2.2 \times 10^{-4}$. Хотя мы обсудили только аппроксимацию $F'(x_k)$, аналогичные аппроксимации можно получить для второй производной $F''(x_k)$, которая входит в формулу Ньютона.

Если использовать эти аппроксимации, оценки производных будут менее точными, чем значения функции. Например, значение первой производной будет иметь только половину точных значащих цифр по сравнению с $F(x_k)$. Вдали от решения это будет слабо влиять на поведение алгоритма, однако это может ограничить точность, с которой удастся вычислить решение.

Метод Ньютона был получен путем построения квадратичной аппроксимации функции $F(x)$ с использованием значений функции и ее производных в точке. Другой итеративный метод, основанный на квадратичных аппроксимациях, использующих только значения функции, можно получить следующим образом. Начинаем с трех произвольных вещественных чисел x_1, x_2 и x_3. На шаге с номером $k-2$ имеем x_{k-2}, x_{k-1} и x_k. Пусть x_{k+1} минимизирует квадратичную функцию, которая интерполирует $(x_i, F(x_i))$, $i = k-2, k-1, k$. Теперь отбросим старую точку x_{k-2} и продолжим итерации с x_{k-1}, x_k и x_{k+1}. Этот алгоритм называется *последовательной параболической интерполяцией*. Можно показать, что вблизи решения x^* скорость сходимости итераций примерно равна 1.324... при условии $F''(x^*) > 0$. Заметим, что такое же требование включает и метод Ньютона.

9.2.2. Поиск «золотого сечения»

Теперь мы обсудим более надежные, но и более медленные методы одномерной минимизации, для которых не требуется производных функции $F(x)$. Методы этого раздела не находят соответствия среди методов решения нелинейных уравнений. Предположим, что F – вещественная функция, заданная на $[0, 1]$. Предположим далее, что имеется единственное такое значение x^*, что $F(x^*)$ является минимумом $F(x)$ на $[0, 1]$ и что $F(x)$ строго убывает при $x \leqslant x^*$ и строго возрастает при $x^* \leqslant x$. Такая функция называется *унимодальной* и имеет график одного из трех типов, показанных на рис. 9.4. Ограничение интервалом $[0, 1]$ не существенно, поскольку можно рассматривать и другие интервалы, если сделать замену переменной; см. § 3 гл. 5. Важно только, чтобы было задано некоторое ограничение на решение.

Заметим, что унимодальная функция не обязана быть гладкой или даже непрерывной, однако она имеет очень полезное для минимизации свойство: для любых двух точек x_1, x_2, таких, что $x_1 < x_2 \leqslant x^*$, можно заключить, что $F(x_1) > F(x_2)$. Аналогично, если $x^* \leqslant x_1 < x_2$, то $F(x_1) < F(x_2)$. Обратно, если $x_1 < x_2$ и $F(x_1) > F(x_2)$, то $x_1 \leqslant x^* \leqslant 1$.

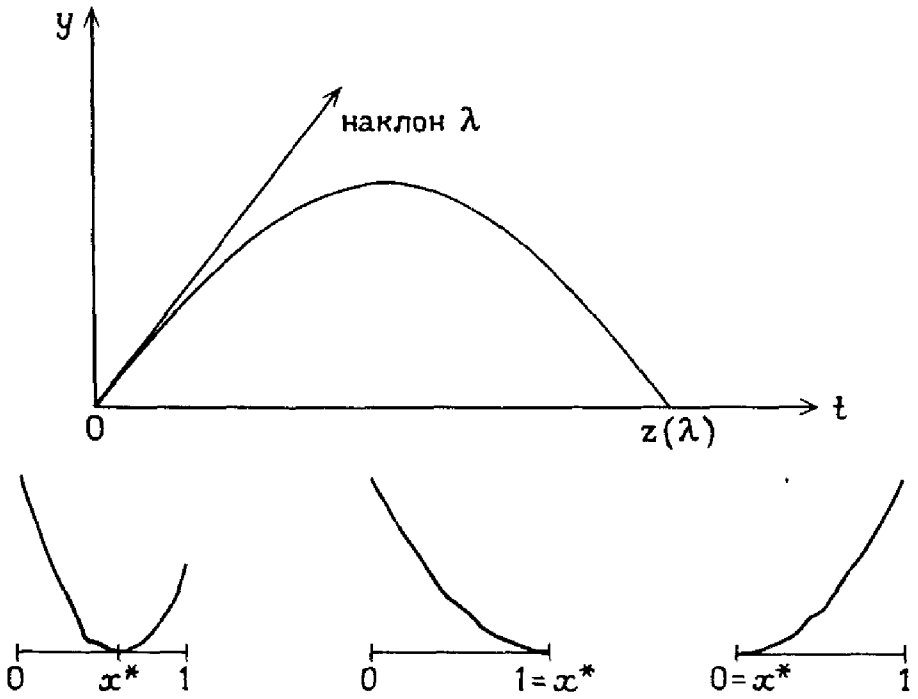


Рис. 9.4. Унимодальные функции.

а если $F(x_1) < F(x_2)$, то $0 \leqslant x^* \leqslant x_2$ (конечно, если $F(x_1) = F(x_2)$, мы имеем дополнительную информацию, что $x_1 \leqslant x^* \leqslant x_2$, но нам не придется этим пользоваться). Эти факты позволяют улучшить оценку точки, минимизирующей $F(x)$, используя только значения функции. Проблема в том, чтобы найти множество точек $x_1, x_2, \ldots, x_k$ и соответствующие значения функции $F(x_1), F(x_2), \ldots, F(x_k)$, а затем определить, что оптимальное значение F лежит в интервале $x_{i-1} \leqslant x^* \leqslant x_{i+1}$ для некоторого i. Такой интервал называется *интервалом неопределенности.*

Предположим, что нам разрешено вычислить функцию k раз, где $k > 1$ задано. Как использовать эти вычисления, чтобы локализовать x^* внутри наименьшего возможного интервала неопределенности?

Алгоритм выбора точек x_i $(i = 1, \ldots, k)$ называется *планом поиска.* Если мы знаем лишь то, что функция F унимодальна, какова оптимальная стратегия нахождения x^*? *Оптимальным* планом поиска для заданного числа вычислений функции называется такой план, который приводит к наименьшему интервалу неопределенности. Чтобы получить оптимальный план, будем предполагать, что функция F вычисляется последовательно, т. е. только в одной точке за один раз.

Начало теории выбора планов последовательного поиска положили работы Дж. Кифера, опубликованные в начале 50-х годов. Алгоритм оптимальной стратегии последовательно дает k точек испытаний, которые мы пронумеруем $x_k, x_{k-1}, \ldots, x_2, x_1$. Сначала дадим общее описание алгоритма, а затем приведем пример его применения. На первом шаге одновременно выбираются две точки x_k и x_{k-1}, причем

$x_{k-1} < x_k$. Если $F(x_{k-1}) > F(x_k)$, то интервал неопределенности уменьшился до $[x_{k-1}, 1]$ *и у нас уже есть значение* $F(x_k)$, *с помощью которого можно дальше уменьшать интервал неопределенности.* С другой стороны, если $F(x_{k-1}) < F(x_k)$, то интервал неопределенности уменьшился до $[0, x_k]$ и мы уже знаем $F(x_{k-1})$. Ключом к оптимальному выбору x_{k-1} и x_k является то, что точка испытания, которая окажется внутри уменьшенного интервала неопределенности, сама должна располагаться в оптимальной позиции для продолжения поиска.

Для того чтобы описать алгоритм более детально и уметь применять его на произвольном интервале, а не только на $[0, 1]$, полезно определить *координату r точки x относительно интервала* $[a, b]$ как число $r = (x - a)/(b - a)$. Тогда a относительно интервала $[a, b]$ имеет координату 0, а b–координату 1, в то время как средняя точка имеет координату $\frac{1}{2}$.

Пусть $f_0 = f_1 = 1$, $f_2 = 2$, $f_3 = 3$, $f_4 = 5$, $f_5 = 8$, ..., $f_k = f_{k-1} + f_{k-2}$. Значения f_i представляют собой знаменитые числа Фибоначчи. Оптимальная стратегия последовательного поиска минимума называется *методом Фибоначчи*, поскольку она тесно связана с этими числами. В оптимальной стратегии выбирают $x_{k-1} = f_{k-2}/f_k$ и $x_k = f_{k-1}/f_k$. Какой бы из интервалов, $[0, x_k]$ или $[x_{k-1}, 1]$, ни стал новым интервалом неопределенности, унаследованная точка будет иметь одну из следующих двух координат относительно нового интервала: f_{k-3}/f_{k-1} или f_{k-2}/f_{k-1}. Тогда точка x_{k-2} выбирается так, чтобы она имела другую из этих двух координат относительно нового интервала. Располагая значением $F(x_{k-2})$ и значением функции, унаследованным от первого интервала, можно снова уменьшить интервал неопределенности и сохранить в нем одно из этих значений функции.

На последнем шаге мы придем к некоторому интервалу неопределенности $[a, b]$, в котором средняя точка унаследована от предыдущего шага. Тогда в качестве x_1 выбирается точка с относительной координатой $\frac{1}{2} + \varepsilon$ (где ε–некоторое малое положительное число), и конечным интервалом неопределенности будет либо $\left[0, \frac{1}{2} + \varepsilon\right]$, либо $\left[\frac{1}{2}, 1\right]$ относительно $[a, b]$. Пример 9.2 демонстрирует применение этого метода.

Пример 9.2 Метод Фибоначчи.

Рассмотрим минимизацию функции $F(x) = (x - 0.65)^2$ на интервале $[0, 1]$ с помощью метода Фибоначчи, который использует $f_7 = 21$ в качестве начальной точки (см. рис. 9.5 и табл. 9.2). Используется значение $\varepsilon = 0.01/21$. На первой итерации функция вычисляется в точках $f_6/f_7 = 13/21$ и $f_5/f_7 = 8/21$; вычисления дают значения 0.000958 и 0.0724 соответственно. Поскольку функция унимодальна, минимум находится в интервале $[8/21, 1]$ и т. д.

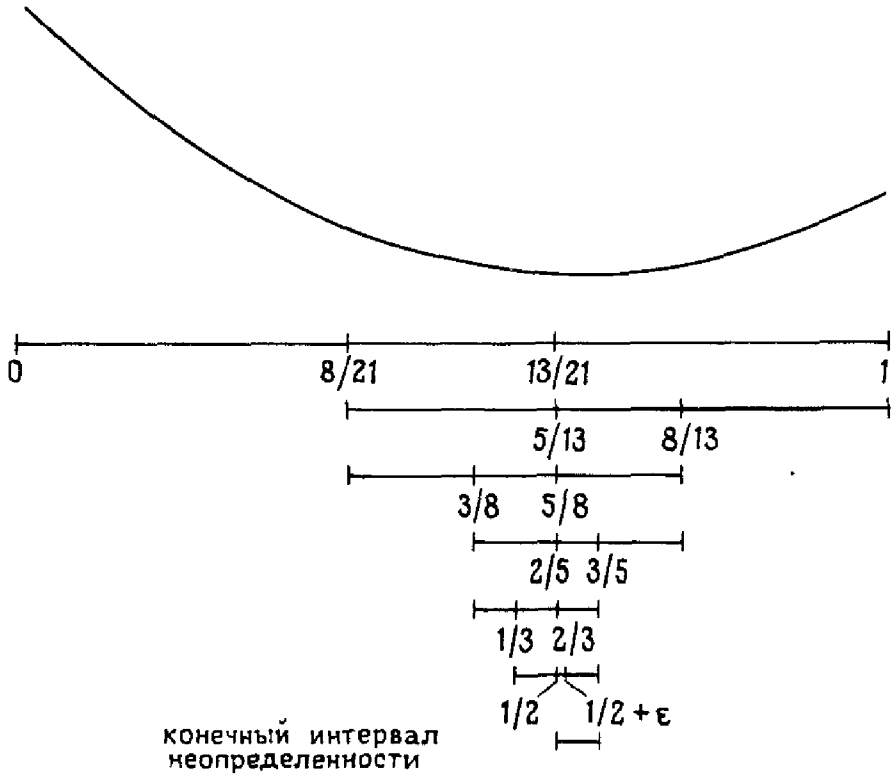


Рис. 9.5. Метод Фибоначчи.

В результате последней итерации интервал неопределенности сокращается до $[13/21, 14.01/21] \approx [0.62, 0.67]$. Средняя точка интервала 0.645 дает хорошее приближение к решению 0.65. □

Таблица 9.2

Итерация	Интервал	(x_{k-1}, x_k) относительные координаты	(x_{k-1}, x_k) абсолютные координаты	$(F(x_{k-1}), F(x_k))$
1	[0/21, 21/21]	(8/21, 13/21)	(8/21, 13/21)	(.0724, .000958)
2	[8/21, 21/21]	(5/13, 8/13)	(13/21, 16/21)	(.000958, .0125)
3	[8/21, 16/21]	(3/8, 5/8)	(11/21, 13/21)	(.0159, .000958)
4	[11/21, 16/21]	(2/5, 3/5)	(13/21, 14/21)	(.000958, .000278)
5	[13/21, 16/21]	(1/3, 2/3)	(14/21, 15/21)	(.000278, .000413)
6	[13/21, 15/21]	$(1/2, 1/2 + \epsilon)$	(14/21, 14.01/21)	(.000278, .000294)

У метода Фибоначчи есть один недостаток: общее число вычислений функции должно быть выбрано заранее. В большинстве задач мы предпочли бы продолжать итерации до тех пор, пока значение функции в точке x не будет получено с достаточной точностью. Близкий к поиску Фибоначчи метод, который называется *методом золотого сечения*, решает эту проблему. Чтобы получить его, рассмотрим метод Фибоначчи более подробно.

На первом шаге длина интервала неопределенности уменьшается

с 1 до f_{k-1}/f_k. На последующих шагах длина интервала умножается на числа

$$\frac{f_{k-2}}{f_{k-1}}, \quad \frac{f_{k-3}}{f_{k-2}}, \quad \ldots, \quad \frac{f_2}{f_3}, \quad \frac{f_1}{f_2}(1 + 2\varepsilon).$$

Итак, длина конечного интервала неопределенности составляет $(1 + 2\varepsilon)/f_k$. Пренебрегая ε, можно показать, что $1/f_k \approx r^k$ при $k \to \infty$, где

$$r = \frac{\sqrt{5} - 1}{2} \approx 0.6180.$$

Таким образом, для больших k асимптотически на каждом шаге метода Фибоначчи длина интервала неопределенности умножается на 0.6180. Это аналогично уменьшению длины интервала неопределенности в два раза в методе бисекции поиска нуля функции (см. раздел 2.1 гл. 7).

Для больших значений k координаты точек x_{k-1} и x_k относительно начального интервала близки к $1 - r \approx 0.3820$ и $r \approx 0.6180$ соответственно. Если начать план поиска с этих значений, а не с тех, которые основаны на числах Фибоначчи, план будет близок к оптимальной стратегии. Чтобы увидеть, как продолжать дальше, предположим, например, что $F(0.3820) < F(0.6180)$. Известно, что x^* находится в интервале $[0, 0.6180]$. Следовательно, нужно вычислить F в точках $(1 - r) * 0.6180 \approx 0.3820 * 0.6180$ и $r * 0.6180 \approx 0.6180 * 0.6180$, т. е. в точках $1 - r$ и r относительно уменьшенного интервала. Но поскольку $0.6180 * 0.6180 \approx 0.3820 \approx x_{k-1}$, значение F здесь уже известно. Таким образом, снова на каждой итерации после первой требуется только одно вычисление F и на каждом шаге длина интервала неопределенности умножается на 0.6180. В отличие от метода Фибоначчи здесь не нужно фиксировать число k до начала поиска. По этой причине данный алгоритм более практичен. Пример 9.3 демонстрирует применение метода золотого сечения.

Пример 9.3. Метод золотого сечения.

Снова рассмотрим минимизацию функции $F(x) = (x - 0.65)^2$ на интервале $[0, 1]$, но теперь используем метод золотого сечения вместо метода Фибоначчи (см. табл. 9.3). На первой итерации функция вычисляется в точках $r \approx 0.6180$ и $1 - r \approx 0.3820$, что дает значения 0.001022 и 0.071842 соответственно. Поскольку функция унимодальна, минимум должен лежать в интервале $[a, b] = [0.3820, 1]$. На следующей итерации функция вычисляется в точках $y = (b - a)r + a \approx 0.7639$ и $x = (b - a) \times (1 - r) + a \approx 0.6180$ (это абсолютные координаты точек с относительными координатами r и $1 - r$ относительно нового интервала) и т. д. После заключительной итерации интервал неопределенности сокращается до $[0.6180, 0.7082]$. Среднюю точку этого интервала 0.6631 можно взять в качестве аппроксимации решения 0.65. □

Таблица 9.3

Итерация	Интервал $[a, b]$	(x, y) относительные координаты	(x, y) абсолютные координаты	$(F(x), F(y))$
1	[.0000, 1.0000]	(.3820, .6180)	(.3820, .6180)	(.071842, .001022)
2	[.3820, 1.0000]	(.3820, .6180)	(.6180, .7639)	(.001022, .012981)
3	[.3820, .7639]	(.3820, .6180)	(.5279, .6180)	(.014917, .001022)
4	[.5278, .7639]	(.3820, .6180)	(.6180, .6738)	(.001022, .000565)
5	[.6180, .7639]	(.3820, .6180)	(.6738, .7082)	(.000565, .003388)
6	[.6180, .7082]	(.3820, .6180)	(.6525, .6738)	(.000006, .000565)

Этот метод поиска x^* называется методом золотого сечения из-за истории числа $r \approx 0.6180$. Число $1/r = \varphi = (1 + \sqrt{5})/2 = 1.6180$ называется *золотым сечением*. В гл. 7 мы заметили, что порядок сходимости метода секущих равен φ. Золотое сечение φ является фундаментальным числом, которое появляется во многих областях. Интересное обсуждение числа φ можно найти в книге [Gardner, 1961].

Метод золотого сечения аналогичен методу бисекции для нахождения вещественного нуля функции (см. гл. 7) в том отношении, что он гарантированно сходится в наихудшем возможном случае, а цена этой надежности – медленная сходимость, которая является всего лишь линейной. Поиск золотого сечения не извлекает никакой выгоды из возможной гладкости функции F.

9.3. Подпрограмма FMIN

Брент [Brent, 1973] разработал алгоритм поиска минимума, комбинирующий метод золотого сечения и последовательную параболическую интерполяцию. Этот алгоритм полностью аналогичен алгоритму нахождения нуля FZERO, основанному на комбинации бисекции и обратной квадратической интерполяции. В данном параграфе описывается фортрановская версия алгоритма Брента. Это подпрограмма-функция

FUNCTION FMIN (A, B, F, TOL).

Здесь [A, B] – исходный интервал, на котором задана функция F. Вычисляемое в FMIN значение является оценкой точки x^*, минимизирующей $F(x)$. TOL – входной параметр, который, грубо говоря, дает желаемую длину конечного интервала неопределенности. Итак, если положить TOL равным 10^{-3} и если результат на выходе FMIN имеет величину порядка 1, то ответ имеет примерно три верные значащие цифры. Если положить TOL равным 10^{-6} и ответ будет, скажем, 0.00000385746, то этот результат, вероятно, вообще не имеет верных значащих цифр, но если интерпретировать результат такого порядка малости, как нуль, то никакие значащие цифры не нужны. Подпрограмма FMIN применяет

метод золотого сечения, переходя, когда это возможно, к последовательной параболической интерполяции.

Подпрограммы FZERO и FMIN имеют похожие параметры: интервал поиска, функцию, которую нужно вычислять, и допуск. Обе подпрограммы пытаются уменьшать длину интервала до тех пор, пока она не станет меньше, чем допуск. Однако между двумя процедурами есть важное различие, которое влияет на выбор допуска. Рассмотрим сначала задачу решения нелинейного уравнения. Если $F'(x^*) = 0$ и $F''(x^*) \neq 0$, то для малого ε

$$F(x^* + \varepsilon) = F(x^*) + \varepsilon F'(x^*) + \varepsilon^2 \frac{F''(x^*)}{2} + \ldots \approx c\varepsilon,$$

где $c = F'(x^*)$. Итак, малые изменения в x^* вызывают пропорционально малые изменения в $F(x^*)$. В качестве допуска для FZERO разумно выбрать значение, грубо говоря, порядка ошибок в значениях функции. Часто допуск может быть столь же малым, как уровень машинных округлений $\varepsilon_{\text{маш}}$.

Вернемся к задаче минимизации. Если мы ищем минимум, в котором $F'(x^*) = 0$ и $F''(x^*) = 0$, то член с первой производной в разложении Тейлора исчезает. Для малого ε

$$F(x^* + \varepsilon) = F(x^*) + \varepsilon^2 \frac{F''(x^*)}{2} + \ldots \approx F(x^*) + c\varepsilon^2,$$

где $c = F''(x^*)/2$. Изменение порядка ε в x^* теперь вызывает изменение порядка ε^2 в $F(x^*)$. Поэтому неразумно выбирать в качестве допуска для FMIN значение, меньшее, чем квадратный корень из ошибки в значениях функции. Другими словами, простые нули функции часто могут быть найдены примерно с *полной* машинной точностью, но значения, минимизирующие функцию, могут быть найдены только примерно с *половинной* точностью. Если TOL – входной допуск в FZERO или FMIN, а $\varepsilon_{\text{маш}}$ – машинная точность, то FZERO никогда не вычисляет функцию в точках, расстояние между которыми меньше, чем

$$2 * \text{TOL} * (\text{ABS}(\text{X}) + 1.0),$$

тогда как FMIN никогда не вычисляет функцию в точках, расстояние между которыми меньше, чем

$$\text{SQRT}(\varepsilon_{\text{маш}}) * \text{ABS}(\text{X}) + \text{TOL}/3.$$

Таким образом, малые значения TOL с большей вероятностью будут проигнорированы в FMIN, чем в FZERO.

На практике унимодальные функции встречаются сравнительно редко, и может быть трудно проверить, что данная функция является унимодальной. В сомнительных случаях возможно несколько подходов. Поскольку многие функции унимодальны в окрестности локального минимума, вероятно, применение FMIN будет успешным, если началь-

ный интервал [A, B] представляет собой хорошую оценку решения. Но даже если начальный интервал велик, во многих случаях FMIN сможет найти локальный минимум функции. Вторая возможность – использовать процедуру UNCMIN, описанную в § 5; она предназначается для n-мерной минимизации, но может быть применена и при $n = 1$, если функция достаточно гладкая.

Если функция не дифференцируема и оценки минимизирующего значения нет, то у нас мало теоретических основ для построения алгоритма (поскольку значение функции в одной точке не дает никакой информации о значениях функции в других точках). В этом случае единственная возможность – вычислить функцию в некоторой последовательности точек и выбрать наименьшее значение. Например, программа могла бы выбрать некоторое $a > 0$, а затем вычислять $F(0)$, $F(a)$, $F(2a)$, $F(2^2a)$, $F(2^3a)$ и т. д. до тех пор, пока значения продолжают убывать, принимая в качестве приближения к x^* точку $2^k a$ с наименьшим значением $F(x)$. Или, если $F(a) > F(0)$, программа вычисляет $F(2^{-1}a)$, $F(2^{-2}a)$, ... вплоть до значения $F(2^{-r}a)$, аргумент которого принимается в качестве приближения к x^*. Читатель может придумать различные способы уточнения x^*, полученного с помощью этих грубых методов. Они не эффективны; всякий раз, когда это возможно, следует использовать более тонкие алгоритмы, обсуждаемые в этой главе.

Для того чтобы продемонстрировать использование подпрограммы FMIN, мы выбрали ту же функцию, что и для подпрограммы FZERO, а именно $F(x) = x^3 - 2x - 5$. Поскольку $F'(x)$ – квадратичная функция, экстремумы можно найти аналитически. Заметим, что используемое здесь значение TOL является квадратным корнем из значения, использованного в FZERO.

Выходное значение XSTAR равно 0.8168136.

```
C Пример использования FMIN
C
      REAL A, B, XSTAR, TOL, FMIN
      EXTERNAL F
      A = 0.1
      B = 0.9
      TOL = 1.0E - 5
      XSTAR = FMIN (A, B, F, TOL)
      WRITE (*,*)'XSTAR = ',XSTAR
      STOP
      END
C
      REAL FUNCTION F(X)
      REAL X
      F = X*(X*X - 2.0) - 5.0
      RETURN
      END
```

9.4. Оптимизация в многомерном случае

Локальная минимизация функции n переменных имеет такое большое значение, что алгоритмы для нее разрабатываются уже свыше 140 лет. Многие из них основаны на следующей метафоре: представьте себе график функции как ландшафт с холмами и долинами. Дно долин представляет собой минимумы функции. Если мы не находимся в минимуме, то естественная идея заключается в том, чтобы спускаться с холма, пока не дойдем до минимума.

Эту идею реализует давно известный метод, который называется *методом наискорейшего спуска*. Он был предложен Коши в 1845 г. Обозначим вектор $(x_1, \ldots, x_n)^T$ через x и предположим, что функция $F(x)$ имеет непрерывные частные производные нескольких порядков. Пусть $\nabla F = \nabla F(x)$ означает градиент функции F в точке x, т. е. вектор, i-я компонента которого есть $(\nabla F)_i(x) = \partial F/\partial x_i$. Для фиксированного x и меняющегося α совокупность векторов $x - \alpha\nabla F$ представляет собой луч, исходящий из точки x. Известно, что $-\nabla F(x)$ – это направление «с холма» для функции $F(x)$ в точке x в том смысле, что для достаточно малого положительного α значения функции будут убывать, если мы продвинемся на малое расстояние вдоль направления $-\nabla F(x)$, т. е. гарантировано, что $F(x - \alpha\nabla F) < F(x)$ для малого $\alpha > 0$.

Этот подход может показаться эвристическим, поскольку он предполагает, что любое направление «вниз с холма» будет удовлетворительным. Однако $-\nabla F(x)$ является «наилучшим» направлением спуска, так как значения функции вдоль луча $x - \alpha\nabla F$ будут убывать быстрее, чем вдоль любого другого луча, начинающегося в точке x. Направление $-\nabla F(x)$ также является «наилучшим» и в несколько другом смысле. Когда мы строили метод Ньютона в разд. 2.1, мы аппроксимировали целевую функцию тремя членами ряда Тейлора, чтобы получить шаг. Можно показать, что $-\nabla F(x)$ – это шаг, полученный при минимизации первых двух членов ряда Тейлора. По этим причинам $-\nabla F(x)$ называется направлением наискорейшего спуска для F в точке x.

Коши предложил искать на полупрямой, определенной как $x - \alpha\nabla F$ $(0 < \alpha < \infty)$, значение, минимизирующее F. Это одномерная минимизация типа той, что обсуждалась в § 2. Найдя этот минимум, начинают поиск вдоль полупрямой наискорейшего спуска, исходящей из новой точки x. При некоторых слабых предположениях этот метод будет сходиться к локальному минимуму функции F.

Теоретический анализ предсказал, что в некоторых случаях метод Коши будет сходиться крайне медленно, а опыт подтвердил это во многих практических случаях даже для таких скромных значений n, как 2, 3 и 4. Пример 9.4 демонстрирует плохое поведение метода наискорейшего спуска.

Пример 9.4. Метод наискорейшего спуска.

Воспользуемся методом наискорейшего спуска для решения следующей задачи:

$$\text{минимизировать } F(x_1, x_2, x_3) = x_1^2 + 10x_2^2 + 100x_3^2,$$

исходя из начальной точки (1,1,1). Решением будет $x^* = (0,0,0)^T$. Градиент этой функции имеет вид

$$\nabla F(x) = (2x_1, 20x_2, 200x_3)^T.$$

На каждой итерации новая точка определяется путем решения одномерной задачи

$$\min_{\alpha} F(x - \alpha \nabla F(x)) = (x_1 - 2\alpha x_1)^2 + 10(x_2 - 20\alpha x_2)^2 + 100(x_3 - 200\alpha x_3)^2.$$

Это квадратичная функция от α, которую можно минимизировать, если взять производную и положить ее равной нулю, что дает

$$\alpha = \frac{x_1^2 + 10^2 x_2^2 + 10^4 x_3^2}{2(x_1^2 + 10^3 x_2^2 + 10^6 x_3^2)}$$

(в общем случае найти явную формулу для α не удается). Первые несколько итераций алгоритма показаны в табл. 9.4.

Таблица 9.4

	x	$F(x)$
0	(1.0000, 1.0000, 1.0000)	111.0000
1	(0.9899, 0.8991, −0.0091)	9.0718
2	(0.8981, 0.0661, 0.0751)	1.4148
3	(0.8890, 0.0593, −0.0016)	0.8258
4	(0.7368, −0.0422, 0.0255)	0.6260
5	(0.7287, −0.0375, 0.0027)	0.5458
6	(0.6706, −0.0076, 0.0190)	0.4863

Алгоритм сходится медленно. После 180 итераций получаются приемлемые результаты: $x = (1.07 \times 10^{-4}, 0, 3.01 \times 10^{-6})^T$ и $F(x) = 1.24 \times 10^{-9}$. □

В качестве еще одной иллюстрации на рис. 9.6 приведен график итераций наискорейшего спуска для двумерной задачи. Продвижение к минимуму в центре происходит очень медленно; маршрут постепенно спускается в долину с колебаниями в направлениях локальных градиентов. Здесь явно возникает необходимость ускорения сходимости.

Значительная трудность здесь состоит в том, что локальный градиент даже примерно не указывает направление к решению x^*. Дело можно поправить, рассматривая n-мерный аналог метода Ньютона. Как и прежде, он основан на квадратичной аппроксимации функции $F(x)$, полученной из разложения Тейлора вблизи точки x (его можно также

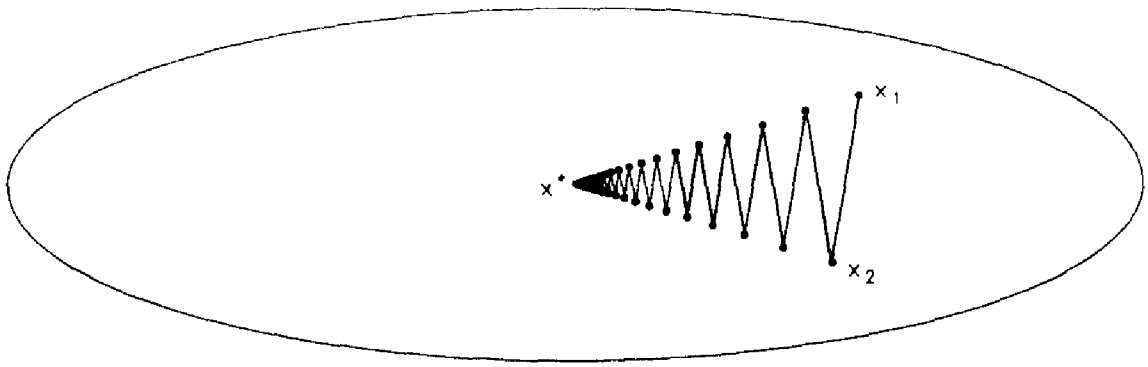


Рис. 9.6. Метод наискорейшего спуска.

получить, применяя методы гл. 7 к системе нелинейных уравнений $\nabla F(x) = 0$).

Будем записывать ряды Тейлора в матрично-векторной форме. В двумерном случае приближение второго порядка с помощью ряда Тейлора имеет вид

$$F(x_1 + p_1, x_2 + p_2) \approx F(x_1, x_2) + p_1 \frac{\partial F(x_1, x_2)}{\partial x_1} + p_2 \frac{\partial F(x_1, x_2)}{\partial x_2} +$$

$$+ \frac{1}{2}\left(p_1^2 \frac{\partial^2 F(x_1, x_2)}{\partial x_1^2} + p_1 p_2 \frac{\partial^2 F(x_1, x_2)}{\partial x_1 \partial x_2} + p_2 p_1 \frac{\partial^2 F(x_1, x_2)}{\partial x_2 \partial x_1} + p_2^2 \frac{\partial^2 F(x_1, x_2)}{\partial^2 x_2^2} \right).$$

Пусть $\nabla^2 F$ – постоянная матрица вторых частных производных функции F в точке x^*, так называемый *гессиан* (*матрица Гессе*):

$$(\nabla^2 F)_{i,j} = \frac{\partial^2 F}{\partial x_i \partial x_j}.$$

Ряд Тейлора, записанный выше, может показаться сложным отчасти из-за количества членов. Однако, если использовать обозначения для градиента и матрицы Гессе, двумерное разложение Тейлора очень напоминает одномерное. Сравнивая члены, мы увидим, что

$$F(x + p) \approx F(x_1, x_2) + (p_1 p_2) \begin{pmatrix} \dfrac{\partial F(x_1, x_2)}{\partial x_1} \\ \dfrac{\partial F(x_1, x_2)}{\partial x_2} \end{pmatrix} +$$

$$+ \frac{1}{2}(p_1 p_2) \begin{pmatrix} \dfrac{\partial^2 F(x_1, x_2)}{\partial x_1^2} & \dfrac{\partial^2 F(x_1, x_2)}{\partial x_1 \partial x_2} \\ \dfrac{\partial^2 F(x_1, x_2)}{\partial x_2 \partial x_1} & \dfrac{\partial^2 F(x_1, x_2)}{\partial x_2^2} \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} =$$

$$= F(x) + p^T \nabla F(x) + \frac{1}{2} p^T \nabla^2 F(x) p.$$

Несмотря на то что этот результат получен в двумерном случае, последнее выражение сохраняется также для n измерений (похожее разложение в ряд Тейлора было получено в § 5 гл. 7).

Для того чтобы воспользоваться этим разложением для метода Ньютона, запишем

$$F(x+p) \approx F(x) + p^T \nabla F(x) + \frac{1}{2} p^T \nabla^2 F(x) p = F(x) + Q(p),$$

где члены более высокого порядка рассматриваются как квадратичная функция от p. Чтобы получить шаг p, минимизируем квадратичную функцию, строя ее градиент по p

$$\nabla_p Q(p) = \nabla_p (p^T \nabla F(x) + \frac{1}{2} p^T \nabla^2 F(x) p) = \nabla F(x) + \nabla^2 F(x) p.$$

Приравнивая его к нулю, получаем

$$\nabla^2 F(x) p = -\nabla F(x).$$

Это система n линейных уравнений относительно n неизвестных $p = (p_1, p_2, \ldots, p_n)^T$, которые называются *уравнениями Ньютона*. Итак, $x_{k+1} = x_k + p = x_k - \nabla^2 F(x_k)^{-1} \nabla F(x_k)$ (в соответствии с обсуждением в гл. 3, шаг p вычисляется как решение системы уравнений без формирования обратной матрицы в явном виде).

Метод Ньютона в n-мерном случае обладает тем же свойством быстрой сходимости, что и в одномерном случае, а именно он сходится квадратично в окрестности решения

$$\| x_{k+1} - x^* \|_2 \leqslant \beta \| x_k - x^* \|_2^2,$$

где β – некоторая неотрицательная константа, зависящая от $F(x)$. В качестве иллюстрации см. пример 9.5. Это не та задача минимизации, которая рассматривалась в примере 9.4. Метод Ньютона решил бы ее за одну итерацию, поскольку это квадратичная функция (почему?).

Пример 9.5. Метод Ньютона.

Рассмотрим следующую задачу:

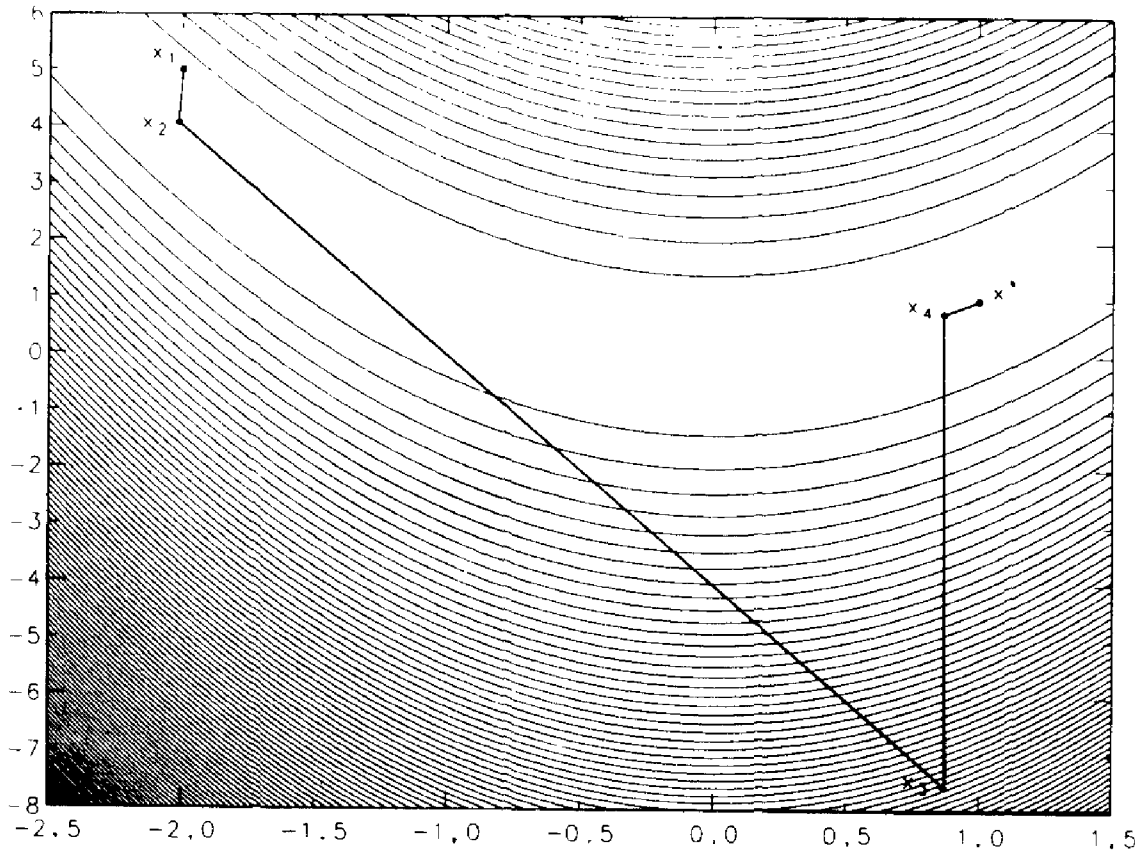
$$\text{минимизировать } F(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

с начальными значениями $x = (-2.0,\ 5.0)^T$. Градиент и гессиан этой функции имеют вид

$$\nabla F(x) = \begin{pmatrix} -400x_1(x_2 - x_1^2) + 2(x_1 - 1) \\ 200(x_2 - x_1^2) \end{pmatrix},$$

$$\nabla^2 F(x) = \begin{pmatrix} 1200x_1^2 - 400x_2 + 2 & -400x_1 \\ -400x_1 & 200 \end{pmatrix}.$$

Линии уровня этой функции показаны на рис. 9.7. На нем также



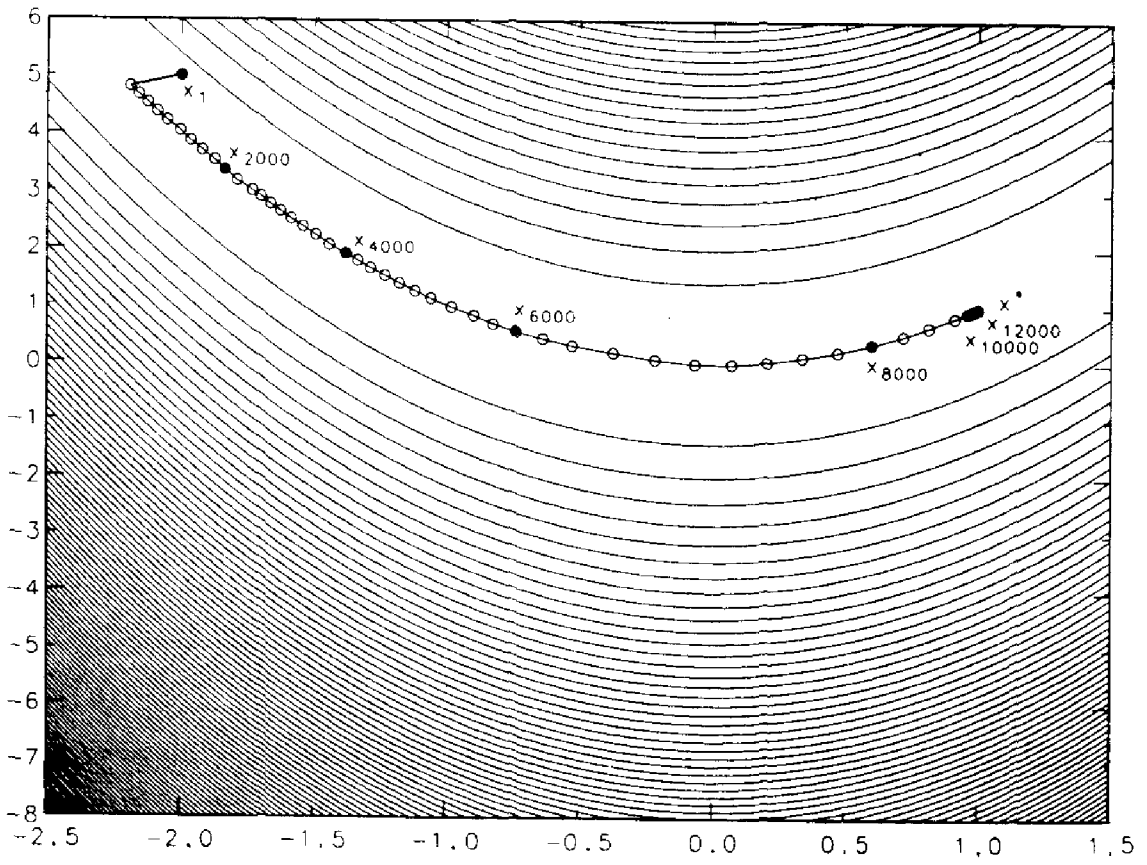


Рис. 9.7. Линии уровня в примере 9.5.

Таблица 9.5

	x	$F(x)$
0	(−2.000000, 5.000000)	$1.09 \times 10^{+2}$
1	(−2.015075, 4.060302)	$9.09 \times 10^{+0}$
2	(0.868913, −7.562379)	$6.92 \times 10^{+0}$
3	(0.868992, 0.755147)	1.70×10^{-2}
4	(0.999999, 0.982837)	2.93×10^{-2}
5	(0.999999, 0.999999)	1.58×10^{-14}
6	(1.000000, 0.999999)	2.51×10^{-26}
7	(1.000000, 1.000000)	0.0

показаны результаты применения алгоритмов Ньютона и наискорейшего спуска. Результаты применения метода Ньютона к этой задаче приведены в табл. 9.5.

Как и в одномерном случае, у метода Ньютона есть недостатки. Например, он может не сходиться. Чтобы поправить дело, метод обычно включают в состав более общего алгоритма. Приведенный ниже алгоритм спуска является совершенно общим; в нем можно использовать и принцип наискорейшего спуска, чтобы получить сходящийся метод. Основное назначение составного алгоритма в том, чтобы обеспечить сходимость; свойства направления спуска, вычисляемого на шаге 2, определяют, насколько быстро сходится весь алгоритм.

Алгоритм спуска для нелинейной минимизации

Задано начальное приближение x_0; полагаем $k \leftarrow 0$.

1. Вычислить $F_k = F(x_k)$, $\nabla F_k = \nabla F(x_k)$, значения функции и градиента. Проверить условие выхода. Если условие выполнено — останов.
2. Найти направление спуска p, т.е. такое направление p, что $F(x_k + \varepsilon p) < F_k$ для малых ε. Это эквивалентно требованию $p^T \nabla F_k < 0$.
3. Линейный поиск: найти такое $\alpha > 0$, что $F(x_k + \alpha p) < F_k$. Положить $x_{k+1} \leftarrow x_k + \alpha p$, $k \leftarrow k + 1$. Перейти к шагу 1.

Используя на каждом шаге подходящие методы, можно гарантировать сходимость такого метода минимизации к локальному минимуму для большого класса задач. Шаг 3 представляет собой задачу приближенной одномерной минимизации, которая решается с использованием методов, подобных описанным ранее. Этот шаг необходим, поскольку даже тогда, когда направление метода Ньютона является удовлетворительным направлением спуска, оно может не иметь правильной величины или масштаба, если точка x_k далека от x^*.

*9.4.1. Модификации метода Ньютона

В идеале при использовании метода Ньютона на втором шаге описанного выше алгоритм спуска $p = -\nabla^2 F^{-1}\nabla F$. Однако это может не быть направлением спуска. В качестве простейшей иллюстрации рассмотрим одномерный пример

$$F(x) = 2x^3 - 5x^2 + 4x$$

с начальным значением $x_0 = 0$. Тогда $F(x_0) = 0$. Вычислим шаг метода Ньютона, используя

$$p = -\nabla^2 F^{-1}\nabla F = -(12x_0 - 10)^{-1}(6x_0^2 - 10x_0 + 4) = 0.4.$$

Тогда $F(x_0 + p) = F(0.4) = 0.9280 > F(x_0)$. В действительности непосредственно проверяется, что $F(x_0 + \alpha p) > F(x_0)$ для всех $\alpha \in [0, 1]$. Итак, p не является направлением спуска.

Иногда можно гарантировать, что метод Ньютона будет давать направление спуска. Предположим, что матрица $\nabla^2 F^{-1}$, обратная матрице Гессе, положительно определена, т. е. удовлетворяет условию $z^T \nabla^2 F^{-1} z > 0$ для всех $z \neq 0$. В этом случае направление метода Ньютона гарантированно будет направлением спуска, поскольку, используя разложение Тейлора (где $\nabla F = \nabla F(x)$ – градиент функции $F(x)$), получим

$$\begin{aligned} F(x + \varepsilon p) &= F(x) + \nabla F^T(\varepsilon p) + O(\varepsilon^2) = \\ &= F(x) + \varepsilon \nabla F^T(-\nabla^2 F^{-1}\nabla F) + O(\varepsilon^2) = \\ &= F(x) - \varepsilon \nabla F^T \nabla^2 F^{-1}\nabla F + O(\varepsilon^2). \end{aligned}$$

Поскольку матрица $\nabla^2 F^{-1}$ положительно определена, $\nabla F^T \nabla^2 F^{-1}\nabla F > 0$, пока $\nabla F \neq 0$. Итак, если ε мало и $\nabla F \neq 0$, то $F(x + \varepsilon p) < F(x)$, т. е. p представляет собой направление спуска. Если же $\nabla F = 0$, то x – критическая точка и нужно проверить другие условия, включающие вторые производные, чтобы определить, минимизирует ли x функцию F.

Поскольку лишь для немногих функций матрица $\nabla^2 F^{-1}(x)$ положительно определена для всех x, требуется некоторая стратегия, которая давала бы гарантированное направление спуска. Удачная идея, предложенная в книге [Gill, Murray, 1974], заключалась в том, чтобы заменить гессиан «близкой» положительно определенной матрицей $\bar{\nabla}^2 F$, если это необходимо, и найти шаг, решив систему $\bar{\nabla}^2 F p = -\nabla F$. Это модифицированное направление будет направлением спуска, если $\nabla F \neq 0$. Модифицированный гессиан используется только на одной итерации. На следующей итерации алгоритм снова попытается использовать истинный гессиан, если он положительно определен.

Однако даже с этими модификациями метод Ньютона все еще имеет ряд недостатков. Во-первых, нужно вычислять матрицу вторых производных. Как и в одномерном случае, $\nabla^2 F$ можно аппроксимировать, используя конечные разности функции или значения градиента. Однако для n измерений это потребует n дополнительных вычислений градиента

или n^2 дополнительных вычислений функции, что чрезмерно дорого, если исключить функции, которые особенно легко вычислять.

Другой недостаток метода Ньютона заключается в том, что вычисление шага p требует решения системы n линейных уравнений с затратой приблизительно $n^3/6$ арифметических операций. При больших n это гигантский объем работы на одну итерацию.

Метод наискорейшего спуска не страдает этими недостатками, но он приводит к большим потерям в скорости сходимости. Для того чтобы получить альтернативные эффективные и практичные методы, можно аппроксимировать гессиан в ходе минимизации функции. Эти методы основаны на аппроксимации гессиана секущими и являются обобщением описанного в разделе 2.3 гл. 7 метода секущих. Если $B_k \approx \nabla^2 F_k = F(x)$, то шаг на k-й итерации будет определяться из системы

$$B_k p = -\nabla F_k.$$

Таким образом, мы получаем вариант метода Ньютона, в котором используется приближенный, а не точный гессиан. Шаг p можно использовать в комбинированном методе спуска, описанном выше. После того как линейный поиск дает новое значение $x_{k+1} = x_k + \alpha p$, приближенный гессиан B_k обновляется, что приводит к новой аппроксимации B_{k+1}; для нее используются значения x_{k+1} и $F(x_{k+1})$.

Если бы функция $F(x)$ была квадратичной, она удовлетворяла бы равенству

$$(\nabla^2 F)(x_{k+1} - x_k) = \nabla F_{k+1} - \nabla F_k.$$

Новая аппроксимация гессиана выбирается так, чтобы

$$B_{k+1}(x_{k+1} - x_k) = \nabla F_{k+1} - \nabla F_k.$$

В одномерном случае это однозначно определяет B_{k+1}. При более высоких размерностях для того, чтобы определить B_{k+1}, необходимы дополнительные условия. Обычно B_k обновляется с помощью простой матрицы малого ранга так, как в разделе 9.1 гл. 3.

Каковы преимущества такого подхода?

(1) В окрестности решения метод сходится быстро, со сверхлинейной (т.е. более быстрой, чем линейная) скорость.

(2) Используются только значения градиента, а не вторых производных.

(3) Матрицу B_k можно выбрать так, чтобы она была положительно определенной, т.е. чтобы всегда получалось направление спуска.

(4) Поскольку B_k модифицируется только матрицей малого ранга, объем работы для выполнения одной итерации (исключая стоимость вычисления функции) можно снизить до $O(n^2)$ операций (см. раздел 9.1 гл. 3).

Методы такого типа называются *квазиньютоновскими*. Один из первых таких алгоритмов был предложен Дэвидоном в 1959 г. Со времени появления статьи Дэвидона было разработано много усовер-

шенствований квазиньютоновских методов. Отсылаем читателя к обзору [Dennis, Moré, 1974], где обсуждается также теория сходимости этих методов.

9.5. Подпрограмма UNCMIN

Подпрограмма UNCMIN разработана для безусловной минимизации вещественной функции n переменных $F(x)$. Ее можно использовать и при $n = 1$, но в этом случае она, вероятно, окажется менее эффективной, чем подпрограмма FMIN. Она основана на комбинированном методе спуска, описанном выше, и реализует квазиньютоновский алгоритм.

Как и для подпрограммы FMIN, пользователь должен задать подпрограмму вычисления функции. Значения градиента, требующиеся для UNCMIN, будут вычисляться с помощью конечных разностей. Вдобавок должно быть обеспечено рабочее пространство для хранения приближенной матрицы Гессе и некоторых вспомогательных векторов.

Как и в случае с FMIN, если мы ищем минимум, в котором $\nabla F(x) = 0$, то изменение порядка ε в x вызывает изменение порядка ε^2 в $F(x)$. Поэтому неразумно выбирать в качестве допуска для UNCMIN значение, меньшее, чем квадратный корень из ошибки в значениях функции.

Подпрограмма UNCMIN найдет локальный минимум функции $F(x)$, если функция ограничена снизу. Невозможно гарантировать, что будет найден глобальный минимум. Если функция $F(x)$ не ограничена снизу, но все же имеет локальные минимумы, подпрограмма UNCMIN может оказаться эффективной при достаточно хорошем начальном приближении.

В демонстрационной задаче ниже мы минимизируем функцию $n = 10$ переменных, известную как обобщенная функция Розенброка. Функция $F(x)$ определяется формулой

$$F(x_1, \ldots, x_n) = 1 + \sum_{i=2}^{10} [\, 100(x_i - x_{i-1}^2)^2 + (1 - x_{i-1})^2 \,].$$

Начальная оценка минимизирующего значения равна

$$\left(\frac{1}{11}, \frac{2}{11}, \frac{3}{11}, \ldots, \frac{10}{11}\right)^T,$$

оптимальным решением является

$$x^* = (1, 1, \ldots, 1)^T,$$

а оптимальным значением функции $F(x^*) = 1$.

```
C Вызывающая программа, минимизирующая функцию, представлен-
C ную процедурой CALCF
C
```

```
      PARAMETER (N = 10, LWORK = N*(N + 10))
      REAL XO (N), X(N), F, WORK((LWORK)
      EXTERNAL CALCF
C
C Задание начальной оценки решения
C
      DO  10 I = 1,N
        XO(I) = 1/FLOAT(N + 1)
   10 CONTINUE
C
C Минимизация функции

C
      CALL UNCMIN (N, XO, CALCF, X; F, IERROR, WORK, LWORK)
C
C Печать результатов
C
      IF (IERROR .NE. 0) WRITE (*,*) 'Код ошибки = ', IERROR
      WRITE (*,*) 'F(X*) = ', F
      WRITE (*,*) 'X* = '
      WRITE (*, 800) (X(I), I = 1, N)
C
      STOP
  800 FORMAT (5F 12.6)
      END
C
C Целевая функция
C
      SUBROUTINE CALCF (N, X, F)
      REAL X(N), F, T1, T2
C
      T1 = 0.0
      T2 = 0.0
      DO   10 I = 2,N
      T1 = T1 + (X(I) - X(I - 1)**2)**2
      T2 = T2 = (1.0 - X(I - 1))**2
   10 CONTINUE
      F = 1.0 + 100.0*T1 + T2
C
C     RETURN
      END
```

Для этой конкретной функции, которая, в сущности, представляет собой сумму квадратов, результаты сильно зависят от точности машинной арифметики с плавающей точкой (см. § 5 гл. 2 и § 3 настоящей главы). Например, если используется компьютер CDC Cyber с $\varepsilon_{маш} = 6 \times 10^{-15}$, вычисленное оптимальное решение согласуется с x^* более чем

в шести десятичных цифрах. Если же используется IBM PC/AT ($\varepsilon_{маш} = 1 \times 10^{-7}$), та же самая программа дает

UNCMIN WARNING--INFO = 3: Точку с меньшим значением найти не удается
Код ошибки = 3
$F(X^*) = 1.00116$
$X^* =$

0.999745	0.999598	0.999370	0.998893	0.998021
0.996177	0.992586	0.985264	0.970914	0.942591

В таком случае часто бывает полезным использовать арифметику с двойной точностью.

9.6. Нелинейное приближение данных

Одной из наиболее часто встречающихся задач безусловной минимизации является аппроксимация данных нелинейной моделью. Если для решения этой задачи подходит метод наименьших квадратов (см. гл. 6), то задачу можно записать в виде

$$\min_x F(x_1, \ldots, x_n) = \sum_{j=1}^{m} [f_j(x_1, \ldots, x_n)]^2,$$

где каждая нелинейная функция $f_j(x_1, \ldots, x_n)$ представляет собой невязку для одной из точек данных.

Пример 9.6. Нелинейная аппроксимация данных.

Рассмотрим задачу нелинейного метода наименьших квадратов, основанную на экспоненциальной модели из § 1

$$b_j \approx x_1 \exp(x_2 t_j),$$

если заданы значения $\{(t_j, b_j)\}_{j=1}^{m}$. Функции-невязки равны

$$f_j(x_1, x_2) = b_j - x_1 \exp(x_2 t_j),$$

и целевая функция наименьших квадратов имеет вид

$$F(x_1, x_2) = \sum_{j=1}^{m} [b_j - x_1 \exp(x_2 t_j)]^2.$$

Для данных $\{(t_j, b_j)\} = \{(0,20), (1,9), (2,3), (3,1)\}$ получаем (с помощью UNCMIN) решение $x_1^* = 19.9900$, $x_2^* = -20.6230$. □

Нелинейная задача наименьших квадратов является задачей минимизации без ограничений, и, следовательно, для ее решения можно использовать методы, обсуждавшиеся в этой главе. Ниже приводится вызывающая программа для примера 9.6.

```
C Вызывающая программа аппроксимации данных с помощью нели-
C нейного метода наименьших квадратов
C
      PARAMETER (N = 2, LWORK = N*(N + 10), MD = 4)
      REAL XO(N), X(N), F, WORK(LWORK), T(MD), B(MD)
      COMMON/EXPDAT/T, B, M
      EXTERNAL CALCF
C
C Приближаемые данные
C
      T(1) = 0.0
      T(2) = 1.0
      T(3) = 2.0
      T(4) = 3.0
      B(1) = 20.0
      B(2) = 9.0
      B(3) = 3.0
      B(4) = 1.0
C
C Задание начальной оценки для решения
C
      M = 4
      XO(1) = 1.0
      XO(2) = 1.0
C
C Минимизация функции
C
C     CALL UNCMIN (N, XO, CALCF, X, F, IERROR, WORK, LWORK)
C Печать результатов
C
      IF (IERROR.NE. 0) WRITE (*,*) 'Код ошибки = ', IERROR
      WRITE (*,*)' F(X*) = ', F
      WRITE (*,*)' X* = ', (X(I),I = 1,N)
C
      STOP
      END
C Целевая функция
C
      SUBROUTINE CALCF (N,X,F)
      REAL X(N), F, T(4), B(4)
      COMMON/EXPDAT/T, B, M
C
      F = 0.0
      DO 10 J = 1, M
      F = F + (B(J) - X(1)*EXP(X(2)*T(J)))**2
```

```
 10 CONTINUE
C
      RETURN
      END
```

Нужно отметить несколько моментов. Во-первых, эта программа не очень сильно отличается от демонстрационной программы в § 5. Во-вторых, необходимо передать точки данных $\{(t_j, b_j)\}$ в подпрограмму CALCF, но это нельзя сделать с помощью оператора вызова подпрограммы. Нормальным образом точки данных передавались бы подпрограмме как параметры, но, поскольку CALCF вызывается в UNCMIN, множество параметров для CALCF не должно модифицироваться. Возможны следующие варианты: либо использовать оператор COMMON, как это сделано здесь, либо использовать операторы DATA, PARAMETER или присваивания внутри подпрограммы CALCF.

Этот подход дает хорошие результаты для многих задач и обладает тем преимуществом, что не требует дополнительного программного обеспечения. Однако это не идеальный подход к задаче нелинейной аппроксимации данных. Он аналогичен формированию нормальных уравнений в линейной задаче наименьших квадратов (см. § 3 гл. 6), и все замечания о нормальных уравнениях применимы и здесь. В этом случае может оказаться полезной версия с двойной точностью UNCMND. Более специализированные алгоритмы нелинейного приближения данных обходят эти проблемы; такое программное обеспечение можно найти, например, в пакете Minpack (см. список литературы).

9.7. Историческая справка: сэр Исаак Ньютон (1642–1727)

В 1543 г. Николай Коперник совершил революцию в научном мышлении, провозгласив, что Земля вращается вокруг Солнца. Раньше в течение многих сотен лет думали, что Земля – центр Вселенной, а человек – венец творения, и открытие Коперника в проникнутом религией обществе того времени представлялось «отпадением от благодати».

Это открытие также положило начало периоду, продолжающемуся по сей день, в котором для объяснения мира вокруг нас используется научный анализ. Работа Коперника была продолжена Галилеем, который усовершенствовал телескоп и открыл концепцию инерции. Хотя Галилей работал почти столетием позже Коперника, его идеи все еще считались радикальными, и Ватикан преследовал Галилея за «еретические» учения.

Галилей умер в 1642 г., Исаак Ньютон родился на Рождество того же года. Если не считать переживаний, вызванных смертью отца и повторным замужеством матери, его детство было ничем не примечательно. К счастью, его интеллектуальные интересы не прошли незамеченными,

и в 1661 г. его послали в Кембриджский университет. Официально университетское обучение еще было основано на моделях и представлениях греков, но при этом широко обсуждались новые идеи Коперника и других ученых. В Кембридже Ньютон изучал работы французского философа Рене Декарта (по имени которого названы «декартовы координаты») и химика Роберта Бойля. Ньютон окончил университет в 1665 г., не стяжав особой известности.

В том же году университет был закрыт из-за чумы. В течение двух лет Ньютон работал в изоляции, опасаясь болезни. Именно в эти два года он сделал большинство своих главных открытий. Он разработал дифференциальное исчисление; он открыл в ходе экспериментов с призмами, что свет состоит из цветов; и он обнаружил, что радиальная сила, действующая на планету, убывает обратно пропорционально квадрату ее расстояния от Солнца, что стало основой для открытого им позже закона всемирного тяготения.

Когда университет открылся вновь в 1667 г., Ньютону было предоставлено место в нем. Он читал там лекции по оптике, публично провозглашая свои новые идеи о природе света, идеи, прямо противоположные принятым тогда теориям Декарта. Однако его лекции, очевидно, не произвели большого впечатления. Только в 1671 г., когда в Королевском обществе узнали об изобретении им телескопа-рефлектора, работы Ньютона стали привлекать внимание.

Восприятие работ Ньютона не было полностью благожелательным. Роберт Гук, известный в Королевском обществе ученый, критиковал его идеи. Ньютон реагировал раздраженно, неадекватно степени критики. Когда в 1675 г. Ньютон представил работу по оптике, Гук обвинил его в плагиате. Эта и другая критика так возмутила Ньютона, что в 1678 г. он, очевидно, испытал нервный срыв и на шесть лет изолировался от общества.

В течение этого периода Ньютон продолжал исследовать движение планет, что привело к написанию трактата "Philosophiae Naturalis Principia Mathematica" («Математические начала натуральной философии»), который стал наиболее важной книгой в современной науке. В нем Ньютон установил три закона движения: (1) тело остается в покое, если на него не действует сила; (2) сила пропорциональна массе, умноженной на ускорение; (3) для каждого действия имеется равное и противоположное по направлению противодействие. От этих трех законов он переходит к выводу закона всемирного тяготения: любые два тела притягивают друг друга с силой, пропорциональной произведению их масс, деленному на квадрат расстояния между ними.

Гук снова обвинил Ньютона в плагиате. В ответ Ньютон изъял из своей книги все ссылки на Гука и отложил публикацию своей книги «Оптика» до смерти Гука. Несмотря на обвинения Гука, "Principia" принесли Ньютону небывалую всемирную славу. Он был избран президентом Королевского общества, посвящен в рыцари (первый ученый, который удостоился такой чести) и сделан смотрителем монетного

двора – положение, которое принесло ему солидный доход.

В поздние свои годы Ньютон провел много часов, изучая алхимию, потратив на это так же много усилий, как на свои более респектабельные научные изыскания. Он занимался также религиозными исследованиями. Он отверг понятие Троицы – мнение, которое могло бы привести к его изгнанию из университета, если бы стало известно. Он также активно занимался управлением монетным двором. В последние годы жизни он продолжал издавать свои работы, по-прежнему сердился на тех, кто его критиковал, и председательствовал в Королевском обществе. Он умер в 1727 г.

9.8. Некоторые дополнительные сведения

В этой главе обсуждалась только оптимизация без ограничений. Задачи с ограничениями обычно являются значительно более трудными для решения, чем задачи без ограничений. Если функция F и все ограничения g_i линейны, это задача *линейного программирования*; можно показать, что множество S в этом случае будет выпуклым многогранником в n-мерном пространстве, а решение – *вершиной* этого многогранника. Обычный метод решения состоит в обходе вершин с перемещением на каждом шаге от текущей вершины к смежной. Трудности линейного программирования связаны главным образом с рещением задач для очень больших n, которые приводят к разреженным матрицам. Такие задачи оказываются трудными из-за комбинаторной сложности многогранника общего вида в n-мерном пространстве. Если функция или какое-то из ограничений нелинейны, задача относится к *нелинейному программированию*. Линейное и нелинейное программирование выходят за рамки данной книги.

Методы решения задач с ограничениями, как линейных, так и нелинейных, описываются в книге [Luenberger, 1984]; дополнительный материал по этой теме можно найти в книгах [Orchard-Hays, 1968], [Dantzig, 1963] и [Gill et al., 1981].

Следует также заметить, что обсуждавшиеся здесь методы подходят только для решения небольших задач, где число переменных не намного превышает, скажем, 100. Это обусловлено тем, что они требуют хранения и обработки матриц, и связанные с этим затраты быстро растут с увеличением размера задачи. В некоторых случаях для эффективного решения таких задач можно использовать методы для разреженных матриц в сочетании с методом Ньютона, однако существуют методы с меньшими запросами к памяти и меньшим объемом вычислений на итерацию, имеющие более широкую область применения. Обзор методов для задач больших размеров можно найти в [Gill et al., 1981].

Подпрограммы нелинейной оптимизации можно классифицировать в соответствии с тем, имеют ли они дело с задачами без ограничений или с ограничениями, является ли целевая функция суммой квадратов или нелинейной функцией общего вида и требуется ли вычисление производ-

ных. Пакеты таких подпрограмм обычно включают также процедуры для решения систем нелинейных уравнений.

Ряд процедур оптимизации можно найти в библиотеке программ, разработанной Лабораторией оптимизации систем при Станфордском университете. Для больших задач эффективной процедурой является подпрограмма MINOS (описанная в статье [Murtagh, Saunders, 1980]). Для решения нелинейных уравнений и нелинейных задач приближения данных можно использовать подпрограммы из пакета Minpack.

Методы поиска глобального оптимума сейчас находятся в стадии активного исследования. Дальнейшую информацию о существующих методах см. в статьях в книге [Boggs et al., 1985].

9.9. Задачи

9.1. При расширении области определения интеграла ошибок $\operatorname{erf}(x)$ на комплексные значения аргумента приходится рассматривать вещественную функцию вещественного аргумента, известную как интеграл Доусона,

$$D(x) = e^{-x^2} \int_0^x e^{t^2}\, dt.$$

Поскольку при x, стремящемся к 0 или ∞, $D(x)$ стремится к 0 и поскольку $D(x) > 0$ при промежуточных значениях x, функция $D(x)$ должна иметь конечный максимум. Используйте подпрограмму FMIN для нахождения максимума функции $D(x)$.

9.2. Используйте подпрограмму FMIN для нахождения максимума функции

$$F(x) = (\sin(x))^6 \operatorname{tg}(1 - x) e^{30x}$$

на интервале [0,1].

9.3. Используйте подпрограмму UNCMIN для минимизации функции

$$F(x_1, x_2) = \frac{1}{2}(x_1^2 + x_2^2) \exp(x_1^2 - x_2^2)$$

с начальными значениями $(1, 1)^T$ и $(0.1, 0.1)^T$.

9.4. Для любого $\lambda > 0$ обозначим через $z(\lambda)$ наименьший положительный нуль функции $y(t)$, которая является решением начальной задачи для обыкновенного дифференциального уравнения

$$y''(t) + I_0(y) + t/10 = 0,$$
$$y(0) = 0, \quad y'(0) = \lambda,$$

где $I_0(t)$ – модифицированная функция Бесселя нулевого порядка (см. книгу [Abramowitz, Stegun, 1965], с. 374–375). Примерный график решения $y(t)$ показан на рис. 9.8.

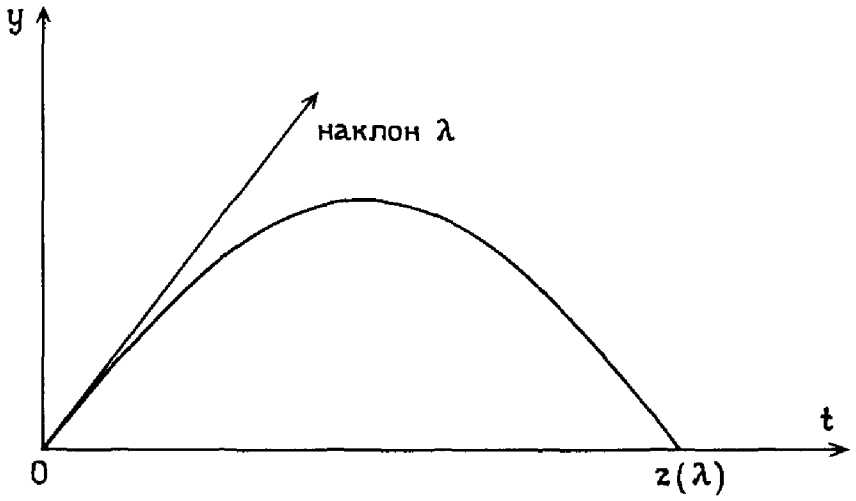


Рис. 9.8.

(а) Найдите λ_{max}, единственное такое значение λ, что $z(\lambda)$ является максимумом.

(б) Найдите также $z(\lambda_{max})$.

(в) Постройте таблицу значений $y(t)$ для $t = 0(0.1)\, z(\lambda_{max})$ для функции $y(t)$, максимизирующей $z(\lambda)$, т. е. для функции $y(t)$, удовлетворяющей условию $y'(0) = \lambda_{max}$.

(г) Обсудите точность ваших результатов.

9.5. Следующая задача была поставлена в 1696 г. Иоганном Бернулли и Готфридом Лейбницем, считавшими, что Ньютон сможет ее решить; он сделал это, создав тем самым *вариационное исчисление*. Предположим, что на стене заданы две точки, назовем их P_1 и P_2, причем P_1 выше, чем P_2. Если уронить шар из точки P_1, каким путем ему нужно следовать, чтобы достичь P_2 как можно быстрее? Если бы точка P_1 находилась по вертикали над P_2, этот путь был бы прямой линией, но в общем случае он более сложен. Приближенное решение этой задачи можно получить следующим образом. Рассмотрим рис. 9.9. Обозначим путь, дающий решение задачи, через $y = p(t)$. Если бы мы знали значения $p(t)$ в дискретные времена t_0, t_1, ..., то могли бы аппроксимировать решение, используя интерполяцию. Точное решение задачи будет удовлетворять условию

$$\min_{p(t)} \int_{P_1}^{P_2} \sqrt{\frac{1 + (p'(t))^2}{-p(t)}}\, dt.$$

Чтобы аппроксимировать решение, обозначим через t_0, t_1, ..., t_{n+1} равноотстоящие точки в интервале $[0, x_2]$, где $t_0 = 0$ и $t_{n+1} = x_2$. Пусть $p_i = p(t_i)$. Используйте подпрограмму UNCMIN для решения задачи

$$\min_{p_1, \ldots, p_n, p_{n+1}} \int_0^{x_2} \sqrt{\frac{1 + (g'(t))^2}{-g(t)}}\, dt,$$

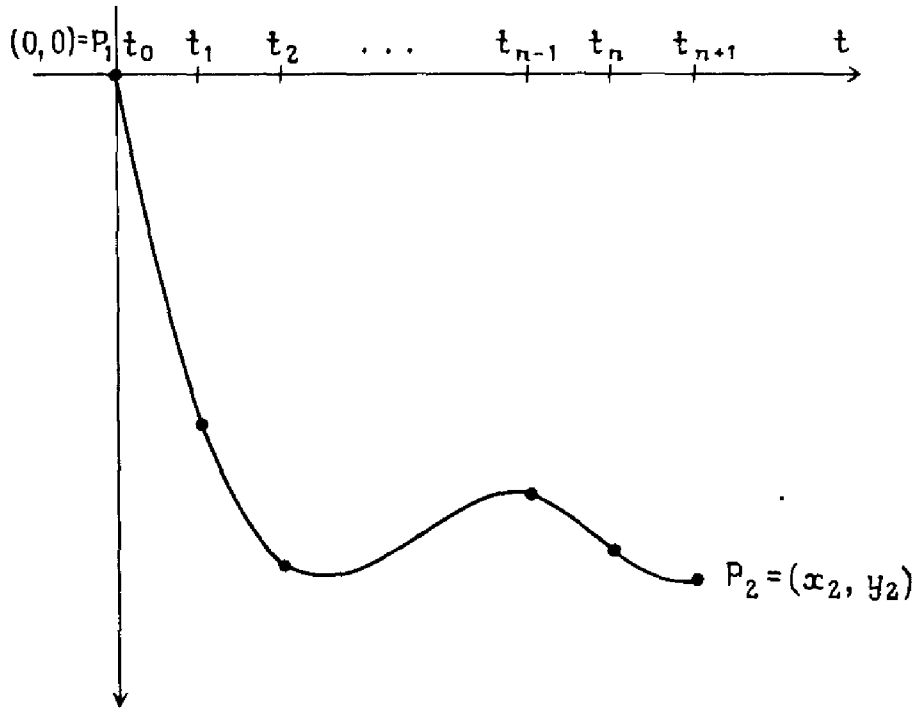


Рис. 9.9.

где $g(t) = -p_{n+1}\sqrt{t} + h(t)$, а $h(t)$ – кубический эрмитов интерполянт для точек $(t_0, 0 + p_{n+1}\sqrt{t_0})$, $(t_1, p_1 + p_{n+1}\sqrt{t_1})$, ..., $(t_n, p_n + p_{n+1}\sqrt{t_n})$, $(t_{n+1}, y_2 + p_{n+1}\sqrt{t_{n+1}})$, который вычисляется подпрограммой PCHEZ. Член $-p_{n+1}\sqrt{t}$ включен, чтобы моделировать вырождение производной в точке (0, 0). Вычислите интеграл с помощью подпрограммы QK15; для вычисления подынтегрального выражения воспользуйтесь подпрограммой PCHEV. В качестве начального приближения используйте отрезок прямой линии, соединяющей P_1 и P_2; положите $n = 5$. Попробуйте вашу программу на трех задачах, взяв $P_1 = (0, 0)$, а $P_2 = (1, -5)$, $(1, -1)$ и $(5, -1)$. Эта задача известна как проблема «брахистохроны» («наименьшего времени»), а точное ее решение – кривая, называемая *циклоидой*.

9.6. Разделимые наименьшие квадраты. Предположим, что нужно аппроксимировать m точек данных (t_i, y_i), $i = 1, \ldots, m$, методом наименьших квадратов с помощью функции

$$b(t) = c_1 + c_2 t + c_3 t^2 + c_4 e^{\lambda t}.$$

Эта функция содержит 5 параметров: c_1, c_2, c_3, c_4 и λ. Параметры c_i входят линейно, однако λ входит нелинейным образом. Пусть $A(\lambda)$ – матрица размера $m \times 4$ с элементами

$$a_{i,j} = t_i^{j-1}, \quad j = 1, 2, 3,$$
$$a_{i,4} = e^{\lambda t_i}.$$

Пусть b – заданный вектор данных b_i длины m, и пусть c – неизвестный

вектор коэффициентов c_j длины 4. Тогда мы имеем следующую задачу оптимизации:

$$\min_{\lambda} \min_{c} \| A(\lambda)c - y \|_2^2 .$$

Внутренний минимум, включающий четыре линейных параметра, можно найти для любого λ с помощью подпрограммы SQRLS. Внешний минимум, включающий единственный нелинейный параметр λ, можно найти с помощью подпрограммы FMIN. Заметим, что SQRLS будет вызываться подпрограммой-функцией, которую вызывает FMIN.

Таблица 9.6

t	y Группа 1	y Группа 2
0.00	20.00	20.00
0.25	51.58	24.13
0.50	68.73	26.50
0.75	75.46	27.13
1.00	74.36	26.00
1.25	67.09	23.13
1.50	54.73	18.50
1.75	37.98	12.13
2.00	17.28	4.00

В таблице 9.6 приводятся две группы данных, к которым нужно применить этот подход. Первая группа не должна представлять никакой трудности, но вторая приводит к вырожденности, которую можно обнаружить путем проверки значений внутреннего минимума и ранга матрицы $A(\lambda)$. Найдите c_i и λ, дающие наилучшее приближение. Являются ли эти значения c_i единственным решением? Предполагается, что данные имеют только два верных десятичных знака, так что y_i могут иметь ошибки, достигающие 0.005.

9.7. Аппроксимируйте данные переписи населения из задачи 4.2 с помощью модели

$$b(t) \approx c_1 + c_2(t - 1900) + c_3 e^{\lambda(t - 1900)}$$

с меняющимся λ, как описано в предыдущей задаче. Дайте прогноз численности населения в 1980 г.

9.8. Аппроксимируйте функцию $\mathrm{erf}(t)$ той же моделью, что и в задаче 6.1 (в), но с

$$z = \frac{1}{1 + \lambda t}.$$

Нелинейный параметр λ можно найти, используя FMIN и метод, описанный в задаче 9.6.

9.9. Установлено, что сверхновой звезде типа I соответствует специальная модель светимости (яркости). Спустя несколько дней после максимума светимости эту модель можно описать так:

$$L(t) = C_1 \exp(-t/\alpha_1) + C_2 \exp(-t/\alpha_2),$$

где t – время (в днях) с момента максимальной светимости, а $L(t)$ – светимость относительно максимальной светимости. В табл. 9.7 приводятся данные по относительной светимости для сверхновой I939A6 типа I, полученные в 1939 г. Пик светимости приходится на день 0.0, но все наблюдения, полученные до дня 7.0, опущены, поскольку указанная модель не описывает светимости перед и непосредственно после максимума.

Таблица 9.7

День	Светимость	День	Светимость	День	Светимость
7.0	0.6310	19.0	0.1318	57.0	0.05754
7.0	0.8318	20.9	0.1585	85.0	0.03631
14.8	0.2754	25.8	0.1096	109.0	0.02291
16.0	0.1445	26.8	0.1445	110.0	0.02291
16.9	0.2089	28.0	0.09120	141.0	0.01738
17.0	0.1585	53.0	0.06310	142.0	0.01585
18.8	0.1585	54.0	0.06918	168.0	0.009120

(а) Нанесите данные на график. Вы заметите две различных области; таким образом, для того чтобы обеспечить адекватное приближение, требуются две экспоненты.

(б) Используйте подпрограмму UNCMIN для приближения данных указанной моделью. Нанесите приближение на график данных. Изобразите также невязки. Выглядят ли они случайными? При выборе начальных значений большую роль играет опыт. Известно, что константы времени α_1 и α_2 приблизительно равны 5.0 и 60.0 соответственно. Попробуйте различные начальные значения для C_1 и C_2. Насколько чувствительно результирующее приближение к этим значениям?

9.10. Обратимся к задаче 6.8, в которой говорится о модели содержания CO_2 в атмосфере

$$y(t) = B + d\exp(\alpha t) + \sum_{i=1}^{n} A_i \sin\frac{2\pi}{(T/i)}(t + \varphi_i),$$

где t измеряется в месяцах, а другие параметры неизвестны. Используйте метод из задачи 6.8 и подпрограмму UNCMIN, чтобы найти эти параметры для различных n. Изобразите приближение и невязки на графике. Хорошо ли эта модель аппроксимирует данные? (Заметьте, что член $\exp(\alpha t)$ чувствителен к малым изменениям в α. Мы предлагаем вам решать эту задачу в два этапа. Сначала зафиксируйте $\alpha = 0.0037$ и оцените другие параметры, а затем используйте эти оценки как начальное приближение для полной модели. Однако даже в этом случае могут возникнуть трудности из-за переполнений.)

9.11. Примените метод Ньютона к решению задачи

$$\min F(x) = (x - 2)^4 - 9$$

с начальным приближением $x = 1$. Почему метод сходится так медленно?

9.10. Прологи: FMIN и UNCMIN

```
      REAL FUNCTION FMIN (AX, BX, F, TOL)
C*** НАЧАЛО ПРОЛОГА FMIN
C*** ДАТА НАПИСАНИЯ 730101 (ГГММДД)
C*** ДАТА ПЕРЕСМОТРА 730101 (ГГММДД)
C*** КАТЕГОРИЯ NO. GIA2
C*** КЛЮЧЕВЫЕ СЛОВА: одномерная минимизация, унимодальная
C                     функция
C*** АВТОР: BRENT R.
C*** НАЗНАЧЕНИЕ: значением функции FMIN является аппроксима-
C                ция точки, в которой F достигает минимума на
C                интервале (AX, BX).
C*** ОПИСАНИЕ
C*** Из книги D. Kahaner, C. Moler, S. Nash
C              "Numerical Methods and Software"
C              Prentice-Hall, 1988.
C     Используемый метод является комбинацией метода золотого
C     сечения и последовательной параболической интерполяции.
C     Сходимость не намного медленнее, чем для метода Фибоначчи.
C     Если F имеет непрерывную вторую производную, положительную
C     в точке минимума (которая не совпадает с XH или BH), сходимость
C     является сверхлинейной и обычно имеет порядок I.324....
C     Функция F никогда не вычисляется в двух точках, расстояние
C     между которыми меньше, чем EPS*ABS(FMIN) + (TOL/3), где
C     EPS примерно равно квадратному корню из относительной
C     машинной точности. Если F - унимодальная функция и вычисляе-
C     мые значения F сохраняют унимодальность, когда аргументы
C     различаются не менее чем на EPS*ABS(XSTAR) + (TOL/3),
C     то FMIN приближает абсциссу глобального минимума F на
```

```
C     интервале AX, BX с погрешностью, меньшей чем 3*EPS*
C     *ABS(FMIN) + TOL. Если F не унимодальна, то FMIN может
C     аппроксимировать локальный, но, возможно, не глобальный ми-
C     нимум с той же точностью.
C     Эта подпрограмма-функция является слегка модифицированной
C     версией процедуры LOCALMIN на Алголе-60, приведенной в книге
C     Richard Brent, Algorithms for Minimization Without Derivatives,
C     Prentice-Hall, Inc., 1973.
C     Входные параметры:
C       AX  REAL
C                левый конец начального интервала.
C       BX  REAL
C                правый конец начального интервала.
C       F        Вещественная функция вида REAL FUNCTION
C                F(X), которая вычисляет F(X) для любого X из
C                интервала (AX, BX).
C                Должна быть объявлена в операторе EXTERNAL
C                вызывающей программы.
C       TOL      REAL
C                желательная длина интервала неопределенности окон-
C                чательного результата (≥ 0.0).
C     Выходные параметры:
C       FMIN     Приближенная абсцисса минимума функции.
C       AX       Нижняя граница абсциссы минимума.
C       BX       Верхняя граница абсциссы минимума.
C*** ССЫЛКИ: RICHARD BRENT, ALGORITHMS FOR MINIMI-
C                ZATION WITHOUT DERIVATIVES, PRENTICE-
C                HALL, INC., 1973.
C
C*** ВЫЗЫВАЕМЫЕ ПОДПРОГРАММЫ: НЕТ
C*** КОНЕЦ ПРОЛОГА FMIN
C***   SUBROUTINE UNCMIN (N, XO, FCN, X, F, INFO, W, LW)
C
C*** НАЧАЛО ПРОЛОГА UNCMIN
C*** ДАТА НАПИСАНИЯ 870923 (ГГММДД)
C     ДАТА ПЕРЕСМОТРА 871222 (ГГММДД)
C*** КАТЕГОРИЯ NO, GIBIAL
C*** КЛЮЧЕВЫЕ СЛОВА: минимизация без ограничений
C*** АВТОР: Nash S.G. (George Mason University)
C*** НАЗНАЧЕНИЕ: UNCMIN минимизирует гладкую нелинейную
C     функцию n переменных. Требуется подпрограмма, которая вы-
C     числяет значение функции в любой точке, однако значения произ-
C     водных не требуются.
```

```
C     UNCMIN обеспечивает простой интерфейс для более гибких
C     процедур нижнего уровня. Пользователь не имеет контроля над
C     опциями.
C
C*** ОПИСАНИЕ
C     Из книги D. Kahaner, C. Moler, S. Nash "Numerical Methods and
C                    Software" Prentice-Hall, 1988
C     Эта процедура использует квазиньютоновский алгоритм с линей-
C     ным поиском, чтобы минимизировать функцию, представленную
C     подпрограммой FCN. На каждой итерации нелинейная функция
C     аппроксимируется квадратичной функцией, полученной с помо-
C     щью разложения Тейлора. Направление поиска определяется пу-
C     тем минимизации квадратичной функции, а приближенный мини-
C     мум нелинейной функции вдоль направления поиска находится
C     с помощью метода линейного поиска. Алгоритм вычисляет ап-
C     проксимацию для матрицы вторых производных нелинейной
C     функции, используя квазиньютоновские методы.
C     Пакет UNCMIN весьма универсален и обеспечивает пользова-
C     телю много опций. Однако эта подпрограмма спроектирована
C     для легкого использования и предусматривает не много вариан-
C     тов выбора. Например:
C
C     1. Необходимо вычислять только значения функции. Значения
C        первых производных получаются в результате конечно-разност-
C        ного дифференцирования. Это может стоить очень дорого,
C        когда число переменных велико.
C
C     2. Предполагается, что значения функции можно получить с
C        хорошей точностью (сравнимой с точностью компьютерной
C        арифметики).
C
C     3. Разрешается не более 150 итераций.
C
C     4. Предполагается, что значения функции хорошо шкалированы,
C        т.е. оптимальное значение функции не является патологичес-
C        ки большим или маленьким.
C
C     Более подробную информацию см. в работе, указанной ниже.
C
C     Описание параметров:
C       N       INTEGER
C               Размерность задачи.
C      XO(N)    REAL
C               Начальная оценка минимума.
C       FCN     Имя подпрограммы для вычисления минимизируемой
C               функции. Эта подпрограмма должна быть объявлена
```

```
C                в операторе EXTERNAL в вызывающей программе
C                и иметь заголовок SUBROUTINE FCN (N, X, F)
C                где N и X имеют тот же смысл, как здесь а F–
C                значение вычисляемой функции.
C       X(N)     REAL
C                Локальный минимум.
C       F        REAL
C                Значение функции в локальном минимуме X
S       INFO     INTEGER
C                Код завершения
C                INFO =  0: Найдено оптимальное решение.
C                INFO =  1: Завершение с малым градиентом; X, веро-
C                           ятно, оптимально.
C                INFO =  2: Завершение с малым размером шага; X,
C                           вероятно, оптимально.
C                INFO =  3: Точку с меньшим значением F найти не
C                           удается; X, вероятно, оптимально.
C                INFO =  4: Исчерпан лимит (150) числа итераций.
C                INFO =  5: Слишком много больших шагов; функция
C                           может быть неограниченной.
C                INFO = –1: Недостаточно места для рабочих масси-
C                           вов.
C       W(LW)    REAL
C                Рабочий массив.
C       LW       INTEGER
C                Размер рабочего массива: не меньше чем N*(N + 10).
C
C       ССЫЛКИ: R. B. SCHNABEL,  J. E. KOONTZ,  BE. E. WEISS
C                A MODULAR SYSTEM OF ALGORITHMS FOR
C                UNCONSTRAINED MINIMIZATION, REPORT
C                CU-CS-240-82, COMP. SCI. DEPT., UNIV. OF
C                COLORADO AT BOULDER, 1982.
C*** ВЫЗЫВАЕМЫЕ ПОДПРОГРАММЫ: OPTDRV, XERROR
C*** КОНЕЦ ПРОЛОГА UNCMIN
```

Глава 10
Моделирование и случайные числа

10.1. Введение

Когда мы говорим о «случайности» в нашей повседневной жизни, то подразумеваем непредсказуемое событие, такое, например, как удар молнии или результат подбрасывания монеты, приход в магазин очередного покупателя или время между приходом двух посетителей. В действительности же ни одно из этих событий не является непредсказуемым; например, результат подбрасывания монеты зависит от ее начальной ориентации, силы броска, высоты над землей, сопротивления воздуха и т. д., но эта зависимость весьма сложна. Поэтому проще рассматривать результат как случайный, особенно если нас интересует средний результат, а не исход очередного бросания. «Ничто в природе не случайно... Кажущаяся случайность событий есть лишь проявление неполноты нашего знания о них» [1]). Помехи на экране телевизора и продолжительность работы электрической лампочки случайны именно в этом смысле. В этих примерах известны долговременные или средние свойства, но нет определенности в отношении конкретных событий. Мы можем знать, что ночью в пятницу приемная больницы обслуживает в среднем 200 пациентов, но мы не можем сказать, сколько больных поступит в больницу в пятницу 13 декабря – 192 или 225. Известно, что вероятность выпадения «змеиных глаз» (двух единиц) при бросании двух костей равна 1/36, но мы не знаем, при каком броске они выпадут – при 7-м или при 11-м, и т. д.

Информация об исходах, которые более вероятны, чем другие, важна для предсказаний. Например, фермеры, строительные компании, авиалинии принимают решения на основе прогноза погоды. Игроки изобретают системы успешной игры в рулетку, черное и белое, покер. Городские власти решают, сколько пожарных команд нужно иметь в городе, больницы определяют количество обслуживающего персонала для приемных покоев, владельцы парикмахерских решают, сколько парикмахеров им нанимать. Если один парикмахер может подстричь 20 человек за день и ожидается приход 60 человек в день, значит ли это, что владелец должен нанять 3 парикмахеров? Если за день придут 64 клиента, не начнут ли они ворчать из-за слишком долгого ожидания?

Вообще любой интересующий нас вопрос в этих ситуациях можно записать в форме уравнений, обычно достаточно сложных и редко поддающихся точному решению. Для разрешения таких ситуаций часто

[1]) Спиноза, Этика, I. Читатели, знакомые с квантовой механикой, могут не согласиться с этим утверждением.

используют моделирование. Типичная модель содержит утверждения на машинном языке, которые очень похожи на моделируемую ситуацию. В случае парикмахерской модель должна включать в себя следующие сегменты:

```
      Сегмент ПАРИКМАХЕР (парикмахер готов к приему нового
      заказчика)
            ЕСЛИ по крайней мере один заказчик ждет
                  ТО
                        взять заказчика, который ожидает дольше
*                       определить время выполнения стрижки
                  ИНАЧЕ
                        пометить парикмахера как свободного
            Конец цикла ЕСЛИ
      Конец сегмента ПАРИКМАХЕР
      Сегмент ЗАКАЗЧИК (заказчик только что пришел)
*           Определить время до прихода следующего заказчика
            ЕСЛИ по крайней мере один парикмахер свободен
                  ТО
*                    выбрать парикмахера
*                    определить время выполнения стрижки
                  ИНАЧЕ
                        поставить заказчика в очередь
            Конец цикла ЕСЛИ
      Конец сегмента ЗАКАЗЧИК
```

Перед нами типичный алгоритм моделирования, записанный с помощью псевдокода. Эти сегменты составляют логический базис моделирования очереди, но в дополнение потребуются еще многие детали, такие, как планирование событий и сбор статистики. Приведенную выше псевдопрограмму можно переписать на Фортране или на каком-нибудь другом языке, но существуют и специальные языки моделирования, такие, как GPSS [Schriber, 1974], Simscript [Russel, 1983], SLAM [Pritsker, 1986]. Хорошим общим справочником по моделированию является книга [Bratley et al., 1987].

В приведенном выше тексте четыре оператора отмечены звездочками. Все остальные операторы детерминированы. Выделенные операторы представляют собой вероятностные события. Наша задача здесь заключается в том, чтобы генерировать моменты прихода клиентов и выбирать парикмахеров, причем все это должно быть похоже на реальную ситуацию. Отсюда следует, что значения времен между прибытиями клиентов и моменты завершения стрижки должны быть похожи на некоторый случайный процесс, для которого известны, например, среднее и дисперсия (см. § 2 гл. 6). А вот пример правила выбора парикмахера: если два парикмахера свободны, то младший из них выбирается в двух случаях из трех. Чтобы достичь поставленной цели, нам нужен метод генерирования времен прихода клиентов и правило выбора,

причем их средние свойства должны совпадать со средними свойствами реального процесса. Для этого нужны случайные числа.

Существуют и задачи другого типа, которые сами по себе не содержат случайного элемента, но где для тех или иных целей используется случайная выборка. Например, в период кампании по выборам президента США было бы слишком дорого, да и излишне опрашивать каждого потенциального избирателя для достоверного предсказания результатов выборов. Вместо этого по тщательно продуманным правилам опрашивается небольшая выборка, состоящая из нескольких тысяч человек, и по этим данным делают заключение обо всем населении. В другой ситуации в гл. 5 было показано, что метод Монте-Карло можно использовать для оценки значения интеграла любого числа измерений, и объяснялось, как этот метод применялся и продолжает применяться для проектирования сложных ядерных реакторов.

10.2. Случайные числа

Во многих случаях приходится генерировать на компьютере длинные последовательности чисел, которые ведут себя так, как если бы они были случайными. Формальное определение случайности трудно применить на практике, так как оно означает отсутствие определенного образца в поведении. Однако интуитивное определение заключается в том, что последовательность чисел является случайной, если простейший способ ее описания состоит в прямом выписывании этой последовательности. Так, последовательность

0101010101010101010101010101010101010101010101010101

можно определить кратким алгоритмом, тогда как последовательность

0110110010000010101000010101001011111010100001010101

определить кратким алгоритмом не удается.

Это определение случайности имеет следующие недостатки: (1) оно не говорит нам, как построить случайную последовательность, и (2) в основе своей неверно, так как создает впечатление, что можно определить, случаен ли процесс, рассматривая конкретную последовательность. На самом деле это не так. Если при подбрасывании монеты мы обозначим «решку» через 0, а «орла» через 1, то вероятность того, что при 52 подбрасываниях получится первая последовательность, равна $(1/2)^{52}$; вероятность получить вторую последовательность та же самая. И тем не менее мы отклонили бы первую последовательность и допустили бы вторую! По-видимому, то, что нам нужно,– это последовательность, прошедшая статистические тесты на случайность и имеющая большой интервал между повторениями. Этот интервал называется *периодом* и играет в теории важную роль. Мы обсудим некоторые из этих тем в § 3.

Существует два хорошо известных класса методов генерирования случайных чисел.

(1) *Методы, основанные на использовании реальных явлений.* Они включают в себя, например, подсчет времени между двумя щелчками счетчика Гейгера, подбрасывание монеты или вытаскивание шара из урны. Эти методы медленны и могут не удовлетворять статистическим тестам на случайность. В любом случае физический процесс, будучи случайным, не поддается воспроизведению, и при отладке программ, использующих эти методы, возникают большие трудности. Более того, для таких приложений, как квадратурный метод Монте-Карло, теоретические результаты показывают, что подобные случайные числа являются менее эффективными, чем искусственно сгенерированные последовательности, описываемые в этой главе. Физические устройства для генерирования случайных чисел сегодня редко используются для вычислений, так как они работают очень медленно. Однако в лотереях по-прежнему используется выбор предметов из урны, поскольку полного доверия к компьютеру все еще нет.

(2) *Методы, основанные на алгоритмах.* С помощью алгоритмов можно получать последовательности случайных чисел независимо от их последующего потребления программой, хранить их в виде файла на диске или ленте и использовать тогда, когда это необходимо. Практическая потребность в случайных числах существовала задолго до появления компьютеров и до изобретения метода Монте-Карло. Таблица из 40 000 случайных цифр, опубликованная в 1927 г., была основана на информации о переписи населения. Позже для этих целей использовались специальные машины, и в 1955 г. компания RAND Corporation опубликовала «Миллион случайных цифр и 100 000 случайных чисел с нормальным распределением». Такие совокупности хороши тем, что поддаются воспроизведению и, следовательно, вычисления можно повторить и проверить. Обычно эти числа записывались на ленту и эксплуатировались слишком часто. Один из путей решения возникающих при этом проблем состоит в том, чтобы начинать использовать файл со случайного места, но тогда мы снова приходим к проблеме выработки случайной позиции в файле. В прошлом статистики изобретали изощренные методы пользования таблицами случайных чисел.

В настоящее время чаще всего применяются алгоритмы, которые генерируют случайные числа в режиме реального времени, и эти числа тут же потребляются программой. Числа выдаются по мере необходимости и вполне поддаются воспроизведению, что важно при отладке программы [1]. Поскольку такие числа генерируются с помощью алгоритма, на самом деле они являются детерминированными, а вовсе не случайными, и поэтому называются *псевдослучайными.* Программу,

[1] В будущем возможен возврат к генераторам, «просматривающим таблицу»: компакт-диск может хранить около 125 миллионов хорошо проверенных случайных чисел и стоить при этом всего несколько долларов.

которая порождает эти числа, называют *генератором случайных чисел*. Мы считаем последовательность псевдослучайных чисел случайной, если «каждый ее член непредсказуем для непосвященного и она удовлетворяет ряду традиционных статистических тестов» [Lehmer, 1951]. Однако мы располагаем лишь конечным числом тестов, и поэтому существует вероятность того, что какой-то изъян останется невыявленным.

Кроме того, большинство алгоритмических генераторов *цикличны*, или *периодичны*, т.е. числа в конце концов начинают повторяться. Тем не менее этот подход, по-видимому, является наиболее удобным, и поэтому в нашей книге будут обсуждаться только такие методы.

Должно быть очевидно, что, хотя математическое определение случайности можно сделать точным, практическое определение (особенно для алгоритмических генераторов) зависит от конкретного применения. Генератор, используемый в игре, не обязательно должен быть таким же хорошим, как генератор для многомерных квадратур. Для оценки качества генераторов предложено много критериев. В общем случае один генератор не может всем им удовлетворять. Вот эти требования.

(а) *Высокое качество*. Генератор должен пройти все статистические тесты и иметь очень длинный период.

(б) *Эффективность*. Исполнение должно быть быстрым, а размеры занимаемой памяти – минимальными.

(в) *Повторимость*. Точно воспроизводя начальные данные, мы должны получать одну и ту же последовательность. Пользователь должен иметь возможность перезапустить генератор в любой момент, но при этом можно обойтись без явной инициализации. Незначительные изменения в начальной процедуре будут приводить к разным последовательностям.

(г) *Машинная независимость и переносимость*. Алгоритм должен работать на машинах различных типов; в частности, не должно быть операций, приводящих программу к остановке. При одинаковой инициализации генератора он должен давать одну и ту же последовательность случайных чисел на разных компьютерах.

(д) *Простота*. Алгоритм должен быть прост в реализации и использовании.

Не существует генераторов, удовлетворяющих всем этим критериям. Например, моделирование на суперкомпьютере требует особенно эффективного генератора, но от него можно не требовать, чтобы он давал те же самые числа (или даже просто работал) на другом компьютере.

10.3. Генерирование равномерно распределенных чисел

Наиболее часто используются *равномерно распределенные* числа, действительные или целые. Генератор равномерно распределенных чисел порождает значения, равномерно распределенные на интервале $[0, 1)$. Это означает, что вероятность числа попасть в подынтервал $0 \leqslant a < b \leqslant 1$ равна длине интервала, т.е. $b - a$. Таким образом, если

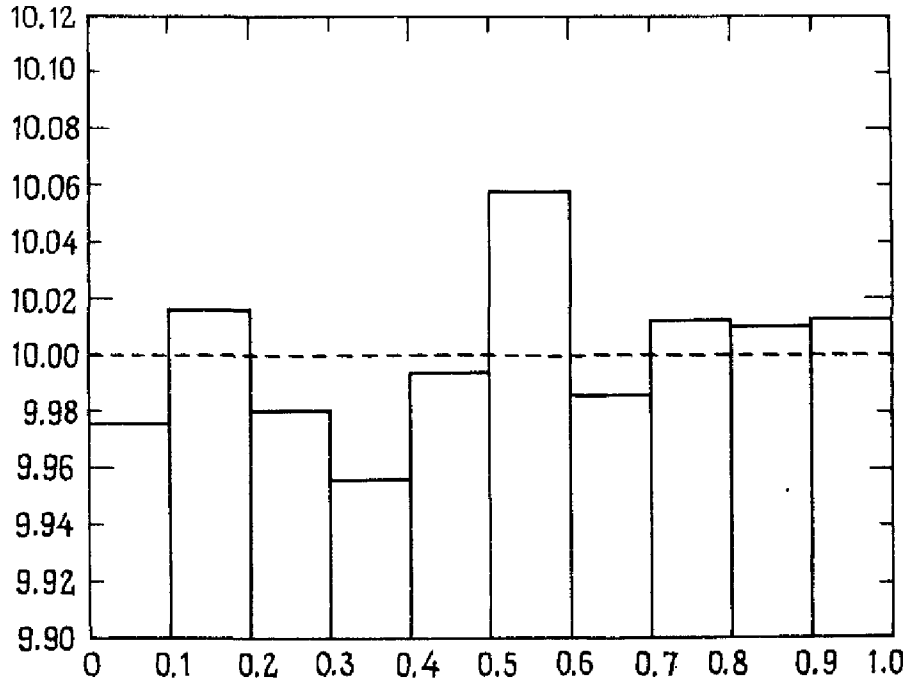


Рис. 10.1. Распределение 50 000 чисел, сгенерированных программой RND, по 10 подынтервалам.

мы генерируем 50 000 чисел, то около 25 000 из них будут лежать в интервале [0.4, 0.9).

Например, рис. 10.1 иллюстрирует один эксперимент с генератором случайных чисел RND, который вместе с интерпретатором Бейсика входит в состав программного обеспечения персональных компьютеров фирмы IBM. Мы разбили интервал [0, 1) на 10 подынтервалов [0, 0.1), [0.1, 0.2), ... , [0.9, 1.0), сгенерировали 50 000 случайных чисел и подсчитали, сколько их лежит в каждом подынтервале. Теоретически в каждом подынтервале их должно быть примерно по 5000. Конечно, их не будет в точности по 5000, но интуиция подсказывает, что доля чисел в каждом подынтервале не должна сильно отличаться от 1/10. Можно использовать специальные статистические тесты, такие, как тест χ^2 [Kennedy et al., 1980], для подтверждения того, что в данном случае результаты согласуются с предположением о равномерности генератора. Рисунок, который показывает число или долю предметов, попадающих в каждое из подмножеств, в виде диаграммы со столбцами, называется *гистограммой*.

Если равномерный генератор порождает целые числа, то они будут равномерно распределены в указанном выше смысле в границах множества машинно представимых неотрицательных целых чисел. Если максимальное представимое целое число обозначить через IMAX (см. разд. 2.1 гл. 2), то генератор будет порождать целые числа из интервала [0, IMAX]. Генератор целых чисел можно использовать и для порождения действительных чисел, например производя деление с плавающей точкой всех целых чисел на IMAX, хотя одна эта операция может потребовать столько же времени, что и весь остальной алгоритм.

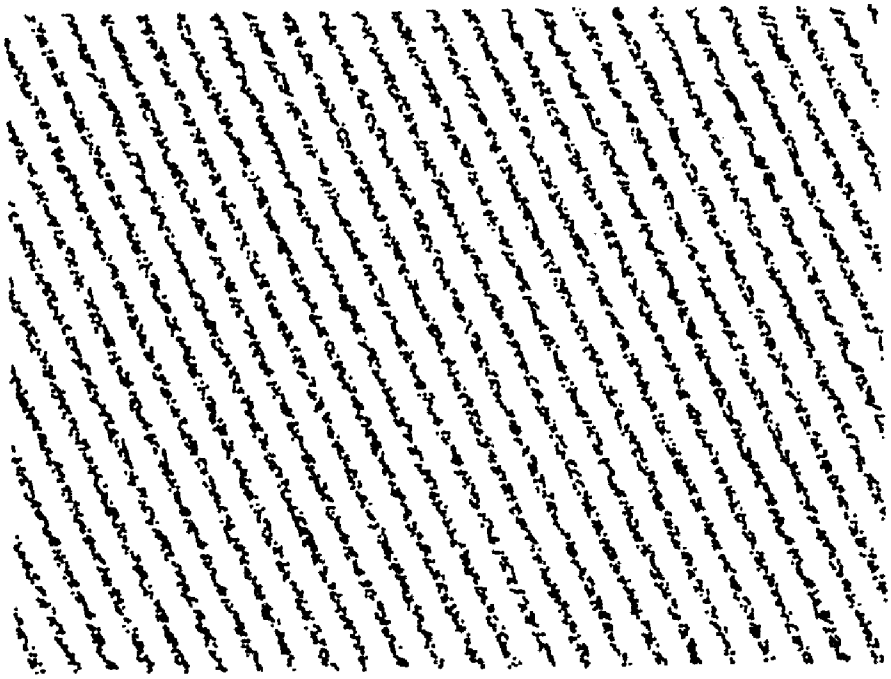


Рис. 10.2. Генератор равномерно распределенных случайных чисел с последовательными корреляциями.

Часто равномерно распределенные целые числа оказываются не менее полезными, чем действительные, но, поскольку диапазон целых чисел на разных компьютерах весьма различен, программа, использующая целые случайные числа, может работать по-разному на разных машинах. Одно время операции с целыми числами производились гораздо быстрее, чем операции с плавающей точкой, что способствовало развитию алгоритмов для целых чисел. Но сегодня это зависит от конкретного компьютера. До середины 1980 гг. генераторы работали с целыми числами. Теперь положение меняется, и наиболее популярными становятся такие генераторы, как UNI, которые используют исключительно операции с плавающей точкой.

Числа, порождаемые генератором, должны быть не только равномерно распределены, но и независимы. Это означает, что в поведении последовательных чисел не должно быть какого-либо различимого шаблона. Если же последовательность не удовлетворяет этому условию, то говорят о наличии *последовательных корреляций*. Известный пример такого генератора дает ранняя версия программы RND. На рис. 10.2 те же самые 50 000 чисел, что и на предыдущем рисунке, представлены в другом виде. Генерируемые числа группируются парами (x_1, x_2), ..., (x_{49999}, X_{50000}), и каждая пара изображается точкой единичного квадрата. Очевиден «полосатый» шаблон в расположении точек, объясняемый наличием последовательных корреляций. Представьте себе эффект использования такого генератора для оценки двумерного интеграла по единичному квадрату (см. § 11 гл. 5), если подынтегральная функция имеет крутой пик в одной из непокрытых областей. В работе [Marsaglia,

1968] показано, что большинство используемых в настоящее время генераторов до некоторой степени страдает последовательными корреляциями. Название этой работы кратко описывает ситуацию: «Случайные числа в основном лежат в плоскостях», но для хорошо разработанных генераторов такой эффект значительно слабее, чем для других. Эта версия RND была настолько плохой, что фирма IBM в более поздних вариантах своего Бейсика заменила ее другой. Однако мы настоятельно рекомендуем вам не использовать RND или какой-либо другой встроенный генератор до тех пор, пока вы не проверите его на модельной задаче из вашей области с известным ответом.

Почти универсальным является метод, который использует для генерации псевдослучайных чисел некоторое начальное, или *зерновое*, значение, обычно целое, и затем вычисляет x_0 в соответствии с некоторым правилом. Далее, $(k+1)$-е число x_{k+1} находится по предыдущим числам. Иногда для получения следующего числа нужно только предыдущее. Вначале правила выбирались интуитивно с таким расчетом, чтобы они были как можно более сложными, путаными и малопонятными, например метод «середины квадрата», в котором x_k возводится в квадрат и в качестве x_{k+1} берется число, образованное фиксированным количеством цифр из середины результата. Но отсутствие теории для этих методов часто приводило к катастрофическим последствиям. В методе середины квадрата в непредсказуемый момент x_k может оказаться равным нулю, и тогда все последующие числа x_i также будут нулями. Этот метод был предложен фон Нейманом, который осознавал его недостатки. Сказанное объясняет, почему уже довольно давно произошел переход к правилам, чьи свойства изучены как можно более полно. Между прочим, индексированное обозначение x_k принято лишь для удобства; генераторы случайных чисел хранят не все порожденные ими числа, а лишь те несколько, которые необходимы для получения следующего.

Еще одной важной характеристикой генератора случайных чисел является его *период p*. Это максимальная длина последовательности $\{x_k\}$ до того, как она начинает повторяться. Независимо от того, является генератор целочисленным или нет, существует лишь конечное число M возможных значений x_k. Если правило вычисления x_{k+1} зависит только от x_k, то стоит x_k вновь принять какое-то из предыдущих значений x_i, как вся последовательность повторится. Таким образом, $p \leqslant M$. Период может быть и гораздо меньше. Если x_{k+1} зависит от x_k и x_{k-1}, то максимальный период равен M^2, хотя отдельные значения x_k могут повторяться и чаще. С помощью теории чисел можно выбрать правило так, чтобы изначально известный период достигал или почти достигал максимального возможного значения. Дальнейшее использование теории чисел может в некоторой степени помочь предсказать характер последовательности и сообщить пользователю известную уверенность в том, что эта последовательность будет достаточно хорошо заменять случайную последовательность чисел.

Неизвестен способ, с помощью которого можно было бы убедиться, что данный генератор удовлетворяет всем возможным критериям случайности. Определение, данное в § 2 (см. [Lehmer, 1951]), является эвристическим. Числа, генерируемые ранней версией RND, для многих приложений оказались неприемлемыми. Теория чисел может помочь в исключении некоторых плохих генераторов и построении хороших, но приобрести репутацию надежного генератор может лишь после того, как пройдет проверку рядом все более жестких тестов.

10.4. Использование случайных чисел: броуновское движение и фракталы

В начале девятнадцатого столетия шотландский ботаник Роберт Броун исследовал каплю воды, которая миллионы лет была заключена в куске кварца. Под микроскопом он обнаружил множество маленьких частичек, находящихся в непрерывном сложном и нерегулярном движении. Наблюдая такое же движение в жидкостях и ранее, Браун объяснял его тем, что «жизнеспособность ⟨молекул растений⟩ сохраняется долгое время после смерти ⟨растения⟩», но происхождение и возраст этой капли исключали любое подобное ботаническое объяснение. Теперь нам известно, что это *броуновское движение* возникает вследствие бомбардировки взвешенной пылинки молекулами жидкости, происходящей случайным образом: путь рандомизируется в результате случайных флуктуаций в скоростях сталкивающихся молекул. Интересно, что в 1905 г. в ходе исследований по атомной теории Эйнштейн писал: «Я обнаружил, что должно иметь место ⟨видимое⟩ движение взвешенных частиц, не зная о том, что наблюдения, относящиеся к броуновскому движению, уже давно известны».

К счастью, для изучения броуновского движения нет необходимости смотреть в микроскоп. Используя генератор случайных чисел, мы можем моделировать движение такой частицы с большой точностью. Рис. 10.3 показывает такой броуновский путь, вычисленный на компьютере, и вы можете проделать то же самое, решая задачу 10.3.

Смоделированное движение еще только начинает приближаться по сложности к истинному пути. Если бы мы сфотографировали реальную частицу, а затем увеличили фотографию в 1000 раз, то смогли бы увидеть эффект бомбардировки меньшими группами молекул. Отрезок реальной траектории, который прежде выглядел прямолинейным, теперь оказался бы столь же изломанным и нерегулярным, как и весь начальный путь. Сотрудник фирмы IBM Мандельброт (см. [Mandelbrot, 1982]) ввел термин «фрактал» для геометрического объекта, который остается самоподобным при любых увеличениях, и броуновское движение было одним из первых естественных явлений, подошедшим под это определение. Конечно, «никакая реальная структура не может сохранять свой внешний вид при бесконечном увеличении. Но тем не менее принцип самоподобия приближенно реализуется в природе: в береговых

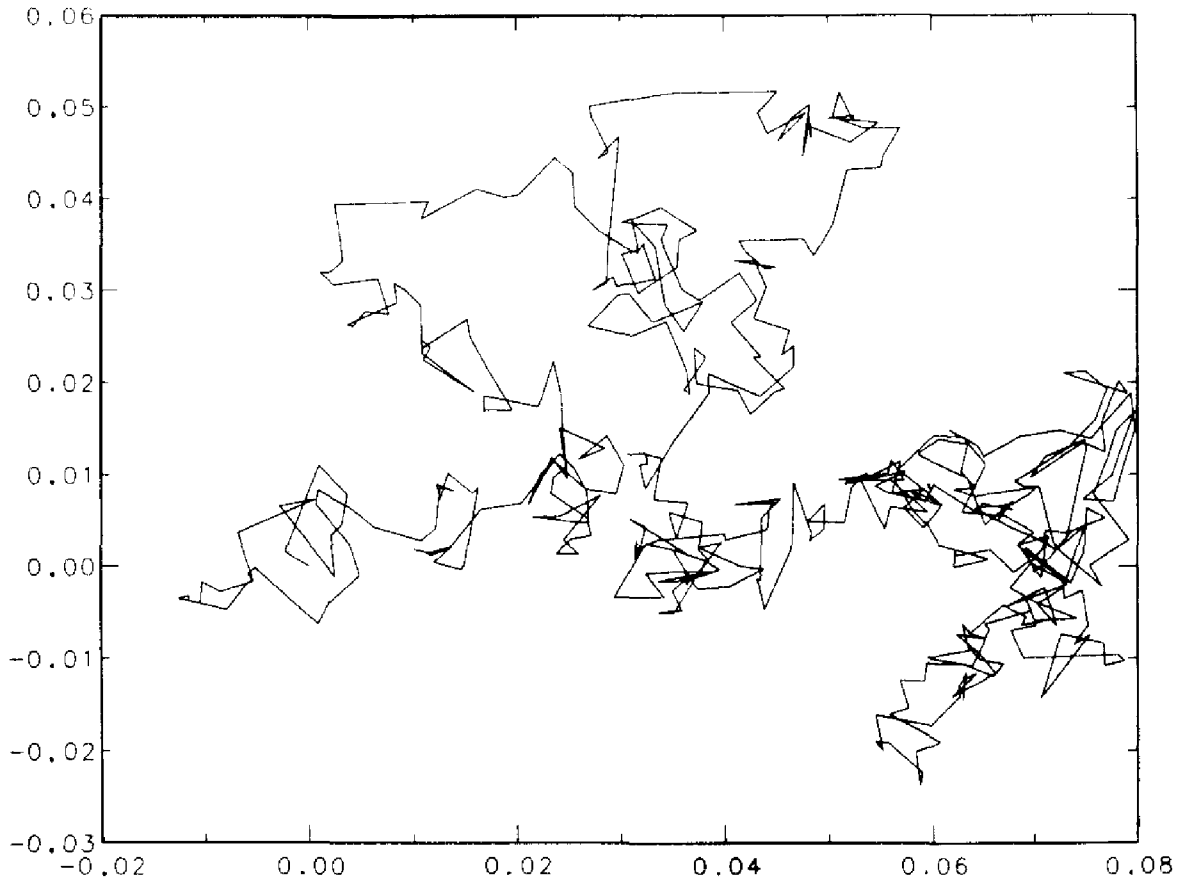


Рис. 10.3. Броуновское движение.

линиях и руслах рек, очертаниях облаков и деревьев, турбулентных потоках жидкостей и иерархических структурах жизнедеятельных систем» [Peitgen, Richter, 1986].

Один из наиболее интересных фактов, относящихся к фракталам, заключается в том, что для их порождения не всегда нужны случайные числа, т.е. они являются детерминированными. Это аналогично полностью воспроизводимым методам, которые мы используем для генерирования псевдослучайных чисел на компьютере. Проиллюстрируем сказанное одним из наиболее известных примеров. Пусть c и z_0 — два *комплексных* числа:

$$c = p + iq, \quad z_0 = x_0 + iy_0 .$$

Рассмотрим итерацию

$$z_{n+1} = z_n^2 + c$$

или ее эквивалентную запись для действительной и мнимой частей:

$$x_{n+1} = x_n^2 - y_n^2 + p, \quad y_{n+1} = 2x_n y_n + q.$$

Меняя конкретные значения c и z_0, мы можем получать различное поведение последовательности. Например, если $c = 0$ и $|z_0| < 1$, то $z_n \to 0$, но если $|z_0| > 1$, то последовательность z_n расходится или, в терминоло-

гии динамики, «сходится к бесконечности». С помощью двух следующих алгоритмов можно получать сложные фигуры необычайной красоты.

(1) Зафиксируем c и будем проводить цикл по различным значениям z_0 в некоторой правильной области, например в единичном квадрате. Для каждого z_0 произведем достаточное количество итераций, чтобы понять, сходится ли последовательность к бесконечности. Обычно для этого проверяют, будет ли достигнуто выполнение неравенства $x_n^2 + y_n^2 > R$ (где R – некоторая константа, например $R = 100$) за фиксированное число N итераций, скажем 100. Если последовательность сходится к бесконечности в этом смысле, то точка $z_0 = (x_0, y_0)$ изображается на графическом устройстве цветной, а в противном случае черной. Обычно цвета кодируются в соответствии со скоростью сходимости к бесконечности: более насыщенный цвет означает, что неравенство $x_n^2 + y_n^2 > R$ выполняется раньше, чем для цвета менее насыщенного. Граница, отделяющая область сходимости к бесконечности от других точек, называется *множеством Жюлиа*.

(2) Зафиксируем $z_0 = 0$ и повторим (1) для ряда значений c, возведенных в квадрат. Граница соответствующей области называется *множеством Мандельброта*.

В типичных компьютерных экспериментах квадрат в (1) и (2) ассоциируется с графическим монитором. Если ваш компьютер снабжен

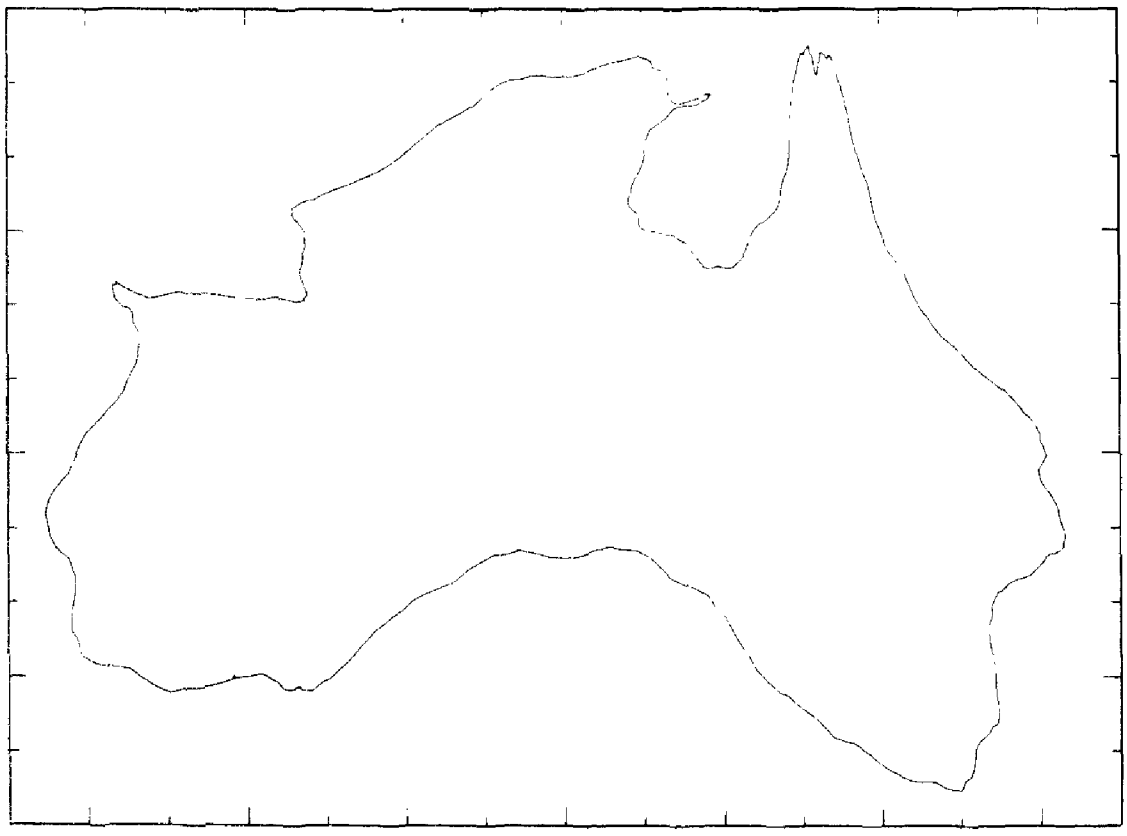


Рис. 10.4. Фрактальная береговая линия.

программами для окрашивания точки с координатами (I, J) в цвет K, то для реализации обоих алгоритмов достаточно всего нескольких строк текста, но результат может получиться очень эффектным (см. задачу 10.12).

Фракталы (с некоторой рандомизацией) нашли приложение в компьютерной графике и мультипликации в качестве способа описания контуров воображаемых, но очень реалистичных горных цепей и береговых линий. Рис. 10.4 показывает фрактальную береговую линию. Для того чтобы изображение было достаточно подробным, генератор случайных чисел должен обладать очень большим периодом, не иметь существенного смещения или корреляций и работать максимально быстро. Вы можете проверить все это сами, решив задачу 10.4.

Обычно алгоритмическое описание фрактала бывает кратким, хотя результирующая фигура может оказаться очень сложной. Например, алгоритм получения множества Жюлиа задается точным указанием всего нескольких чисел. Была проделана определенная работа для проверки наличия обратной связи. Верно ли, что всякую сложную фигуру, например фотоизображение, можно описать как фрактал? Этот факт мог бы найти применение при сжатии образов (см. [Barnsley, Sloan, 1988]). Простой эксперимент для читателей предлагается в задаче 10.13.

10.5. Конгруэнтные генераторы и генераторы Фибоначчи

Одним из наиболее часто используемых целочисленных генераторов является *конгруэнтный* генератор, который был предложен математиком Лемером. В нем используется следующий алгоритм:

$$x_k = (ax_{k-1} + c) (\mathrm{mod}\, M), \qquad x_0, a, c < M \leqslant \mathrm{MAX},$$

где x_0, a, c – заданные целые числа. Функция mod в записи $\alpha (\mathrm{mod}\, \beta)$ определяется как остаток от деления α на β. Например, $15 (\mathrm{mod}\, 4) = 3$. В качестве M берется некоторое большое целое число. Следующий простой физический эксперимент подсказывает, почему такой генератор может быть полезен. Установим фотокамеру над бильярдным столом и будем делать снимки через фиксированные интервалы времени, например каждые 5 с. Шар ударяется и отскакивает от бортов без трения. Позиция шара каждые 5 с полностью определена, но последовательность фотографий придает этим позициям случайный вид, особенно если не показаны бортики стола. Функция mod аналогична этим бортикам.

В зависимости от выбора a и c генератор может оказаться хорошим или плохим. Например, выбор $a = c = 1$ очевидно плох. Некоторые другие значения, с первого взгляда не столь плохие, могут привести к последовательным корреляциям, описанным в § 3. Некоторые значения a, c и M порождают приемлемые случайные числа. Обычно используют значения $a = 69069$, $c = 0$, $M = 2^{32}$ или $M = 2^{31}$. В книге [Knuth, 1969] изложены сведения из теории чисел, необходимые для правильно-

го выбора a и c. Хорошая реализация генератора такого типа может дать почти M элементов, прежде чем встретится первое повторение. Также очень хорошими являются и другие статистические характеристики. Конгруэнтные генераторы идеальны для многих процессов моделирования. Так как эти генераторы изучаются уже долгое время, вы можете найти их в большинстве стандартных статистических пакетов. Но более мощные и быстрые компьютеры позволяют нам браться за бо́льшие задачи, требующие более длинных циклов и еще лучших статистических свойств.

Для порождения случайных чисел с плавающей точкой из интервала [0,1) мы предпочитаем генераторы *Фибоначчи*. Они вычисляют новый элемент как разность, сумму или произведение двух ранее порожденных элементов. Например, в подпрограмме UNI (см. § 6) применяется генератор Фибоначчи вида

$$x_k = x_{k-17} - x_{k-5}.$$

Значения членов последовательности зависят от начальных семнадцати x_i и стратегии, которую мы примем в отношении ситуации $x_i < 0$. Стандартным решением относительно таких x_i является правило $x_i + 1 \to x_i$; тогда все элементы будут принадлежать единичному интервалу. Поскольку x_k зависит от 17 предыдущих значений, теоретически период p может быть очень велик: $p = (2^{17} - 1)2^n$, где n – число битов в дробной части x_i, т. е. в мантиссе со знаком.

Для стандартной (в смысле IEEE) 32-битовой арифметики с плавающей точкой $n = 24$; таким образом, p примерно равно $2^{17} \cdot 2^{24} \approx 10^{12}$.

Говорят, что данный генератор имеет *лаги* 17 и 5. Генераторы Фибоначчи с лагом 1 давно изучены и вышли из употребления, поскольку имеют очень высокую степень последовательной корреляции. Сравнительно недавно были вновь открыты генераторы с большими лагами. Они чрезвычайно эффективны и имеют отличные статистические свойства. В настоящее время генераторы Фибоначчи представляются наилучшим выбором для приложений, требующих интенсивного использования случайных чисел, даже несмотря на то, что для них нужно больше памяти, чем для конгруэнтных генераторов.

При использовании генератора Фибоначчи, требующего более трех или четырех предыдущих значений, эти значения обычно помещают в небольшой массив, края которого как бы склеены. Две целые переменные I и J используются для указания тех элементов, которые должны вычитаться. Например, подпрограмма UNI использует 17-элементный действительный массив U(1), ..., U(17), причем начальные значения I и J равны соответственно 17 и 5:

```
UNI = U(I) – U(J)
IF(UNI.LT.0.0) UNI = UNI + 1
U(I) = UNI
I = I – 1
J = J – 1
```

```
IF(I.EQ.0)I = 17
IF(J.EQ.0)J = 17
```

Такой генератор порождает числа, которые удовлетворяют почти всем известным тестам на случайность, однако и его можно улучшить, например увеличив значения лагов. На некоторых суперкомпьютерах используются генераторы Фибоначчи с лагами 97 и 33. Кроме того, обнаружено, что если скомбинировать два хороших генератора, то комбинация будет лучше, чем каждый генератор в отдельности. В подпрограмме UNI значение u на выходе генератора Фибоначчи комбинируется с выходным значением v конгруэнтного генератора с плавающей точкой и полагается UNI $= u - v$ или UNI $= u - v + 1.0$ в зависимости от знака разности $u - v$. Такая комбинация выдерживает все известные статистические тесты и имеет период, равный произведению периодов каждого из генераторов, т.е. $(2^{17} - 1)2^{48} \approx 10^{19}$.

Чтобы начать последовательность, необходимо сгенерировать 17 «случайных» значений в интервале [0, 1). В UNI это происходит следующим образом: каждый элемент начального массива представляется в двоичной форме:

$$s = \frac{s_1}{2} + \frac{s_2}{2^2} + \ldots + \frac{s_m}{2^m}, \quad s_i = 0 \text{ или } 1, \; m \leqslant n.$$

Алгоритм инициализации задает случайным образом значение 0 или 1 для каждого бита s_i; то или иное значение выбирается в соответствии с тем, больше или меньше нуля число на выходе простого встроенного целочисленного генератора. Единственное, что должен сделать пользователь,– ввести начальное значение для этого последнего генератора.

Если параметр m в выражении для s равен n (числу битов в мантиссе числа с плавающей точкой), то этот генератор будет порождать различные последовательности при работе с одинаковыми входными данными на компьютерах с разной длиной слова, скажем на суперкомпьютере Cray и на персональном компьютере IBM. На каждом из этих компьютеров алгоритм имеет период $(2^{17} - 1)2^m$. Если m – меньшая из двух длин мантиссы ($m = 24$), то будут получаться одинаковые последовательности. В UNI $m = 24$ из соображений переносимости, но это легко можно изменить.

10.6. Функция UNI

Использование UNI требует одного начального вычисления USTART(ISEED) с целым входным числом. USTART переводит заданное целое число в представление с плавающей точкой. Пользователь может случайно забыть инициализировать генератор, поэтому внутренний массив U содержит значения, разумно определенные по умолчанию. В последующем каждое вычисление UNI( ) без аргумента дает число из равномерного распределения на [0, 1). Подпрограмма USTART факти-

чески является «входом» в UNI, поэтому здесь, по существу, одна подпрограмма, а не две. Фортран-77 допускает функции, не имеющие аргументов. Поскольку инициализация производится другим входом, задача UNI сужается до единственной функции, и отсутствие аргументов обеспечивает наиболее эффективный тип связи между UNI и программой, которая к ней обращается.

```
C  Пример типичного использования подпрограммы
C
      REAL U, UNI, USTART, USEED
      INTEGER ISEED, I
C  Задать начальное значение
      ISEED = 305
      USEED = USTART (ISEED)
C  USTART переводит начальное значение в число с плавающей точкой
      WRITE(*,*) ISEED, USEED
      DO 1 I = 1, 1000
            U = UNI( )
1     CONTINUE
      WRITE(*,*)U
      END
C
C  Будут получены следующие результаты
C            305        305.000
C              0.157039
C
```

10.7. Выборка из других распределений

Часто генерирование случайного числа из интервала [0, 1) является просто средством принятия некоторого случайного решения. В качестве примера предположим, что в некоторой программе моделирования желательно при случайном выборе попадать на какую-то ветвь один раз из десяти. Этого легко добиться, выбирая случайное число y из равномерного распределения на [0, 1) и беря данную ветвь, если $y < 0.1$. Более быстрым способом является выбор случайного целого числа x из интервала [0, IMAX) и переход на данную ветвь, если $x <$ IMAX/10. Таким же образом из равномерно распределенных случайных целых чисел можно получить любое конечное распределение с различными весами.

Часто бывает необходимо сгенерировать последовательность случайных чисел, равномерно распределенных на данном конечном интервале $[a, b)$. Если числа u_i генерируются, скажем, подпрограммой UNI и равномерно распределены на [0, 1), то числа $(b-a)u_i + a$ будут равномерно распределены на $[a, b)$.

Гораздо сложнее получить случайные числа с неравномерным непрерывным распределением, например с нормальным. Если $f(x)$ – плотность

распределения (в случае нормального распределения это хорошо известная колоколообразная кривая), то *функция распределения* определяется следующим образом:

$$F(x) = \int_{-\infty}^{x} f(t)\,dt.$$

Для нормального распределения $F(x)$ имеет вид S-образной кривой, которая стремится к 0 при $x \to -\infty$ и к 1 при $x \to \infty$. Общий метод получения чисел с распределением такого вида заключается в нахождении y, равномерно распределенного на [0, 1), и определении единственного x, для которого $F(x) = y$ (рис. 10.5). Это можно пояснить следующим образом. Представим, что мы выбираем y многократно; каждому y мы ставим в соответствие единственное x. Выбору 50 000 игреков соответствует выбор 50 000 иксов. Таким образом, доля выбранных y, лежащих между двумя фиксированными значениями, скажем y_1 и y_2, в точности равна доле x, лежащих в интервале $[x_1, x_2]$. Но вероятность первого из названных событий равна $|y_2 - y_1|$, если y распределено равномерно, следовательно, верна цепочка равенств

$$\begin{aligned}
&\text{доля чисел в интервале } [x_1, x_2] = \\
&\quad = \text{доля чисел в интервале } [y_1, y_2] = y_2 - y_1 = \\
&\quad = F(x_2) - F(x_1) = \int_{-\infty}^{x_2} f(t)\,dt - \int_{-\infty}^{x_1} f(t)\,dt = \\
&\quad = \int_{x_1}^{x_2} f(t)\,dt,
\end{aligned}$$

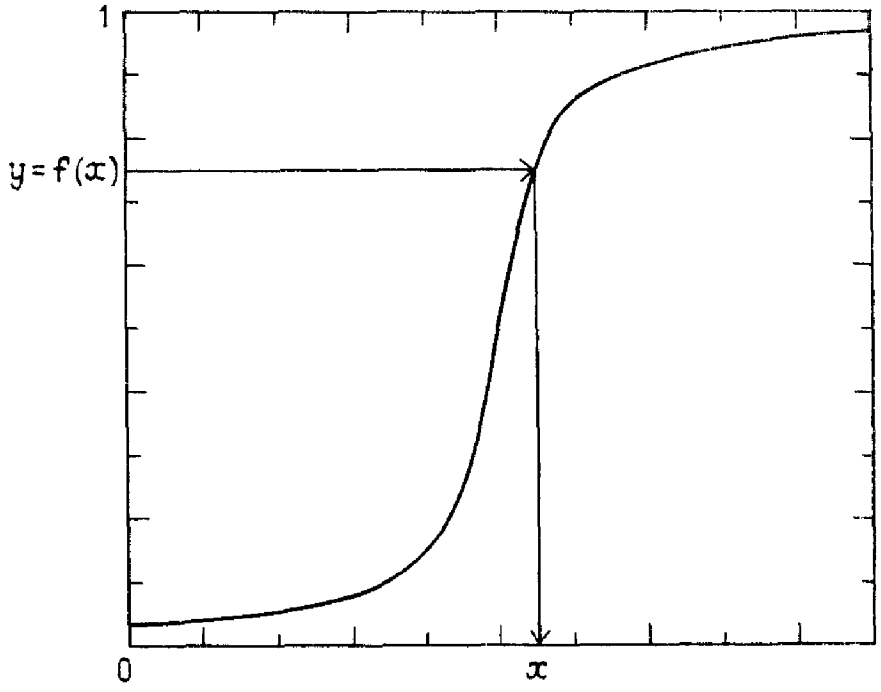


Рис. 10.5. Получение случайных чисел с распределением общего вида.

которая и показывает, что в случае равномерного распределения игреков x имеет распределение с плотностью $f(t)$. Этот метод эквивалентен вычислению обратной функции $F^{-1}(x)$. Сложной проблемой при этом является достаточно быстрое и точное формирование $F^{-1}(y)$.

Для некоторых часто встречающихся распределений развита специальная техника обращения функций. Все эти методы опираются на использование быстрых и точных равномерных генераторов. Обзор методов, используемых во многих генераторах, можно найти в книгах [Kennedy, Gentle, 1980] и [Rubinstein, 1981]. В качестве примера рассмотрим *экспоненциальное* распределение с параметром $\lambda > 0$. Это распределение часто используется в задачах массового обслуживания для моделирования интервалов между прибытиями и времен обслуживания. Экспоненциальное распределение имеет плотность

$$f(t) = \lambda \exp(-\lambda t), \quad t > 0,$$

и функцию распределения

$$F(x) = \int_0^x \lambda \exp(-\lambda t)\, dt = 1 - \exp(-\lambda x).$$

Для этого распределения легко получить $F^{-1}(y)$, т. е. разрешить уравнение $F(x) = y$. Решение имеет вид

$$x = -\frac{\ln(1-y)}{\lambda}.$$

Для получения x с искомым распределением нужно сгенерировать y, равномерно распределенное на $[0, 1)$, и применить эту формулу. Если говорить о практической стороне дела, то существуют более эффективные способы, в которых не используется медленная операция вычисления логарифма для каждого случайного числа.

Экспоненциальное распределение можно использовать для моделирования работы парикмахерской, описанной в § 1. Если среднее время выполнения стрижки парикмахером равно двадцати минутам и он может сделать сто стрижек в неделю, то время выполнения стрижки этим парикмахером можно моделировать экспоненциальным распределением с параметром $\lambda = 1/20$. Используя подпрограмму UNI для получения равномерно распределенных чисел и приведенную выше формулу для вычисления x, мы можем получить список времен для ста стрижек. На рис. 10.6 показана гистограмма для этих значений. Как вы можете видеть, бóльшая часть стрижек требует не более восемнадцати минут; одного привередливого клиента стригут в течение полутора часов.

В отличие от экспоненциального распределения, для нормального распределения трудно вычислить $F^{-1}(y)$. Вы уже знаете из задач, которые рассматривались в предыдущих главах, что невозможно получить явную формулу для решения уравнения $F^{-1}(y) = x$. Прямолинейные численные методы могут оказаться медленными. Алгоритм, при-

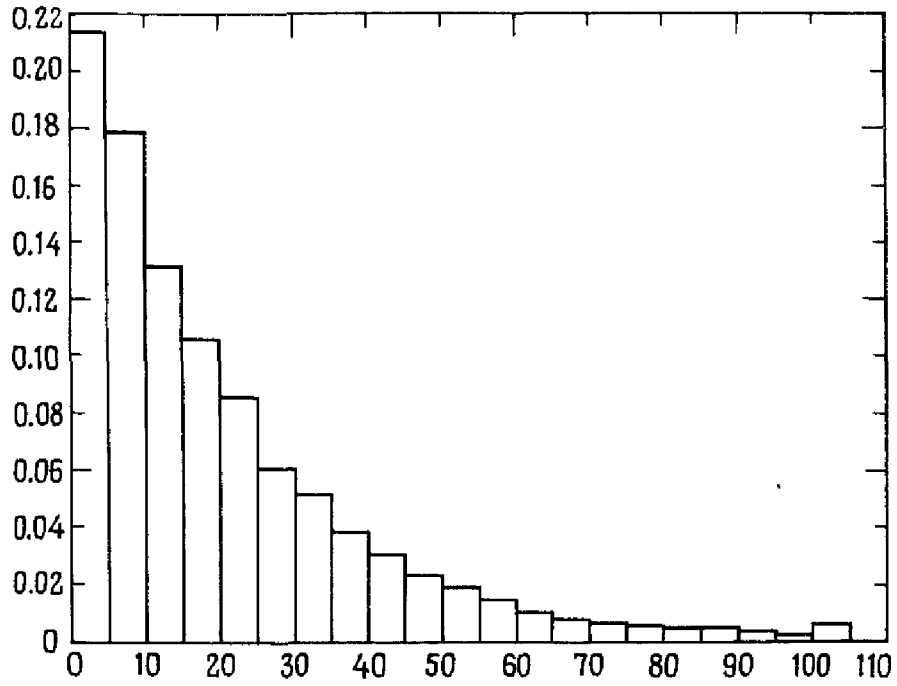


Рис. 10.6. Гистограмма экспоненциального распределения.

надлежащий Марсалье, генерирует пару чисел, скажем u и v, равномерно распределенных на $[-1, 1)$, и вычисляет $r^2 = u^2 + v^2$. Если $r > 1$, то u и v отбрасываются и генерируется новая пара. При этом мы теряем 22% эффективности. Если $r \leqslant 1$, то вычисляется $w = \sqrt{-2(\ln r)/r}$. Теперь можно показать, что $x = uw$ и $y = vw$ нормально распределены со средним $\mu = 0$ и дисперсией $\sigma^2 = 1$. Это распределение обозначается через $N(\mu, \sigma^2)$. Наиболее медленная часть алгоритма – это вычисление логарифма, но если скорость не очень важна, то рекомендуется применять именно этот метод, поскольку он очень прост.

Иногда бывает необходимо генерировать нормально распределенные случайные числа со средним и стандартным отклонением, отличными от 0 и 1 соответственно. В этом случае, если r_i принадлежит $N(0, 1)$ и получено, например, на выходе программы RNOR, обсуждаемой ниже, то $r_i\sigma + \mu$ принадлежит $N(\mu, \sigma^2)$. Таким образом, с помощью RNOR можно генерировать любые нормально распределенные числа.

10.8. Функция RNOR

Функция, которую мы применяем для генерирования чисел со стандартным нормальным распределением $N(0, 1)$, основана на методе, впервые описанном в работе [Marsaglia, 1984]. Этот метод использует массив заранее вычисленных значений, которые разбивают нормальное распределение на части, равные 1/32 его размаха, и является очень быстрым. Для большей части вычислений RNOR( ) требуется только одно равномерно распределенное число и не требуются ни логарифмы, ни квадратные корни. Для некоторых вычислений могут потребоваться

несколько равномерно распределенных чисел и другие более сложные вычисления, но такое происходит нечасто. Чтобы еще больше увеличить скорость вычислений, равномерно распределенные числа генерируются датчиком типа UNI, встроенным в тело RNOR, а не путем явного обращения к подпрограмме UNI( ). Так же как и UNI, подпрограмма RNOR требует инициирующего вычисления RSTART (ISEED) с начальным значением ISEED, после чего обращения к RNOR не требуют задания аргумента.

```
C     Пример типичного использования подпрограммы RNOR
        REAL R, RNOR, RSTART, RSEED
        INTEGER ISEED, I
C     Задать начальное значение
        ISEED = 305
        RSEED = RSTART (ISEED)
C     RSTART переводит начальное значение в число с плавающей
C     точкой
        WRITE (*,*) ISEED, RSEED
        DO 1 I = 1,3
            R = RNOR( )
            WRITE (*,*) R
      1 CONTINUE
        END
C
C     Будут получены следующие результаты (для разных компьютеров
C     они могут быть различными)
C
C                305        305.00
C                  0.335785
C                  0.230143
C                  1.02517
```

Пример 10.1. Гистограмма нормального распределения.

Используя RNOR, получим 10000 значений и сгруппируем их в 32 интервалах $(-\infty, -3.0]$, $(-3.0, -2.8]$, $(-2.8, -2.6]$, ..., $(2.8, 3.0]$, $(3.0, \infty)$, а затем подсчитаем число и долю значений, находящихся в каждом интервале. Программа, приводимая ниже, показывает, как это сделать. Графическое изображение долей приводит к гистограмме, представленной на рис. 10.7. На этом рисунке показана также «кумулятивная гистограмма», которая получается из первой суммированием находящихся слева долей. Вид гистограммы существенно зависит от числа подынтервалов. Если мы решим использовать 5000 подынтервалов, то гистограмма будет выглядеть хаотической, так как многие интервалы не будут содержать ни одного числа. Кумулятивная гистограмма менее чувствительна к числу подынтервалов.

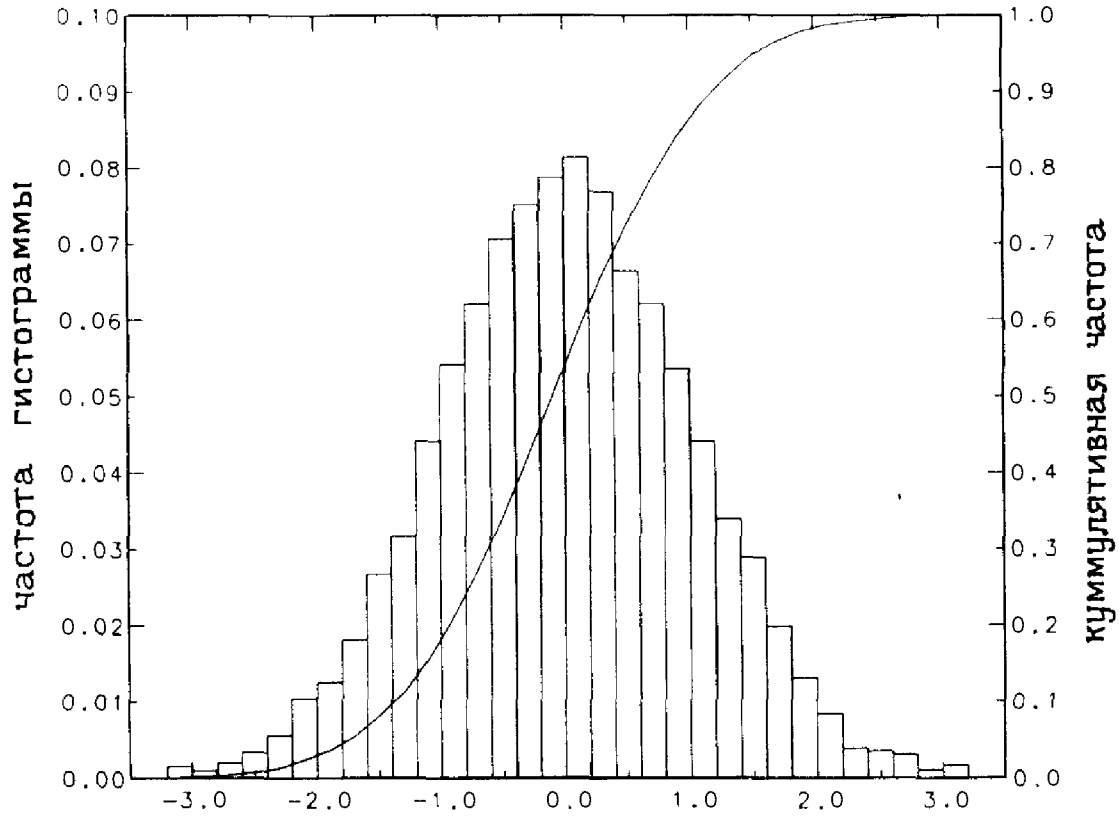


Рис. 10.7.

```
C       Гистограмма для подпрограммы RNOR
        PARAMETER(NBINS = 32, A = -3.0, B = 3.0)
        INTEGER ISEED, I, J, H(NBINS), NR, INBIN
        REAL R, RNOR, RSTART, RSEED, WIDTH
C
        ISEED = 305
        RSEED = RSTART(ISEED)
        WIDTH = (B - A)/(NBINS - 2)
100     WRITE (*,*) 'Задача 10.1: ввести число нормально распре-
        деленных значений'
        DO 200 I = 1, NBINS
            H(I) = 0
200     CONTINUE
        READ(*,*) NR
        IF (NR.LE.0) STOP
        DO1 I = 1,NR
            R = RNOR( )
            J = INBIN(A,B,NBINS,WIDTH,R)
            H(J) = H(J) + 1
1       CONTINUE
C
        WRITE (*,*) 'Гистограмма для RNOR: числа в интервалах
        1, ..., 32'
```

```
      WRITE (*,*) '(-INFINITY, -3], (-3, -2.8], ..., (2.8, 3], (3,
     INFINITY)'
      WRITE (*,*) 'Значения в некоторой степени зависят от ком-
     пьютера'
      WRITE (*,' (918)') (H(I), I = 1, NBINS)
      WRITE (*,*)
      GO TO 100
      END
C
      INTEGER FUNCTION INBIN (XMIN, XMAX, NBINS,
     WIDTH, DATA)
C
C     Эта функция находит правильный интервал для вещественного
C     параметра DATA. Значения, меньшие XMIN, относятся к ин-
C     тервалу 1.
C     Значения, бóльшие XMAX, относятся к интервалу NBINS.
C
      INTEGER NBINS
      REAL XMIN, XMAX, WIDTH, DATA
      IF (DATA.LT.XMIN) THEN
         INBIN = 1
      ELSE IF (DATA.GE.XMAX) THEN
         INBIN = NBINS
      ELSE
         INBIN = 2 + (DATA-XMIN)/WIDTH
      ENDIF
      RETURN
      END
```

* 10.9. Пример: радиационная защита и критичность реактора

Наиболее ранним крупномасштабным приложением метода Монте-Карло было его применение к изучению движения незаряженных частиц в некоторой среде. (Незаряженные частицы между столкновениями движутся по отрезкам прямых.) В любой заданный момент нейтрон имеет определенное местоположение, направление движения и энергию. Если он не сталкивается в среде с атомом, то продолжает свой путь в том же направлении и с той же энергией. Однако в каждой точке существует вероятность столкновения; при прохождении пути длины δs вероятность столкновения равна $\sigma_c \delta s$. Величина σ_c называется «сечением» и обычно зависит от энергии частицы и свойств среды. Для определения этой величины используются эксперименты и теория, и для точного описания сложной среды нужны обширные таблицы. Хотя обычно сечение при данном значении энергии постоянно для любого материала, оно может резко меняться при переходе от одной области

к другой. Реакторы состоят из секций с различными сечениями, а именно из урановых стержней, погруженных в воду, труб, стенок, свинцовых щитов и т. д. Обычно предполагается, что величина пути, проходимого частицей между столкновениями, задается экспоненциальным распределением с параметром $\lambda = \sigma_c$. Если путь между двумя столкновениями не пересекает границу сред, то столкновения не запрещаются; в противном случае частица продолжает движение только до границы. Если эта граница наружная, то говорят, что частица испускается. (В программах такого типа больше всего времени тратится на интерполирование в таблицах при вычислении сечения в данной точке и при данном значении энергии, а также на решение уравнений, определяющих, пересекла ли частица границу среды.) Если частица испытывает столкновение, то она может быть поглощена средой, и тогда ее жизненный цикл заканчивается. В этом случае энергия обращается в тепло; в случае большого выделения тепла система будет «таять». В противном случае частица может претерпеть рассеяние, т. е. покинуть точку столкновения с новым направлением и новым значением энергии, или деление, когда точку столкновения покидают несколько новых частиц с различными значениями энергии. Каждое из этих событий происходит с определенной вероятностью.

Интерес представляют две задачи:

(1) Какая часть нейтронов будет испускаться и какова их средняя энергия? Это задача радиационной защиты.

(2) Растет ли общее число нейтронов, т. е. превышает ли число нейтронов, порождаемых делением, число испускаемых электронов? Это проблема «критичности» реактора.

Более подробное рассмотрение этих вопросов выходит далеко за пределы данного учебника, однако задача 10.11 даст вам представление о том, с какими проблемами сталкиваются конструкторы реакторов. Книги [Carter, 1975] и [Bell, Glasstone, 1970] являются популярными справочниками в этой области.

10.10. Задачи

10.1. Если x – случайная величина с нормальным распределением $N(0, 1)$, а z – некоторая константа, то вероятность того, что x меньше z, обозначается через $P[x < z]$ и может быть вычислена по формуле

$$P[x < z] = \frac{1}{2\pi} \int_{-\infty}^{z} \exp(-t^2/2)\, dt = \frac{1}{2}\left(1 + \operatorname{erf}\frac{z}{\sqrt{2}}\right).$$

Напишите программу HIST (VAL, H, N, HX), которая поможет вам в формировании гистограммы. Входной массив HX должен содержать N возрастающих абсцисс; если значение входного параметра VAL лежит между HX(I – 1) и HX(I), то программа HIST должна увеличивать значение H(I – 1) на 1, I = 2, ..., N. Пусть M – некоторое большое целое

число, скажем 10 000. Сгенерируйте M случайных чисел, принадлежащих $N(0, 1)$, используя RNOR, а затем с помощью HIST постройте гистограмму на 32 подынтервалах, как в примере 10.1. Пусть z принимает значения от -3.0 до 3.0 с шагом 0.2. Для каждого z выведите на печать и сравните две величины: (число $x_i < z)/M$ и $\frac{1}{2}(1 + \text{erf}(z/\sqrt{2}))$. Найти значения функции erf можно с помощью любого из методов, описанных в предыдущих главах нашего учебника. Попытайтесь поэкспериментировать несколько раз с различными начальными значениями для RNOR и сравните результаты. Насколько согласованны результаты разных экспериментов?

10.2. Используя равномерный генератор UNI, получите 10 000 чисел.

(а) Используя подпрограмму HIST из задачи 10.1, подсчитайте количество чисел, попадающих в 10 одинаковых подынтервалов отрезка [0, 1]. Найдите в книгах по прикладной статистике критерий χ^2 и примените его для проверки того, действительно ли сгенерированные числа распределены равномерно.

(б) Сформируйте 5000 пар из чисел, полученных в пункте (а). Используя программную графику вашего компьютера, изобразите эти пары как точки единичного квадрата. Замечаете ли вы какую-нибудь закономерность в расположении точек?

10.3. Используя RNOR, получите две последовательности случайных чисел a_i и b_i с нормальным распределением $N(0, s)$. Пусть $x_1 = y_1 = 0$,

$$x_i = x_{i-1} + a_i, \qquad y_i = y_{i-1} + b_i.$$

Нанесите каждую точку (x_i, y_i) на график и соедините ее с предыдущей отрезком прямой линии, чтобы получить «броуновское движение».

Поэкспериментируйте с различными значениями s, стараясь удержать график на расстоянии, не превышающем единицы, от начала координат.

10.4. Карту Австралии наложили на координатную сетку и измерили координаты следующих 13 точек на побережье: (0,0), (20,7.4), (38.7,–4.8), (47,4), (50,13.5), (41.5,27), (36,40.5), (32,30), (25,34.5), (26.8,39), (9,34.5), (2,25.5), (–7.5,20). Если требуется точное изображение побережья, например для топографических целей, то нужно взять значительно больше точек. Однако для некоторых приложений, таких, как мультипликационные фильмы, достаточно, чтобы побережье напоминало очертания какой-то страны, и не важно, какой именно. Основная идея заключается в том, что при взгляде издалека тринадцати точек, соединенных прямыми линиями, достаточно для того, чтобы изобразить побережье, но по мере приближения наблюдателя изображение должно становиться более подробным, т.е. нужно все больше точек. Исследуйте описываемый ниже метод, который генерирует «фрактальную» береговую линию, содержащую заданные точки, а также произвольное число других точек, получаемых техникой рандомизации. Таким образом можно определить

столько точек, сколько необходимо при данном масштабе обзора. Единственным критерием является «правдоподобный вид» изображения, даже если оно не точно. Мы используем алгоритм из задачи 4.11

$$\mathbf{p}_{i+1/2} = (1/2 + w)(\mathbf{p}_i + \mathbf{p}_{i+1}) - w(\mathbf{p}_{i-1} + \mathbf{p}_{i+2}),$$

который порождает новую точку между $\mathbf{p}_i$ и $\mathbf{p}_{i+1}$, причем выбираем w не как константу, а как нормально распределенную случайную величину со средним μ и дисперсией σ^2, т. е. с распределением $N(\mu, \sigma^2)$. Поэкспериментируйте с различными значениями μ и σ. Разумно выбирать эти значения с соблюдением условий $0 \leqslant \mu \leqslant 0.25$ и $\sigma \approx \mu^2$, хотя и при $\mu \leqslant 0.333$ может получиться интересный результат. Изобразите некоторые из ваших береговых линий (см. работу [Fournier et al., 1982]).

10.5. Генерирование точек внутри круга.

(а) Используйте UNI для получения $N = 10\,000$ пар точек X и Y, равномерно распределенных в интервале $[-1, 1]$. Изобразите (X, Y) как точку в единичном круге, если $X*X + Y*Y < 1$, в противном случае отбростье эту пару и получите новую. Какова эффективность этого метода, т. е. какая часть получающихся пар попадает внутрь круга? Если можете, изобразите точки в круге. Равномерно ли они распределены?

(б) Используйте аналогичный метод для получения точек внутри единичной сферы. Какова эффективность этого метода?

(в) Модифицируйте метод, предложенный в пункте (а), следующим образом: выберите число X из равномерного распределения на интервале $[-1, 1]$, а затем (с помощью UNI) число Y из равномерного распределения на интервале $[-P, P]$, где $P = \sqrt{1 - X^2}$. Этот метод имеет эффективность 100%, но точки (X, Y) распределены в круге не равномерно. Продемонстрируйте это, построив изображения точек. Можете ли вы дать объяснение этому факту?

(г) Вернемся к пункту (а), но не будем отбрасывать точку автоматически, если она выходит за пределы круга. Рассмотрим единичный круг как часть единичного квадрата. В каждом углу квадрата можно поместить четверть круга с радиусом $\sqrt{2} - 1$. Если пара (X, Y) попадает внутрь одной из этих четвертей, то переведем эту точку в единичный круг путем переноса и масштабирования. Измените вашу программу, используя этот алгоритм. Какой процент точек теперь попадает в круг? Какой из алгоритмов, по вашему мнению, позволяет быстрее получить 10 000 точек в круге?

10.6. Используя программу UNI и метод, описанный в § 7, получите гистограмму 1000 точек, принадлежащих экспоненциальному распределению с параметром $\lambda = 1$. Сравните площадь каждой секции гистограммы с соответствующей площадью под функцией плотности.

10.7. Две группы стрелков стоят друг перед другом по разные стороны водного рубежа; в группе «красных» – 100 лучников, в группе «белых» – 45. Каждую минуту один лучник стреляет. Стрела летит от «красных» или «белых» в соответствии с пропорцией оставшихся лучников. Так, сначала «красные» стреляют $100/(100 + 45) * 100\%$ всего времени.

(а) Если обе группы всегда попадают в противника, то «красные» почти всегда побеждают «белых». Каково среднее число лучников, остающихся у «красных» после истребления всех «белых»?

(б) Командир группы «белых» осознает необходимость затруднить действия «красных», а поэтому размещает свою группу так, что «красные» теперь попадают в «белых» с некоторой вероятностью, скажем $0 \leqslant p \leqslant 1$. Какое приблизительно значение должно иметь p, чтобы «белые» истребляли «красных» в среднем в половине всех боев? Решения задач этого типа в замкнутой форме можно получить с помощью уравнений Ланчестера.

10.8. Задача парковки. Автомобили, имеющие единичную длину, пытаются случайным образом припарковаться на улице длиной $N = 10$ единиц. Машина может парковаться, если по каждую сторону от ее центра есть свободное пространство, не меньшее половины единицы, другими словами, не нужно дополнительного места для маневрирования. Какое среднее число автомобилей может припарковаться? Для получения среднего результата вам нужно смоделировать эту ситуацию много раз. (Ответ: $0.7459N$. В двумерном случае, когда автомобили рассматриваются как квадраты со стороной единица, а парковка происходит на стоянке с размерами $N \times N$, эта задача не решена.)

10.9. Простой тест для обезьяны.

(а) Обезьяна, сидящая за пишущей машинкой, случайным образом выстукивает буквы (мы считаем, что на клавиатуре отсутствуют числа, пробелы, знаки препинания и нижний регистр). Известно, что среднее число ударов до появления определенной последовательности из трех символов равно 26^3. Используя UNI, сгенерируйте некоторые последовательности букв и подсчитайте число букв до появления слова CAT. Согласуется ли это число с предсказанием? Попробуйте получить некоторые другие слова, такие, как SEX, DOG, WOW и т. д. (Следует ожидать заметного разброса в подсчетах для разных последовательностей. Если вы располагаете достаточным количеством машинного времени, проведите несколько десятков испытаний и усредните результаты.)

(б) Если вы имеете доступ к другому генератору, например если генератор входит в комплект поставки вашего транслятора с Фортрана, то воспользуйтесь им и сравните результаты.

10.10. Ускоритель генерирует поток атомов H^- (т. е. атомов водорода, слабо связанных с дополнительными электронами). Мы хотели

бы, чтобы эти атомы поразили цель, расположенную на расстоянии 1000 километров от источника. Угол, с которым частица покидает ускоритель, имеет небольшую случайную компоненту. Если бы путь к цели был прямолинейным, то можно было бы ожидать, что обе координаты x и y точки, в которой частица соударяется с мишенью, будут нормально распределены с известными средним и дисперсией.

(а) Постройте картину распределения частиц на плоскости мишени, если координаты x и y принадлежат нормальному распределению $N(0, 1)$. Сделайте это, вызывая программу RNOR 5000 раз и нанося каждую пару на график.

К несчастью, путь заряженной частицы изменяется магнитным полем Земли. Для устранения этой трудности частицы пропускаются через емкость, где под низким давлением находятся нейтральный кислород или азот, которые отбирают электроны у H^- и порождают пучок нейтральных частиц, в дальнейшем движущихся прямолинейно. Однако потеря электрона приводит к угловому отклонению от направления основного луча. В этой задаче мы предполагаем, что положение частицы на плоскости мишени x_1, x_2 зависит от двух факторов – ее направления в момент выхода из ускорителя и рассеивания при прохождении нейтрального газа:

$$x_1 = y_1 + z_1, \qquad x_2 = y_2 + z_2,$$

где (y_1, y_2) отражают эффект воздействия ускорителя, т. е. принадлежат $N(0, 1)$, а (z_1, z_2) отражают дополнительный (независимый) эффект соударений при прохождении газа. (Стандартное отклонение 1.0 от цели соответствует предположению о том, что стандартное отклонение на любом заданном расстоянии d метров от ускорителя равно $10^{-6}d$ и что цель находится на расстоянии 1000 километров от ускорителя.) Что касается z_1 и z_2, то ученые утверждают, что газ отклоняет частицы равномерно по отношению к углу и с радиусом r, который имеет функцию распределения

$$F(r) = 1 - \frac{1}{\sqrt{r^2 + 1}}, \qquad r > 0.$$

Таким образом, z_1 и z_2 можно получить с помощью формул

$$z_1 = r\cos(\theta), \qquad z_2 = r\sin(\theta),$$

где угол θ равномерно распределен на $[0, 2\pi)$. (Иногда это называют распределением лоренцева рассеивания.)

(б) Постройте картину распределения частиц на плоскости мишени, учитывая отклонения, приобретенные в ускорителе и в газе. Значения r можно получать с использованием обратной функции, как описано в § 7.

(в) Постройте гистограмму распределения точек на плоскости мишени как функцию расстояния от центра.

10.11. Эта задача даст вам возможность немного поразмышлять над проблемами, с которыми сталкиваются конструкторы реакторов. Пусть имеется бесконечная однородная пластина толщиной T сантиметров. На одной стороне пластины расположен источник нейтронов. Наша задача – проанализировать нейтроны, покидающие этот источник. Сделаем следующие предположения.

(1) Источник испускает нейтроны только в пластину. Если μ – косинус угла между нормалью к грани пластины и направлением движения нейтрона, то предположим, что μ равномерно распределено на интервале [0, 1]. Таким образом, более вероятно, что нейтроны будут двигаться по нормали, чем перпендикулярно к ней. Это определяет направление движения нейтрона в конусе с вершиной в источнике. Позиция относительно вершины конуса задается азимутальным углом φ, который предполагается равномерно распределенным на интервале $[0, 2\pi]$.

(2) Испускаемый нейтрон имеет энергию E, причем $E_{\min} = 0.001 \leqslant E \leqslant E_{\max} = 2.5$ МэВ (миллион электронвольт). Эта энергия распределена в соответствии с функцией плотности $f(E) = c/\sqrt{E}$. Это означает, что $\int f(E)dE = 1$. Таким образом, нейтрон с большей вероятностью имеет низкую энергию.

(3) Нейтрон проходит расстояние D сантиметров до своего поглощения, испускания или рассеивания. Деления не происходит. Величина D имеет экспоненциальное распределение с параметром σ (сечением) и называется *расстоянием до соударения*.

(4) Если нейтрон изменяет свой статус, то мы прежде всего проверяем, не покинул ли он пластину. Если нет, то мы считаем, что с вероятностью 1/10 он поглощается, и его путь завершается. С вероятностью 9/10 нейтрон рассеивается.

(5) Рассеивание «изотропично», т. е. не существует преимущественных направлений.

(6) После рассеивания нейтрон обычно имеет меньшую энергию. Новое значение его энергии равномерно распределено на интервале $[0.3E, E]$.

Для заданной толщины пластины нужно написать программу, моделирующую большое число испускаемых нейтронов и собирающую некоторую статистику. В частности, нужно найти процент нейтронов, которые поглощаются, проходят сквозь пластину (если начало координат находится в источнике, а положительное направление оси x указывает внутрь пластины, то нейтрон проходит сквозь пластину, если его координата x больше чем T) и отражаются ($x < 0$). В дополнение требуется найти среднюю энергию поглощенных, отраженных и прошедших сквозь пластину частиц и их стандартные отклонения.

(а) Считайте сечение константой $\sigma = 2.0$. Вам поможет программа из файла REACTOR.FOR, записанная на дискете, приложенной к книге. (Вследствие симметрии эта задача, по существу, является одномерной, но расчеты реакторов почти всегда трехмерны, и

программа REACTOR.FOR написана именно для этого случая.)

(б) Более реалистичное (хотя все еще упрощенное) сечение задается в Фортране функцией CROSS, приводимой ниже. Изобразите это сечение как функцию энергии на интервале $[E_{\min}, E_{\max}]$. Используйте эту функцию в вашей программе. Какой приблизительно толщины должна быть пластина, чтобы сквозь нее проходило не более 1% нейтронов? Используйте для решения не меньше 1000 частиц. Для реалистического моделирования нужно по крайней мере 100 000 частиц.

```
      REAL FUNCTION CROSS (E)
C     Вычисляет (воображаемое) сечение как функцию от энергии в
C     интервале [EMIN, EMAX]
      REAL E, S, ABS, SIN, Y, EXP
      S = ABS(SIN(100.*(EXP(E) - 1.0)) + SIN(18.81*(EXP(E) - 1.0)))
      Y = MAX(0.02, S)
      CROSS = 10.0*EXP(-0.1/Y)
      RETURN
      END
```

10.12. Фракталы.

(а) Напишите программу, изображающую множество Жюлиа для итераций

$$z_{n+1} = z_n^2 + c,$$

где $c = 0.27334 + 0.00742i$. Организуйте цикл по точкам области $-1 \leqslant \mathrm{Real}(z_0) \leqslant 1$, $-1.3 \leqslant \mathrm{Imag}(z_0) \leqslant 1.3$. Объем вычислений зависит от выбора R и, что еще более важно, N. Для точек, близких к границе множества, необходимо проделать много шагов, чтобы определить, сходятся ли итерации. Тем не менее $R = 100$ и $N = 100$ позволяет получить интересное изображение.

(б) Для той же самой итерации, что и в (а), постройте множество Мандельброта. Организуйте цикл по значениям c из области $-2.25 \leqslant \mathrm{Re}(c) \leqslant 0.75$, $-1.5 \leqslant \mathrm{Im}(c) \leqslant 1.5$.

10.13. Система итерируемых функций. *Аффинное* преобразование — это правило, которое ставит в соответствие произвольной точке плоскости (x, y) точку плоскости (x', y') следующим образом:

$$x' = ax + by + e,\ y' = cx + dy + f,$$

где a, b, c, d, e, f — числа, определяющие преобразование. *Системой итерируемых функций* называется множество n аффинных преобразований и частот p_i, $i = 1, \ldots, n$. Преобразования применяются итеративно, каждое преобразование используется $p_i \times 100\%$ всего времени, причем выбор преобразования случаен. Начинают с точки $(x, y) = (0, 0)$, генерируют случайное число, на его основе выбирают преобразование, вычисляют следующую точку (x, y) в соответствии с этим преобразованием и затем повторяют процесс, скажем, 2500 раз. Другими словами,

$$x' = a_k x + b_k y + e_k,\ y' = c_k x + d_k y + f_k,$$

где k – индекс выбираемого преобразования. Напишите программу, реализующую этот алгоритм. Она должна считывать параметры преобразования (т.е. значения n, a_i, b_i, e_i, c_i, d_i, f_i и p_i, $i = 1, \ldots, n$) из файла данных. После того как ваша программа будет отлажена, проведите итерации с преобразованиями, задаваемыми параметрами

Таблица 10.1

k	a	b	c	d	e	f	p
1	0.5	0.0	0.0	0.5	0.0	0.0	0.33
2	0.5	0.0	0.0	0.5	1.0	0.0	0.33
3	0.5	0.0	0.0	0.5	0.5	0.5	0.34

(они порождают «треугольник Серпинского») и параметрами

Таблица 10.2

k	a	b	c	d	e	f	p
1	0.00	0.00	0.00	0.16	0.0	0.00	0.01
2	0.20	−0.26	0.23	0.22	0.0	1.60	0.07
3	−0.15	0.28	0.26	0.24	0.0	0.44	0.07
4	0.85	0.04	−0.04	0.85	0.0	1.60	0.85

(они порождают «папоротник»). *Указание.* Наибольшее и наименьшее x и y, которые получаются в результате таких преобразований, нельзя определить заранее. Поэкспериментируйте с различными значениями или запоминайте все пары (x, y) до конца процесса, а затем определите нужные границы. Не изображайте на графике первые десять пар, так как они необходимы для становления процесса.

10.14. Часто требуется произвести случайную выборку k человек из n индивидуумов без возвращения. Фрагмент стандартного алгоритма, который производит такую выборку, выглядит следующим образом:

(а) Пронумеровать индивидуумов числами 1, 2, ..., n.
(б) Положить $i = 1$, $j = 0$.
(в) Если $j = k$, то останов. Выборка закончена. В противном случае перейти к шагу (г).
(г) Обратиться к генератору случайных чисел, равномерно распределенных на (0, 1).
(д) Выбрать i-го индивидуума с вероятностью $(k - j)/(n - i + 1)$.
(е) Если i-й индивидуум выбран, увеличить значение j на 1.
(ж) Увеличить значение i на 1.
(з) Перейти к шагу (в).

Ключевым критерием случайной выборки является то, что любая возможная выборка может быть получена с одинаковой вероятностью. Используя приведенный выше алгоритм, получите 1000 выборок размера 2 из 6 человек. Существует 15 различных возможностей. Каждая ли из возможных выборок появляется с одинаковой частотой? Одинакова ли вероятность выбора 1 и 2 и 1 и 6? Будет ли ваше утверждение верным, если использовать в качестве датчика случайных чисел старую версию RND? Почему?

10.15. Рассмотрим подбрасывание правильной монеты. Найдите среднее число подбрасываний до

(а) появления последовательности ООРР;

(б) появления последовательности ОРОР;

(в) появления последовательности ОООО.

(Символы О и Р обозначают соответственно выпадение «орла» и «решки».— *Перев.*) (*Указание.* Вероятности этих событий различны.) Задача может быть решена аналитически, но приближенные значения вероятностей нетрудно найти с помощью моделирования. По поводу аналитического решения задачи см. книгу [Ross, 1983], с. 231.

10.16. Находящийся несколько навеселе официант в ресторане принимает заказ от n посетителей. Каждый из них заказывает себе блюдо, отличное от выбранных другими. Официант берет на кухне нужные блюда, но забывает, кто что заказал, и поэтому раздает их случайным образом. Какова вероятность того, что ни один из посетителей не получит того блюда, которое заказал? Для моделирования этой задачи рассмотрите массив x и n элементов. Наша задача – сопоставить случайным образом каждому элементу число в диапазоне от 1 до n, причем ни одно число не может быть использовано повторно. (Это соответствие называется *перестановкой*, и оно также связано с выборкой без возвращения.) Произведя перестановку, считаем, что i-й заказчик получил заказанное блюдо, если $x_i = i$. Рассмотрите эту задачу для $n = 5$, 10, 20 и 50. При возрастании n ответ должен приближаться к некоторой знаменитой константе. Найдите эту константу.

10.11. Прологи: UNI и RNOR

```
      REAL FUNCTION UNI( )
C*** НАЧАЛО ПРОЛОГА UNI
C*** ДАТА НАПИСАНИЯ 810915 (ГГММДД)
C*** ДАТА ПЕРЕСМОТРА 871210 (ГГММДД)
C*** КАТЕГОРИЯ NO. L6A21
C*** КЛЮЧЕВЫЕ СЛОВА: случайные числа, равномерно распреде-
C     ленные числа
C*** АВТОРЫ: KAHANER D., SCIENTIFIC COMPUTING
C                DIVISION, NBS; MARSAGLIA G., SUPERCOMPUTER
```

```
C                RES. INST., FLORIDA ST.U.
C
C*** НАЗНАЧЕНИЕ: Эта программа генерирует вещественные слу-
C                чайные числа (одинарной точности), равномерно
C                распределенные на [0, 1).
C*** ОПИСАНИЕ
C     Вычисляет вещественные случайные числа (одинарной точности),
C     равномерно распределенные на [0, 1).
C     Из книги D. Kahaner, C. Moler, S. Nash "Numerical Methods and
C              Software" Prentice-Hall, 1988.
C
C
C     ИСПОЛЬЗОВАНИЕ
Ц     Инициализация генератора: оператор
C          (USEED = USTART(ISEED)
C     (где ISEED – произвольное ненулевое целое число)
C     дает представление числа ISEED с плавающей точкой.
C
C     В дальнейшем обращение
C          U = UNI( )
C     порождает вещественное число из равномерного распределения
C     на [0, 1). Хотя бы одна инициализация необходима, но число
C     обращений к UNI и порядок их могут быть произвольными.
C     Замечание. В зависимости от значения K (см. ниже) для разных
C                машин результаты, выдаваемые подпрограммой UNI,
C                могут быть разными.
C     Типичное использование:
C
C          REAL U, UNI, USTART, USEED
C          INTEGER ISEED
C             Задать начальное значение
C          ISEED = 305
C          USEED = USTART (ISEED)
C          DO 1 I = 1, 1000
C             U = UNI( )
C     1       CONTINUE
C     Замечание. Если K = 24 (значение по умолчанию; см. ниже), то
C                значение U на выходе равно 0.1570390462475...
C     WRITE(*,*) U
C     END
C
C     Замечание о переносимости. Пользователь может выбрать один
C                из двух режимов работы UNI: режим умолчания (не
C                требующий от пользователя никаких действий), порож-
C                дающий одну и ту же последовательность чисел на
C                любом компьютере, для которого мантисса числа с
```

```
C               плавающей точкой имеет по меньшей мере 24 бита,
C               и режим, позволяющий генерировать последователь-
C               сти с бо́льшим периодом, если для данного компью-
C               тера длина мантиссы превышает 24.
C               Чтобы использовать второй режим, прежде чем об-
C               ратиться к USTART, вставьте оператор UBITS =
C               = UNIB(K), K > = 24, где K - число битов в мантиссе
C               слова с плавающей точкой вашего компьютера
C               (K = 48 для машин Cray, Cyber 205). UNIB переводит
C               значение K в форму с плавающей точкой; в даль-
C               нейшем реально используется именно эта форма.
C               Если входное значение K.LE. 24, то UBITS = 24.
C               Если входное значение K.GT. 24, то UBITS =
C               = FLOAT(K).
C               Если K > 24, то последовательности чисел, генериру-
C               мые подпрограммой UNI, для разных машин могут
C               быть разными.
C*** ССЫЛКИ: MARSAGLIA G., COMMENTS ON THE PERFECT
C               UNIFORM RANDOM NUMBER GENERATOR,
C               НЕОПУБЛИКОВАННЫЕ ЗАМЕТКИ, WASH S.U.
C*** ВЫЗЫВАЕМЫЕ ПОДПРОГРАММЫ: НЕТ
C*** КОНЕЦ ПРОЛОГА UNI
      REAL FUNCTION RNOR( )
C*** НАЧАЛО ПРОЛОГА RNOR
C*** ДАТА НАПИСАНИЯ 810915 (ГГММДД)
C*** ДАТА ПЕРЕСМОТРА 870419 (ГГММДД)
C*** КАТЕГОРИЯ NO. L6A14
C*** КЛЮЧЕВЫЕ СЛОВА: случайные числа, нормальное распределе-
C                     ние
C*** АВТОРЫ: KAHANER D., SCIENTIFIC COMPUTING DIVISI-
C               ON, NBS; MARSAGLIA G., SUPERCOMPUTER RES.
C               INST., FLORIDA ST. U.
C
C   НАЗНАЧЕНИЕ: Генерирует нормально распределенные случай-
C               ные числа со средним 0 и стандартным от-
C               клонением 1; это распределение часто обозначае-
C               ется через N(0, 1).
C   Из книги D. Kahaner, C. Moler, S. Nash «Numerical Methods and
C            Software».
C            Prentice-Hall, 1988.
C   ИСПОЛЬЗОВАНИЕ
C   При первом обращении
C        Z = RSTART(ISEED)
C   Здесь ISEED - любое
C   ненулевое целое число. Этот оператор осуществляет инициали-
C   зацию программы. RSTART переводит ISEED в вещественное
```

```
C     представление
C     (одинарной точности).
C     При последующих обращениях
C           Z = RNOR(  ).
C     Значением Z будет очередное случайное вещественное число
C     (одинарной точности).
C
C     ТИПИЧНОЕ ИСПОЛЬЗОВАНИЕ
C           REAL RSTART, RNOR, Z
C           INTEGER ISEED, I
C           ISEED = 305
C           Z = RSTART(ISEED)
C           DO 1 I = 1.10
C                 Z = RNOR(  )
C                 WRITE(*,*)Z
C     1     CONTINUE
C           END
C
C
C*** ССЫЛКИ: MARSAGLIA & TSANG. A FAST, EASILY
C                 IMPLEMENTED METHOD FOR SAMPLING FROM
C                 DECREASING OR SYMMETRIC UNIMODAL DEN-
C                 SITY FUNCTIONS, SIAM J SISC 1983.
C*** ВЫЗЫВАЕМЫЕ ПОДПРОГРАММЫ: НЕТ
C*** КОНЕЦ ПРОЛОГА RNOR
```

Глава 11

Аппроксимация тригонометрическими функциями и быстрое преобразование Фурье

11.1. Введение

Многие физические явления в природе имеют циклический характер. Ежедневные и сезонные колебания, повторения в структуре материалов, например в кристаллах, указывают на важность периодических функций для изучения окружающего нас мира. Сущность анализа Фурье состоит в том, что он позволяет представить сложную циклическую структуру в виде подходящей комбинации более простых периодических функций, таких, как синусы и косинусы. При этом исходная информация преобразуется в другую, потенциально более удобную форму. Как только это сделано, изменяются перспективы исследований и появляется возможность более глубокого понимания объекта. Вы уже знакомы по крайней мере с одним таким преобразованием – логарифмическим – и оценили его важность. Если вы когда-то что-то читали об амплитудной или частотной характеристике стереоусилителя или покупали графический эквалайзер для своего автомобиля, то вы при этом думали в стиле анализа Фурье. В этой главе мы рассмотрим несколько примеров такого подхода, а в § 9 применим его для анализа природного явления, получившего название "el Niño".

Инструменты, которыми мы будем пользоваться,– это ряды Фурье и преобразование Фурье. Они представляют собой одно из крупнейших достижений прикладной математики и были впервые введены в нее Даниилом Бернулли в 50-х годах восемнадцатого века при изучении колебаний струны. Почти 60 годами позднее в процессе изучения теплопроводности Жан Батист Фурье использовал аналогичный, но в большей степени формализованный метод. Ряды и преобразования Фурье используются почти во всех областях науки, включая теорию электрических цепей, теорию механических и колебательных систем, оптику, акустику и термодинамику. По общим вопросам отсылаем читателя к книге [Weaver, 1983]. Анализ Фурье применялся также при изучении временных рядов – последовательностей числовых величин, упорядоченных некоторым естественным образом. Обычно каждое наблюдение такого рода отвечает определенному моменту времени (скажем, речь может идти об объемах еженедельной продажи некоторого товара); именно это и задает упорядочение. Элементарное обсуждение этого вопроса можно найти в книге [Bloomfield, 1976].

Мы обсудим три основные темы, тесно связанные между собой.

(1) Интегральное преобразование Фурье, которое переводит вещественную функцию в пару вещественных функций или одну комплексную функцию в другую.

(2) Ряд Фурье, который ставит в соответствие периодической функции последовательность ее коэффициентов Фурье.
(3) Дискретное преобразование Фурье, переводящее одну последовательность чисел в другую. Его можно выполнить при помощи вычислительного метода, который называется быстрым преобразованием Фурье; он хорошо описан в книге [Brigham, 1974].

В этой главе используется материал нескольких предыдущих глав. Для приближенного вычисления интегралов используется правило трапеций из гл. 5. Частичные суммы рядов Фурье удовлетворяют условию наименьших квадратов (гл. 6), но в то же время могут применяться для интерполяции (гл. 4). Обсуждаемые понятия могут оказаться достаточно тонкими, а связанные с ними доказательства могут потребовать применения весьма сложных математических приемов. Мы опустим большинство доказательств и при объяснении будем полагаться на эвристические соображения и диаграммы. Предполагается, что читатель знаком с некоторым элементарным изложением идей комплексной арифметики и анализа Фурье.

11.2. Интегральное преобразование Фурье, дискретное преобразование Фурье и ряды Фурье

Функция $t(x)$ называется *периодической* с периодом $p > 0$, если $t(x + p) = t(x)$ для любого x. Если функция $t(x)$ имеет период p, то она имеет и периоды $2p$, $3p$ и т. д., но мы закрепляем термин «период» за наименьшим периодом функции. Отметим, что функция $T(x) \equiv t(fx)$ имеет период p/f, поскольку $T(x + p/f) = t(f[x + p/f]) = t(fx + p) = t(fx) = T(p)$. Следовательно, можно построить функции с каким угодно периодом. Как правило, бывает легко установить, периодична ли функция. Предположение о периодичности переносится и на данные, но установить этот факт обычно труднее. Если анализируемые данные со всей очевидностью не являются периодическими, например данные по концентрации CO_2 в задаче 6.4, то в качестве первого важного шага надо выявить (на глаз, методом наименьших квадратов и т. д.) непериодический тренд и вычесть его из данных.

Основная идея этой главы состоит в том, что функцию или набор данных можно разложить по определенной системе тригонометрических функций. Центральную роль в этих разложениях играют коэффициенты разложения, и мы займемся изучением их свойств. Этим ряды Фурье отличаются, скажем, от рядов Тейлора, коэффициенты которых часто не представляют самостоятельного интереса. Начнем с определений интегрального преобразования Фурье, дискретного преобразования Фурье и ряда Фурье. Все они обладают особыми свойствами, но, как будет показано в следующих параграфах, тесно взаимосвязаны. В данной главе рассматриваются только вещественные функции и вещественные данные, хотя при необходимости используются и комплексные перемен-

ные. Сначала сформулируем (без доказательств) основные утверждения, которые нам потребуются.

(1) Вещественную функцию $g(x)$ можно представить в виде разности двух несобственных интегралов:

$$g(x) = \int_0^\infty A(\omega)\cos(2\pi\omega x)\,d\omega - \int_0^\infty B(\omega)\sin(2\pi\omega x)\,d\omega,$$

где функции A и B определяются равенствами

$$A(\omega) = 2\int_{-\infty}^{\infty} g(x)\cos(2\pi\omega x)\,dx,$$

$$B(\omega) = 2\int_{-\infty}^{\infty} g(x)\sin(2\pi\omega x)\,dx,\ \omega \geqslant 0.$$

Функции A и B называются *интегральными косинус-преобразованием и синус-преобразованием Фурье* функции g, а сама функция g называется *интегральным обратным преобразованием Фурье* функций A и B.

Пример 11.1. Интегральное преобразование Фурье функции Рунге на $(-\infty, \infty)$.

Функция Рунге $R(x) = 1/(1 + 25x^2)$ является четной, следовательно, $B(\omega) = 0$. Можно показать, что ее интегральное косинус-преобразование Фурье имеет вид

$$A(\omega) = 2\int_{-\infty}^{\infty} \frac{\cos 2\pi\omega x}{1 + 25x^2}\,dx = \frac{2\pi}{5}\exp(-2\pi\omega/5).$$

Тогда, согласно утверждению (1),

$$\frac{1}{1 + 25x^2} = \frac{2\pi}{5}\int_0^\infty \exp(-2\pi\omega/5)\cos(2\pi\omega)\,dx.$$

В некоторых учебниках функцию $1/(1 + ax^2)$ называют *распределением Коши*. □

(2) Если $g_0, g_1, \ldots, g_{N-1}$ – произвольные вещественные числа, то их можно представить в виде конечных сумм, используя формальную запись

$$g_j = a_0 + \sum_{k=1}^{N/2} a_k \cos\left(kj\frac{2\pi}{N}\right) + \sum_{k=1}^{N/2} b_k \sin\left(kj\frac{2\pi}{N}\right),$$

$$j = 0, \ldots, N-1.$$

Это требует некоторых разъяснений. Во-первых, вместо верхнего предела суммирования $N/2$ следовало бы писать $[N/2]$ – целую часть числа $N/2$, но мы будем пользоваться более коротким обозначением. Таким образом, в случае нечетного N вместо $N/2$ следует брать $(N-1)/2$. Во-вторых, если N четное, член $b_{N/2}$ нужно опустить. Числа a_k и b_k

определяются по формулам

$$a_0 = \frac{1}{N}\sum_{j=0}^{N-1} g_j, \qquad a_{N/2} = \frac{1}{N}\sum_{j=0}^{N-1} g_j \cos(j\pi) = \frac{1}{N}\sum_{j=0}^{N-1} (-1)^j g_j$$

и

$$a_k = \frac{2}{N}\sum_{j=0}^{N-1} g_j \cos\left(jk\frac{2\pi}{N}\right), \qquad b_k = \frac{2}{N}\sum_{j=0}^{N-1} g_j \sin\left(jk\frac{2\pi}{N}\right).$$

$$j \leqslant k < N/2.$$

Последовательности чисел a и b называются *конечными* или *дискретными косинус-преобразованием и синус-преобразованием Фурье* набора чисел g, а сам набор g называется *обратным конечным* (*дискретным*) *преобразованием Фурье* наборов a и b. Для обозначения дискретного преобразования Фурье часто используют сокращение ДПФ.

(3) Если $g(x)$ – периодическая функция, заданная на интервале $[a, b]$, то ее можно представить в виде бесконечного ряда по синусам и косинусам

$$g(x) = \frac{A_0}{2} + \sum_{n=1}^{\infty} A_n \cos\left(n\frac{2\pi x}{b-a}\right) + \sum_{n=1}^{\infty} B_n \sin\left(n\frac{2\pi x}{b-a}\right),$$

где *коэффициенты Фурье* A_n и B_n задаются выражениями

$$A_n = \frac{2}{b-a}\int_a^b g(x)\cos\left(n\frac{2\pi x}{b-a}\right)dx,$$

$$B_n = \frac{2}{b-a}\int_a^b g(x)\sin\left(n\frac{2\pi x}{b-a}\right)dx.$$

Функция $\sin 2\pi\mu x$ принимает одинаковые значения через каждые $1/\mu$ радиан, поэтому говорят, что она имеет *радиальную, или угловую, частоту* $2\pi\mu$ *и круговую частоту* μ. Если x представляет собой время, измеряемое в секундах, то $\sin 2\pi\mu x$ имеет радиальную частоту $2\pi\mu$ радиан в секунду и круговую частоту μ оборотов в секунду. Период $1/\mu$ функции $\sin 2\pi\mu x$ является величиной, обратной к круговой частоте. Если x измеряется в единицах длины, например в миллиметрах, то период называется *длиной волны*, а радиальная частота – *волновым числом*. Когда термин «частота» используется отдельно, он может означать и ту и другую частоты, но мы будем употреблять его как сокращенное название круговой частоты.

Каждое из указанных трех представлений функции g можно интерпретировать как сумму по частотам.

(1) Для интегрального преобразования Фурье ω является частотной переменной, а интегралы, дающие значение g в точке x, представляют собой «сумму» по всем возможным частотам.

(2) В случае дискретного преобразования Фурье удобно представлять себе g_i как величины, наблюдаемые через каждые Δ единиц времени.

Величина $T \equiv N\Delta$ называется *длиной записи* или *основным периодом*. *Основная круговая частота* определяется как $1/T$. В дискретном преобразовании Фурье выражения вроде $kj2\pi/N$ можно записывать в виде $j\Delta k2\pi/T$, и тогда при определении значения g_j (в точке $j\Delta$) суммирование производится по частотам, которые являются целыми кратными основной частоты. Каждая из них на основную частоту меньше следующей. Наибольшим множителем перед основной частотой будет $N/2$, что соответствует максимальной частоте $(N/2) \cdot 1/T = 1/(2\Delta)$, которая называется *частотой Найквиста* – в честь инженера Гарри Найквиста (1889–1976), введшего это понятие при решении задач о телеграфных сообщениях. Частоты, более высокие, чем частота Найквиста, данными определяться не могут. Заметим, что a_0 представляет собой среднее арифметическое всех величин g и называется *постоянной компонентой* [1]. Утверждение о дискретном преобразовании Фурье означает, что, в точках $k = 0, \ldots, N - 1$ функцию g можно представить в виде суммы постоянной компоненты и конечного числа членов с разными частотами.

(3) В случае ряда Фурье суммирование также производится по целым кратным основной частоты $1/(b - a)$. Каждая частота меньше следующей на эту величину. Член $A_0/2$ представляет собой среднее значение функции $g(x)$ на $[a, b]$. Разложение в ряд Фурье показывает, что на $[a, b]$ функцию $g(x)$ можно записать в виде суммы постоянной компоненты и бесконечного числа членов с разными частотами. Эти частоты не являются произвольными, они находятся в отношении $1:2:3:4:\ldots$. Не каждая периодическая функция представляет собой синусоиду, но каждую периодическую функцию (удовлетворяющую некоторым дополнительным условиям.– *Перев.*) можно записать в виде ряда из синусоид с частотами, равными целым кратным основной частоты.

В каждом из трех случаев существует определенная эквивалентность между g и ее синус- и косинус-преобразованиями, между g и ее коэффициентами Фурье: одно можно получить из другого. В математике есть много пар преобразований; например, вы могли встречаться с преобразованиями Лапласа, Ханкеля, Z и т. п. Работать ли с функцией g, с ее синус- и косинус-преобразованиями или с ее рядом Фурье – это вопрос удобства, решение которого зависит от конкретных приложений.

В некоторых приложениях интегральные преобразования являются ключевыми математическими инструментами. В оптике при помощи преобразований Фурье можно изучать действие линзы. Точным оптическим измерительным прибором является интерферометр; чтобы проанализировать результаты измерений, надо вычислить преобразование Фурье выходных данных этого прибора. В других областях, таких, как цифровая обработка сигналов, изучаемые величины являются дискретными, и поэтому интегралы заменяются бесконечными суммами. Но независимо от своего происхождения доступные данные на практике

[1] В оригинале используется термин DC component – компонента постоянного тока.– *Прим. перев.*

всегда образуют набор чисел, представляющий собой некоторую выборку экспериментальных наблюдений. Естественно, это будет конечный набор. Инженеры постоянно спрашивают: «Какие выводы можно сделать о поставленной задаче по конечному числу наблюдений?»

Обычно научным работникам приходится вычислять дискретное преобразование Фурье. Для этого разработаны высокоэффективные новаторские приемы, среди которых самый главный – быстрое преобразование Фурье; но для приложений это всего лишь первый шаг. С помощью этих приемов можно получить оценки ряда Фурье исходной функции или ее интегральных преобразований Фурье. Интервал между точками выборки определяется измерительной аппаратурой, но инженер должен понимать его роль. Подробное обсуждение этих приложений выходит за рамки нашей книги, поэтому мы ограничимся несколькими характерными примерами. Нашей главной целью будет показать, как пользоваться программами и как различные преобразования связаны между собой.

11.3. Энергия и мощность

Когда говорят о взрыве или о выходном сигнале стереоусилителя, часто используют термины «энергия» и «мощность». Они имеют как интуитивно ясное, так и техническое значение. Чтобы придать этим терминам точность, можно использовать ряды и преобразования Фурье; это и будет темой данного параграфа.

Мощность на частоте ω определяется как

$$P(\omega) = \frac{1}{2}(A(\omega)^2 + B(\omega)^2).$$

Величину $P(\omega)$ также называют *энергией*, приходящейся на единицу частоты, на частоте ω. Для дискретной последовательности мощность на частоте k составляет

$$P_0 = Na_0^2, \qquad P_k = \frac{N}{2}(a_k^2 + b_k^2), \qquad 0 < k \leqslant N/2,$$

но $P_{N/2} = Na_{N/2}^2$, если N четно; для периодических функций мощность на частоте k составляет

$$P_0 = \frac{b-a}{2} A_0^2, \qquad P_k = \frac{b-a}{2}(A_k^2 + B_k^2), \qquad k > 0.$$

В любом случае *общая энергия* равна интегралу (или сумме) от мощности по всем частотам.

Если x представляет собой время, то $[g(x)]^2$ является мощностью в момент времени x, или, что то же самое, энергией в единицу времени в момент x. Общая энергия равна интегралу (или сумме) от мощности по

всему отрезку времени. Связь выражений для энергии через время и через частоты устанавливает *закон сохранения энергии*, который утверждает, что общую энергию можно вычислить как по коэффициентам Фурье, так и по исходной функции. Математически это записывается в виде равенства

$$E = \int_{-\infty}^{\infty} [g(x)]^2\, dx = \int_{0}^{\infty} P(\omega)\, d\omega,$$

или

$$E = \sum_{k=0}^{N-1} g_k^2 = \sum_{k=0}^{N/2} P_k,$$

или

$$E = \int_{0}^{b} [g(x)]^2\, dx = \sum_{k=0}^{\infty} P_k,$$

в зависимости от того, с каким преобразованием или рядом приходится иметь дело. Эти равенства можно интерпретировать следующим образом: энергия в g распределена по всем частотам, присутствующим в частотном представлении исходной функции.

Зависимость мощности от частоты часто изображают на графике. Эта зависимость называется *спектром мощности* в случае интегрального преобразования Фурье и *периодограммой* в случае дискретного преобразования Фурье или ряда Фурье. Для дискретного преобразования Фурье мощность или энергию изображают при значениях частоты k/T, которая изменяется от 0 до $1/(2\Delta)$. Во многих приложениях член, отвечающий $k = 0$, не изображают. Другое часто применяемое соглашение состоит в использовании обратного масштаба по горизонтальной оси без изменения периодограммы. При этом правый конец отрезка абсцисс превращается в 2Δ, а левый конец – в ∞, и абсциссы измеряются в длинах периодов. Иными словами, синусоиды с более длинными периодами вносят свой вклад в левую часть периодограммы.

Пример 11.2. Спектр мощности функции Рунге.

Интегральное преобразование Фурье функции Рунге было вычислено в примере 11.1. На рис. 11.1 изображен график спектра мощности этой функции.

Можно также проверить выполнение закона сохранения энергии для этой функции. Нетрудно показать, что

$$E = \int_{-\infty}^{\infty} \left(\frac{1}{1+25x^2}\right)^2 dx = \frac{\pi}{10};$$

в результате непосредственного вычисления получаем

$$\frac{1}{2}\int_{0}^{\infty} [A(\omega)]^2\, d\omega = \frac{2\pi^2}{25}\int_{0}^{\infty} \exp(-4\pi\omega/5)\, d\omega = \frac{\pi}{10}. \qquad \square$$

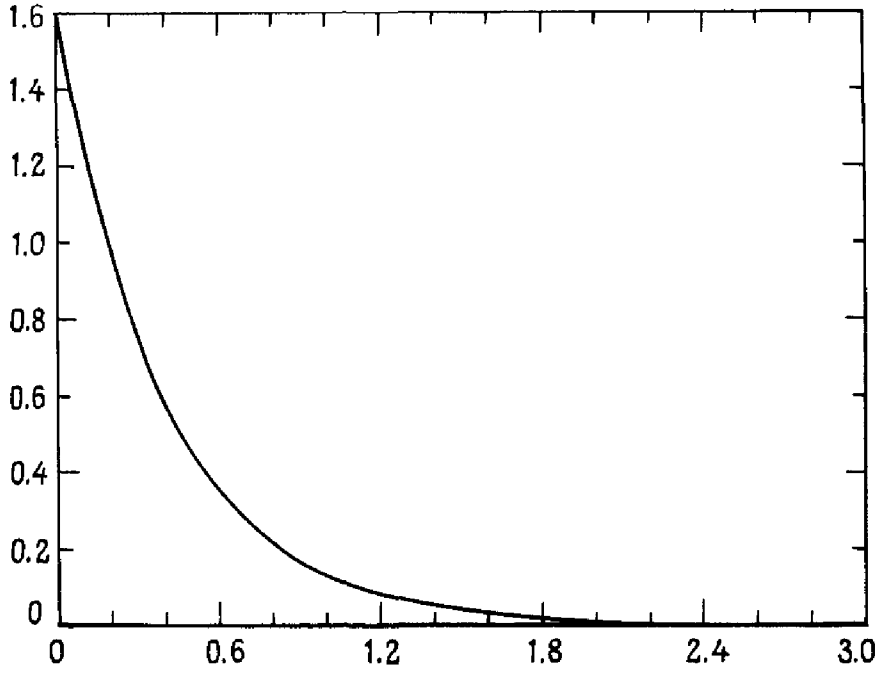


Рис. 11.1. Спектр мощности функции Рунге.

В задаче 6.8 объяснялась важность измерений концентрации углекислоты в атмосфере и было показано, что данные этих измерений можно описать возрастающей экспонентой в комбинации с несколькими синусоидальными членами. Чтобы провести их дальнейший анализ в контексте этой главы, понадобится удалить из модели данных все, кроме синусоид, т. е. вычесть из каждого заданного значения величину $B + d\exp(\alpha t)$, $t = 0, 1, \ldots, 215$, где $B \approx 300.81$, $\alpha \approx 0.0037$ и $d \approx 14.18$ (вспомните, что дано 216 точек). Результат изображен в верхней части рис. 11.2. После вычитания можно вычислить дискретное преобразование Фурье данных и найти из него периодограмму, график которой приведен в остальной части рис. 11.2. У периодограммы есть два отчетливых пика при частотах 18/216 и 36/216 и, возможно, еще один возле левого края. Большие пики отвечают частотам в восемнадцать и в тридцать шесть раз бóльшим, чем основная частота 1/216, или длинам периодов в двенадцать месяцев и в шесть месяцев соответственно. Наибольший пик связан с годовым циклом (двенадцать месяцев), который ясно виден на верхнем графике. Второй пик соответствует полугодовому циклу (шесть месяцев). Полезно попытаться понять, почему всего этого следовало ожидать. Сильные ежегодные изменения в данных невозможно представить чистой синусоидой с частотой 1/12, но их можно описать суммой синусоид с частотами 1/12, 2/12, 3/12, 4/12 и т. д., а эти частоты отвечают периодам в двенадцать, шесть, четыре, три месяца и т. д. Шестимесячный пик вполне реален; есть также и пик на трех месяцах, но он слишком мал, чтобы принимать его во внимание.

На нижнем рисунке используется более крупный масштаб по горизонтали, поэтому лучше видны пики вблизи оси ординат. Пик слева

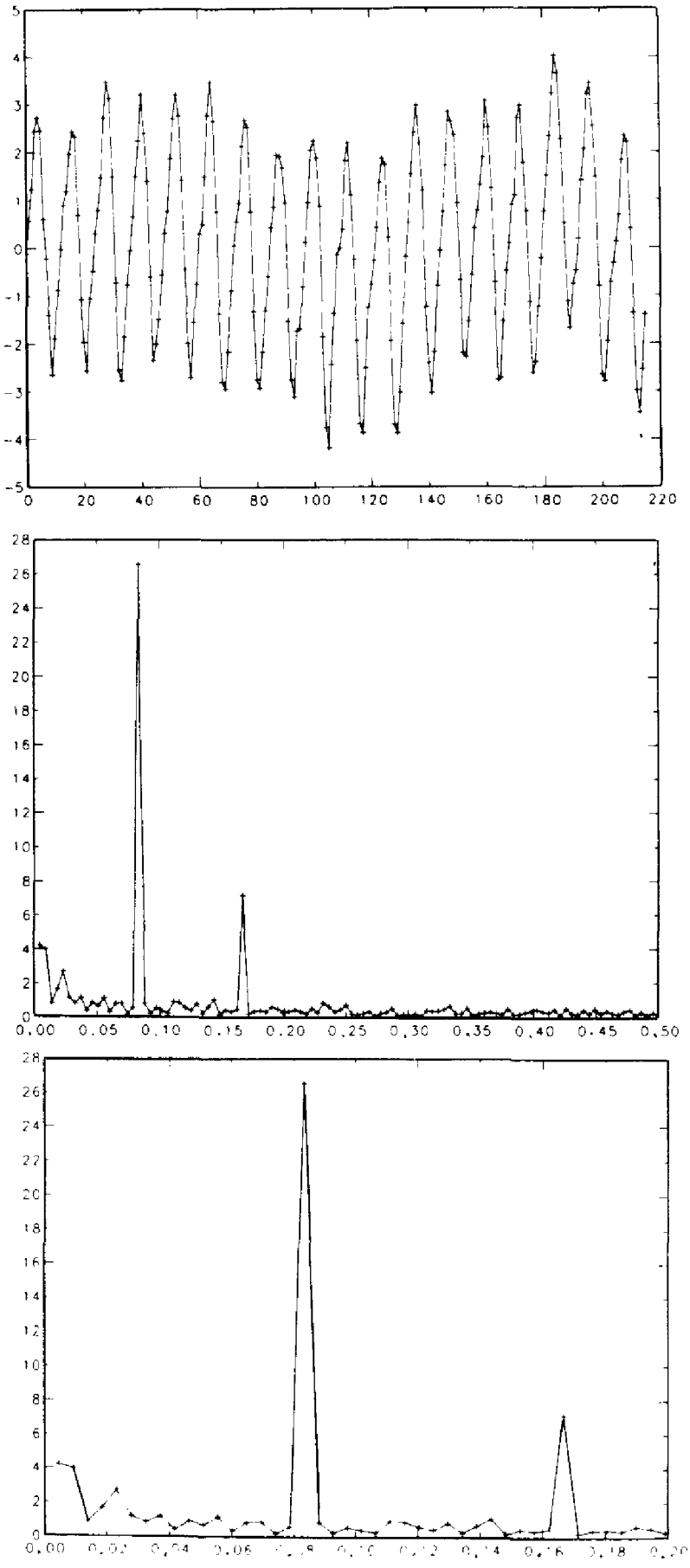


Рис. 11.2. Углекислота в атмосфере (после вычитания тренда) и ее периодограмма.

возле самого края периодограммы соответствует очень длинному циклу в 100–110 месяцев, который едва регистрируется данными. Пик поменьше, чуть правее предыдущего, соответствует периоду около 40–44 месяцев. Эти циклы вполне реальны, хотя для того, чтобы убедиться, что они не являются простыми статистическими флуктуациями, требуется более аккуратный анализ (см., например, статью [Rust et al., 1979]). Мы привели этот пример только для того, чтобы продемонстрировать полезность понятия периодограммы; дальнейшее обсуждение можно найти в указанной литературе.

11.4. Историческая справка: Фурье (1786–1830)

На одной из самых памятных сессий Французской академии наук 21 декабря 1807 года математик и инженер Жан Батист Жозеф Фурье, которому был тогда 21 год, сделал утверждение, открывшее новую главу в истории математики. «Фурье заявил, что произвольную функцию, заданную на конечном интервале произвольным, сколь угодно причудливым графиком, всегда можно разложить в сумму чистых синусоидальных функций. Академикам, включая великого аналитика Лагранжа, это показалось совершенно невероятным. В конце концов, любая суперпозиция таких функций не может дать ничего другого, кроме бесконечно дифференцируемой функции, называемой «аналитической», что очень далеко от заявленной Фурье произвольности функции. Конечно, последующие исследования показали, что утверждение Фурье было абсолютно справедливо, хотя сам он не мог предложить строгих доказательств из-за отсутствия методов работы с бесконечными рядами» [1]. Дальнейшее развитие этой области связано с такими величайшими математическими умами, как Дирихле, Фейер, Лагранж, Риман, Лебег, Гаусс и Борель. Но метод Фурье, разработанный им в связи с теоретическими исследованиями теплопроводности, и особенно интеграл Фурье не имели предшественников.

Сегодня имя Фурье известно и почитаемо, но и в профессиональной, и в личной жизни Фурье познал много трудностей. Он родился в маленькой деревне на юге Франции. Когда ему было девять лет, умер его отец, и мальчик попал в сиротский приют. Затем Фурье был помещен в городскую военную школу. Его возрастающий интерес к математике сочетался с активным участием в местных делах: во время Великой Французской революции он был ненадолго арестован за мужественную защиту жертв террора. Его карьера учителя была прервана, как только стали известны его организаторские способности, и в 1798 г. он был отобран для участия в египетской кампании Наполеона. Фурье стал секретарем Египетского института, занимал и другие дипломатические посты. Он составил обширное описание всех сокровищ, открытых во время египетской кампании; это был первый из когда-либо опублико-

[1] C. Lanczos, Discourse on Fourier Series, Oliver and Boyd, 1966.

ванных полных списков такого рода. По возвращении во Францию он получил место префекта недалеко от Гренобля, а в 1808 г. Наполеон даровал ему титул барона. Но после конца наполеоновской империи Фурье был вынужден уйти в отставку. Людовик XVIII сначала противился его назначению в Академию наук из-за его былых связей с Наполеоном, но в 1817 г. смягчился. С этого момента и до самой смерти Фурье активно участвовал в развитии научной мысли. В течение всей своей карьеры Фурье пользовался верной дружбой молодых коллег, благодарных за его бескорыстную поддержку и ободрение; большинство старших коллег восхищались его достижениями. Одним из немногих исключений был Пуассон, постоянно вступавший в полемику с Фурье, всегда настроенный критически и неприязненно.

В последние несколько лет жизни Фурье его здоровье постоянно ухудшалось из-за болезни (возможно, микседемы), которую он получил во время своего пребывания в Египте. Но даже в эти годы он продолжал публиковать работы по механике, термодинамике, теории уравнений и статистике. Память о нем была увековечена в различных мемориалах, а его средняя школа в Оксере была переименована в Лицей Фурье. Интересно, что одно из его наиболее выдающихся деяний наименее известно: Шампольон [1], сумевший в конце концов расшифровать иероглифы на Розеттском камне, был студентом Фурье и восторгался открытиями в Египте, которые Фурье каталогизировал. Эта расшифровка считается наиболее важным достижением в изучении древнеегипетской письменности.

11.5. Практическое вычисление коэффициентов Фурье; дискретное преобразование Фурье

Чтобы построить ряд Фурье заданной функции g, надо сначала вычислить интегралы, которые определяют его коэффициенты. Найти точные явные выражения для этих интегралов удается редко, чаще их приходится вычислять приближенно с помощью квадратурных формул. В этом параграфе будет показано, что дискретное преобразование Фурье является такой аппроксимацией интегралов.

Частичная сумма ряда Фурье $T_n(x)$, включающая члены вплоть до A_n и B_n, называется *тригонометрическим полиномом* степени n. Например, тригонометрический полином первой степени имеет вид

$$T_1(x) = \frac{A_0}{2} + A_1 \cos 2\pi fx + B_1 \sin 2\pi fx.$$

В дальнейшем мы будем различать ряды Фурье с точными и приближенными коэффициентами, в последнем случае обозначая суммы рядов строчными буквами t_j.

[1] Жан Франсуа Шампольон (1790-1832) - автор первой грамматики древнеегипетского языка, почетный член Петербургской академии наук.- *Прим. перев.*

Пример 11.3. Коэффициенты Фурье и их приближенное вычисление.

Функцию Рунге нельзя разложить в ряд Фурье, потому что она не является периодической. Можно определить новую функцию, периодическую и совпадающую с $R(x)$ на $[-1, 1]$. Она называется *периодическим продолжением* функции $R(x)$ с отрезка $[-1, 1]$. Вычислим ее коэффициенты Фурье и сравним частичные суммы T_n с функцией Рунге на $[-1, 1]$.

Как функция Рунге, так и ее периодическое продолжение – четные функции, поэтому $B_j = 0$. Выберем отрезок $[a, b] = [-1, 1]$; основная частота равна 1/2. Коэффициенты Фурье задаются выражениями

$$A_j = \int_{-1}^{1} \frac{\cos(j\pi x)}{1 + 25x^2}\, dx = 2\int_{0}^{1} \frac{\cos(j\pi x)}{1 + 25x^2}\, dx, \quad j = 0, 1, \ldots .$$

Для не очень больших величин j интегралы в правой части последнего равенства можно с хорошей точностью вычислить с помощью программы Q1DA из гл. 5. Эти значения перечислены в третьем столбце приводимой ниже таблицы со всеми верными десятичными знаками. Коэффициенты Фурье, занумерованные индексом j, относятся к соответствующим частотам $\omega_j = j \cdot f = j/2$. Эти частоты приводятся во втором столбце таблицы.

Таблица 11.1

j	ω_j	A_j Q1DA	Правило трапеций $N = 100$ $\Delta = .02$	Правило трапеций $N = 50$ $\Delta = .04$	Правило трапеций $N = 20$ $\Delta = .1$	Правило трапеций $N = 9$ $\Delta = .2222$
0	0	5.49360×10^{-1}	5.49355×10^{-1}	5.49341×10^{-1}	5.49242×10^{-1}	5.44373×10^{-1}
1	1/2	3.44106×10^{-1}	3.44111×10^{-1}	3.44126×10^{-1}	3.44235×10^{-1}	3.39447×10^{-1}
2	2/2	1.75749×10^{-1}	1.75745×10^{-1}	1.75730×10^{-1}	1.75632×10^{-1}	1.66760×10^{-1}
3	3/2	9.69064×10^{-2}	9.69114×10^{-2}	9.69263×10^{-2}	9.70499×10^{-2}	8.28955×10^{-2}
4	4/2	5.00138×10^{-2}	5.00089×10^{-2}	4.99939×10^{-2}	4.99077×10^{-2}	2.18574×10^{-2}
5	5/2	2.77280×10^{-2}	2.77330×10^{-2}	2.77481×10^{-2}	2.79186×10^{-2}	-2.18574×10^{-2}
6	6/2	1.40807×10^{-2}	1.40757×10^{-2}	1.40604×10^{-2}	1.40275×10^{-2}	-8.28955×10^{-2}
7	7/2	8.02722×10^{-3}	8.03220×10^{-3}	8.04772×10^{-3}	8.36474×10^{-3}	-1.66760×10^{-1}
8	8/2	3.89227×10^{-3}	3.88728×10^{-3}	3.87152×10^{-3}	4.05238×10^{-3}	-3.39447×10^{-1}
9	9/2	2.38204×10^{-3}	2.38705×10^{-3}	2.40309×10^{-3}	3.20095×10^{-3}	-5.44373×10^{-1}
10	10/2	1.02504×10^{-3}	1.02001×10^{-3}	1.00367×10^{-3}	1.98009×10^{-3}	-3.39447×10^{-1}

Принимая эти числа за коэффициенты Фурье, построим графики $T_n(x)$ на отрезке $0 \leqslant x \leqslant 1$. Для $n = 0, 1, 2, 4, 8$ эти графики изображены в правой части рис. 11.3. Заметим, что функция T_8 практически неотличима от функции Рунге. Наконец, в качестве проверки выполнения закона сохранения энергии вычислим (при помощи Q1DA) величину

$$E^2 = \int_{-1}^{1} \left(\frac{1}{1 + 25x^2}\right)^2 dx = 0.31314169\ldots ,$$

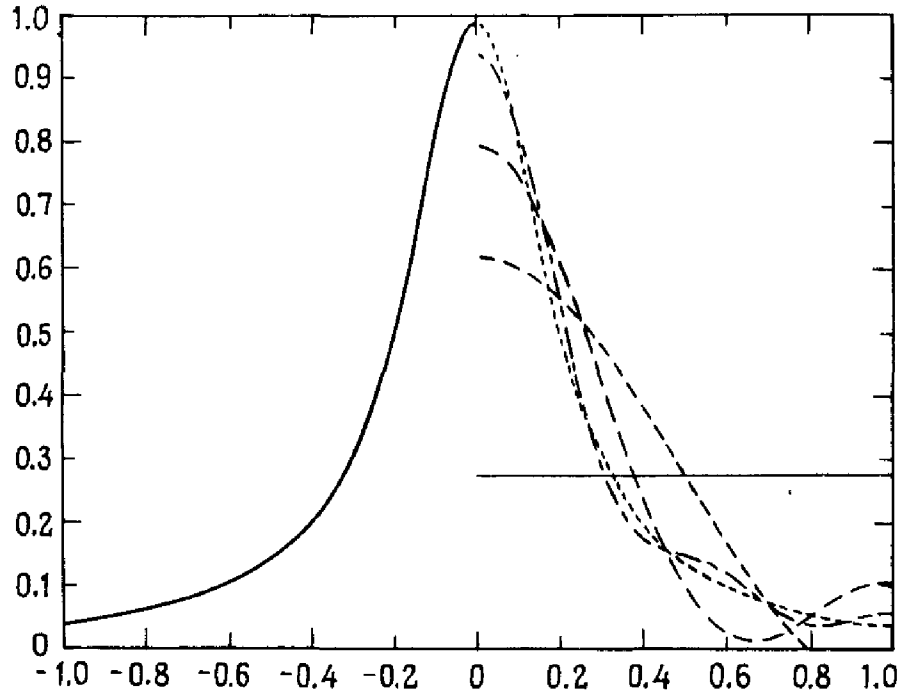


Рис. 11.3. Аппроксимация функции Рунге. Слева – t_8 на $[-1, 0]$, справа – T_0, T_1, T_2, T_4, T_8 на $[0, 1]$.

которая должна совпасть с суммой ряда

$$(b-a)\left(\left(\frac{A_0}{2}\right)^2+\frac{1}{2}\sum_{k=1}^{\infty}(A_k^2+B_k^2)\right)\approx 2\left(\left(\frac{A_0}{2}\right)^2+\frac{1}{2}\sum_{k=1}^{10}A_k^2\right)=0.3131410\ldots,$$

полученной приближенно из третьего столбца таблицы.

На рис. 11.1 показана периодограмма периодического продолжения; сравните ее со спектром мощности функции Рунге на $(-\infty, +\infty)$. □

Существуют и другие способы вычисления коэффициентов Фурье, которые могут оказаться проще, чем применение программы Q1DA. Для каждого из коэффициентов подынтегральное выражение представляет собой периодическую функцию, продолженную с $[a, b]$. В разделе 5.8.1. гл. 5 отмечалось, что для таких функций особенно эффективно правило трапеций, и поэтому оно представляется хорошим выбором. Хотя приближенное вычисление каждого коэффициента можно рассматривать как отдельную задачу, воспользуемся более прямым подходом: применим одну и ту же формулу трапеций (с одним и тем же числом значений интегрируемой функции) для всех вычисляемых коэффициентов. Если обозначить точки разбиения отрезка $[a, b]$ для формулы трапеций через $a = x_0, x_1, \ldots, x_N = b$ и положить $f = 1/(b-a)$, получим

$$A_j=\frac{2}{b-a}\int_a^b g(x)\cos(j2\pi fx)\,dx\approx$$

$$\approx\frac{2}{b-a}\cdot\frac{b-a}{N}\left[\frac{g(x_0)\cos(j2\pi fx_0)}{2}+g(x_1)\cos(j2\pi fx_1)+\ldots\right.$$

$$\ldots + g(x_{N-1})\cos(j2\pi f x_{N-1}) + \frac{g(x_N)\cos(j2\pi f x_N)}{2}\Bigg] =$$

$$= \frac{2}{N}\sum_{k=0}^{N-1} g(x_k)\cos(j2\pi f x_k) = \frac{2}{N}\sum_{k=0}^{N-1} g(x_k)\cos(j2\pi f[x_0 + k(b-a)/N]).$$

Заметим, что последнее слагаемое в этой сумме имеет индекс $N-1$, поскольку как g, так и косинус имеют период $(b-a)$ [1]. Вводя обозначение $g_k = g(x_k)$, по формуле косинуса суммы получаем

$$A_j \approx \frac{2}{N}\cos\left(\frac{2\pi j x_0}{b-a}\right)\sum_{k=0}^{N-1} g_k \cos(2\pi kj/N) - \frac{2}{N}\sin\left(\frac{2\pi j x_0}{b-a}\right)\sum_{k=0}^{N-1} g_k \sin(2\pi kj/N).$$

Важно, что написанное выше выражение содержит j-е компоненты дискретных косинус- и синус-преобразования Фурье чисел $g_0, \ldots, g_{N-1}$:

$$A_0 \approx 2a_0, \qquad A_j \approx a_j\cos\left(\frac{2\pi j x_0}{b-a}\right) - b_j\sin\left(\frac{2\pi j x_0}{b-a}\right), \qquad 0 < j \leqslant N/2.$$

Применяя аналогичным образом правило трапеций для приближенного вычисления B_j, получаем

$$B_j \approx a_j\sin\left(\frac{2\pi j x_0}{b-a}\right) + b_j\cos\left(\frac{2\pi j x_0}{b-a}\right).$$

Если $x_0 = 0$, то

$$A_0 \approx 2a_0, \qquad A_j \approx a_j, \qquad B_j \approx b_j, \qquad 0 < j \leqslant N/2.$$

Таким образом, дискретное преобразование Фурье можно рассматривать как способ приближенного вычисления точных коэффициентов Фурье периодической функции. То, что $a_n \approx A_n$ только при $x_0 = 0$, вполне естественно. Поскольку функция $g(x)$ предполагалась периодической, интегралы можно вычислять по любому отрезку длину $b-a$. Однако коэффициенты A_j, B_j и A'_j, B'_j, рядов Фурье функций $g(x)$ и $g(x+x_0)$ соответственно неодинаковы, они связаны соотношениями

$$A'_j = A_j\cos\frac{2\pi j x_0}{b-a} - B_j\sin\frac{2\pi j x_0}{b-a}$$

и аналогичными соотношениями для B'_j. Формулы для приближенного вычисления A_j и B_j при помощи дискретных преобразований Фурье отражают этот сдвиг исходной функции.

Пример 11.4. Приближенное вычисление коэффициентов Фурье функции Рунге (продолжение).

Вернемся к таблице из предыдущего примера и рассмотрим ее четвертый, пятый и шестой столбцы. Числа в этих столбцах были

[1] Здесь подразумевается, что $g(a) = g(b)$, а для косинуса величина $b-a$ является одним из периодов (не обязательно наименьшим).– *Прим. перев.*

получены при помощи правила трапеций с N точками, $N = 100$, 50 и 20. Заметим, что дискретное преобразование Фурье дает превосходное приближение величин A_j. □

При приближенном вычислении коэффициентов Фурье по квадратурным формулам величины коэффициентов зависят от числа точек, в которых вычисляются значения интегрируемой функции. Для правила трапеций это число равно N. Сколько коэффициентов Фурье можно вычислить приемлемым образом при заданном N? Уже отмечалось, что самый прямой способ заключается в том, чтобы взять одно и то же число точек для вычисления по правилу трапеций всех намеченных коэффициентов. Приближенные значения первых $N/2$ коэффициентов A_j и B_j получаются по N точкам при помощи дискретного преобразования Фурье. Что получится, если попытаться вычислить по тем же N точкам большее число коэффициентов? Получатся «ничего не стоящие» числа. Чтобы убедиться в этом, посмотрим на последний ($N = 9$) столбец в приведенной выше таблице. В нем стоят величины, полученные в результате применения формулы трапеций с $N = 9$ точками. Согласно формуле из § 2, первые пять элементов последнего столбца должны давать приближения к $A_0, \ldots, A_4$. Из таблицы видно, что первые четыре элемента дают приемлемые приближения, а следующие – нет. Строки с шестой по одиннадцатую занимают числа, противоположные по знаку числам из первых пяти строк. (Обсуждение элемента из пятой строки ($2.18574 \times \times 10^{-2}$) отложим до разд. 7.2.)

Почему так происходит? Чтобы объяснить это, снова посмотрим на формулу трапеций, записанную для $j = 5$:

$$A_5 \approx \frac{2}{N}\left[\frac{g(x_0)\cos(5\cdot 2\pi f x_0)}{2} + g(x_1)\cos(5\cdot 2\pi f x_1) + \ldots + \frac{g(x_N)\cos(5\cdot 2\pi f x_N)}{2}\right].$$

Когда мы пытаемся вычислить A_j для $j > N/2$, каждое слагаемое в формуле трапеций в точности противоположно некоторому члену из суммы в аналогичной формуле трапеций для коэффициентов Фурье с меньшим индексом $j' < N/2$. При $N = 9$ и $j = 5$ записанная выше сумма равна

$$A_5 \approx -\frac{2}{N}\left[\frac{g(x_0)\cos(4\cdot 2\pi f x_0)}{2} + \right.$$

$$\left. + g(x_0)\cos(4\cdot 2\pi f x_1) + \ldots + \frac{g(x_N)\cos(4\cdot 2\pi f x_N)}{2}\right] = -a_4.$$

Равенство этих сумм имеет место из-за того, что

$$\cos(5\pi[x_0 + 2k/9]) = -\cos(4\pi[x_0 + 2k/9]), \qquad k = 0, \ldots, 9.$$

Последнее равенство можно проверить при помощи тригонометрических тождеств. Дело обстоит так, как если бы по квадратурной формуле интегрировались разные функции, но совпадающие в узлах квадратуры (с точностью до знака). Иными словами, косинусоида с большей частотой $\cos(5\cdot 2\pi f x)$ принимает в узлах квадратуры те же значения (без

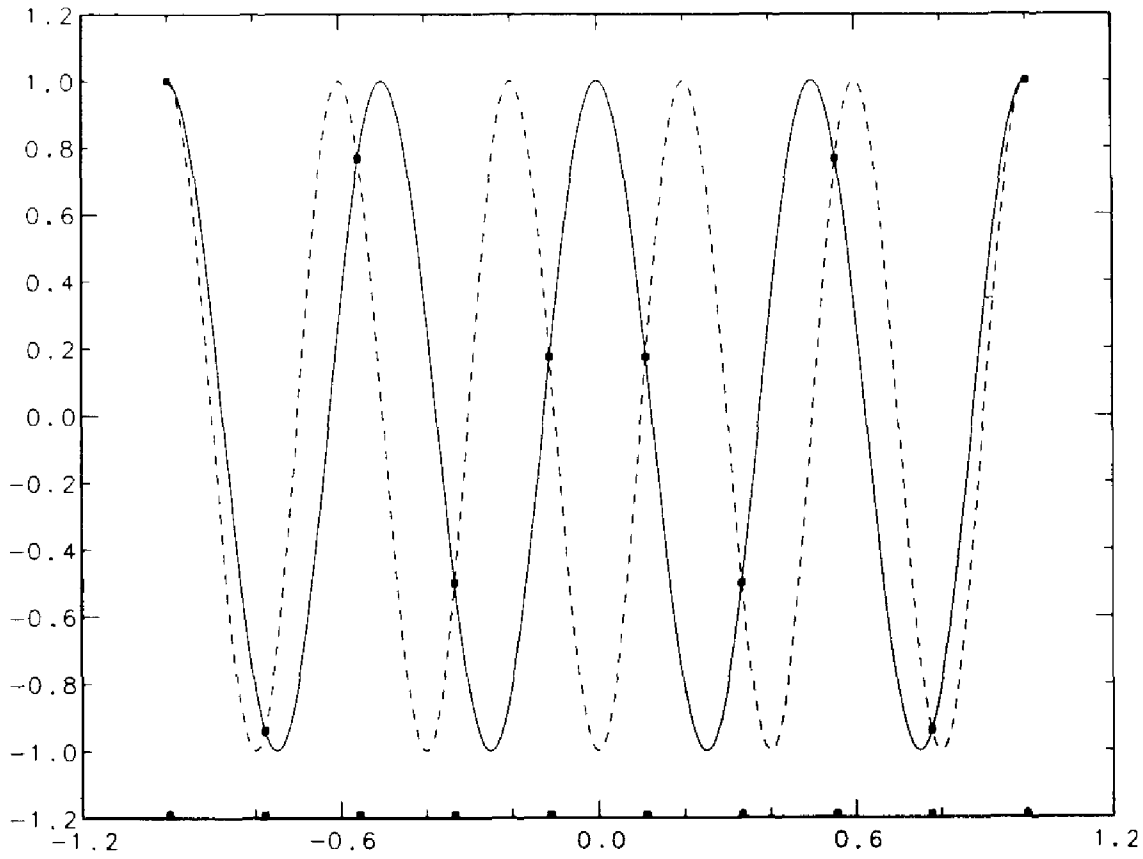


Рис. 11.4. Функции cos 4πx и — cos 5πx совпадают в 10 точках на отрезке [— 1, 1].

учета знака), что и косинусоида с меньшей частотой $\cos(4 \cdot 2\pi f x)$. Для этого явления статистик Джон Тьюки придумал новый термин: aliasing[1]). На рис. 11.4 изображены графики функций $cos(4\pi x)$ и $-\cos(5\pi x)$ на отрезке $[-1, 1]$. На рисунке отмечены десять равномерно распределенных на этом отрезке точек, в которых эти функции совпадают. Явление подмены обусловлено тем, что в точках равномерной сетки значения синуса (или косинуса) могут порождаться многими другими синусами с более высокими частотами. На этом принципе основана работа прибора, называемого стробоскопом, в котором можно увидеть кажущееся «замедление» быстро вращающегося или колеблющегося механизма. Чтобы найти другой пример, непременно прочитайте последнюю часть разд. 7.2.

Из этих рассуждений видно, что применение формулы трапеций, т. е. дискретного преобразования Фурье по N точкам, позволяет вычислять коэффициенты Фурье вплоть до отвечающего частоте Найквиста $1/(2\Delta)$ или периоду 2Δ.

Если увеличить число точек N, то можно вычислить большее количество коэффициентов Фурье, но *максимальная частота будет расти*

[1]) Слово alias (англ.) означает «вымышленное имя, прозвище». Придуманный Тьюки термин можно перевести как *подмену* частот.— *Прим. перев.*

только в том случае, если шаг сетки Δ *будет уменьшаться.* Как раз этой ситуации соответствует таблица из примера 11.3. Если аналоговый телефонный сигнал наблюдается с частотой 5000 наблюдений в секунду, то можно получить информацию только о тех присутствующих в сигнале частотах, которые не превосходят 2500 колебаний в секунду вне зависимости от продолжительности наблюдений с указанной частотой. В большинстве практических задач интересующая нас область значений частот известна, и частота наблюдений значительно более важна, чем число доступных для аппроксимации коэффициентов Фурье.

11.6. Подпрограммы EZFFTF и EZFFTB

Поскольку дискретное преобразование Фурье имеет много приложений, почти на всех компьютерах есть один или несколько пакетов для проведения этих преобразований. Быстрое преобразование Фурье (БПФ) – это алгоритм, который позволяет выполнять такие операции очень быстро. (В § 11.12 кратко описана работа этого алгоритма.) Некоторые фирмы продают даже специализированные устройства для БПФ, так что и БПФ, и родственные ему операции можно выполнять быстро, в режиме реального времени. Приводимые здесь подпрограммы являются частью пакета [Swarzthrauber, 1975]. Вот что это за программы:

EZFFTF – вычисляет дискретные синус- и косинус-преобразование заданных N вещественных чисел;

EZFFTB – вычисляет N вещественных величин по их заданным дискретным синус- и косинус-преобразованию Фурье (это обратное преобразование);

EZFFTI – инициализирующая программа, которую нужно вызвать один раз, прежде чем EZFFTF или EZFFTB смогут работать с новым значением N.

В пакете, из которого взяты указанные подпрограммы, содержатся также программы для работы с комплексными данными (см. разд. 11.1), программы, менее простые в обращении, но более гибкие, и программы для работы с данными, которые можно представить с помощью только синусов или только косинусов. Пакеты для БПФ часто требуют, чтобы N было целой степенью двойки; в этих же программах такого ограничения нет, но они работают наиболее эффективно, когда N является произведением большого числа небольших натуральных чисел.

Дадим сводку необходимых фактов из § 5.

(1) Если можно считать, что значения данных начинаются с абсциссы $x_0 = 0$, то дискретное преобразование Фурье, получаемое на выходе подпрограммы EZFFTF дает приближение для коэффициентов ряда Фурье периодической функции, которая порождает данные.

(2) В противном случае, если $x_0 \neq 0$, необходимы еще некоторые манипуляции с элементами получаемых на выходе массивов.

Чтобы более ясно показать связь между данными на выходе и на входе, начнем индексацию не с 0, а с 1, т.е. предположим, что дано N вещественных величин R(1), R(2), ..., R(N), отвечающих абсциссам на сетке с постоянным шагом Δ. Не предполагается, что первая и последняя из заданных величин совпадают, однако период исходной функции должен быть равен $N\Delta$, следовательно, $f = 1/(N\Delta)$. На выходе подпрограммы EZFFTF получается число AZERO и два массива A(J) и B(J), J = 1, ..., N/2, причем B(N/2) = 0, если N четно. Теперь можно записать тригонометрический полином степени $N/2$, дающий приближение к частичной сумме ряда Фурье исходной функции:

$$t_{N/2}(x) = \text{AZERO} + \sum_{j=1}^{N/2} (\text{A(J)}\cos(j2\pi f x_0) - \text{B(J)}\sin(j2\pi f x_0))\cos j2\pi f x + \\ + (\text{A(J)}\sin(j2\pi f x_0) + \text{B(J)}\cos(j2\pi f x_0))\sin j2\pi f x.$$

(3) Числа A(J) и B(J) соответствуют частоте, в J раз большей, чем основная частота, т.е. $J \times 1/(N\Delta)$.

Приведенная ниже программа показывает, как пользоваться подпрограммой EZFFTF. Вычислим приближенные коэффициенты Фурье функции Рунге на отрезке [−1, 1] по 16 и по 17 заданным точкам, т.е. найдем приближенные значения для $A_0, \ldots, A_8$ и для $B_1, \ldots B_7$, а также для B_8 в случае $N = 17$. Кроме этого программа вычислит приближения для значений ряда Фурье в 101 точке между 0 и 1. Именно эта программа и использовалась с разными значениями N при построении таблицы в примере 11.3 и графиков на рис. 11.3.

```
C Применение действительного дискретного преобразования Фурье для
C вычисления приближенных коэффициентов Фурье функции Рунге на
C [−1, 1] с N = 16 и N = 17.
C
      PARAMETER (MCOEF = 17)
      REAL A(MCOEF/2), B(MCOEF/2), R(MCOEF),
     WSAVE(3 * MCOEF + 15)
      REAL DFTA (MCOEF/2), DFTB(MCOEF/2), C(MCOEF/2),
     S(MCOEF/2)
C
C Арифметическая функция-оператор для вычисления значений функ-
C ции Рунге.
      RUNGE(X) = 1.0/(1 + 25.0 ** X * X)
C
C
      X0 = − 1.0
      PI = ASIN(1.0) * 2.0
C
      DO 10 N = MCOEF − 1, MCOEF
         CALL EZFFTI (N, WSAVE)
```

```
C Функция предполагается периодической, заданной на отрезке [-1, 1]
C длины 2.
      DEL = 2.0/N
      F = 2.0 * PI/(N * DEL)
      DO 1 J = 1, N
C Первое значение функции наблюдается в точке -1, последнее- в
C точке 1-DEL.
            HX = (- 1.0) + (J - 1) * DEL
            R(J) = RUNGE(XJ)
C Вычисление синусов и косинусов для получения приближенных коэф-
C фициентов Фурье с использованием выходных данных EZFFTF.
            IF   (J.LE.N/2) THEN
                 C(J) = COS(J * F * X0)
                 S(J) = SIN(J * F * X0)
            END IF
    1 CONTINUE
      CALL EZFFTF (N, R, AZERO, A, B, WSAVE)
C
C Ради удобства записи пусть этот цикл выполняется до N/2.
C Если N четное, то последний элемент B равен нулю.
      DO 11 1 = 1, N/2
           DFTA(J) = A(J) * C(J) - B(J) * S(J)
           DFTB(J) = A(J) * S(J) + B(J) * C(J)
   11 CONTINUE
      WRITE(*,*) 'RESULTS FOR N = ', N'AZERO = ', AZERO
      WRITE(*,*) 'J DFTA(J) DFTB(J)'
      DO 12 J = 1, N/2
           WRITE(*,*) J, DFTA(J), DFTB(J)
   12 CONTINUE
      M = 101
C
C Вычисление интерполянта в 101 точке на [-1, 1].
      WRITE(*,*) 'RESULTS FOR N =', N
      DO 20 K = 1, M
           X = - 1.0 + 2.0 * (K - 1.0)/(M - 1.0)
           TN = AZERO
   DO 19 J = 1, N/2
        TN = TN + DFTA(J) * COS(J * F * X) + DFTB(J) * SIN(J * F * X)
     19          CONTINUE
                 ER = TN - RUNGE(X)
                 WRITE(*,*)X, TN, ER
     20 CONTINUE
C
           WRITE(*,*)
C
     10 CONTINUE
```

```
      STOP
      END
C
C Будут получены следующие результаты (результаты вычислений в
C 101 точке опущены):
C
C          RESULTS FOR N = 16 AZERO = 0,274611
C                  J    DFTA(J)          DFTB(J)
C                  1    0.344365       - 0.100161E-07
C                  2    0.175654       - 0.307123E-07
C                  3    0.972947E-01   - 0.203292E-07
C                  4    0.501320E-01   - 0.175307E-07
C                  5    0.285903E-01   - 0.176853E-07
C                  6    0.149957E-01   -0.786582E-08
C                  7    0.105194E-01     0.327562E-08
C                  8    0.383778E-02   - 0.268407E-08
C
C
C          RESULTS FOR N = 17 AZERO = 0.274581
C                  J    DFTA(J)          DFTB(J)
C                  1    0.344243       - 0.350539E-07
C                  2    0.175520       - 0.364332E-07
C                  3    0.969902E-01   - 0.295525E-07
C                  4    0.496441E-01   - 0.205552E-07
C                  5    0.275968E-01   - 0.149523E-07
C                  6    0.132331E-01   - 0.103949E-07
C                  7    0.709945E-02   - 0.110340E-07
C                  8    0.141324E-02   - 0.116999E-07
```

11.7. Аппроксимация частичными суммами ряда Фурье

В некоторых приложениях важно иметь значения ряда Фурье, в других – только его коэффициенты. Отбрасывание остатка ряда Фурье приводит к погрешности, а приближенное вычисление коэффициентов ряда при помощи дискретного преобразования Фурье вносит дополнительные ошибки. Пример 11.3 показывает, что частичные суммы ряда с точными коэффициентами хорошо аппроксимируют исходную функцию. В этом параграфе объясняется, как именно частичные суммы ряда Фурье аппроксимируют исходную функцию $g(x)$, которая считается периодически продолженной с фиксированного интервала $[a, b]$.

11.7.1. Случай точных коэффициентов Фурье

Что означает выражение «бесконечный ряд Фурье равен $g(x)$»? Это значит, что на $[a, b]$ частичные суммы все ближе и ближе подходят

к $g(x)$, когда мы суммируем все больше и больше членов ряда, хотя для любого конкретного x частичные суммы могут никогда не совпасть с $g(x)$. Иными словами, частичные суммы ряда сходятся к $g(x)$. В инженерных задачах эта сходимость имеет место во всех точках x, в которых функция $g(x)$ является «гладкой». Там, где g быстро изменяется, ряд сходится медленно. В отличие от приближений некоторыми другими рядами, ряд Фурье часто можно дифференцировать (и неоднократно) для разложения в ряды производных его суммы. Суммы T_n стремятся к g с ростом n. В дополнение к этому, при каждом фиксированном n сумма T_n обладает еще одним важным свойством. Если $T_n(x)$ считать произвольным тригонометрическим полиномом, содержащим члены вплоть до $\cos(n2\pi fx)$ и $\sin(n2\pi fx)$, то любой выбор его коэффициентов, отличных от коэффициентов Фурье, приведет к худшему приближению функции g *в определенном смысле*. В этом смысле (описанном ниже) частичные суммы ряда Фурье дают наилучшее приближение к g. Точный смысл выражения «наилучшее приближение» таков. Мерой среднего отклонения T_n от g является *среднеквадратическое отклонение*

$$\int_a^b [g(x) - T_n(x)]^2 \, dx.$$

Это отклонение, зависящее от величин коэффициентов A_i и B_i, будет наименьшим, когда этими коэффициентами являются коэффициенты Фурье функции g. Это можно интерпретировать следующим образом: в среднем частичные суммы ряда Фурье хорошо аппроксимируют гладкие периодические функции, а насколько хорошо – зависит от g. Грубо говоря, «чем глаже, тем лучше», т. е., чем более гладкой является функция g, тем быстрее сходятся к нулю коэффициенты Фурье и, вероятно, тем лучшее приближение к g дает T_n. В частности, это означает, что если функция $g(x)$ будет гладкой при $x = a$ и $x = b$, то сходимость будет лучше. Например, функция $g(x) = \exp(-x^2)$, заданная на отрезке $[a, b] = [-5,5]$, гладко примыкает к своему периодическому продолжению. С другой стороны, если g периодически продолжить с отрезка $[a, b] = [0,5]$, то в каждом конце отрезка это периодическое продолжение будет иметь скачок. На первом из двух указанных отрезков ряд Фурье сходится намного быстрее, чем на втором.

11.7.2. Случай приближенных коэффициентов Фурье

Пусть $T_n(x)$ – частичная сумма ряда Фурье с точными коэффициентами, а $t_n(x)$ – аналогичная частичная сумма с коэффициентами, полученными применением дискретного преобразования Фурье. В последнем случае подразумевается, что использовано $N = 2n + 1$ точек. Очевидны два свойства этих тригонометрических многочленов:

(1) Если известно T_n, то для вычисления T_{n+1} нужно только два новых коэффициента: A_{n+1} и X_{n+1}, но оба представляют собой

интегралы. Для вычисления t_{n-1} требуется задать две дополнительные точки и пересчитать заново все коэффициенты (a_0, ... a_{n+1}, b_1, b_{n+1}), которые являются суммами.

(2) Нет причин ожидать, что t_n будет удовлетворять тому же условию минимальности среднеквадратического отклонения, которому удовлетворяет T_n. Интуиция подсказывает, что t_n может и не быть таким же хорошим приближением к g, каким является T_n. В левой части рис. 11.3 изображен график t_8 с коэффициентами, посчитанными по формуле трапеций с 17 узлами. Получается, что t_8 дает приближение совершенно такого же качества, что и точная частичная сумма ряда Фурье. Замечательно, что построенный таким образом ряд сходится к функции g так же быстро, как и обычный ряд Фурье. В данном пункте мы попытаемся объяснить, почему так получается.

Можно заменить условие минимальности среднеквадратического отклонения на его дискретный аналог, используя вместо интеграла

$$\sum_{k=0}^{N-1} [g(x_k) - t_n(x_k)]^2$$

сумму по всем N точкам. Можно показать, что минимум этой суммы достигается, когда коэффициенты в t_n получаются в результате дискретного преобразования Фурье. Таким образом, в указанном смысле t_n тоже дает наилучшее приближение. На самом деле минимум суммы квадратов равен нулю, поскольку

$$g(x_k) = t_n(x_k), \quad k = 0, ..., N-1.$$

Используя терминологию гл. 4, можно сказать, что функция t_n интерполирует $g(x)$ в узлах равномерной сетки x_k. Функция T_n – с точными коэффициентами Фурье – также обладает свойством интерполянта, но не со столь предсказуемым набором точек интерполяции.

Это можно ясно увидеть в случае $x_0 = 0$, записав

$$T_{N/2}(x) = \frac{A_0}{2} + \sum_{j=1}^{N/2} A_j \cos(2\pi jx/(b-a)) + B_j \sin(2\pi jx/(b-a)).$$

Теперь, если $x_0 = 0$, то

$$t_{N/2}(x) = a_0 + \sum_{j=1}^{N/2} a_j \cos(2\pi jx/(b-a)) + b_j \sin(2\pi jx/(b-a)) =$$

$$= a_0 + \sum_{j=1}^{N/2} a_j \cos(2\pi jx/(N\Delta)) + b_j \sin(2\pi jx/(N\Delta)).$$

Если положить $x = x_k = k\Delta$, то

$$t_{N/2}(x_k) = g_k = g(x_k)$$

в силу формулы дискретного преобразования Фурье, см. пункт (2) из § 2.

Это показывает, что $t_{N/2}$ интерполирует $g(x)$ в точках x_k.

Вспомним, что увеличение числа точек интерполяции не всегда приводит к хорошему результату. В случае полиномиальной интерполяции добавление новой точки часто ухудшает поведение интерполянта, но для тригонометрической интерполяции это не так. Рассмотрим частичную сумму $t_m(x)$ с $m < n$ членами и будем по-прежнему измерять качество аппроксимации при помощи дискретного среднеквадратического отклонения по исходным $N = 2n + 1$ точкам. Это традиционная задача о наименьших квадратах в смысле гл. 6 с синусами и косинусами в качестве модельных функций. Замечательно, что решением этой задачи, т. е. коэффициентами в t_m, являются числа $a_0, a_1, \ldots, a_m$ и $b_1, b_2 \ldots \ldots, b_m$, в точности совпадающие с первыми m коэффициентами t_n. Это происходит из-за того, что нормальные уравнения в этой задаче (которыми обычно не рекомендуют пользоваться) составляют диагональную систему, которую можно сразу решить. Конечно, t_m не является интерполянтом, за исключением случая $m = n$, но, поскольку это решение получено методом наименьших квадратов, оно должно давать приемлемое приближение. Таким образом, интерполяция и метод наименьших квадратов работают вместе.

Чтобы помочь объяснить, почему t_n является хорошим приближением к $g(x)$, можно воспользоваться законом сохранения энергии из § 3. Известно, что

$$\frac{b-a}{N}\sum_{k=0}^{N-1} g_k^2 \approx \int_a^b [g(x)]^2\,dx.$$

Если внимательно посмотреть на формулы из § 3, выражающие закон сохранения энергии, то можно увидеть, что записанная выше сумма равна

$$(b-a)\left(a_0^2 + \frac{1}{2}\sum_{k=1}^{N/2}(a_k^2 + b_k^2)\right),$$

в то время как интеграл в правой части равен

$$(b-a)\left(\left(\frac{A_0}{2}\right)^2 + \frac{1}{2}\sum_{k=1}^{\infty} A_k^2 + B_k^2\right).$$

Поскольку они приблизительно совпадают, общая энергия, распределенная по всем частотам, должна как-то перераспределиться в случае приближенных коэффициентов. Можно показать, что при $x_0 = 0$ имеют место следующие равенства:

$$a_s = A_s + A_{N+s} + A_{N-s} + A_{2N+s} + A_{2N-s} + \ldots,$$
$$b_s = B_s + B_{N+s} - B_{N-s} + B_{2N+s} - B_{2N-s} + \ldots, \quad s \leqslant N/2.$$

Это аналитическое выражение понятия «подмены»: приближенный коэффициент Фурье возмущается точными коэффициентами Фурье, соответствующими более высоким частотам. Говорят, что высшие частоты

$(N + s)f$, $(N - s)f$ и т. д. «подменяются» частотой sf. Различие между двумя коэффициентами A_s и a_s возникает из-за присутствия высших гармоник, которыми в обычных частичных суммах ряда Фурье T_n просто пренебрегают, в то время как в t_n они учитываются. Процесс отбора, которому фактически соответствует вычисление дискретного преобразования Фурье, превращает высокочастотное колебание в низкочастотное, и любой коэффициент Фурье с индексом, превосходящим $N/2$, влияет на коэффициент a_s, отвечающий некоторой низшей частоте.

Это можно сформулировать и в более практическом стиле. Ряд Фурье представляет собой бесконечный ряд. Если оборвать суммирование на N-м члене без поправки коэффициентов, то какая-то информация о функции g будет потеряна. Приближенные коэффициенты позволяют простым и однозначным образом сохранить некоторую часть этой информации, что и показывают записанные выше формулы. Вообще, если взять конечную дискретную последовательность наблюдений, которые являются значениями непрерывной функции, произойдет потеря какой-то информации о функции. Частично это вызвано тем, что такая дискретная последовательность имеет конечное число членов. Преимущество тригонометрических функций состоит в том, что потеря информации вследствие дискретизации проявляется в форме подмены частот.

С помощью этих же формул можно завершить объяснение разницы между $A_4 = 5.00138 \cdot 10^{-2}$ и $a_4 = 2.18574 \cdot 10^{-2}$ в примере 11.3 при $N = 9$. Для этого формулы надо видоизменить с учетом того факта, что $x_0 = -1$. Тогда можно показать, что указанные два элемента таблицы связаны соотношением

$$a_4 = 2.18574 \cdot 10^{-2} = A_4 - A_{N-4} - A_{N+4} + \ldots \approx$$
$$\approx A_4 - A_5 = 5.00138 \cdot 10^{-2} - 2.77280 \cdot 10^{-2} = 2.22858 \cdot 10^{-2}.$$

Другими словами, происходит подмена первой пропущенной частоты 5/2 частотой 4/2. Более низкие частоты тоже подвержены такому влиянию, но не столь сильно. Например, оценка постоянного члена по правилу трапеций дает

$$a_4 = 5.44373 \cdot 10^{-1} = A_0 - 2A_N - 2A_{2N} - \ldots \approx$$
$$\approx A_0 - 2A_9 = 5.49360 \cdot 10^{-1} - 2 \cdot 2.38204 \cdot 10^{-3} = 5.44596 \cdot 10^{-1}.$$

Если периодическая функция $g(x)$ имеет большие по абсолютной величине коэффициенты Фурье с индексами выше $N/2$, то между A_s и a_s будет заметное различие. В этом случае частичные суммы ряда Фурье не будут хорошими приближениями к g независимо от того, точные или приближенные используются коэффициенты. Но если N настолько велико, что остаток ряда мал, то высокие гармоники должны иметь небольшие и быстро уменьшающиеся амплитуды, а разность между A_s и a_s, видимо, будет невелика.

Пример подмены частот показан на рис. 11.5. На рис. 11.5(а) показано 36 наблюдаемых значений функции $\sin(\pi x/4)$, $x = 0, \ldots, 35$. Синусоида нарисована, чтобы показать, как были получены эти значения,

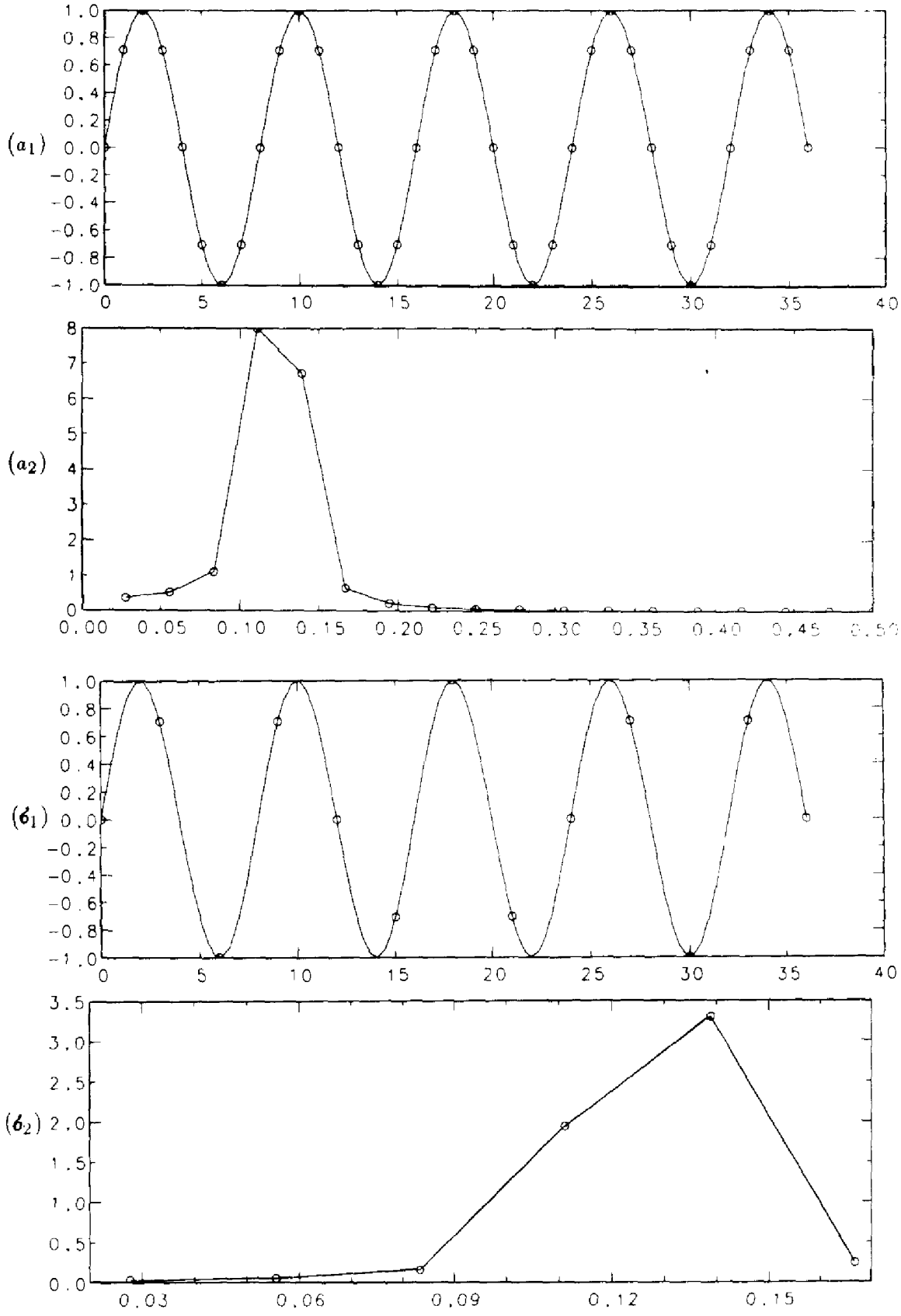


Рис. 11.5. Подмена высокой частоты низкой частотой.

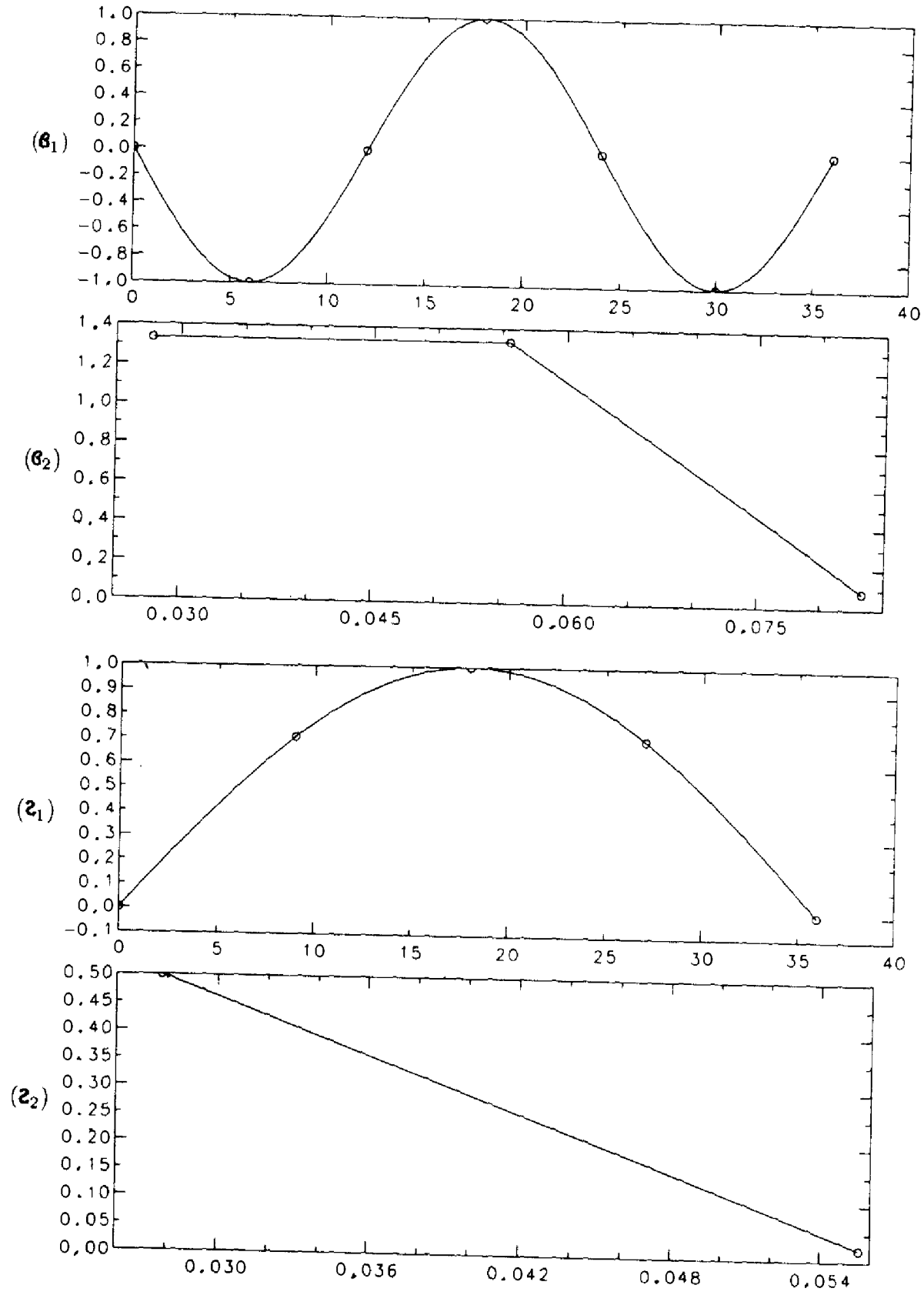


Рис. 11.5. (окончание).

и имеют частоту 1/8. Вычислим с помощью подпрограммы EZFFTF дискретное преобразование Фурье и периодограмму для этих величин; периодограмма изображена непосредственно под графиком данных. Длина записи здесь составляет $N\Delta = 36$, так что основная частота равна 1/36, а частота Найквиста $1/_{(2\Delta)} = 1/2$. Обратите внимание на масштаб по горизонтальной оси периодограммы: ее значения изображены при частотах, кратных основной частоте, вплоть до частоты Найквиста 1/2. Функция $\sin(\pi x/4)$ имеет частоту 1/8; это меньше чем 1/2, и поэтому на периодограмме видны большие величины значений при частотах около 1/8. Поскольку частота 1/8 точно на периодограмме не представлена, велик вклад от частот вблизи 1/8. На рис. 11.5(б) показаны результаты аналогичных вычислений, но с 12 точками, заданными на сетке с шагом $\Delta = 3$. Частота Найквиста 1/6 по-прежнему больше, чем частота исходной функции, и поэтому периодограмма снова имеет большие значения возле 1/8. На рис. 11.5(в) шаг сетки $\Delta = 6$ и $N = 6$. Теперь частота Найквиста 1/12 меньше, чем частота заданного синуса. Происходит подмена высокочастотных данных более низкой частотой, т. е. те же самые данные могут порождаться синусоидой с меньшей частотой, а именно функцией $-\sin(2\pi x/24)$, которая и нарисована проходящей через точки данных. Вследствие этого в периодограмме присутствуют большие величины, которые отвечают двум частотам (1/36 и 2/36), находящимся рядом с 1/24. На рис. 11.5(г) $\Delta = 9$, $N = 4$ и частота Найквиста 1/18 меньше частоты первоначальной функции. Здесь данные тоже могут порождаться синусоидой с меньшей частотой $\sin(2\pi x/72)$, которая на рисунке проходит через точки данных. Периодограмма имеет большое по величине значение при частоте 1/36, ближайшей к 1/72.

Выводы:

(1) Дискретное преобразование Фурье дает приближенные коэффициенты ряда Фурье. Если используется N точек, то $a_s \approx A_s$, $b_s \approx B_s$ для $s \leqslant N/2$.

(2) Коэффициенты приближенной частичной суммы t_n подвержены влиянию всех коэффициентов Фурье. Вследствие этого t_n часто содержит больше информации, чем T_n.

(3) Приближенная частичная сумма t_n является интерполянтом по $2n + 1$ точкам на сетке по x с постоянным шагом; сумма T_n не обязана давать такую интерполяцию.

(4) Точные и приближенные частичные суммы T_n и t_n являются наилучшими приближениями к $g(x)$ по интегральному критерию среднеквадратического отклонения и в смысле наименьшей суммы квадратов соответственно.

Конкретная иллюстрация вывода (2) возникает, когда мы видим в кино движущийся вагон, спицы колес которого кажутся движущимися в замедленном темпе, остановившимися или даже вращающимися назад. Вращение колеса можно описать высокочастотной синусоидой: если

отметить на колесе точку, то ее положение в зависимости от времени описывается функцией $g(x) = A_j \cos(j 2\pi f x)$ с некоторым большим j. Ряд Фурье функции g имеет только один ненулевой член. Частичные суммы ряда Фурье $T_n(x)$ (с точными коэффициентами) тождественно равны нулю при $n < j$. Кинокамера делает снимки значений $g(x)$ примерно через 1/24 с. По утверждению Найквиста, невозможно найти какой-либо коэффициент Фурье, отвечающий частоте, большей чем двенадцать оборотов в секунду. Но из-за подмены частот дискретизация возмущает приближенные коэффициенты Фурье для значительно более низких частот. Приближенная частичная сумма содержит долю этой высокочастотной информации. Поскольку мы «видим» в кино вращение колеса, наш мозг, вероятно, также снимает эти дискретные величины и формирует ту же приближенную частичную сумму, даже если значения между точками данных вызывают удивление.

11.8. Связь между преобразованиями Фурье и рядами Фурье

В § 7 была показана связь между дискретным преобразованием Фурье и рядом Фурье. Теперь мы хотим показать, как связаны ряды Фурье и интегральные преобразования Фурье. Здесь важной идеей является то, что интегральное преобразование Фурье представляет собой ряд Фурье функции с очень большим (бесконечным) периодом. Продемонстрируем это, рассматривая заново пример 11.3 и изучая эффект изменения периода при расширении интервала $[a, b]$ от $[-1, 1]$ до $[-2, 2]$. Период функции при этом удваивается. Изменяются и частоты членов ряда Фурье: теперь шкала частот имеет вдвое меньший шаг, поскольку основная частота уменьшилась в два раза с 1/2 до 1/4. В некотором смысле новый ряд Фурье имеет «вдвое больше» членов, чем старый. Первые два столбца приведенной ниже таблице, в сущности, повторяют второй и третий столбцы таблицы из примера 11.3. После расширения интервала в новой таблице коэффициентов Фурье должны быть отражены вклады от всех перечисленных частот, включая те, которым соответствуют пропуски во втором столбце.

Как можно определить функцию на расширении интервала, т. е. при $-2 \leqslant x \leqslant -1$ и $1 \leqslant x \leqslant 2$? Один из способов заключается в том, чтобы взять там значение функции Рунге, другой – в том, чтобы доопределить функцию нулем. Второй способ особенно прост, так как можно связать новые коэффициенты Фурье со старыми, отвечающими тем же частотам. Например, если A_5 – старый коэффициент, соответствующий частоте $5 \cdot {}^1/_2$, то новый коэффициент с индексом 10 соответствует той же самой частоте $10 \cdot {}^1/_4$. Этот новый коэффициент вычисляется по формуле

$$\frac{2}{4}\int_{-2}^{2} \cos(10 \cdot 2\pi x/4) g(x) dx = \frac{1}{2}\int_{-1}^{1} \cos(5x\pi) g(x) dx = \frac{A_5}{2}$$

(вспомните, что функция g равна нулю на расширении интервала), т.е. в точности равен половине старого. Для того чтобы элементы таблицы можно было сравнивать, надо пронормировать их в соответствии с отвечающими им интервалами изменения частот, т.е. разделить элементы второго столбца на π, а третьего - на $\pi/2$. Вот тогда в обоих столбцах величины, отвечающие одинаковым частотам, совпадут. Разумеется, при промежуточных частотах сравнивать будет нечего. В третьем столбце приводятся величины, аналогичные перечисленным во втором столбце; они найдены с помощью программы Q1DA. Наибольшая из указанных частот, 4, в восемь раз больше основной частоты для интервала $[-1, 1]$ и в шестнадцать раз больше основной частоты для интервала $[-2, 2]$. Заметим, что шаг по частоте не изменился бы, если бы приводилось больше членов.

Таблица 11.2 Сравнение коэффициентов Фурье для двух интервалов

ω	$1/\pi$ x коэффициент для $[-1, 1]$	$2/\pi \times$ коэффициент для $[-2, 2]$ при задании функции нулем вне $[-1, 1]$	$2/\pi \times$ коэффициент для $[-2, 2]$ при задании функций как $R(x)$ на $[-2, 2]$
0/4	1.7487×10^{-1}	1.7487×10^{-1}	1.8731×10^{-1}
1/4		1.5395×10^{-1}	1.4755×10^{-1}
2/4	1.0953×10^{-1}	1.0953×10^{-1}	1.0618×10^{-1}
3/4		7.3552×10^{-2}	7.8184×10^{-2}
4/4	5.5943×10^{-2}	5.5943×10^{-2}	5.6774×10^{-2}
5/4		4.4470×10^{-2}	4.1673×10^{-2}
6/4	3.0846×10^{-2}	3.0846×10^{-2}	3.0299×10^{-2}
7/4		2.0043×10^{-2}	2.2231×10^{-2}
8/4	1.5920×10^{-2}	1.5920×10^{-2}	1.6162×10^{-2}
9/4		1.3521×10^{-2}	1.1864×10^{-2}
10/4	8.8261×10^{-3}	8.8261×10^{-3}	8.6178×10^{-3}
11/4		4.9203×10^{-3}	6.3334×10^{-3}
12/4	4.4820×10^{-3}	4.4820×10^{-3}	4.5934×10^{-3}
13/4		4.5517×10^{-3}	3.3826×10^{-3}
14/4	2.5551×10^{-3}	2.5552×10^{-3}	2.4470×10^{-3}
15/4		7.6741×10^{-4}	1.8078×10^{-3}
16/4	1.2389×10^{-3}	1.2389×10^{-3}	1.3024×10^{-3}

Если задавать g на расширении интервала как функцию Рунге, то нельзя ожидать точного совпадения на одинаковых частотах, но все же числа не должны отличаться очень сильно. Эти величины показаны в четвертом столбце.

Обобщая эту идею, надо устремить $[a, b]$ к $(-\infty, \infty)$. Каждый новый ряд Фурье будет при этом содержать все больше и больше членов, все

плотнее заполняя область изменения частот. В пределе основная частота стремится к нулю, а ряд Фурье, представляющий собой сумму по целым кратным основной частоты, превращается в интеграл Фурье по всем возможным частотам. Интегралы по $[a, b]$, задающие коэффициенты Фурье, превращаются в интегралы по $(-\infty, \infty)$; при этом индекс k, ранее обозначавший k-й коэффициент Фурье, заменяется на непрерывно меняющуюся переменную ω. Если продолжить функцию нулем вне $[a, b]$, то видно, что *пронормированные* коэффициенты Фурье при частотах $k \cdot 1/(b-a)$ совпадают во всех последующих рядах и, следовательно, должны быть равны значению предельной функции – интегрального преобразования Фурье, т. е.

$$A_k \cdot (b-a) = A\left(\frac{k}{b-a}\right) \text{ и } B_k \cdot (b-a) = B\left(\frac{k}{b-a}\right).$$

Следовательно, коэффициенты Фурье можно рассматривать как частные значения интегрального преобразования Фурье, взятого от функции, которая равна нулю вне $[a, b]$. Еще раз сформулируем этот факт: для функции, обращающейся в нуль вне интервала $[a, b]$, коэффициенты ряда Фурье, отвечающие одинаковым частотам, совпадают с точностью до множителя $b-a$. В этом случае дискретные преобразования Фурье a_k, b_k дают приближенные значения преобразований Фурье: $a_k(b-a) \approx A(k/(b-a))$, $b_k(b-a) \approx B(k/(b-a))$. Хорошим примером такой функции является $\exp(-x^2)$ на интервале $[a, b] = [-5, 5]$.

* 11.8.1. Функции, не имеющие преобразований Фурье или ряда Фурье

Не всякую функцию можно записать в виде интеграла Фурье. В § 2 мы не задавались вопросом, какие функции имеют интегральные преобразования Фурье. Проще всего увидеть, что в формулах для энергии фигурируют несобственные интегралы от $g^2(x)$, $A^2(\omega)$ и $B^2(\omega)$, поэтому для существования интегралов эти функции должны стремиться к нулю при больших $|x|$ и $|\omega|$. Такому требованию не удовлетворяют периодические функции вроде $g(x) = \sin x$ и даже функция, тождественно равная константе. В обычном смысле их интегральные преобразования Фурье не существуют, но это понятие столь полезно, что было дано его обобщение и на указанные случаи.

Что, например, надо понимать под интегральным преобразованием Фурье функции $\sin 2\pi\alpha x$? Интуитивно мы представляем себе интегральное преобразование Фурье как сумму по всем возможным частотам. При этом вклад от частоты α равен единице, а от остальных частот – нулю. Другой подход заключается в том, чтобы изучить поведение интегралов, определяющих $A(\omega)$ и $B(\omega)$, когда они вычисляются по большим, но конечным интервалам, а затем устремить длину интервала к бесконечности. Поскольку $\sin 2\pi\alpha x$ – нечетная функция, преобразование $A(\omega)$

будет равно нулю. Легко вычислить

$$B(\omega) \approx 2 \int_{-U}^{U} \sin(2\pi\alpha s) \sin(2\pi\omega s) ds = \frac{\sin(2\pi U[\alpha - \omega])}{\pi[\alpha - \omega]} - \frac{\sin(2\pi U[\alpha + \omega])}{\pi[\alpha + \omega]}.$$

Это нечетная функция от ω, которая имеет пики высоты примерно $\pm 2U$[1]) при $\omega \approx \pm\alpha$ и исчезающе мала при больших ω. При $U \to \infty$ получаем «функцию», равную нулю всюду, за исключением точек $\pm\alpha$, в которых она обращается в $\pm\infty$. Обычно это формулируют так: интегральным преобразованием Фурье функции $\sin(2\pi\alpha x)$ является пара δ-функций в точках $\pm\alpha$. Аналогично, любая *линейная* комбинация тригонометрических функций имеет интегральное преобразование Фурье в виде линейной комбинации δ-функций.

Точно так же не для любой функции существует ряд Фурье. Например, функция Рунге непериодична. Записанный для нее в примере 11.3 ряд Фурье сходится к $R(x)$ на интервале $[-1, 1]$, но больше нигде. Нас обычно интересуют функции, заданные на конечной области, и нам безразлично, что происходит вне этой области. С такой ситуацией можно справиться тремя связанными между собой способами:

(1) Предположить, что изучаемая функция тождественно равна нулю вне интересующей нас области или очень быстро стремится там к нулю. Такая функция будет иметь интегральное преобразование Фурье, но не будет иметь ряда Фурье.

(2) Предположить, что функция периодична. Такая функция будет иметь ряд Фурье, но не будет иметь интегральнсго преобразования Фурье (в обычном смысле).

(3) Предположить, что функция определена на всей прямой от $-\infty$ до ∞, но нам доступно лишь ее произведение с другой функцией, равной единице на интересующем нас интервале и нулю вне его. Такая функция со значениями нуль или единица называется *окном*.

Сравнение этих трех подходов выходит за рамки нашего учебника. Заинтересованных читателей отсылаем к книге [Weaver, 1983].

Есть два вида функций, которые играют важную роль в понимании свойств рядов и преобразований Фурье,–это финитные по времени функции и функции с ограниченной полосой частот. *Финитные по времени* функции отличны от нуля лишь в некотором конечном ин-

---

[1]) Точное значение интеграла при $\omega = \pm\alpha$ равно $B(\omega) = 2V - \frac{1}{2\pi\alpha}\sin(4\pi\alpha V)$.– *Прим. перев.*

тервале. (Если аргументом функции является пространственная переменная, то отличную от нуля лишь в конечном интервале функцию называют *финитной по пространству*.) У финитной по времени функции не существует ряда Фурье, поскольку она непериодична. Функция *с ограниченной полосой частот* – это функция, у которой интегральные преобразования Фурье $A(\omega)$ и $B(\omega)$ не равны нулю только на некотором конечном интервале, т. е. $A(\omega) = B(\omega) = 0$ при $|\omega| > R > 0$. В этом случае говорят, что $g(x)$ имеет *ширину полосы* 2Ω. Существует два основных теоретических результата, относящихся к таким функциям (за подробностями вновь отсылаем читателя к книге [Weaver, 1983]).

(1) Принцип неопределенности Гейзенберга: функция не может одновременно быть финитной по времени и иметь ограниченную полосу частот. Если существует конечный интервал, вне которого функция g равна нулю, то не может быть такого конечного интервала для ее преобразования Фурье, и наоборот. Во вводных курсах физики это утверждение формулируют следующим образом: невозможно определить с абсолютной точностью положение и скорость атомной частицы одновременно.

(2) Теорема о наблюдениях: если для функции с ограниченной полосой частот известны ее значения во всех точках $k/(2\Omega)$, $k = -\infty, \ldots, \infty$, то (теоретически) можно найти ее значения при *всех* x. Частота наблюдений $1/(2\Omega)$ обеспечивает два наблюдаемых значения за период, отвечающий наивысшей из присутствующих частот. Как отмечалось ранее, она называется *найквистовой частотой наблюдений*, а последовательность получаемых с такой частотой значений – *выборкой Найквиста*.

В большинстве случаев мы не можем управлять частотой выборки значений функции. Но утверждение (2) означает, что если наблюдения отстоят друг от друга на 2Ω, т. е. производятся с частотой $1/(2\Omega)$, то можно полностью восстановить любую функцию, у которой наивысшая частота равна Ω. Значение этой идеи очень велико. Она допускает простое приложение к передаче телефонных сигналов. Предположим, телефонная компания хочет обеспечить передачу звуковых сигналов вплоть до частоты 2500 колебаний в секунду. В соответствии с теоремой о наблюдениях сигнал от источника можно кодировать 5000 дискретными точками в секунду и при этом точно восстанавливать его на выходе. Это значит, что не обязательно передавать полный аналоговый сигнал от источника к приемнику, надо передавать только 5000 его значений в секунду. Это может показаться огромным объемом информации, но на самом деле линия связи остается незанятой большую часть каждой секунды. Разумеется, линия может при этом использоваться для передачи другого сообщения.

Если наблюдения отстоят друг от друга на 2Ω, то о частотах выше

Ω ничего сказать нельзя. Если все же попытаться сделать это, то получатся неверные результаты. Мы уже сталкивались с этой ситуацией в примере 11.3 (см. последний столбец таблицы).

11.9. Применение метода наименьших квадратов: модель el Niño

Вспомним, что в ряды Фурье раскладываются функции, заданные на интервале (см. § 7). Как только мы переходим к оценкам коэффициентов по правилу трапеций, ряд становится зависящим лишь от дискретного набора данных. Частичная сумма $t_n(x)$ с коэффициентами, вычисленными по $(2n + 1)$-точечной формуле трапеций, интерполирует исходные данные. Как вы уже видели, в некотором смысле она более полезна, чем «обычный» ряд, и не следует думать о ней только как о способе аппроксимации – это полноправная математическая конструкция. Например, в разд. 7.2 было объяснено, что если отбросить некоторые высокочастотные члены, то при $m < n$ сумма t_m является наилучшим в смысле метода наименьших квадратов приближением к данным g_k, $k = 0, \ldots, N - 1$, использующим первые m синусов и косинусов из ряда Фурье.

В качестве типичного приложения рассмотрим модель "el Niño". Ученые заметили, что в южной части Тихого океана преобладают ветры, дующие с востока на запад. Это приводит к движению поверхностных слоев воды в том же направлении и к поднятию холодной воды с нижних слоев возле западного побережья Южной Америки. Холодная вода богаче питательными веществами, поэтому на континентальном шельфе у этих берегов благоденствует множество морских животных. Преобладание восточных ветров оказывает также влияние на погоду в прибрежных районах. Это влияние непостоянно, оно изменяется в течение года и от года к году, причем изменения образуют более или менее регулярный цикл, называемый *el Niño*. Ученые пытаются анализировать эти циклы, так как они имеют огромное экономическое значение для сельского хозяйства и для прогнозирования погоды. Один из способов анализа основан на применении так называемого «Указателя южных колебаний», в котором фиксируются разности атмосферных давлений на острове Пасхи и в дарвиновской Австралии, измеренных на уровне моря в одно и то же время. В этом справочнике содержатся ежемесячные данные, представляющие собой средние значения измерений, которые можно отнести к середине месяца. На рис. 11.6 показаны данные из этого «Указателя» за 14-летний период с 1962 по 1975 г. На график нанесено 168 точек с полуцелыми абсциссами (представляющими середины каждого месяца) между 0 и 168.

Рассмотрим различные модели этих данных. На рис. 11.6 изображен также график тригонометрического интерполянта t_n, $n = 84$, коэффициенты которого были найдены с помощью программы EZFFTF. Значения функции t_n вычислялись в достаточно большом числе допол-

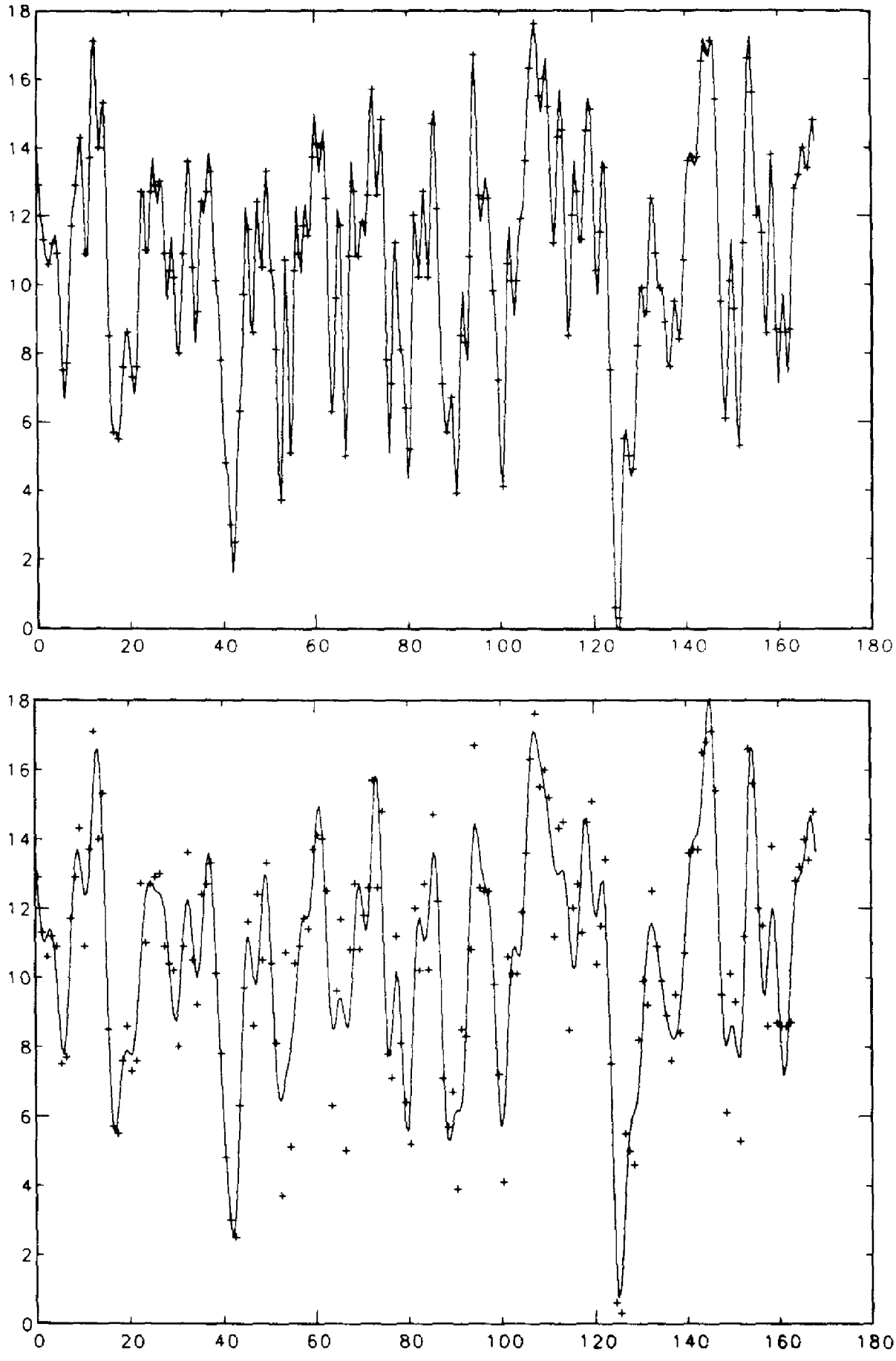


Рис. 11.6 (а) и (б). Данные из «Указателя южных колебаний» и тригонометрические приближения к ним.

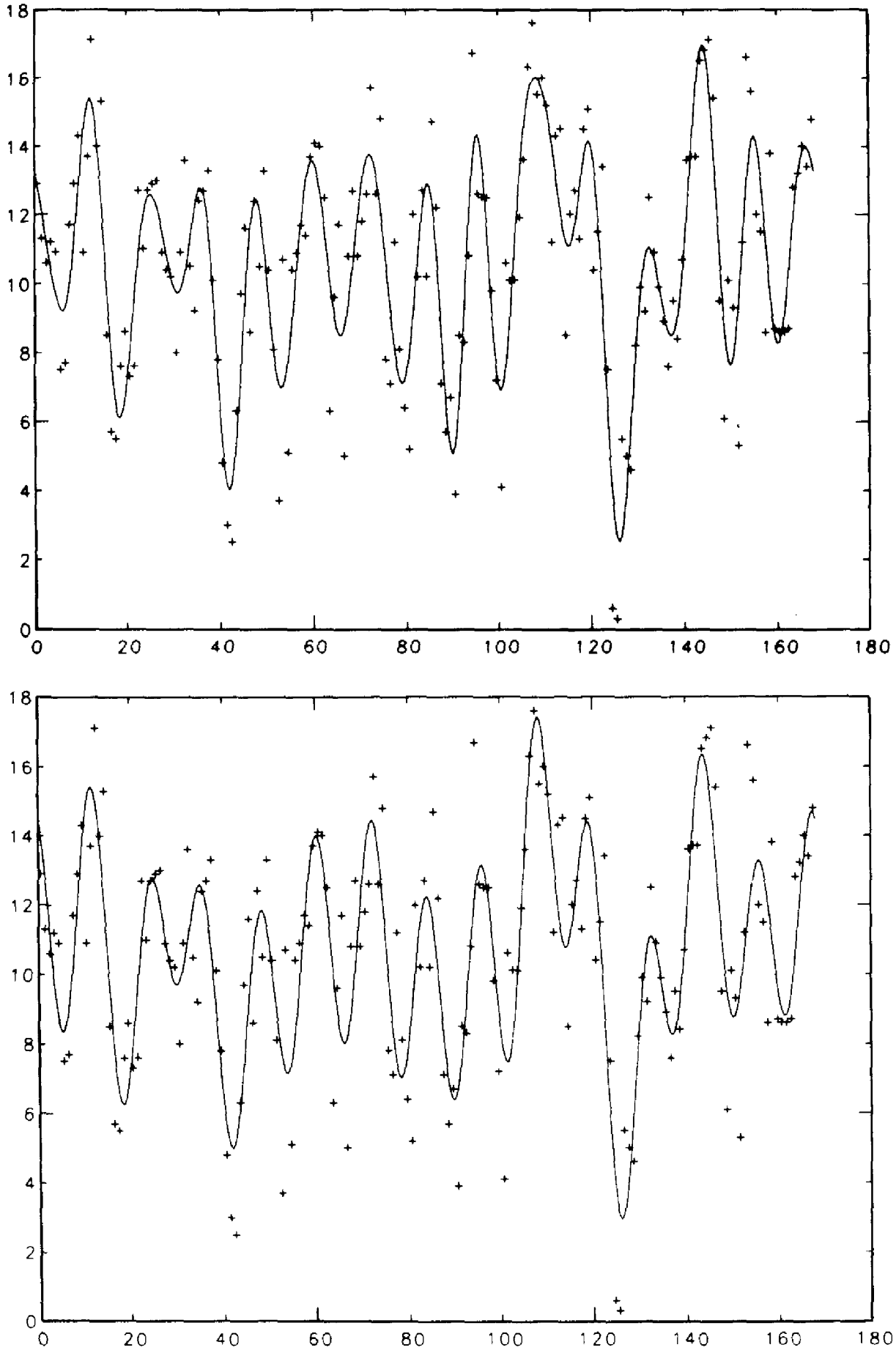


Рис. 11.6 (в) и (г). Данные из «Указателя южных колебаний» и тригонометрические приближения к ним.

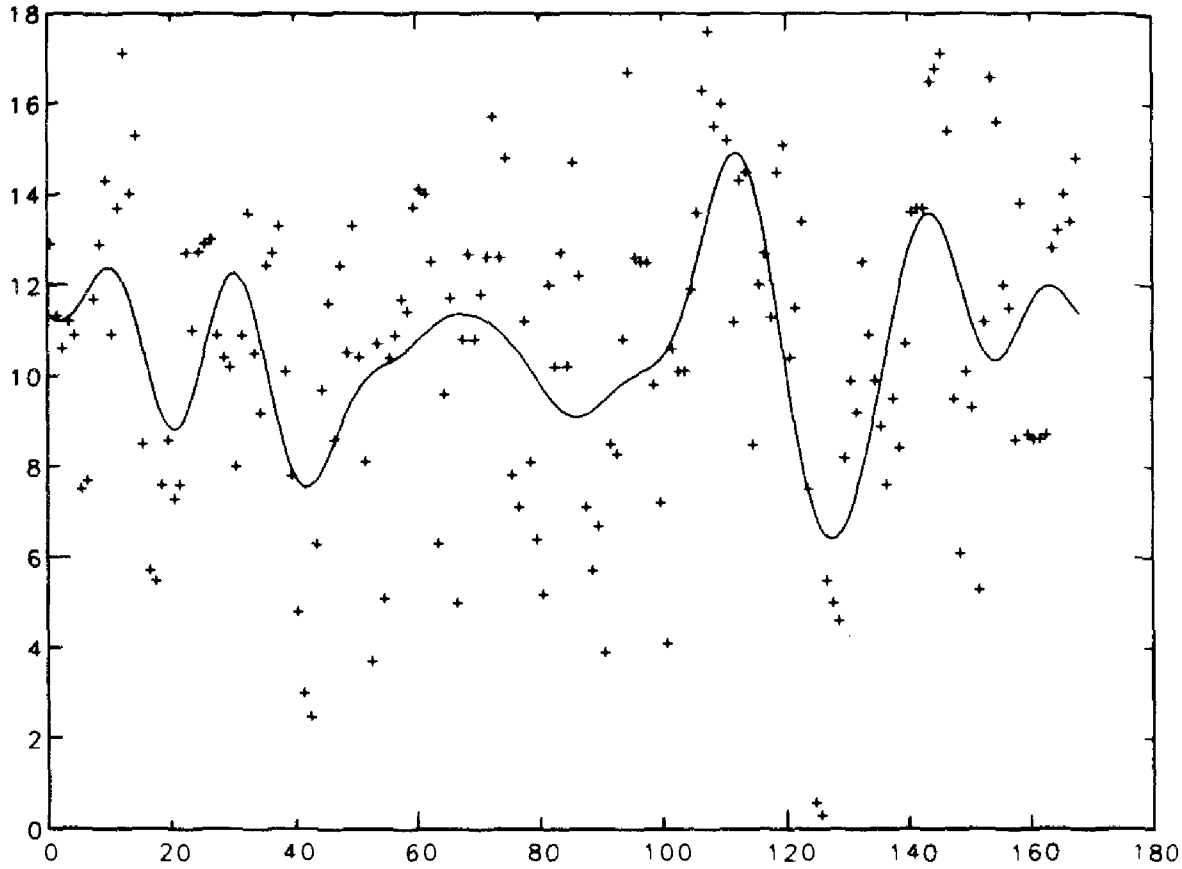


Рис. 11.6 (д). Данные из «Указателя южных колебаний» и тригонометрические приближения к ним.

нительных точек, чтобы можно было построить приведенный на рисунке график. Видно, что t_n производит интерполяцию и удовлетворительно ведет себя между точками данных. Интерполировать по 168 точкам при помощи одного алгебраического полинома – дело безнадежное. На остальных частях того же рисунка изображены графики функций t_m с числами членов $m = 42$, 21, 15 и 10. В каждом случае от исходного тригонометрического интерполянта отбрасывалось все больше и больше высокочастотных членов. Полиномы t_m одновременно являются наилучшими приближениями с заданным числом членов в смысле метода наименьших квадратов. Вполне приемлемый результат показан на рис. 11.6(г), отвечающем модели с 15 членами; на нем можно различить даже наиболее важные колебания, вызванные ежегодными вариациями.

На рис. 11.6(д) показана функция t_{10}. Почему она дает столь плохое приближение к данным? Ежегодные изменения данных требуют учета в t_n одного особенно большого коэффициента, соответствующего синусоиде с периодом в один год. Это можно увидеть из периодограммы, показанной на рис. 11.7(а). При $j = 14$ она имеет большой пик. Каждая точка с абсциссой j на периодограмме отвечает синусоиде, которая в пределах 168 месяцев проведения измерений имела j периодов. Поскольку 14 периодов за 168 месяцев соответствуют одному периоду за

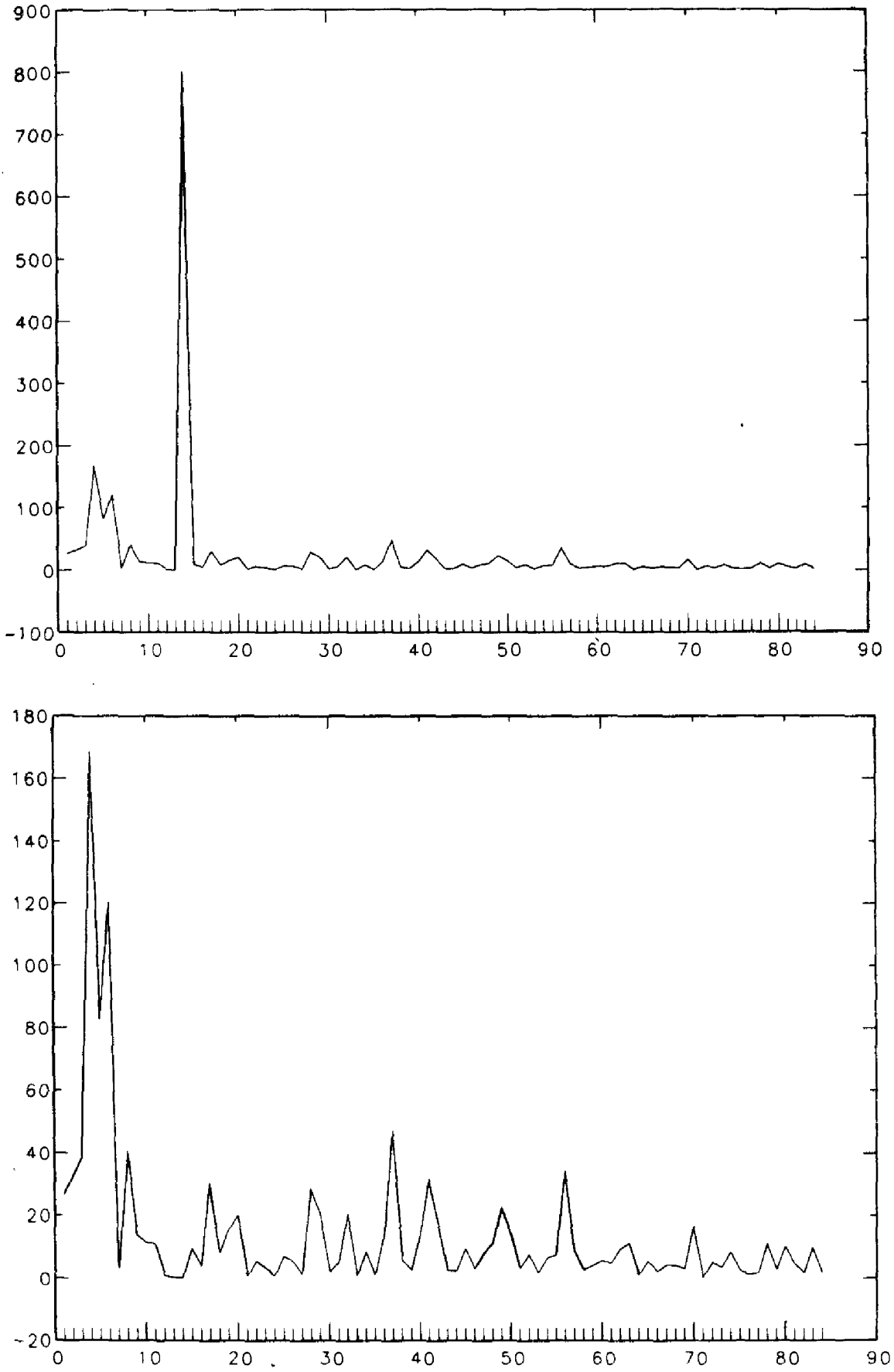


Рис. 11.7 (а) и (б). Периодограммы el Niño.

12 месяцев, указанный большой пик обусловлен регулярными ежегодными колебаниями. Если тригонометрический полином не содержит такого члена, то от него нельзя ожидать хорошего приближения к данным.

Что произойдет, если циклические колебания в данных не будут соответствовать какой-либо частоте, кратной основной? В конце концов, природа не всегда оперирует теми же единицами, что и мы. Отбираемый периодограммой ансамбль частот может давать столь широкую вилку частот, присутствующих в данных, что они не смогут описать полное изменение данных. Можно догадаться, что при этом получится, рассматривая закон сохранения энергии в дискретной форме. Сумма квадратов данных должна равняться сумме квадратов коэффициентов Фурье. Мощность, соответствующая пропущенной частоте, распределяется по другим частотам. В основном это относится к близлежащим частотам, но в какой-то степени влиянию подвергаются все частоты. На рис. 11.7(б) приведена другая периодограмма el Niño; при ее построении 14-й член суммы был намеренно положен равным нулю, чтобы он не заглушил остальные компоненты. Видно, что велики члены суммы, соответствующие четвертой, пятой и шестой компонентам периодограммы при длинах периодов в 42, 33.4 и 28 месяцев. Дополнительный анализ тех же данных показывает, что наибольший пик, соответствующий циклу продолжительностью около 44 месяцев, «размазался» по ближайшим компонентам. Изучались и другие пики, для них были получены более точные оценки. Один из выводов состоит в том, что пики этих синусоид в 1982 г. совпали, что давало возможность предсказать грязевые оползни в Калифорнии, которые снесли множество домов и причинили убытки в миллионы долларов. Возможно, в следующий раз такое явление удастся предсказать заранее. О других интересных вещах, связанных с el Niño, см. статью [Rasmusson, 1985] и обзорные статьи в журнале Journal Oceanus, том 27 (1984).

11.10. Быстрое преобразование Фурье

Посмотрите на формулы из § 2 и проверьте, что даже если числа $\cos(jk\,2\pi/N)$ и $\sin(jk2\pi/N)$ даны, вычисление всех a_k и b_k требует около N^2 умножений и подобного же числа сложений. В гл. 3 мы видели, что для решения системы N линейных уравнений методом гауссова исключения приходится выполнять около $N^3/3$ операций, поэтому N^2 не кажется большим числом. Но приложения этих двух алгоритмов совершенно различны. Во многих ситуациях приходится вычислять дискретные преобразования Фурье тысяч и даже миллионов точек. Решать системы линейных уравнений таких больших размеров приходится редко, разве только с матрицей специальной структуры, свойствами которой можно воспользоваться. Если N велико, то для вычисления дискретного преобразования Фурье по его определению даже на суперкомпьютере могут потребоваться часы работы. Быстрое преобразование Фурье

(БПФ) позволяет произвести те же вычисления, выполнив всего лишь около $N \log_2 N$ операций. Например, если $N = 1024$, то отношение числа выполняемых операций при счете по явным формулам к числу операций в БПФ равно $N^2/(N \cdot \log_2 N) = 102.4$. Таким образом, БПФ требует примерно на два порядка меньшего объема вычислений. Фактическое соотношение времен счета может быть лучше или хуже в зависимости от особенностей программной реализации.

Пример 11.5. Непосредственное вычисление дискретного преобразования Фурье и вычисление с использованием БПФ.

Для иллюстрации различия времени вычислений с использованием БПФ и с использованием определения дискретного преобразования Фурье ниже приводится программа непосредственного выполнения дискретных синус- и косинус-преобразований по формулам из § 2. Это один из *наименее* эффективных алгоритмов, поскольку в нем не используется БПФ и требуется неоднократно вычислять одни и те же синусы и косинусы. Рис. 11.8 показывает времена счета для программ EZFFTF и DIRECT при разных значениях N. Заметим, что EZFFTF всегда работает быстрее, чем DIRECT, даже если N – простое число. Это происходит из-за того, что программа DIRECT вычисляет все синусы

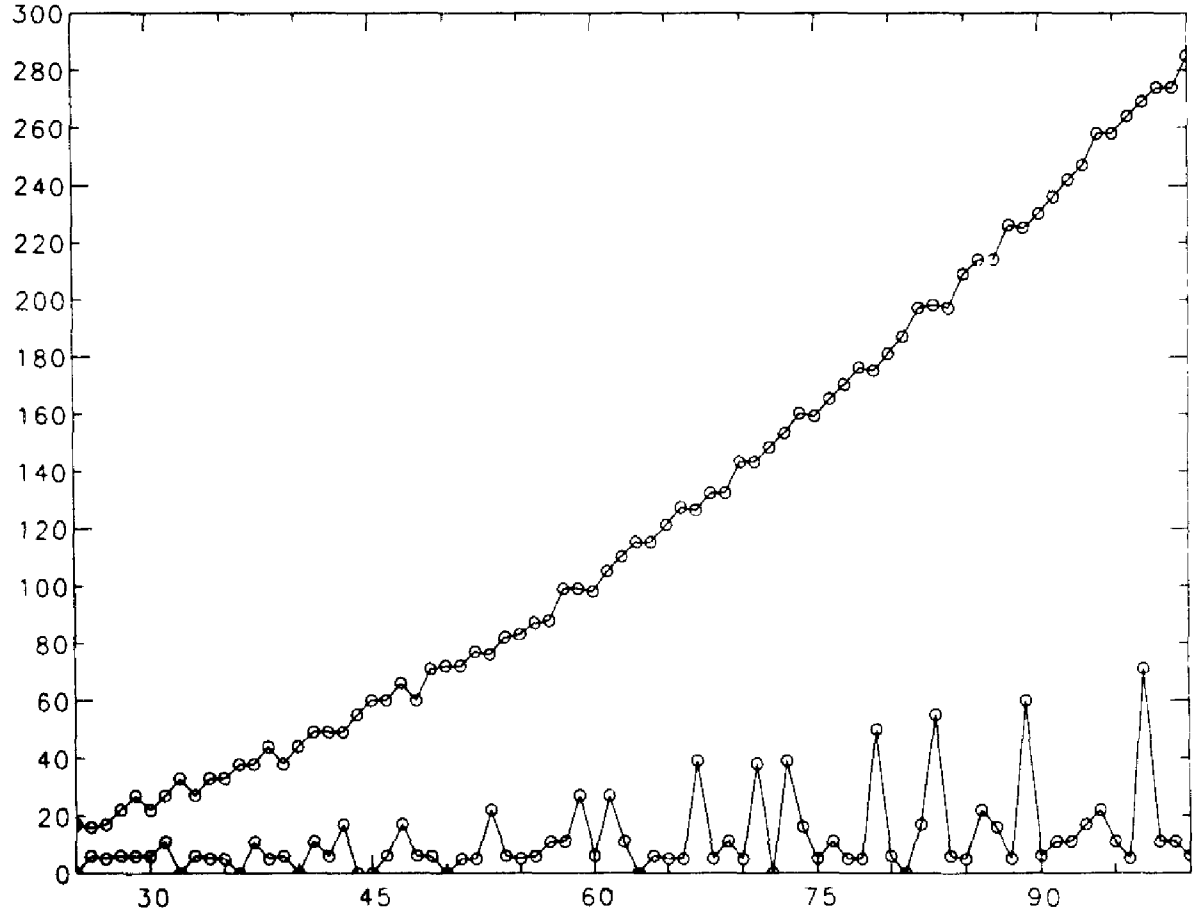


Рис. 11.8. Времена счета с применением подпрограммы EZFFTF и прямого дискретного преобразования Фурье.

и косинусы по несколько раз. Если N является произведением многих небольших целых чисел, то EZFFTF работает значительно быстрее, чем DIRECT. (Обе кривые подвержены известной «нервной дрожи», связанной с малыми неточностями таймера).

```
      SUBROUTINE DIRECT (N, DATA, AZERO, A, B)
C     Непосредственное использование определения для вычисления
C     вещественных дискретных преобразований Фурье.
C     Никаких упрощений. ПРОГРАММА МЕДЛЕННАЯ.
      REAL DATA (0:*), A(*), B(*)
      AZERO = 0.0
      DO 1 J = 0, N - 1
        AZERO=AZERO + DATA(J)
1     CONTINUE
      AZERO = AZERO/N
      TPN = 2*ASIN(1.0)*2./N
      DO 20 K = 1, N/2
        A(K) = 0
        B(K) = 0
        DO 10 J = 0, N - 1
          A(K) = A(K) + DATA(J)*COS(J*K*TPN)
          B(K) = B(K) + DATA(J)*SIN(J*K*TPN)
10      CONTINUE
        IF(K.NE.N/2)THEN
          A(K) = A(K)*(2./N)
          B(K) = B(K)*(2./N)
        ELSE
          A(K) = A(K)*(1./N)
          B(K) = B(K)*(1./N)
        END IF
20    CONTINUE
      RETURN
      END
```

□

Существенной идеей алгоритма БПФ является то, что N-точечное дискретное преобразование Фурье можно выполнить, произведя два $N/2$-точечных дискретных преобразования Фурье над слегка модифицированными исходными данными. Хотя это и кажется более запутанным, но на самом деле требует меньшего объема работы, чем непосредственное вычисление. Конечно, каждое из этих $N/2$-точечных преобразований можно также разбить на два $N/4$-точечных и т. д. В § 6 отмечалось, что для применения некоторых программ требуется, чтобы N было целой степенью двойки; теперь причина этого становится ясной. Если N простое, то единственный способ снижения затрат заключается в отбрасывании некоторых данных или добавлении (фиктивных) нулевых значений. Это иногда называют *нулевой подкладкой*, и нужно всегда быть начеку, так как некоторые программы БПФ делают ее

автоматически, изменяя тем самым ожидаемые результаты.

Существует несколько способов более подробного описания БПФ. Вот некоторые из них (но не все):

(1) Схема «разделяй и властвуй» ([Aho et al., 1974]).
(2) Матричные факторизации ([Kahaner, 1970]).
(3) Вычисление полиномов ([Kahaner, 1978]).
(4) Схема «бабочки» ([Brigham, 1974]).

В некоторых приложениях не обязательно знать детали алгоритма, но важно уметь интерпретировать результаты его работы. С другой стороны, если вы заинтересованы в применении БПФ на новом компьютере, то детали алгоритма существенны, поскольку разное оборудование может приводить к различиям в структуре данных и особенностях программирования. В этом учебнике мы не будем обсуждать данный аспект; отсылаем читателя к списку литературы.

* 11.11. Комплексное представление

В § 2 описаны интегральные синус- и косинус-преобразования Фурье, дискретные преобразования Фурье и ряды Фурье вещественных функций и наборов вещественных данных. Их можно обобщить на случай комплекснозначных функций и комплексных значений данных. Это имеет определенные преимущества.

(1) Используя понятие комплексной переменной, можно записать большинство формул из предыдущих параграфов более компактно.
(2) При обсуждении двумерных преобразований, например в обработке изображений, используются почти исключительно комплексные обозначения. Они также широко применяются в цифровой обработке сигналов.
(3) Многие ученые полагают, что комплексная плоскость является естественной средой для изложения идей Фурье.

В области своей специализации вы, возможно, сможете придерживаться подхода с использованием синусов и косинусов, но для изучения литературы или новых разработок придется освоить комплексные формулировки. В этом параграфе приводятся основные сведения.

Напомним формулу Эйлера: для произвольных чисел f и t

$$e^{i2\pi ft} = \cos 2\pi ft + i \sin 2\pi ft, \quad i = \sqrt{-1}\ .$$

(1) Если $g(t)$ – комплекснозначная функция, то ее комплексное преобразование Фурье обозначается через $G(\omega)$. Функции g и G связаны парой формул

$$G(\omega) = \int_{-\infty}^{\infty} g(t)\, e^{-2\pi i\omega t}\, dt, \quad g(t) = \int_{-\infty}^{\infty} G(\omega)\, e^{2\pi i\omega t}\, d\omega .$$

Заметим, что $G(\omega)$ определено как для положительных, так и для отрицательных значений ω. Если g–вещественная функция, то у нее есть и комплексное интегральное преобразование Фурье $G(\omega)$, и интегральные косинус- и синус-преобразование Фурье $A(\omega)$, $B(\omega)$ (см. § 2), которые связаны между собой. Если определить $A(\omega)$ и $B(\omega)$ при отрицательных ω формулами $A(-\omega) = A(\omega)$, $B(-\omega) = -B(\omega)$, то можно показать, что $G(\omega) = (A(\omega) - iB(\omega))/2$. Комплексное преобразование вещественных функций может смутить тем, что g представляется «суммой» по положительным и отрицательным частотам. Мощность на частоте $|\omega|$ равна $|G(\omega)|^2 + |G(-\omega)|^2 = (A^2(\omega) + B^2(\omega))/2$. Различать положительные и отрицательные частоты приходится редко; о них часто говорят как об одной и той же частоте.

(2) Если величины g_j представляют собой набор N комплексных чисел, то комплексным дискретным преобразованием Фурье для этого набора является другой набор G_j. Числа g_j и G_j связаны парой формул

$$G_j = \frac{1}{N}\sum_{k=0}^{N-1} g_k \exp\left(-ijk\frac{2\pi}{N}\right),$$

$$g_j = \sum_{k=0}^{N-1} G_k \exp\left(ikj\frac{2\pi}{N}\right), \quad j = 0, 1, \ldots, N-1.$$

Заметим, что G_j определены для частот от 0 до $N-1$. Если величины g_j представляют собой набор вещественных чисел, то у этого набора есть как комплексное дискретное преобразование Фурье, так и дискретные косинус- и синус-преобразования Фурье a_k, b_k, $k = 0, \ldots, N/2$ (см. § 2). Как они связаны между собой? Можно показать, что

$$G_0 = a_0, \quad G_k = (a_k - ib_k)/2, \quad 1 \leqslant k \leqslant N/2,$$

кроме случая четного N, в котором надо опустить 1/2 в $G_{N/2}$. Остальные значения G_k являются комплексно сопряженными к числам из первой половины набора, и их обычно игнорируют. Конкретно,

$$G_k = (a_k + ib_k)/2, \quad 1 \leqslant k \leqslant N/2.$$

Говорят, что последовательности G_j и g_j являются парой дискретных преобразований Фурье, прямого и обратного. Различие между случаями четных и нечетных N в комплексной формулировке не проявляется, поэтому такие формулы намного легче запомнить. Комплексное дискретное преобразование Фурье вещественных данных g_j может смутить тем, что вроде бы порождает вдвое больше компонент в области частот, чем можно ожидать. Но из написанных формул видно, что эти величины не несут новой информации. В качестве иллюстрации вычислим a, b и G для последовательности $g_0 = 1$, $g_1 = g_2 = g_3 = 0$. По формулам из § 2 получаем

$$a_0 = \frac{1}{4}, \quad a_1 = \frac{2}{4}, \quad a_2 = \frac{1}{4}, \quad b_1 = b_2 = 0.$$

Согласно написанной выше формуле суммирования, $G_i = 1/4$ для всех i. Теперь по формуле $G_k = (a_k - ib_k)/2$ для первой половины набора G получаем

$$G_0 = a_0, \quad G_1 = \frac{a_1 - ib_1}{2}, \quad G_2 = a_2 - ib_2;$$

по формуле $G_{N-k} = (a_k + ib_k)/2$ для второй половины набора получаем

$$G_3 = \frac{a_1 + ib_1}{2}, \quad G_2 = a_2 + ib_2.$$

Поскольку N четно, b_2 автоматически равно нулю, и мы получаем $G_i = 1/4$.

(3) Если g – комплекснозначная периодическая функция, заданная на $[a, b]$, то ее можно разложить в комплексный ряд Фурье

$$g(t) = \sum_{j=-\infty}^{\infty} C_j e^{ij2\pi ft},$$

$$C_j = \frac{1}{b-a} \int_a^b g(t) \exp(-ij2\pi ft)\, dt,$$

где $f = 1/(b-a)$. Заметим, что и здесь функция g представлена в виде суммы по положительным и отрицательным частотам. Если g – вещественная периодическая функция, то она имеет вещественный ряд Фурье с коэффициентами A_j и B_j, которые связаны с C_j. Определим A_j и B_j для $j < 0$ при помощи равенств

$$A_{-j} = A_j, \quad B_{-j} = -B_j, \quad j > 0.$$

Тогда можно показать, что

$$C_0 = \frac{A_0}{2}, \quad C_j = \frac{1}{2}(A_j - iB_j), \quad j \neq 0.$$

Комплексная форма записи не добавляет никакой новой информации. В некоторых учебниках изложение начинается с комплексного представления, потому что оно более краткое и намного упрощает доказательства и алгебраические преобразования. Однако отрицательные индексы, появляющиеся в комплексной форме ряда Фурье, равно как и отрицательные частоты в интегральном преобразовании Фурье, часто вызывают замешательство. Комплексная форма важна для двумерных приложений, что будет показано в § 12. Программы вычисления дискретных преобразований Фурье строят либо вещественные a и b по заданным вещественным g, либо комплексные G по произвольным (вещественным или комплексным) g. В данной главе подпрограмма EZFFTF является примером программы первого типа, а подпрограмма CFFTF – второго.

Часто используется в приложениях и другая форма преобразования (2). Снова отметим, что в (1) и (3) частоты могут быть отрицательными. Дискретное преобразование Фурье в (2) можно определить аналогичным образом (вспомним соглашение о записи $N/2$ вместо $[N/2]$, см. разд. 11.2.–*Перев.*):

$$G'_j = \frac{1}{N}\sum_{k=0}^{N-1} g_k \exp\left(-ijk\frac{2\pi}{N}\right), \quad j = -N/2, \ldots, 0, 1, \ldots, M,$$

где $$M = \begin{cases} N/2, & \text{если } N \text{ нечетно,} \\ N/2 - 1, & \text{если } N \text{ четно.} \end{cases}$$

После изучения этой формулы становится ясно, что $G_0 = G'_0$, $G_1 = G'_1, \ldots, G_{N/2} = G'_{N/2}$. Величины G'_{-1}, G'_{-2} и т. д. равны числам из второй части набора G_s без штрихов. Например, $G'_{-1} = G_{N-1}$, поскольку $\exp(-i(-1)k(2\pi/N)) = \exp\left(-i(N-1)k\frac{2\pi}{N}\right)$. Аналогично, $G'_{-2} = G_{N-2}$ и т. д. Теперь можно записать

$$g_j = \sum_{k=M}^{N/2} G'_k \exp\left(ijk\frac{2\pi}{N}\right), \quad j = 0, \ldots, N-1.$$

Чем хороша эта форма записи? Выше отмечалось, что если данные g_j вещественны, то вторую половину массива G обычно игнорируют. Если забыть об этом и построить график изменения величин всех коэффициентов (периодограмму), то обычно до середины он будет уменьшаться, а затем расти. Фактически в случае вещественных данных график симметричен относительно своей середины. Если данные комплексны, то в целом график имеет такую же форму, но уже не симметричен. Такой график было бы проще интерпретировать, если бы пик был в середине, а не на обоих концах. Указанное выше представление позволяет это сделать, потому что нулевая частота попадает в середину: $-N/2, \ldots, -1, 0, 1, \ldots, M$. На практике при четных N используют небольшую хитрость, чтобы избежать циклической перестановки массива G на выходе комплексного БПФ. Исходные данные перед вводом в программу вычисления комплексного БПФ часто изменяют, умножая их на $(-1)^k$. Поскольку $(-1)^k = \cos(k\pi) = \exp(ik\pi)$, в действительности вычисляется

$$\frac{1}{N}\sum_{k=0}^{N-1}(-1)^k g_k \exp\left(-ijk\frac{2\pi}{N}\right) = \frac{1}{N}\sum_{k=0}^{N-1} g_k \exp\left(-ik\frac{2\pi}{N}\left(j - N/2\right)\right).$$

При $j = N/2$ это G'_0, при $j = 1 + N/2$ это G'_1, при $j = (N/2) - 1$ это G'_{-1} и т. д. Таким образом, величины в середине выходного массива соответствуют низким частотам. Пример этого мы увидим в разделе 12.1, где будет рассматриваться применение анализа Фурье к обработке изображений.

* 11.11.1. Подпрограммы CFFTF и CFFTB

Три подпрограммы CFFTF, CFFTB и CFFTI являются комплексными аналогами программ из § 6, работающих с действительным представлением. В предыдущем разделе было сказано, что комплексное дискретное преобразование Фурье всегда используется в приложениях многомерного анализа. Однако, поскольку в программе CFFTF используются комплексные входные данные g_i, ее легко применить для проведения вещественного одномерного преобразования, положив мнимые части вводимого массива равными нулю. Комплексный массив на выходе содержит коэффициенты a_i и b_i как вещественные и мнимые части первых $N/2$ компонент преобразования. На выходе из CFFTF вычисленные значения делятся на N и записываются в память на место входных данных. В остальных компонентах массива содержатся комплексно сопряженные к этим числам величины. Как и в случае вещественного представления, если первое значение абсциссы $t = 0$, то результат работы CFFTF дает приближения для коэффициентов комплексного ряда Фурье периодической функции, породившей входные данные. В противном случае надо приписать веса компонентам выходного массива, умножая j-ю компоненту на $\exp(-ij2\pi ft_0)$, $j = 1, \ldots, N/2$. Если индекс массива преобразуемых данных изменяется начиная с нуля, то $C(J)$ соответствует частоте, в J раз большей, чем основная частота, т. е. $J \times 1/(N\Delta)$. В тех приложениях, где наибольшая эффективность программы не является существенным требованием, часто применяется комплексное преобразование.

В приведенной ниже программе показано, как использовать подпрограмму CFFTF для вычисления комплексного дискретного преобразования Фурье функции Рунге. Сравните эту программу с аналогичной программой из § 6, в которой было обращение к EZFFTF.

```
C     При помощи комплексного дискретного преобразования Фурье
C     найти приближенные коэффициенты Фурье заданной на [-1, 1]
C     функции Рунге для N = 16 и N = 17.
C
      REAL WSAVE(150)
      COMPLEX COEFF(0:16), SQTM1
C
C     Обратите внимание на начинающуюся для простоты с 0 индексацию, разрешенную в языке FORTRAN 77.
C
C     Арифметическая оператор-функция для вычисления значений
C     функции Рунге.
      RUNGE(X) = 1.0/(1 + 25.0*X*X)
C
      X0 = -1.0
      PI = ASIN(1.0)*2.0
      SQTM1 = CMPLX(0.0, -1.0)
```

```
C
      DO 10 N = 16, 17
       CALL CFFTI (N, WSAVE)
C     Функция предполагается периодической, заданной на отрезке
C     [-1, 1] длины 2.
       DEL = 2.0/N
       F = 2.0 * PI/(N * DEL)
       DO 1 J = 0, N - 1
C     Первое значение функции наблюдается в точке -1, последнее -
C     в точке 1 - DEL.
        XJ = (-1.0) + J * DEL
        COEFF(J) = CMPLX(RUNGE(XJ), 0.0)
    1  CONTINUE
       CALL CFFTF (N, COEFF, WSAVE)
C     Для правильной нормировки выдаваемые коэффициенты надо
C     разделить на N.
C
C     Обратите внимание на повторения коэффициентов после N/2.
C     Весовые множители, введенные для учета ненулевого X0, до не-
C     торой степени нарушают этот порядок.
C
       WRITE (*,*) 'RESULTS FOR N = ', N
       WRITE (*,*) 'CZERO = ', COEFF(0)/N*2
       WRITE (*,*) 'J OUTPUT FROM CFFTF SCALED
       COEFFICIENTS'
       DO 11 J = 1, N - 1
        WRITE (*,'(I5,2E15.6,5X,2E15.6)')
         J, COEFF(J), EXP(-SQTM1 * J * F * X0) * COEFF(J)/N * 2
   11  CONTINUE
       WRITE (*,*)
   10 CONTINUE
      STOP
      END
```

```
C
C   Будут получены следующие результаты[1])
C
C     RESULTS FOR N = 16
C     CZERO = (0.549222, 0.000000)
C  J                OUTPUT FROM CFFTF                SCALED COEFFICIENTS
C  1  -0.275492E+01   0.520041E-07   0.344365E+00  -0.366058E-07
C  2   0.140523E+01   0.000000E+00   0.175654E+00  -0.307123E-07
C  3  -0.778358E+00   0.730062E-08   0.972947E-01  -0.264299E-07
C  4   0.401056E+00   0.000000E+00   0.501320E-01  -0.175307E-07
C  5  -0.228722E+00  -0.730062E-08   0.285903E-01  -0.115846E-07
C  6   0.119966E+00   0.000000E+00   0.149957E-01  -0.786582E-08
C  7  -0.841548E-01  -0.730062E-08   0.105194E-01  -0.552484E-08
C  8   0.614045E-01   0.000000E+00   0.767556E-02  -0.536815E-08
```

```
C  9   -0.841548E-01   -0.374029E-07    0.105194E-01   -0.360132E-08
C 10    0.119966E+00    0.000000E+00    0.149957E-01   -0.131097E-07
C 11   -0.228722E+00    0.730062E-08    0.285903E-01   -0.284064E-07
C 12    0.401056E+00    0.000000E+00    0.501320E-01    0.525921E-07
C 13   -0.778358E+00   -0.730062E-08    0.972947E-01   -0.109662E-06
C 14    0.140523E+01    0.000000E+00    0.175654E+00    0.214986E-06
C 15   -0.275492E+01   -0.730062E-08    0.344365E+00   -0.450667E-06
C
C      RESULTS FOR N = 17
C      CZERO = (0.549161, 0.000000)
C  J       OUTPUT FROM CFFTF          SCALED COEFFICIENTS
C  1   -0.292606E+01   -0.421539E-07    0.344243E+00   -0.251354E-07
C  2    0.149192E+01    0.488273E-07    0.175520E+00   -0.249445E-07
C  3   -0.824417E+00   -0.349775E-07    0.969903E-01   -0.213225E-07
C  4    0.421975E+00    0.271580E-07    0.496441E-01   -0.141651E-07
C  5   -0.234573E+00   -0.245598E-07    0.275968E-01   -0.917357E-08
C  6    0.112482E+00    0.293560E-07    0.132331E-01   -0.348762E-08
C  7   -0.603455E-01   -0.568597E-07    0.709948E-02    0.234479E-08
C  8    0.120123E-01    0.910478E-07    0.141321E-02    0.972312E-08
C  9    0.120123E-01   -0.910478E-07   -0.141321E-02    0.118234E-07
C 10   -0.603455E-01    0.568597E-07   -0.709948E-02    0.128959E-07
C 11    0.112482E+00   -0.293560E-07   -0.132331E-01    0.161793E-07
C 12   -0.234573E+00    0.245598E-07   -0.275968E-01    0.318405E-07
C 13    0.421975E+00   -0.271580E-07   -0.496441E-01    0.596154E-07
C 14   -0.824417E+00    0.349775E-07   -0.969903E-01    0.122823E-06
C 15    0.149192E+01   -0.488273E-07   -0.175520E+00    0.235911E-05
C 16   -0.292606E+01    0.421539E-07   -0.344243E+00    0.486474E-06
```

[1]) Scaled coefficients (англ.) – взвешенные коэффициенты. – *Прим. перев.*

Более эффективный способ выполнения вещественного преобразования с помощью CFFTF заключается в том, чтобы поместить нулевое, второе, четвертое, шестое и т. д. заданные значения в массив вещественных компонент g_i, а первое, третье, пятое и т. д. – в массив мнимых компонент. При этом достаточно задать комплексный массив длины $N/2$. На выходе массив можно будет разбросать в нужном порядке, чтобы получить преобразование исходных данных; см. [Brigham, 1974], с. 169.

* 11.12. Двумерные преобразования

В программах обработки изображений широко применяется БПФ. Двумерное фотографическое изображение представляется в виде матрицы $N \times N$ положительных чисел g_{ij}, $i, j = 0, \ldots, N-1$, в которой каждый элемент представляет собой среднюю освещенность (степень потемнения фотопленки) малого прямоугольного участка изображения – точки растра, или пиксела. Для конкретных приложений может понадобиться улучшить качество изображения, увеличивая или уменьшая его резкость, или выделить важные характеристики, например края заснятого объекта и т. п. Все понятия из § 2 можно обобщить на случай двух и более измерений. В таких ситуациях описание становится намного проще в комплексном представлении, и поэтому в программах для

решения двумерных задач применяются комплексные преобразования.

Интегральные преобразования Фурье функции $g(x, y)$ задаются парой формул

$$G(\omega, \mu) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x, y) \exp(-2\pi i [\omega x + \mu y])\, dx\, dy$$

и

$$g(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} G(\omega, \mu) \exp(2\pi i [\omega x + \mu y])\, d\omega\, d\mu .$$

Дискретное преобразование Фурье массива чисел g_{mn} размера $M \times N$ задается парой формул

$$G_{uv} = \frac{1}{NM} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} g_{mn} \exp\left(-2\pi i \left[\frac{mu}{M} + \frac{nv}{N}\right]\right),$$
$$u = 0, \ldots, M-1,\ v = 0, \ldots, N-1,$$

и

$$g_{mn} = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} G_{uv} \exp\left(2\pi i \left[\frac{mu}{M} + \frac{nv}{N}\right]\right),$$
$$m = 0, \ldots, M-1,\ n = 0, \ldots, N-1 .$$

Так же как и в одномерном случае, величину $|G_{uv}|$ обычно интерпретируют как вклад от частоты (u, v), или, иначе говоря, от частоты, в u раз превосходящей основную частоту по x и в v раз превосходящей основную частоту по y. В обработке изображений x и y обычно являются расстояниями, а (u, v) называется *пространственной частотой*. Заметим, что $\omega x + \mu y = \sqrt{\omega^2 + \mu^2}\,(x \cos\theta + y \sin\theta)$, где $\operatorname{tg}\theta = \mu/\omega$. Поскольку $x' = x\cos\theta + y\sin\theta$ задает поворот на угол θ, выражение $\cos(2\pi[\omega x + \mu y])$ описывает колебания с частотой $\sqrt{\omega^2 + \mu^2}$ по направлению x' и постоянно в перпендикулярном направлении. Поэтому разложение функции g часто интерпретируют как запись этой функции в виде комплексной линейной комбинации синусов и косинусов повернутого аргумента (см. [Rosenfeld, Kak, 1982], с. 18).

Одним из наиболее важных свойств двумерных преобразований является то, что их можно вычислять с помощью программ типа CFFTF для одномерных преобразований. Если переписать формулу для G в виде

$$G_{uv} = \frac{1}{M} \sum_{m=0}^{M-1} \left[\frac{1}{N} \sum_{n=0}^{N-1} g_{mn} \exp\left(-inv\frac{2\pi}{N}\right)\right] \exp\left(-imu\frac{2\pi}{M}\right),$$

то выражение в квадратных скобках будет одномерным дискретным преобразованием Фурье m-й строки массива g_{mn}. Поэтому вычисление двумерного дискретного преобразования Фурье массива g_{mn} можно разбить на следующие шаги:

(1) Вычислить одномерные дискретные преобразования для всех

строк массива g и записать эти преобразования на места исходных строк.

(2) Затем вычислить одномерные дискретные преобразования для всех столбцов и записать эти преобразования на места исходных столбцов.

*11.12.1. Подпрограмма CFFT2D

Эта подпрограмма вычисляет двумерное дискретное преобразование Фурье (или обратное к нему) квадратного комплексного массива. В приведенном ниже примере показано ее применение для преобразования массива 64 × 64 положительных чисел, которые представляют собой значения освещенности изображения в точках (I, J). Освещенность везде нулевая, кроме маленького квадрата в центре, где она равна единице. Это изображение показано на рис. 11.9(а). На рис. 11.9(б) показаны величины компонент дискретного преобразования Фурье в том виде, как их выдает CFFT2D. Обратите внимание на пики в четырех углах, соответствующие низким частотам. Это типичное явление, которое иллюстрирует замечание, сделанное в конце § 11.11. Большинство пользователей предпочло бы работать с тем, что изображено на рис. 11.9(в). Чтобы построить нарисованную там поверхность, вычисления были проведены заново, но с измененными данными: значения G_{ij} были умножены на $(-1)^{i+j}$. Графики такого типа, как на последнем рисунке, называются *аксонометрическими*; они более полно описываются в задаче 11.6.

Приведенная ниже программа иллюстрирует работу подпрограммы CFFT2D при вычислении дискретного преобразования Фурье массива 64 × 64, кодирующего изображение. Вычисляются также величины его коэффициентов Фурье, а затем подпрограмма CFFT2D применяется для обратного преобразования, приводящего к исходным данным.

```
C     Пример 11.8: построение графиков изображения от источника
C     размера 8 × 8 в массиве 64 × 64 (на нулевом фоне) и его преоб-
C     разования.
C
      PARAMETER (N = 64)
      REAL      A(N, N), A2(N, N)
      COMPLEX   IMAGE(N, N), IMAGE2(N, N), W(3 * N + 8)
      LOGICAL   FORWD
C
      LDA = N
C     Распределение данных. IMAGE – исходные данные.
C     В IMAGE2 они умножены на (-1)**(I + J), чтобы разместить
C     коэффициенты Фурье более удобным для рассмотрения образом.
C
      DO 1 I = 1, N
```

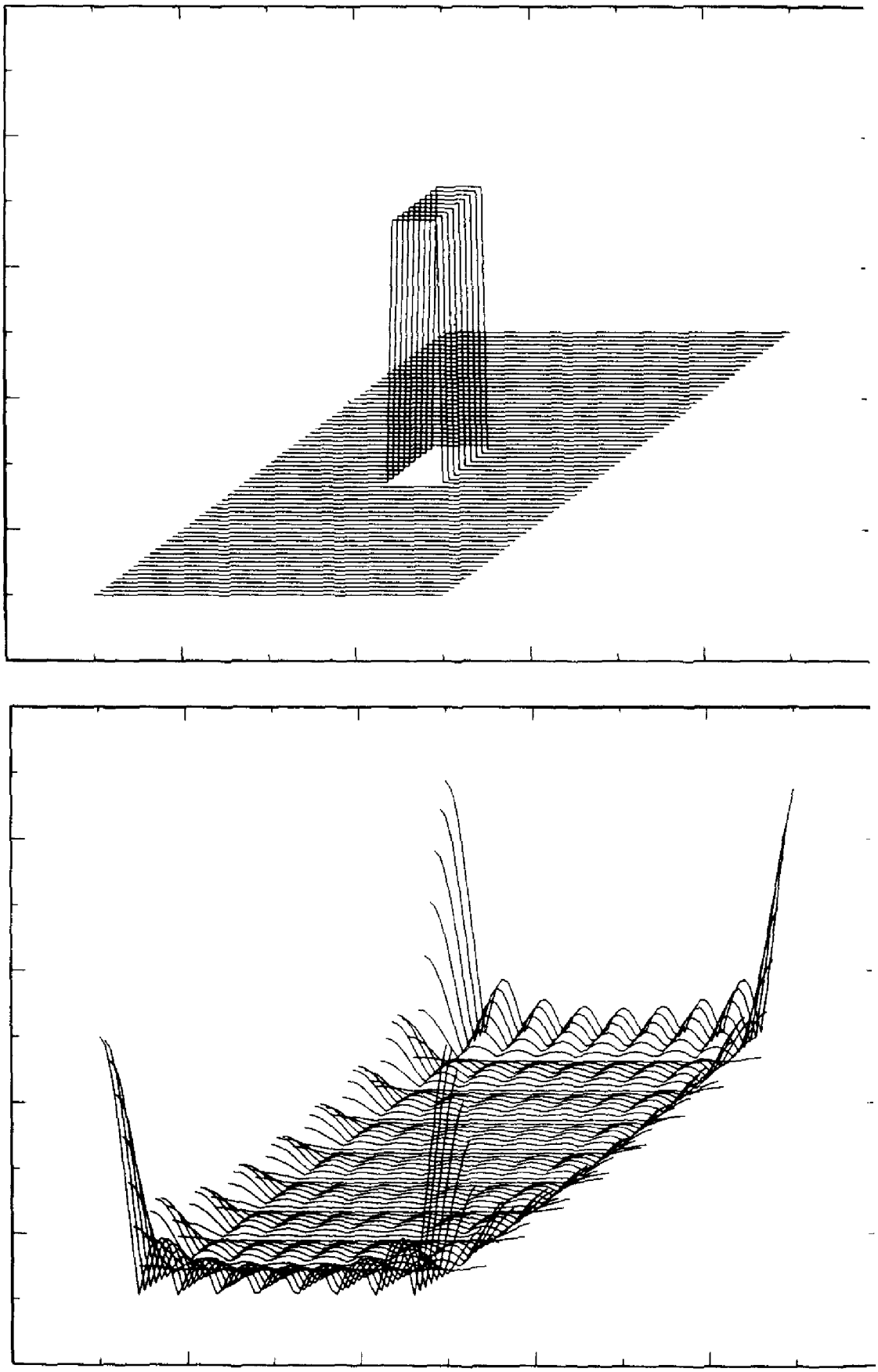


Рис. 11.9. (а) Изображение; (б) периодограмма изображения.

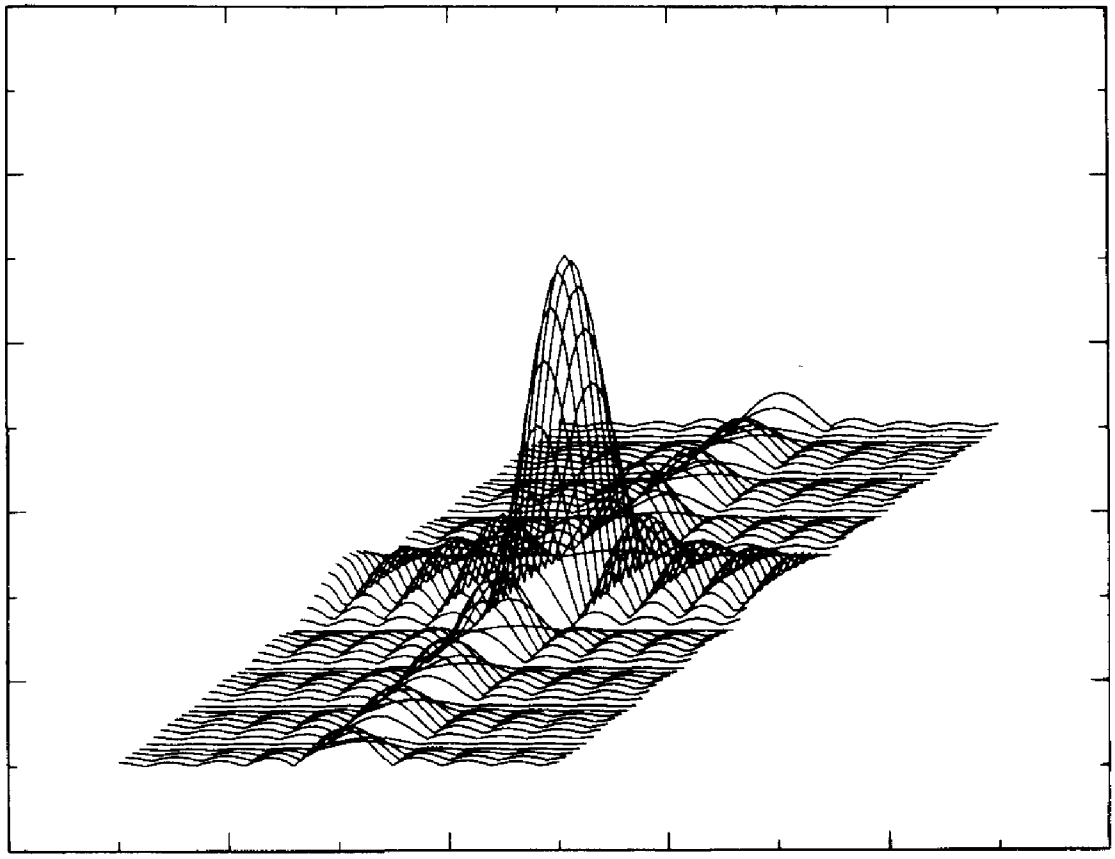


Рис. 11.9. (в) Периодограмма после введения весовых множителей.

```
      DO 1 J = 1,N
       A(I, J) = 0.
       IF(I.GE.(N/2 - 4).AND.I.LE.(N/2 + 4).AND.J.GE.(N/2 - 4)
     *          .AND.J.LE.(N/2 + 4)) A(I, J) = 1.0
       IMAGE(I, J) = CMPLX(A(I, J), 0.0)
       IMAGE2(I, J) = IMAGE(I, J)*(-1)**(I - 1 + J - 1)
1     CONTINUE
C
C     Построение графика изображения по программе аксонометри-
C     ческого проектирования.
      CALL AXNPLT (A, LDA, N)
C
C     Вычисление преобразований Фурье массивов IMAGE и IMAGE2
C
      FORWD = .TRUE.
      CALL CFFT2D (N, IMAGE, LDA, W, FORWD)
      CALL CFFT2D (N, IMAGE2, LDA, W, FORWD)
C
C     Вычисление величин компонент преобразований.
C     Полученные преобразования не имеют весовых множителей, их
C     надо разделить на N*N.
```

```
C
      DO 2 I = 1,N
        DO 2 J = 1,N
          A(I,J) = ABS(IMAGE(I, J))
          A2(I, J) = ABS(IMAGE2(I, J))
    2 CONTINUE
C
C     Построение графиков модулей преобразований.
      CALL AXNPLT (A, LDA, N)
      CALL AXNPLT (A2, LDA, N)
C
C     Вычисление обратного преобразования изображения и сравнение
C     его с исходными данными.
C
      FORWD = .FALSE.
      CALL CFFT2D (N, IMAGE, LDA, W, FORWD)
      ERMAX = 0.0
      DO 3 I = 1,N
        DO 3 J = 1,N
        DATA = 0.0
        IF(I.GE.(N/2 - 4).AND.I.LE.(N/2 + 4).AND.J.GE.(N/2 - 4)
     *          .AND.J.LE.(N/2 + 4)) DATA = 1.0
        ERR = ABS(DATA - ABS(IMAGE(I, J))/(N * N))
        IF(ERR.GT.ERMAX) ERMAX = ERR
    3 CONTINUE
      WRITE (*,*) 'MAXIMUM ERROR IS', ERMAX
C
      END
```

*11.13. Свертка и корреляция

Свертка двух функций – это важнейшее понятие, которое используется почти во всех областях науки и техники. В механике она называется принципом суперпозиции или интегралом Дюамеля, в теории систем – интегралом отклика на внешний импульс, в оптике – функцией пятна или функцией точечного источника. Свертку двух функций обычно можно представить себе как операцию осреднения или сглаживания. В практических задачах, связанных с измерениями, экспериментатор часто полагает, что каждая из измеренных величин является некоторым средним значением, и тогда результаты измерений можно рассматривать как свертку. Примером может служить цифровой экспонометр, якобы показывающий интенсивность света в точке, а на самом деле – среднюю величину (свертку) нескольких близлежащих значений.

Если $f(x)$ и $g(x)$ – функции, определенные на $(-\infty, \infty)$, то их *сверткой* $h(x)$, которая обозначается через $f*g$, называется функция, заданная

формулой

$$h(x) = f * g \equiv \lim_{T \to \infty} \left[\int_{-T}^{T} f(t)\, g(x - t)\, dt \right].$$

Другой очень важный интеграл описывает корреляцию двух функций. Если $f(x)$ и $g(x)$ – функции, определенные на $(-\infty, \infty)$, то их *ковариацией*, которая обозначается через $f \circ g$, называется функция

$$z(x) = f \circ g \equiv \lim_{T \to \infty} \left[\int_{-T}^{T} (f(t) - \bar{f})(g(x + t) - \bar{g})\, dt \right],$$

где

$$\bar{f} = \lim_{T \to \infty} \left[\frac{1}{2T} \int_{-T}^{T} f(t)\, dt \right] \quad \text{и} \quad \bar{g} = \lim_{T \to \infty} \left[\frac{1}{2T} \int_{-T}^{T} g(t)\, dt \right]$$

– *средние значения* f и g. Если $f = g$, то $z/z(0)$ называют *автокорреляционной функцией* и часто обозначают через $\rho(x)$. Деление на $z(0)$ представляет собой нормировку, так что $-1 \leqslant \rho(x) \leqslant 1$.

Свертка и корреляция допускают графическую интерпретацию; такая интерпретация дается в любом стандартном справочнике, например в [Weaver, 1983] или [Brigham, 1974].

Наиболее важное свойство свертки $f * g$ состоит в том, что ее преобразование Фурье является произведением преобразования Фурье функции f и преобразования Фурье функции g. Обратно, преобразование Фурье произведения $f \cdot g$ является сверткой преобразования Фурье функции f и преобразования Фурье функции g. На с. 58 книги [Brigham, 1974] сказано, что это свойство, возможно, является самым важным и мощным инструментом современного научного анализа; и, действительно, это одно из наиболее интенсивно используемых средств анализа. Аналогичный результат имеет место и для ковариации: преобразование Фурье функции $f \circ g$ равно произведению преобразования Фурье функции f на функцию, комплексно сопряженную к преобразованию Фурье функции g.

Если функции f и g имеют нулевые средние, т. е. $\bar{f} = \bar{g} = 0$, то формулы для свертки и ковариации различаются только в одном знаке, а автокорреляция отличается от корреляции двух разных функций весовым множителем. Можно подумать, что эти различия несущественны. На самом же деле свертка и ковариация интерпретируются по-разному. Как отмечалось выше, свертку часто интерпретируют как осреднение. Важность этого наблюдения состоит в том, что иногда можно «снять» осреднение при помощи преобразования Фурье. Автокорреляция является мерой отличия g от такой же, но сдвинутой функции. Они применяются в статистических тестах и для обнаружения скрытой периодичности в данных (см. пример 11.6).

При перенесении этих определений на дискретные последователь-

ности конечной длины необходима некоторая осторожность. Если заменить интегралы суммами, то не очевидно, какими должны быть пределы индексов суммирования. Стандартный подход заключается в том, что последовательности g_i и h_i, $i = 0, \ldots, N-1$, продолжают и делают периодическими, так, чтобы все индексы, положительные и отрицательные, имели смысл. В задачах обработки изображений период оставляют равным N, но при обработке сигналов и в анализе временных рядов часто сначала продолжают последовательности до длины $2N$, присоединяя к ним по N нулей, а затем делают их $2N$-периодическими. Для периодических последовательностей можно определить дискретные аналоги ковариации и свертки. Широко используются оба соглашения, но для последующего изложения мы выбрали второе. Если f_i и g_i – числа, $i = 0, \ldots, N-1$, множества которых расширены до $2N$ элементов добавлением нулей, то *дискретной ковариацией* наборов f и g является набор чисел

$$z_i = \sum_{j=0}^{N-1} (f_i - \bar{f})(g_{i+j} - \bar{g}) = \sum_{j=0}^{N-i-1} (f_i - \bar{f})(g_{i+j} - \bar{g}), \quad i = 0, \ldots, N-1,$$

где

$$\bar{f} = \frac{1}{N} \sum_{i=0}^{N-1} f_i, \quad \bar{g} = \frac{1}{N} \sum_{i=0}^{N-1} g_i .$$

Если $f_i = g_i$, то *дискретной автокорреляцией* набора f является набор чисел z_i/z_0, $i = 0, \ldots, N-1$. Если считать заданные числа периодически продолженными с периодом $2N$, то *дискретной сверткой* (называемой также круговой или циклической сверткой) наборов f и g является набор чисел h_i, $i = 0, \ldots, N-1$, определенных формулой

$$h_i = \sum_{j=0}^{N-1} f_i g_{i-j}, \quad i = 0, \ldots, N-1 .$$

Дискретное преобразование Фурье дискретной свертки двух наборов чисел f_i и g_i равно произведению дискретного преобразования Фурье набора g_i и дискретного преобразования Фурье набора f_i, так же как и в непрерывном случае. Аналогичная связь существует и для дискретной ковариации: дискретное преобразование Фурье от $f \circ g$ равно произведению дискретного преобразования Фурье набора f на набор чисел, комплексно сопряженных к дискретному преобразованию Фурье набора g. В качестве приложения возможно быстрое вычисление автокорреляции последовательности g_i на основе теоремы о свертке:

(1) Вычитаем среднее, чтобы последовательность длины N имела нулевое среднее значение. Добавляем N нулей, тем самым расширяя g до длины $2N$.

(2) Вычисляем дискретное преобразование Фурье $2N$-точечной последовательности g.

(3) Вычисляем квадраты модулей компонент преобразования, что

сводится к умножению преобразования Фурье на сопряженное к нему. Если используется комплексное преобразование G_j, то получается $|G_j|^2$, $j < 2N$. Если используются вещественные преобразования a_j, b_j, то получается $(a_j^2 + b_j^2)/2$ при $j < N$ и в два раза бóльшая величина при $j = N$. В случае использования вещественных преобразований полагаем остальные N компоненты равными нулю.

(4) Вычисляем *обратное преобразование* этой $2N$-точечной последовательности и отбрасываем вторую половину результата.

Пример 11.6. Вычисление автокорреляции данных el Niño непосредственно и с использованием БПФ.

Чтобы показать, как вычислять дискретную автокорреляцию, рассмотрим данные el Niño из § 9 и найдем автокорреляцию этих данных непосредственно, т. е. по приведенной выше формуле, а также с использованием БПФ.

```
C     Поиск автокорреляции данных el Niño непосредственно и с
C     использованием БПФ.
C
      INTEGER N
      PARAMETER (N = 168)
      REAL EL(0:2*N-1), WSAVE(4*(2*N)+15), ACOV(0:N-1),
     *     A(2*N), B(2*N), ACOVR(0:2*N-1)
      COMPLEX CEL(0:2*N-1), CORR(0:2*N-1)
      LOGICAL EX
C
C     В комплексном случае требуется N слов памяти для EL, по
C     2N - для ACOV, CEL, CORR и 4*(2N)+15 для WSAVE.
C     Максимальное значение N равно 168.
      INQUIRE (FILE='ELNINO.DAT', EXIST=EX, ERR=1000)
      IF(.NOT.EX)GO TO 1000
      OPEN (UNIT=8, FILE='ELNINO.DAT', ERR=1000)
C
C     Ввод данных, поиск среднего.
      SUM=0.0
      DO 100 I=0, N-1
        READ(8,*) EL(I)
        SUM=SUM+EL(I)
  100 CONTINUE
      DO 101 I=0, N-1
        EL(I)=EL(I)-SUM/N
C     Вычитание среднего и добавление N нулей для применения
C     вещественного и комплексного БПФ.
        CEL(I)=CMPLX(EL(I),0.0)
        EL(I+N)=0.0
        CEL(I+N)=0.0
```

```
      101 CONTINUE
C
C     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
C     Непосредственное вычисление. Простое суммирование по всем
C     данным.
C     Просто, но медленно.
C
        DO 110 J = 0, N - 1
          ACOV(J) = 0.0
          DO 110 M = 0, N - 1 - J
            ACOV(J) = ACOV(J) + EL(M) * EL(M + J)
      110 CONTINUE
C
C     Вывод автокорреляции после умножения на весовой множитель.
          WRITE (*,*)
     * 'EX 11.6: AUTOCORRELATION (DIRECT)'
          WRITE (*, '(5E14.6)') ((ACOV(I)/ACOV(0)), I = 0, N - 1)
C     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
C     Подход с применением БПФ (комплексного).
C     Вычисление БПФ данных длины 2N.
C     Вычисление квадратов модулей компонент преобразования и
C     запись их в виде вещественных частей комплексного массива.
C     Вычисление обратного преобразования, отбрасывание его второй
C     половины и всех мнимых частей (которые равны нулю), умно-
C     жение на длину последовательности 2N.
          CALL CFFTI (2 * N, WSAVE)
          CALL CFFTF (2 * N, CEL, WSAVE)
C     CFFTF выдает преобразования без весовых множителей. Для
      получения настоящих преобразований их надо разделить на (2N).
          DO 120 I = 0, 2 * N - 1
            CORR(I) = ABS(CEL(I)/(2 * N)) * * 2
      120 CONTINUE
C     Поскольку мы вычисляем произведение преобразования на его
C     сопряженное, каждое из них надо разделить на (2N), т.е. все
C     произведение разделить на (2N) * * 2.
          CALL CFFTB (2 * N, CORR, WSAVE)
C
          DO 121 I = 0, N - 1
            ACOV(I) = REAL(CORR(I)) * (2 * N)
      121 CONTINUE
C     Автокорреляция есть обратное преобразование, умноженное
C     на длину последовательности 2N.
C     В реальной программе введение весовых множителей было бы
C     реализовано единственным делением на 2N. Мы разбили его на
```

```
C     части для наглядности.
      WRITE(*,*)
     * 'EX 11.6: AUTOCORRELATION (COMPLEX FFT)'
      WRITE(*,'(5E14.6)')((ACOV(I)/ACOV(0)), I = 0, N - 1)
C
C     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C
C     Подход с применением БПФ (вещественного).
C     Вычисление БПФ данных длины 2N.
C     EZFFTF выдает A и B с правильными весами, поэтому для
C     получения преобразования дополнительные множители не
C     требуются.
C     Вычисление массива квадратов каждой частотной компоненты
C     и запись их в массив A косинус-преобразований для проведения
C     обратного преобразования. Элементы массива B полагаем
C     равными нулю.
C     B массивах A и B по N компонент.
C     Заметим, что при вычислении этих величин нужна осторожность:
C     это 0.5*(A(I)**2 + B(I)**2) при I < N, но в два раза больше
C     при I = N.
C     Вычисление обратного преобразования, отбрасывание его второй
C     половины.
C
      CALL EZFFTI (2*N, WSAVE)
      CALL EZFFTF (2*N, EL, AZERO, A, B, WSAVE)
      AZERO = AZERO*AZERO
C
      DO 150 I = 1, N
        IF(I.NE.N) THEN
          A(I) = (A(I)**2 + B(I)**2)/2.0
        ELSE
          A(I) = (A(I)**2 + B(I)**2)
        END IF
        B(I) = 0.0
  150 CONTINUE
      CALL EZFFTB (2*N, ACOVR, AZERO, A, B, WSAVE)
      WRITE(*,*)
     * 'EX 11.6: AUTOCORRELATION (REAL FFT)'
      WRITE(*,'(5E14.6)')((ACOVR(I)/ACOVR(0)), I = 0, N - 1)
C
      STOP
 1000 WRITE(*,*)'CANNOT FIND THE DATA FILE: ELNINO.DAT'
      END
```

Полученная автокорреляция показана на рис. 11.10. Обратите внимание на пики, возникающие через каждые двенадцать точек. Из этого факта мы заключаем, что величина, заданная в некоторой точке, сильно

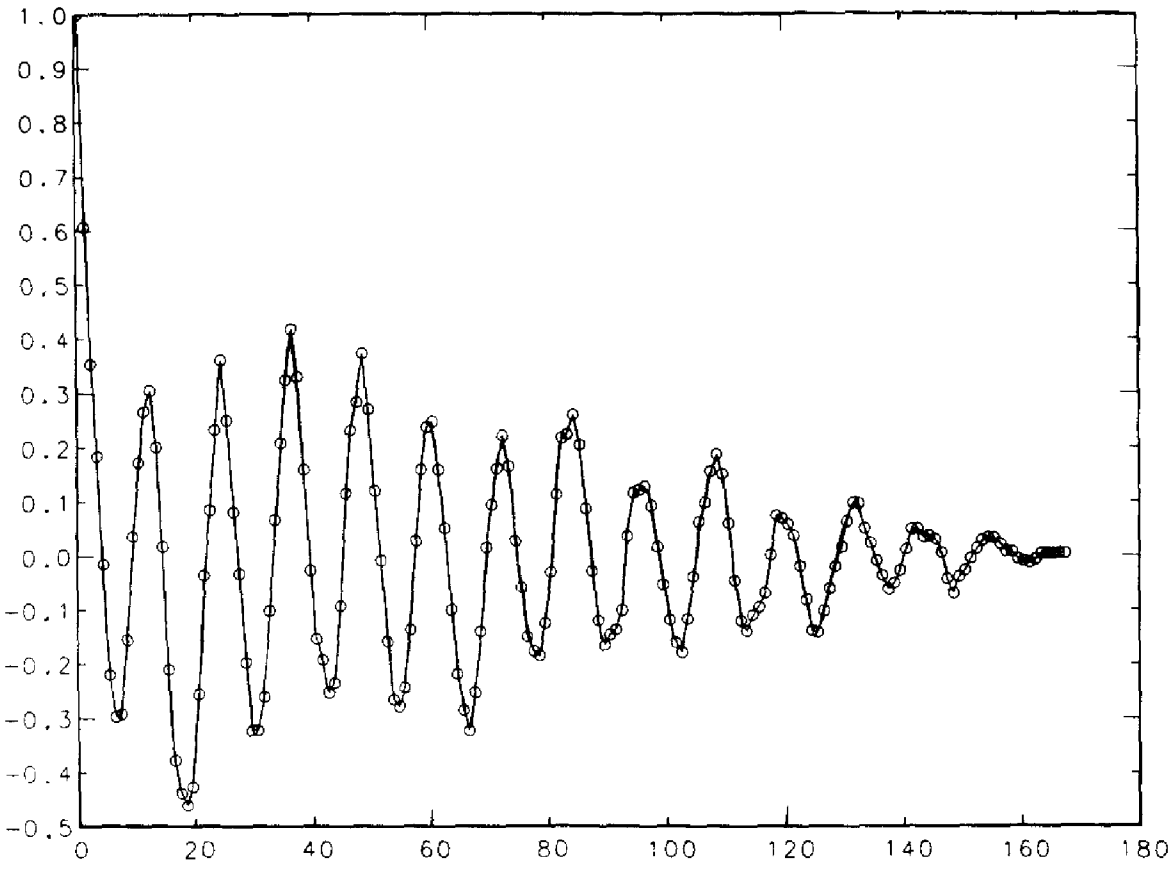


Рис. 11.10. Автокорреляция данных el Niño.

коррелирует с данными того же самого месяца, но одним или более годами позднее. Можно обнаружить и свидетельства существования циклов с бóльшим периодом. □

В качестве последнего примера этого параграфа опишем в упрощенной форме типичное приложение свертки. Задача состоит в том, чтобы сделать неясное, плохо сфокусированное изображение на фотографической пластинке более резким. Подходящим здесь будет понятие точечного источника, т. е. яркой точки на темном фоне. Если произвольный источник света можно рассматривать как сумму точечных источников разных интенсивностей, то для определения результирующего изображения от общего источника можно использовать знание изображений от точечных источников. То, что соответствует точечному источнику, называется *функцией точечного источника* или *функцией влияния единичного импульса*. Ее можно представить себе как меру ослабления источника вследствие прохождения света через физическую аппаратуру перед попаданием на фотопластинку. Эта функция описывает расплывшееся изображение от резкого точечного источника света. В стандартной модели этого явления предполагается:

(1) что влияние точечного источника одинаково с точностью до сдвига, где бы он ни был помещен на плоскости источников (это

называется свойством *инвариантности относительно сдвига*);

(2) что процесс линеен, т. е. результат от добавления новых источников получается сложением изображений от каждого источника.

Тогда можно показать, что изображение описывается сверткой источников с функцией влияния точечного источника.

Сформулируем то же самое иначе. Пусть p – точка на пластине, а $\psi_2(p)$ – наблюдаемая в изображении интенсивность света. Отделенный от изображения источник дает в соответствующей точке интенсивность $\psi_1(p)$. В идеальных условиях без потерь было бы $\psi_2(p) = \psi_1(p)$, но пыльный воздух и линзы между источником и фотопластинкой нарушают это равенство. Если принять, что интенсивность изображения в точке p дается взвешенным средним величин интенсивностей источников, то ее можно представить интегралом

$$\psi_2(p) = \int k(p,q)\ \psi_1(q)\,dq,$$

т. е. влияние на изображение в точке p, оказываемое областью источника вблизи его точки q, составляет $k(p,q)\psi_1(q)\,dq$. Если предположить далее, что размывание происходит во всех точках источника одинаково, то единственное, что имеет значение, – это расстояние между p и q, и, следовательно, $k(p,q) = k(p-q)$. Если размеры линзы бесконечны, то формула для ψ_2 превращается в двумерную свертку. Мы хотим устранить нечеткость изображения, что эквивалентно отысканию ψ_1. Часто мы знаем что-то о k или можем чем-то аппроксимировать эту функцию. В конце концов, функция точечного источника k задает операцию осреднения, и вполне вероятно, что значение $k(p-q)$ велико при $p = q$ и что эта функция быстро убывает к нулю при удалении p и q друг от друга. Используя прописные буквы для обозначения преобразований Фурье, получаем из предыдущего параграфа, что

$$\psi_2 = k * \psi_1 \quad \text{влечет за собой } \Psi_2 = K \cdot \Psi_1,$$

так что для отыскания ψ_1 нужно разделить преобразование Фурье функции ψ_2 на преобразование Фурье функции k и найти обратное преобразование частного.

Настоящие фотографические изображения довольно редко удается описать столь простой моделью. Например, если H обращается в нуль хотя бы в одной точке, то формулы теряют силу; кроме того, они не обеспечивают включение в модель всегда присутствующего «шума». Тем не менее эта модель дает возможность проиллюстрировать применение преобразования Фурье.

11.14. Историческая справка: быстрое преобразование Фурье

Еще в докомпьютерные времена анализ Фурье был существенной частью исследований в некоторых областях, но выполнение 32-точеч-

ного дискретного преобразования Фурье требовало $32^2 = 1024$ арифметических операций – внушительного объема вычислений. Наиболее раннее документированное описание быстрого алгоритма восходит к Гауссу, к работе, написанной им в 1866 г. на латыни. В 1903 г. Рунге описал метод, использующий для сокращения объема вычислений свойства симметрии синусов и косинусов, а в вышедшей позднее книге Кёнига и Рунге было показано, как использовать периодичность этих тригонометрических функций для получения $2N$-точечного дискретного преобразования Фурье по двум N-точечным преобразованиям, затрачивая немногим более N операций. Если длина набора данных равна N и N представляет собой степень двойки, то можно разбить данные на $\log_2 N$ подмножеств и применить для вычисления дискретного преобразования Фурье алгоритм Кёнига–Рунге с $\log_2 N$ сдваиваниями, что потребует общих затрат около $N \log_2 N$ операций вместо N^2. Использование симметрии снижает только константу пропорциональности в оценке числа операций, в то время как последовательные сдваивания приводят к появлению множителя $\log_2 N$. Это различие не было существенным при малых значениях N, с которыми приходилось работать во времена Рунге и Кёнига, и его просто не заметили, хотя метод и был опубликован известными авторами. В 1942 г. Дэниелсон и Ланцош описали метод в более общей форме, явно указав на $N \log_2 N$ операций. Однако выполнение таких вычислений вручную по-прежнему оставалось практически непосильной задачей. В 1948 г. Томас сообщил о преобразовании, выполненном им при помощи бухгалтерской счетной машины в фирме IBM. Он потратил на это преобразование три месяца.

После появления цифровых компьютеров можно было ожидать, что быстрый алгоритм немедленно найдет широкое применение, однако этот алгоритм не пользовался большой известностью, и в 60-е годы дискретное преобразование Фурье оставалось непозволительно дорогим. В начале 60-х годов Ричард Гарвин изучал твердый гелий, и ему потребовалось выполнить некоторое количество преобразований Фурье. Он обратился к своему коллеге Джону Тьюки и попросил его придумать способ эффективного выполнения этих вычислений. Тьюки предложил ему идею алгоритма со сложностью $N \log_2 N$, но Гарвину нужна была помощь в программировании, и он передал в вычислительный центр IBM заказ на программу этого алгоритма. Джон Кули был новичком в вычислительном центре и занимался там некоторыми своими разработками. Поскольку его сочли единственным, кому нечего делать, ему и дали задание Гарвина. Кули оно показалось интересным, но свою собственную работу он считал более важной; потребовался некоторый нажим, чтобы заставить его поработать над этим заданием в свободное время. В конечном счете Кули вручил Гарвину программу и решил, что может вернуться к настоящей работе. Но Гарвин увидел широкую область применения этой программы и указал на нее многим своим коллегам. Кули стал получать заказы на программу и, в конце концов,

на статью с описанием того, что он сделал. Его совместная с Тьюки статья была опубликована в 1965 году и стала общепризнанной отправной точкой современного применения БПФ. Как это было с рядом других хороших идей, вокруг БПФ крутились работы многих ученых, но нужно было определенное сочетание обстоятельств, чтобы привлечь к нему всеобщее внимание. Огромное значение БПФ не вызывает никаких сомнений. Многие знающие люди считают, что это наиболее важный вклад в науку о вычислениях после изобретения хранимой в памяти программы. Напоминая о работе Гаусса, которая прошла практически незамеченной, Кули советует стремящимся к признанию молодым ученым: «Не пишите работы на латыни».

11.15. Задачи

11.1. Вычислите 32 значения функции $\mathrm{erf}(x)$ на $[0,3]$ с постоянным шагом. Примените программу EZFFTF для отыскания $t_n(x)$ при $n = 4$, 8 и 16. Постройте графики этих функций. Насколько хорошо они приближают erf? Повторите вычисления, изменив данные путем вычитания из них $\mathrm{erf}(3)\frac{x}{3}$. При этом первая и последняя из заданных величин обратятся в нуль.

11.2. При помощи программы EZFFTF постройте периодограмму, которая была показана на рис. 11.2. Если величины B, d и α, которые вы вычислили в гл. 9, иные, то используйте их. Отличается ли положение пиков на вашей периодограмме от положения пиков на нашей периодограмме?

11.3. При помощи программы EZFFTF выполните вычисления, необходимые для построения графиков, приведенных на рис. 11.5.

11.4. При помощи программы RNOR из гл. 10 постройте 256 значений $f_i = \sin(i/10) + \eta_i$, $i = 1, \ldots, 256$, при $\eta_i = N(0,0.1)$. Изобразите эти данные.

(а) Используя программу EZFFTF, вычислите синус- и косинус-преобразования Фурье и постройте их графики. Найдите периодограмму, изобразите ее.

(б) Замените некоторые из высокочастотных коэффициентов нулями и после этого примените программу EZFFTF для выполнения обратного преобразования. Изобразите результат.

11.5. Задачи на понимание результатов, выдаваемых программами БПФ.

(а) Напишите программу, строящую выборку значений функции $f(t) = \cos(2\pi t)$ в N точках $t_i = i\Delta$, $i = 0, \ldots, N-1$, с постоянным по t шагом Δ. Примените к этому набору программу EZFFTF и выведите величины вычисленных коэффициентов Фурье для различных значений N и Δ, дайте интерпретацию выведенных чисел, используя ряд Фурье функции f. Для типичного значения Δ,

скажем $\Delta = 1/8$, сравните полученные результаты при $N = 8$ и при $N = 9$. Во втором случае первый и последний элементы выборки значений функции равны.

(б) Повторите пункт (а), используя программу CFFTF. В случае $N = 8$, $\Delta = 1/8$ убедитесь, что вы можете объяснить, какие из комплексных коэффициентов должны быть нулевыми и какие веса надо приписать ненулевым коэффициентам. Повторите вычисления, изменив данные (умножив i-ю заданную величину на $(-1)^i$). Напечатайте абсолютные величины коэффициентов ДПФ и объясните полученные результаты.

(в) Повторите пункт (а) с выборкой значений функции в точках $t_i = 0.25 + i\Delta$. Объясните, почему результаты отличаются от полученных в пункте (а).

(г) Вычислите дискретное преобразование Фурье единичного импульса, т.е. найдите ДПФ массива $(1, 0, \ldots, 0)$, используя программу EZFFTF. Какого вида функция $f(t)$ соответствует такому массиву? Проверьте, выполняется ли закон сохранения энергии. Можете ли вы дать интерпретацию этих результатов?

(д) Преобразованием Фурье функции $f(x) = \exp(-x^2)$ является $A(\omega) = 2\sqrt{\pi} f(\pi\omega)$, $B(\omega) = 0$. Иными словами, с точностью до постоянного множителя получается та же функция. Возьмите частные значения $f(x)$ в $N = 4$, 8, 16 и 32 равномерно распределенных на отрезке $[a, b] = [-5, 5]$ точках, т.е. в точках $a + i\Delta$, $i = 0, \ldots, N - 1$, $\Delta = (b - a)/N$. Для вычисления дискретного преобразования Фурье этого набора примените программу EZFFTF. Сравните полученные результаты с точным преобразованием Фурье исходной функции. Не забудьте учесть, что $a \neq 0$. Какова наибольшая погрешность?

(е) Повторите пункт (д), но с использованием программы CFFTF. Не центрируйте частоты.

(ж) Повторите пункт (е), но с центрированием частот, т.е. перед обращением к CFFT умножьте i-е заданное число на (-1).

11.6. Распространение света.

(а) Две параллельные плоскости находятся на расстоянии z. Световое возмущение ψ_1 когерентного излучения (например, луч лазера) в плоскости 1 распространяется (дифрагирует) к плоскости 2 и порождает световое возмущение ψ_2 во второй плоскости. Все детекторы, включая человеческий глаз, реагируют на квадрат модуля возмущения. Другими словами, мы видим не ψ, а $|\psi|^2$. Можно показать, что если z достаточно велико, то ψ_2 пропорционально преобразованию Фурье функции ψ_1–свойство, называемое дифракцией Фраунгофера. Если возмущение в плоскости 1 представляет собой квадратный источник света единичной интенсивности, то дифракционная картина будет выглядеть так, как показано на рис. 11.9. Рассчитайте дифракционную картину круглого источника света единичной интен-

сивности. Постройте (по аналогии с рис. 11.9) график квадратного корня из найденной величины, используя программу для вычисления аксонометрической проекции. Построить такой график по точкам двумерного массива A неотрицательных элементов легко, если имеется программа построения N-точечного графика PLOT(N, X, Y). Пусть X0 – массив последовательных целых чисел. Добавьте приращение $i\Delta$ к i-му столбцу (или строке) массива A и поместите результат в Y. Добавьте такое же приращение к X0 и поместите результат в X. Затем постройте график по этим двум массивам.

(б) Пусть K = (N − 1)/2 и R = ((I − K)**2 + (J − K)**2)/S**2. Будем считать, что P(I, J) – массив, описывающий возмущение в плоскости 1; зададим P(I, J) = EXP(−R). Такой источник света называется двумерным гауссианом. Положите N = 64, а S пусть изменяется в пределах от 1 до 10. Вычислите дифракционную картину Фраунгофера от такого источника. Вы обнаружите, что с точностью до постоянного множителя она выглядит совершенно так же, как входные данные. При каком значении S изображение больше всего похоже на источник?

(в) Более точное описание возмущения в плоскости 2 в предположении, что плоскости отстоят друг от друга недалеко, дается уравнением дифракции Френеля. Для такого источника света, как в пункте (а), ψ_2 будет пропорционально преобразованию Фурье произведения $\psi_1(x, y)$ и $\exp(2\pi ir/z)$, где r – расстояние от центра плоскости до (x, y). Найдите картину дифракции Френеля для $z = 0.01, 0.1, 1.0, 10$ и 100. Изобразите результаты и убедитесь, что при больших z они аппроксимируют результаты пункта (а).

11.7. Эта задача иллюстрирует эффект удаления высокочастотных компонент изображения. Изображение размера 64 × 64 составьте из двух маленьких квадратов единичной интенсивности на черном фоне. Это значит, что P(I, J) = 0, за исключением точек с координатами $30 \leqslant I \leqslant 34$, $30 \leqslant J \leqslant 34$ и точек с координатами $31 \leqslant I \leqslant 33$, $2 \leqslant J \leqslant 4$, где P = 1.

(а) Постройте аксонометрический график изображения.

(б) При помощи программы CFFTF вычислите ДПФ изображения с такими весами, чтобы нулевая частота оказалась в центре. Затем замените нулями все частотные компоненты на расстоянии от центра, большем R. Выполните обратное преобразование и снова постройте график изображения. Напишите программу так, чтобы R можно было изменять. При уменьшении R разглядеть маленький квадрат будет все труднее. При каком значении R он исчезает?

11.8. Преобразование Фурье производной.

(а) С помощью формулы комплексного преобразования Фурье покажите, что преобразование Фурье функции $f'(t)$ равно $2\pi i\omega F(\omega)$, где $F(\omega)$ – преобразование Фурье функции $f(t)$. Можете

предположить, что $f(t)$ стремится к нулю при стремлении t к плюс или минус бесконечности. Разумно ли такое предположение? То же утверждение имеет место и для рядов Фурье: если коэффициенты комплексного ряда Фурье функции $f(t)$ равны C_j, то коэффициенты ряда Фурье функции $f'(t)$ равны $2\pi ijfC_j$ в предположении, что оба ряда сходятся.

(б) Начните со значений функции $f(t) = \exp(-t^2)$ в N точках на интервале $[a, b] = [-5, 5]$: $a + i\Delta$, $i = 0, \ldots, N-1$, $\Delta = (b-a)/N$. Вычислите дискретное преобразование Фурье с помощью программы CFFTF. Умножьте компоненты этого преобразования на $2\pi i\omega$, а затем выполните обратное преобразование результата. Изобразите то, что у вас получилось, рядом с графиком точной производной. Найдите наибольшую погрешность как функцию от N для $N = 8$, 16, 32 и 64. *Указания.* (1) Перед выполнением преобразования умножьте k-е значение функции на $(-1)^{k-1}$, $k = 1, \ldots, N$, для центрирования частот. (2) Чтобы правильно решить эту задачу, важно сопоставить k-ю компоненту преобразованного массива с ее настоящей частотой. Положим $j = (k-1) - N/2$ при изменении k от 1 до N (см. § 11). Тогда частота ω, отвечающая k-й компоненте преобразования, равна $\omega = j/(b-a)$. (3) После выполнения обратного преобразования снова умножьте результат на $(-1)^{k-1}$ и не забудьте в конце разделить его на N.

11.9. Вычисление неопределенного интеграла. Покажите, что если функция $g(x)$ разложена в сходящийся ряд Фурье

$$g(x) = \sum_{-\infty}^{\infty} C_n \exp(2\pi inx/(b-a)),$$

то разложение в ряд Фурье неопределенного интеграла от g имеет вид

$$g_I(x) \equiv \int_a^x g(t)\,dt = C_0(x-a) + \sum_{n=-\infty,\, n\neq 0}^{\infty} C_n \frac{\exp(2\pi inx/(b-a))}{2\pi in(b-a)}.$$

Предположим, что $C_0(b-a) = \int_a^b g(t)\,dt$ известно. Вот алгоритм оценивания $g_I(x)$. Вычисляем значения $g(x_i)$, $i = 1, \ldots, N$, $x_i = a + (i-1)\Delta$, $\Delta = (b-a)/N$, и выполняем дискретное преобразование Фурье этих чисел. Делим компоненту преобразования, отвечающую частоте $n/(b-a)$, на $2\pi in/(b-a)$; исключение составляет компонента, соответствующая нулевой частоте, которую мы полагаем равной нулю. Вычисляем обратное дискретное преобразование Фурье полученной последовательности. Тогда i-я компонента результата плюс $C_0(x_i - a)$ дает аппроксимацию интеграла $g_I(x_i)$. Примените этот алгоритм к $g(x) = \exp(-x^2)$, $[a, b] = [-5, 5]$ для $N = 8$, 16 и 32 и сравните его с правилом трапеций для тех же значений N. Сделайте то же самое для

$g(x) = 1/(1 + 25x^2)$. Для какой функции результат получился лучше? Можно ли было этого ожидать?

11.10. Пусть $g(x) = \exp(-x)$ при $x \geqslant 0$ и 0 при $x < 0$.

(а) Найдите $A(\omega)$ и $B(\omega)$. *Указание*: Вычислите $G(\omega)$.

(б) Проверьте, что

$$\int_0^{\infty} (A^2(\omega) + B^2(\omega))\, d\omega = 2 \int_{-\infty}^{\infty} g^2(x)\, dx.$$

(в) Найдите спектр мощности функции $g(x)$ и постройте его график.

(г) Найдите значения $g(x)$ в $N = 64$ равномерно распределенных на $[0, 5]$ точках и вычислите $A(K)$ и $B(K)$ с помощью программы EZFFTF. Вычислите периодограмму и постройте ее график на том же рисунке, что в пункте (в).

(д) *Используя коэффициенты из пункта (г)*, вычислите функцию $t_{N/2}(x)$ и постройте ее график; сравните его с графиком функции $g(x)$.

11.11. Файл ELNINO.DAT на дискете, приложенной к этой книге, содержит 168 точек данных, обсуждавшихся в § 9.

(а) Изобразите эти данные. Примените программу EZFFTF для вычисления A_k и B_k. Постройте график периодограммы.

(б) Вычислите с помощью программы CFFTF автокорреляционную функцию для этих данных; нанесите ее на график.

(в) Вычислите функции $t_m(x)$ для $m = 84, 42, 21, 15$ и постройте их графики.

11.12. Люди веками замечали, что лик солнца непостоянен и неравномерен, что циклическим образом на нем появляются случайно расположенные темные области. В 1848 году Рудольф Вольфер предложил правило, которое сводит количество и размеры этих темных пятен к одному числу. С помощью этого правила и архивных записей астрономы определили связанные с солнечными пятнами числа Вольфера начиная с 1700 года. Сегодня их измеряют многие обсерватории. Распространение данных по всему миру координируется Швейцарской федеральной обсерваторией на ежедневной, ежемесячной и ежегодной основе. Солнечная активность является циклической, и были обнаружены корреляции между изменениями чисел Вольфера и погодой, а также другими земными явлениями, важными для экономики; этим объясняется сохранение интереса к пятнам на Солнце. В файле SUNSPOT.DAT на дискете, приложенной к учебнику, содержатся средние числа Вольфера для каждого года с 1700 по 1987. Напишите программу считывания этих чисел и изобразите эти данные. Затем примените к ним программу EZFFTF. В данных нет сколько-нибудь заметного тренда, который пришлось бы вычитать. Постройте график мощности в зависимости от частоты, опустив член с $\omega = 0$. Существует ли одна доминирующая частота? Каков период доминирующей цикличности? В какие годы, начиная с нынешнего и до начала следующего века, солнечная активность будет наибольшей?

11.13. В соответствии с § 10, при простых N на выполнение программы EZFFTF затрачивается время $O(N^2)$. На рис. 11.8 показано, что время работы EZFFTF при простых N больше, чем при других значениях N, но неясно, будет ли тренд квадратическим. Приводимая ниже таблица дает более полное представление о временах работы EZFFTF при простых N.

Таблица 11.3

N	Время	N	Время	N	Время	N	Время
41	22	89	104	149	285	199	506
43	28	97	121	151	291	211	571
47	27	101	132	157	319	223	631
53	33	103	137	163	335	227	659
59	44	107	148	167	357	229	670
61	49	109	149	173	385	233	692
67	55	113	165	179	407	239	725
71	66	127	208	181	418	241	742
73	72	131	220	191	466	251	802
79	83	137	242	193	472		
83	88	139	247	197	495		

При помощи подпрограммы SQRLS подберите параметры модели $a + bt + ct^2$ так, чтобы она аппроксимировала измерения времен вплоть до $N = 151$. Вычислите значения построенной модельной функции в остальных точках N. Получилось ли хорошее соответствие данным?

11.14. Эта простая задача показывает, что наивысшая частота, которую можно анализировать в данных, равна половине частоты выборки. В темной комнате вращается колесо со скоростью 1 оборот в минуту. Фотографии делаются в моменты вспышек - при $t = 0$ и далее через каждые Δ секунд. На первой фотографии темное пятно на колесе появляется в его верхней точке, а на последующих фотографиях – в других положениях. Если $\Delta = 15$, то какой кажется скорость вращения колеса? Какой она кажется при $\Delta = 30, 45, 60$?

11.16. Прологи: EZFFTF, EZFFTB, CFFTF, CFFTB и CFFT2D

```
      SUBROUTINE EZFFTF (N, R, AZERO, A, B, WSAVE)
C*** НАЧАЛО ПРОЛОГА EZFFTF
C*** ДАТА НАПИСАНИЯ 790601 (ГГММДД)
C*** ДАТА ПЕРЕСМОТРА 830401 (ГГММДД)
C*** КАТЕГОРИЯ NO. J1A1
```

```
C*** КЛЮЧЕВЫЕ СЛОВА: Преобразование Фурье
C*** АВТОР: SWARZTRAUBER P.N. (NCAR)
C*** НАЗНАЧЕНИЕ: упрощенное вещественное периодическое пря-
C***                мое преобразование.
C*** ОПИСАНИЕ
C     Подпрограмма EZFFTF вычисляет коэффициенты Фурье дейст-
C     вительной периодической последовательности (анализ Фурье).
C     Преобразование определено ниже при описании выходных пара-
C     метров AZERO, A и B. Подпрограмма EZFFTF - это упрощенная,
C     но более медленная версия подпрограммы RFFTF.
C     Из книги D. Kahaner, C. Moler, S. Nash "Numerical Methods and
C              Software" Prentice-Hall, 1988
C
C     Входные параметры:
C       N - длина преобразуемого массива R. Метод наиболее эффек-
C           тивен, когда N является произведением малых простых
C           чисел.
C       R - вещественный массив длины N, содержащий преобразуемую
C           последовательность. Содержимое R не меняется.
C       WSAVE - рабочий массив, который должен быть описан в
C               программе, вызывающей EZFFTF, и иметь размер
C               по крайней мере 3 * N + 15.
C               Массив WSAVE должен быть инициализирован вы-
C               зовом подпрограммы EZFFTI(N, WSAVE), причем
C               для разных значений N должны использоваться
C               разные массивы WSAVE.
C               Инициализацию не нужно проводить заново, если N
C               остается неизменным. Таким образом, последующие
C               преобразования можно выполнить быстрее, чем
C               первое. Один и тот же массив WSAVE можно
C               использовать при обращении к EZFFTF и EZFFTB.
C     Выходные параметры:
C       AZERO - сумма по I от I = 1 до I = N величин R(I)/N.
C       A, B  - для четного N B(N/2) = 0., а A(N/2) является суммой
C               по I от I = 1 до I = N величин (-1)**(I - 1) * R(I)/N,
C               при четном N определим KMAX = N/2 - 1; при
C               нечетном N определим KMAX = (N - 1)/2, тогда для
C               K = 1, ..., KMAX A(K) равно сумме по I от I = 1
C               до I = N величин
C                  2./N * R(I) * cos(K * (I - 1) * 2 * PI/N), B(K) равно
C               сумме по I от I = 1 до I = N выражений
C                  2./N * R(I) * SIN(K * (I - 1) * 2 * PI/N).
C*** ССЫЛКИ: НЕТ
C*** ВЫЗЫВАЕМЫЕ ПОДПРОГРАММЫ: RFFTF
C*** КОНЕЦ ПРОЛОГА EZFFTF
```

```
      SUBROUTINE EZFFTB (N, R, AZERO, A, B, WSAVE)
C*** НАЧАЛО ПРОЛОГА EZFFTB
C*** ДАТА НАПИСАНИЯ 790601 (ГГММДД)
C*** ДАТА ПЕРЕСМОТРА 830401 (ГГММДД)
C*** КАТЕГОРИЯ NO. J1A1
C*** КЛЮЧЕВЫЕ СЛОВА: Преобразование Фурье
C*** АВТОР: SWARZTRAUBER P.N. (NCAR)
C*** НАЗНАЧЕНИЕ: упрощенное вещественное периодическое обратное преобразование.
C*** ОПИСАНИЕ
C     Подпрограмма EZFFTB вычисляет вещественную периодическую
C     последовательность по ее коэффициентам Фурье (синтез Фурье).
C     Преобразование определено ниже при описании выходного параметра R. Подпрограмма EZFFTB – это упрощенная, но более
C     медленная версия программы RFFTB.
C
C     Из книги D. Kahaner, C. Moler, S. Nash "Numerical Methods and
C          Software" Prentice-Hall, 1988
C
C     Входные параметры:
C       N      – длина выходного массива R. Метод наиболее эффективен, когда N является произведением малых простых чисел.
C       AZERO  – коэффициент Фурье при нулевой частоте.
C       A, B   – массивы, содержащие остальные коэффициенты Фурье.
C                Содержимое этих массивов не меняется.
C                Длина массивов зависит от того, четно N или нечетно.
C                Если N четно, то для A и B требуется по N/2
C                слов памяти.
C                Если N нечетно, то для A и B требуется по (N – 1)/2
C                слов памяти.
C       WSAVE  – рабочий массив, который должен быть описан в программе, вызывающей EZFFTB, и иметь длину по
C                крайней мере 3 * N + 15.
C                Массив WSAVE должен быть инициализирован вызовом подпрограммы EZFFTI(N, WSAVE), причем
C                для разных значений N должны использоваться
C                разные массивы WSAVE. Инициализацию не нужно
C                проводить заново, если N остается неизменным.
C                Таким образом, последующие преобразования можно
C                выполнить быстрее, чем первое.
C                Один и тот же массив WSAVE можно использовать
C                при обращении к EZFFTF и EZFFTB.
C     Выходные параметры:
C       R      – при четном N определим KMAX = N/2, при нечетном
C                N определим KMAX = (N – 1)/2, тогда при I = 1, ..., N
```

```
C                R(I) = AZERO плюс сумма по K от K = 1 до
C                K = KMAX величин A(K)*cos(K*(I - 1)*2*PI/N) +
C                B(K)*SIN(K*(I - 1)*2*PI/N)
C                Комплексная форма записи:
C                Для J = 1, ..., N
C                R(J) равно сумме по K от K = -KMAX до K = KMAX
C                величин C(K)*EXP(I*K*(J - 1)*2*PI/N),
C                где
C                C(K) = .5*CMPLX(A(K), - B(K)) для K = 1, ..., KMAX,
C                C(-K) = CONJG(C(K)),
C                C(0) = AZERO и I = SQRT(-1).
C                Амплитудно-частотная форма записи:
C                Для I = 1, ..., N
C                R(I) равно AZERO плюс сумма по K от K = 1 до
C                K = KMAX величин
C                ALPHA(K)*COS(K*(I - 1)*2*PI/N + BETA(K)),
C                где
C                ALPHA(K) = SQRT(A(K)*A(K) + B(K)*B(K)),
C                COS(BETA(K)) = A(K)/ALPHA(K),
C                SIN(BETA(K)) = - B(K)/ALPHA(K).
C*** ССЫЛКИ: НЕТ
C*** ВЫЗЫВАЕМЫЕ ПОДПРОГРАММЫ: RFFTB
C*** КОНЕЦ ПРОЛОГА EZFFTB

      SUBROUTINE CFFTF(N. C, WSAVE)
C*** НАЧАЛО ПРОЛОГА CFFTF
C*** ДАТА НАПИСАНИЯ 790601 (ГГММДД)
C*** ДАТА ПЕРЕСМОТРА 870721 (ГГММДД)
C*** КАТЕГОРИЯ NO. J1A2
C*** КЛЮЧЕВЫЕ СЛОВА: Преобразование Фурье
C*** АВТОР: SWARZTRAUBER P.N. (NCAR)
C*** НАЗНАЧЕНИЕ: прямое преобразование комплексной периоди-
C                ческой последовательности.
C*** ОПИСАНИЕ
C
C    Из книги D. Kahaner, C. Moler, S. Nash
C                "Numerical Methods and Software"
C                Prentice-Hall, 1988
C    Подпрограмма CFFTF вычисляет прямое комплексное дискретное
C    преобразование Фурье (анализ Фурье), или, что то же самое,
C    вычисляет коэффициенты Фурье комплексной периодической после-
C    довательности.
C    Результат преобразования не нормируется. Для получения нор-
C    мированного преобразования выдаваемые величины надо
C    разделить на N. Иначе, если после вызова CFFTF обратиться
C    к CFFTB, члены исходной последовательности будут умножены
C    на N.
```

```
C     Используемый в подпрограмме CFFTF массив WSAVE должен
C     быть инициализирован вызовом подпрограммы
C     CFFTI(N, WSAVE).
C
C     Входные параметры:
C       N       – длина комплексной последовательности C.
C                 Метод наиболее эффективен, когда N является
C                 произведением малых простых чисел.
C       C       – комплексный массив длины N, содержащий заданную
C                 последовательность. Информация в этом массиве не
C                 сохраняется.
C       WSAVE   – вещественный рабочий массив, который должен быть
C                 описан в программе, вызывающей CFFTF, и иметь
C                 длину по крайней мере 4*N + 15. Массив WSAVE
C                 должен быть инициализирован вызовом подпрограм-
C                 мы CFFTI(N, WSAVE), причем для разных значений
C                 N должны использоваться разные массивы WSAVE.
C                 Инициализацию не нужно проводить заново, если N
C                 остается неизменным. Таким образом, последующие
C                 преобразования можно выполнить быстрее, чем
C                 первое. Один и тот же массив WSAVE можно ис-
C                 пользовать при обращении к CFFTF и CFFTB.
C     Выходные параметры:
C       C       – при J = 1, ..., N
C                 C(J) = сумме по K = 1, ..., N величин
C                 C(K)*EXP(-I*(J-1)*(K-1)*2*PI/N),
C                 где
C                 I = SQRT(-1).
C       WSAVE   – содержит инициализированные значения, которые не
C                 должны изменяться между обращениями к под-
C                 программам CFFTF и CFFTB.
C
C*** ССЫЛКИ: НЕТ
C*** ВЫЗЫВАЕМЫЕ ПОДПРОГРАММЫ: CFFTF1
C*** КОНЕЦ ПРОЛОГА CFFTF

      SUBROUTINE CFFTB(N, C, WSAVE)
C*** НАЧАЛО ПРОЛОГА CFFTB
C*** ДАТА НАПИСАНИЯ 790601 (ГГММДД)
C*** ДАТА ПЕРЕСМОТРА 870721 (ГГММДД)
C*** КАТЕГОРИЯ NO. J1A2
C*** КЛЮЧЕВЫЕ СЛОВА: Преобразование Фурье
C*** АВТОР: SWARZTRAUBER P.N. (NCAR)
C*** НАЗНАЧЕНИЕ: ненормированное обратное по отношению к
C                 CFFTF преобразование.
C*** ОПИСАНИЕ
```

```
C     Из книги D. Kahaner, C. Moler, S. Nash
C                "Numerical Methods and Software"
C                Prentice-Hall, 1988
C
C     Подпрограмма CFFTB вычисляет обратное комплексное дискрет-
C     ное преобразование Фурье (синтез Фурье), или, что то же самое,
C     строит комплексную периодическую последовательность по ее
C     коэффициентам Фурье.
C     Если после вызова CFFTF обратиться к CFFTB, то члены
C     исходной последовательности будут умножены на N.
C     Используемый в подпрограмме CFFTB массив WSAVE должен
C     быть инициализирован вызовом подпрограммы
C     CFFTI(N, WSAVE).
C
C     Входные параметры:
C       N      - длина комплексной последовательности C.
C                Метод наиболее эффективен, когда N является про-
C                изведением малых простых чисел.
C       C      - комплексный массив длины N, содержащий заданную
C                последовательность. Информация в этом массиве не
C                сохраняется.
C       WSAVE  - вещественный рабочий массив, который должен быть
C                описан в программе, вызывающей CFFTB, и иметь
C                длину по крайней мере 4 * N + 15.
C                Массив WSAVE должен быть инициализирован вы-
C                зовом подпрограммы CFFTI(N, WSAVE), причем для
C                разных значений N должны использоваться разные
C                массивы WSAVE. Инициализацию не нужно прово-
C                дить заново, если N остается неизменным. Таким
C                образом, последующие преобразования можно вы-
C                полнить быстрее, чем первое. Один и тот же массив
C                WSAVE можно использовать при обращении к CFFTF
C                и CFFTB.
C
C     Выходные параметры:
C       C      - для J = 1, ..., N
C                C(J) = сумме по K = 1, ..., N выражений
C                C(K) * EXP(I * (J - 1) * (K - 1) * 2 * PI/N),
C                где
C                I = SQRT(-1).
C       WSAVE  - содержит инициализированные значения, которые
C                не должны изменяться между обращениями к под-
C                программам CFFTF и CFFTB.
C
C*** ССЫЛКИ: НЕТ
C*** ВЫЗЫВАЕМЫЕ ПОДПРОГРАММЫ: CFFTB1
```

```
C*** КОНЕЦ ПРОЛОГА CFFTB

      SUBROUTINE CFFT2D(N, F, LDF, W, FORWD)
C*** Начало пролога CFFT2D
C*** ДАТА НАПИСАНИЯ 870811 (ГГММДД)
C*** ДАТА ПЕРЕСМОТРА 870811 (ГГММДД)
C*** КАТЕГОРИЯ NO. J1B
C*** КЛЮЧЕВЫЕ СЛОВА: двумерное преобразование Фурье, БПФ.
C*** АВТОР: KAHANER D.K. (NBS)
C*** НАЗНАЧЕНИЕ: двумерное комплексное быстрое преобразование
C                 Фурье.
C*** ОПИСАНИЕ
C     Прямое или обратное двумерное быстрое преобразование Фурье
C     комплексной матрицы F размера N × N.
C
C     Из книги D. Kahaner, C. Moler, S. Nash
C              "Numerical Methods and Software"
C              Prentice-Hall, 1988.
C
C     Выходные параметры:
C       N(INTEGER)       - число строк и столбцов в преобразуемой
C                          матрице F. Вы должны задать N > 0, ЭТО
C                          НЕ ПРОВЕРЯЕТСЯ.
C       F(COMPLEX)       - преобразуемый массив комплексных ве-
C                          личин размера N × N. Результат преоб-
C                          разования будет записан в него же.
C       LDF(INTEGER)     - ведущая (первая) размерность комплекс-
C                          ного массива в вызывающей CFFT2D
C                          подпрограмме. Например, если массив
C                          F описан в операторе COMPLEX как
C                          F (0:15, 0:20) или F (16, 21), то задайте
C                          LDF = 16. Вы должны задать LDF > = N,
C                          ЭТО НЕ ПРОВЕРЯЕТСЯ.
C       W(COMPLEX)       - рабочий массив. В вызывающей програм-
C                          ме его размер должен быть описан по
C                          крайней мере в 6N + 15 действительных
C                          слов или в 3N + 8 комплексных слов.
C       FORWD(LOGICAL)   - указатель прямого или обратного преоб-
C                          разования. Для прямого преобразования
C                          задайте.TRUE., для проведения обрат-
C                          ного преобразования - .FALSE..
C
C     Выходные параметры:
C       F(COMPLEX)       - прямое или обратное преобразование
C                          заданной матрицы. Результат не содер-
C                          жит весовых множителей, т.е. если после
```

```
C                вызова CFFT2D с FORWD = .TRUE.
C                обратиться к CFFT2D с FORWD =
C                .FALSE., то исходные данные будут
C                умножены на N * N.
C     Замечание.
C
C                В некоторых приложениях желательно
C                иметь такие весовые множители в пре-
C                образовании, чтобы центр квадрата из
C                частот размера N * N соответствовал ну-
C                левой частоте. Пользователь может до-
C                биться этого, заменяя вводимые исход-
C                ные данные
C                F(I, J) на
C                F(I, J) * (-1.) ** (I + J), I, J = 0, ..., N - 1.
C
C*** ССЫЛКИ: НЕТ
C*** ВЫЗЫВАЕМЫЕ ПОДПРОГРАММЫ: CFFTI, CFFTF, CFFTB
C*** КОНЕЦ ПРОЛОГА CFFT2D
```

Литература

Пакеты программ

1. The ACM collected algorithms – ACM Algorithms Distribution Service, IMSL Inc., Sixth Floor, NBC Building, 7500 Bellaire Blvd., Houston TX 77036.
2. Itpack – см. [Kincaid et al., 1982].
3. Linpack – National Energy Software Center, Argonne National Laboratory, Argonne IL 60439.
4. Minpack – National Energy Software Center, Argonne National Laboratory, Argonne IL 60439.
5. Eispack – National Energy Software Center, Argonne National Laboratory, Argonne IL 60439.
6. Quadpack – Prof. R. Piessens, Computer Science Department, Univ. of Leuven, Celestijnenlaan 200, A3030 Heverlee, Belgium.
7. Hompack – см. [Watson et al., 1987].
8. PORT – AT&T Bell Laboratories, 600 Mountain Ave., Murray Hill NJ 07074.
9. SLATEC – см. [Buzbee, 1984].
10. Sparspack – см. [George et al., 1980].
11. Yale sparse matrix package – Prof S.C. Eisenstat, Department of Computer Science, Yale University, New Haven CT 06520.

Книги и статьи

Abramowitz M., Stegun I.A. [1965]. Handbook of Mathematical Functions. Dover. New York. [Имеется перевод: Абрамовиц М., Стиган И. Справочник по специальным функциям.- М.: Наука, 1979.]

Aho A., Hopcroft J., Ullman J. [1974]. The Design and Analysis of Computer Algorithms, Addison Wesley, Reading, Massachusetts. [Имеется перевод: Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов.– М.: Мир, 1979.]

Aiken R. (editor) [1985]. Stiff Computation, Oxford University Press, Oxford, England.

Anderson J. [1984]. Fundamentals of Aerodynamics, McGraw-Hill, New York

Andrews H.C., Patterson C.L. [1975]. Outer Product Expansions and Their Uses in Digital Image Processing, Amer. Math. Monthly 82, pp. 1–13.

Backus J. [1979]. The History of Fortran I, II, and III, Annals of the History of Computing 1, pp. 21–37.

Barnhill R., Riesenfeld A. [1974]. Computer Aided Geometric Design, Academic Press, New York.

Barnsley M., Sloan A. [1988]. A Better Way to Compress Images, Byte 13, pp. 215–223.

Bartels R., Beatty J., Barsky B. [1987]. An Introduction to Splines for Use in Computer Graphics and Geometric Modeling, Morgan Kaufmann, Los Altos, California.

Bell E.T. [1975]. Men of Mathematics, Simon and Schuster, New York

Bell G., Glasstone S. [1970]. Nuclear Reactor Theory, Van Nostrand Reinhold, New York.

Belsley D.A., Kuh E., Welsch R. [1981]. Regression Diagnostics: Identifying Influential Data and Sources of Collinearity, Wiley, New York.

Bloomfield P. [1976]. Fourier Analysis of Time Series: An Introduction, Wiley-Interscience, New York.

Boggs P., Byrd R.H., Schnabel R.B. [1985]. Numerical Optimization 1984, SIAM, Philadelphia, Pennsylvania.

Boisvert R.F., Howe S.E., Kahaner D.K. [1984] The Guide to Available Mathematical Software, National Technical Information Service Report PB 84-171305, Springfield, Virginia.

de Boor C. [1978]. A Practical Guide to Splines, Springer, Berlin. [Имеется перевод: Де Бор К. Практическое руководство по сплайнам. М.: Радио и связь, 1985.]

Bratley P., Fox B., Schrage L. [1987]. A Guide to Simulation, Springer, Berlin

Brent R.P. [1973]. Algorithms for Minimization Without Derivatives, Prentice-Hall, Englewood Cliffs, NJ.

Brigham O. [1974]. The Fast Fourier Transform, Prentice-Hall, Englewood Cliffs, New Jersey.

Buzbee B.L. [1984]. The SLATEC Common Mathematical Library, in Sources and Development of Mathematical Software, W.R. Cowell (editor), Prentice-Hall, Englewood Cliffs, New Jersey.

Byrne G., Hindvarsh A. [1987]. Stiff ODE Solvers: A Review of Current and Coming Attractions, J. Computational Physics 70, pp. 1–62.

Carter L.L., Cashwell E.D. [1975]. Particle-Transport with the Monte Carlo Method, ERDA Critical Review Series, TID-26607, National Technical Information Service, Springfield, Virginia.

Chatterjee S., Price B. [1977]. Regression Analysis by Example, Wiley, New York.

Cline A. [1974]. Scalar-and-Planar-Valued Curve Fitting Using Splines Under Tension, Communications of ACM 17, pp. 218–220.
Concus P., Golub G. H., O'Leary D. P. [1976]. A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose (editors), Academic Press, New York.
Dantzig G. B. [1963]. Linear Programming and Extensions, Princeton University Press, Princeton, NJ. [Имеется перевод: Данциг Дж. Б. Линейное программирование, его применения и обобщения. М.: Прогресс, 1966.]
Davis P. [1963]. Interpolation and Approximation, Blaisdell, New York.
Davis P., Rabinowitz P. [1984]. Methods of Numerical Integration, 2nd Ed., Academic Press, New York.
Dekker T. J. [1969]. Finding a zero by means of successive linear interpolation, in Constructive aspects of the fundamental theorem of algebra, D. Dejon and P. Henrici (editors), Wiley-Interscience, New York.
Dennis J. E., Jr., More J. [1974]. Quasi-Newton methods, motivation and theory, SIAM Review 19, pp. 46–89.
Dennis J. E., Jr., Schnabel R. B. [1983]. Numerical Methods for Unconstrained Optimization and Nonlinear Equations, Prentice-Hall, Englewood Cliffs, NJ. [Имеется перевод: Деннис Дж., Шнабель Р. Численные методы безусловной оптимизации и решения нелинейных уравнений.– М.: Мир, 1988.]
de Doncker E., Robinson I. [1984]. An Algorithm for Automatic Integration Over a Triangle Using Nonlinear Extrapolation, ACM Transactions on Mathematical Software 10, pp. 1–16.
Dongarra J. J., Moler C. B., Bunch J. R., Stewart G. W. [1979]. LINPACK Users' Guide, SIAM, Philadelphia.
Draper N. R., Smith H. [1981]. Applied Regression Analysis, Wiley, New York, 1981 [Имеется перевод: Дрейпер Н., Смит Г. Прикладной регрессионный анализ. Кн. 1, 2.– М.: Финансы и статистика, 1986.]
Dyn N., Gregory J., Levin D. [1987]. A 4-Point Interpolatory Subdivision Scheme For Curve Design, The Weizmann Institute of Science, Department of Applied Mathematics, Rehovot, Israel 76100.
Forsythe G., Malcolm M., Moler C. [1977]. Computer Methods for Mathematical Computations, Prentice-Hall, Englewood Cliffs, New Jersey. [Имеется перевод: Форсайт Дж., Малькольм М., Моулер К. Машинные методы математических вычислений.– М.: Мир, 1980.]
Forsythe G. E., Moler C. B. [1967]. Computer Solution of Linear Algebraic System, Prentice-Hall, Englewood Cliffs, NJ. [Имеется перевод: Форсайт Дж., Моулер К. Численное решение систем линейных алгебраических уравнений.– М.: Мир, 1969.]
Fournier A., Fussell D., Carpenter L. [1982]. Computer Rendering of Stochastic Models, Comm of ACM 25, pp. 371–384.
Fox P. A., Hall A. D., Schryer N. L. [1978]. Framework for a portable library, ACM Transaction on Mathematical Software 4, pp. 177–188.
Fritsch F. N., Carlson R. E. [1980]. Monotone Piecewise Cubic Interpolation, SIAM J. Num. Anal. 17, pp. 238–246.
Gaffney P. [1987]. When Things Go Wrong ..., IBM Bergen Scientific Centre Report BSC87/1.
Gardner M. [1961]. Mathematical Puzzles and Diversions, Simon and Schuster, New York. [Имеется перевод: Гарднер М. Математические головоломки и развлечения.– М.: Мир, 1971.]
George A., Liu J. W. [1981]. Computer Solution of Large Sparse Positive Definite Systems, Prentice-Hall, Englewood Cliffs, NJ. [Имеется перевод: Джордж А., Лю Дж. Численное решение больших разреженных систем уравнений.– М.: Мир, 1984.]
George A., Liu J. W., Ng E. [1980]. User Guide for SPARSPACK: Waterloo Sparse Linear Equations Package, Report CS-78-30 (revised 1980), Computer Science Dept.,

University of Waterloo, Waterloo, Canada.
Gill P. E., Golub G. H., Murray W., Saunders M. A. [1974]. Methods for modifying matrix factorizations, Mathematics of Computation 28, pp. 505–535.
Gill P. E., Murray W. [1974]. Newton-type methods for unconstrained and linearly constrained optimization, Mathematical Programming 28, pp. 311–350.
Gill P. E., Murray W., Wright M. H. [1981]. Practical Optimization, Academic Press, New York. [Имеется перевод: Гилл Ф., Мюррей У., Райт М. Практическая оптимизация. – М.: Мир, 1985.]
Golub G. H., Van Loan C. F. [1983]. Matrix Computations, The Johns Hopkins University Press, Baltimore. [Имеется перевод: Голуб Дж., Ван Лоан Ч. Матричные вычисления. М.: Мир, 1993.]
Haber S. [1970]. Numerical Evaluation of Multiple Integrals, SIAM Review 12, pp. 481–526.
Haberman R. [1977]. Mathematical Models, Prentice-Hall, Englewood Cliffs, NJ.
Hamming R. [1962]. Numerical Methods for Scientists and Engineers. McGraw-Hill, New York. [Имеется перевод: Хемминг Р. В. Численные методы для научных работников и инженеров. – М.: Наука, 1972.]
Hansen E. R. [1969]. Topics in Interval Analysis, Oxford University Press. London.
Hindmarsh A. [1980]. LSODE and LSODEL, Two Initial Value Ordinary Differential Equation Solvers, ACM SIGNUM Newsletter 15, pp. 10–11.
IEEE Standard for Binary Floating Point Arithmetic, ANSI/IEEE Standard 754–1985, Inst. Electrical and Electronics Engineers, Inc., New York, 1985.
Jenkins M. A., Traub J. F. [1970]. A three-stage algorithm for real polynomials using quadratic iteration, SIAM Journal on Numerical Analysis 7, pp. 545–566.
Jones R. E., Kahaner D. K. [1983]. XERROR, the SLATEC Error-handling Package, Software – Practice and Experience 13, pp. 251–257.
Joyce D. [1971]. Survey of Extrapolation Processes in Numerical Analysis, SIAM Rev. 13, pp. 435–490.
Kahaner D. [1970]. Matrix Description of the Fast Fourier Transform, IEEE Transactions on Audio and Electroacoustics AU-18, pp. 442–450.
Kahaner D. [1978]. The Fast Fourier Transform by Polynomial Evaluation, ZAMP 29, pp. 387–394.
Kennedy W. J., Gentle J. E. [1980]. Statistical Computing, Marcel Dekker, New York.
Kincaid D., Respess G., Young D., Grimes R. [1982]. Itpack 2C: A Fortran package for solving large sparse linear systems by adaptive accelerated iterative methods, ACM Transactions on Mathematical Software 8, pp. 302–322.
Knuth D. E. [1969]. The Art of Computer Programming: Volume 2, Seminumerical Algorithms, Addison-Wesley, Reading, MA. [Имеется перевод: Кнут Д. Е. Искусство программирования. Том 2. Получисленные алгоритмы. – М.: Мир, 1977.]
Krogh F. [1970]. VODQ/SVDQ/DVDQ, Variable Order Integrators for the Numerical Solution of Ordinary Differential Equations, Tech Brief NPO-11643, Jet Propulsion Laboratory, California Institute of Technology, Pasadena California.
Lambert J. [1973]. Computational Methods in Ordinary Differential Equations, Wiley, New York.
Lanczos C. [1966]. Discourse on Fourier Series, Oliver and Boyd, Edinburgh.
Lapidus L., Seinfeld J. [1971]. Numerical solution of Ordinary Differential Equations, Academic Press, New York.
Laurie D. [1978]. Automatic Numerical Integration Over a Triangle, Technical Report, National Research Center for Mathematical Sciences of the CSIR, P.O. Box 395, Pretoria, South Africa.
Lawson C. L., Hanson R., Kincaid D., Krogh F. [1979]. Basic Linear Algebra Subprograms for Fortran Usage, ACM Trans Math Software 5, pp. 308–323.
Lawson C. L., Hanson R. J. [1974]. Solving Least Squares Problems, Prentice-Hall, Englewood Cliffs, NJ. [Имеется перевод: Лоусон Ч., Хенсон Р. Численное решение задач метода наименьших квадратов. – М.: Наука, 1986.]
Lehmer D. [1951]. Mathematical Models in Large-scale Computing Units, in

Proceedings of the Second Symposium on Large Scale Digital Computing Machinery, Harvard University Press, Cambridge, Massachusetts, pp. 141–146.
Luenberger D.G. [1984]. Introduction to Linear and Nonlinear Programming, Addison-Wesley, Reading, Massachusetts.
Lyness J. [1983]. AUG2 – Integration Over a Triangle, Argonne National Laboratory Report, ANL/MCS-TM-13.
Mandelbrot B. [1982]. The Fractal Geometry of Nature, Freeman, New York.
Marsaglia G. [1968]. Random numbers fall mainly in the planes, Proc. Nat. Acad. Sci. 61, pp. 25–28.
Marsaglia G. [1984]. A Fast, Easily Implemented Method for Sampling from Decreasing of Symmetric Unimodal Density Functions, SIAM, J. Sci. Stat. Comp. 5, pp. 349–359.
Moore R.E. [1979]. Methods and Applications of Interval Analysis, SIAM, Philadelphia.
More J.J., Garbow B.S., Hillstrom K.E. [1980]. User Guide for MINPACK-1, Report ANL-80-74, Argonne National Laboratory, Argonne, Illinois.
Moler C.B., Solomon L.P. [1970]. Use of Splines and Numerical Integration in Geometrical Acoustics, J. Acoustical Soc. Amer. 48, pp. 739–744.
Moses J. [1971]. Symbolic Integration: The Stormy Decade, Communications of ACM 14, pp. 548–560.
Murota K., Iri M. [1982]. Parameter Tuning and Repeated Application of the IMT Type Transformation in Numerical Quadrature, Numer. Math. 38, pp. 347–363.
Murtagh B.A., Saunders M. [1978]. Large-scale linearly constrained optimization, Mathematical Programming 14, pp. 41–72.
Orchard-Hays W. [1968]. Advanced Linear-programming Computing Techniques, McGraw-Hill, New York.
Ortega J., Poole W. [1981]. An Introduction to Numerical Methods for Differential Equations, Pitman. Marshfield, MA. [Имеется перевод: Ортега Дж., Пул У. Введение в численные методы решения дифференциальных уравнений.– М.: Наука, 1986.]
Ortega J.M., Rheinboldt W.C. [1970]. Iterative Solution of Nonlinear Equations in Several Variables. Academic Press, New York. [Имеется перевод: Ортега Дж., Рейнболдт В. Итерационные методы решения нелинейных систем уравнений со многими неизвестными.– М.: Мир, 1975.]
Osborne M.R. [1966]. On Nordsiek's Method for the Numerical Solution of Ordinary Differential Equations, BIT 6, pp. 51–57.
Peitgen H., Richter P. [1986]. The Beauty of Fractals, Springer, Berlin. [Имеется перевод: Пайтген Х.-О., Рихтер П.-Х. Красота фракталов: образы комплексных динамических систем. М.: Мир, 1994.]
Piessens R., deDoncker-Kapenga E., Uberhuber C., Kahaner D. [1983]. QUADPACK: A Subroutine Package for Automatic Integration, Springer, Berlin.
Prenter P. [1975]. Splines and Variational Methods, Wiley, New York.
Pritsker A. [1986]. Introduction to Simulation and Slam II, Wiley, New York.
Ralston A. [1965]. A First Course in Numerical Analysis, McGraw-Hill, New York.
Rasmusson E. [1985]. *El Niño* and Variations in Climate, Americal Scientist 73, pp. 168–177.
Rosenfeld A., Kak A. [1982]. Digital Picture Processing, Academic Press, New York.
Ross S. [1983]. Stochastic Processes, Wiley, New York.
Roy M.R. [1985]. A History of Computing Technology, Prentice-Hall, Englewood Cliffs, NJ.
Rubinstein R. [1981]. Simulation and the Monte Carlo Method. John Wiley, New York.
Russell E. [1983]. Building Simulation Models with Simscript II.5, Consolidated Analysis Centers Inc., New York.
Rust B.W., Rotty R.M., Maryland G. [1979]. Inferences Drawn From Atmospheric CO_2 Data, J. Geophysical Research 84, pp. 3115–3122.
Schnabel R.B., Koontz J.E., Weiss B.E. [1982]. A modular system of algorithms for unconstrained minimization, Report CU-CS-240-82, Computer Science Department,

University of Colorado, Boulder, Colorado.
Schriber T. [1974]. Simulation Using GPSS, Wiley, New York.
Schultz M. [1973]. Spline Analysis, Prentice-Hall, Englewood Cliffs, New Jersey.
Shampine L., Baca C. [1984]. Error Estimators for Stiff Differential Equations, J. Computational and Applied Mathematics 11, p. 197–208.
Shampine L., Gordon M. [1975]. Computer Solution of Ordinary Differential Equations: the Initial Value Problem, Freeman, San Francisco.
Skeel R.D. [1979]. Equivalent Forms of Multistep Formulas, Math. Comp. 33, pp. 1229–1250.
Smith B.T., Boyle J.M., Garbow B.S., Ikebe Y., Klema V.C., Moler C.B. [1974]. Matrix Eigensystem Routines – EISPACK Guide, Springer, Berlin.
Sterbenz P. [1974]. Floating Point Computation, Prentice-Hall, Englewood Cliffs, New Jersey.
Stroud A. [1972]. Approximate Calculation of Multiple Integrals, Prentice-Hall, Englewood Cliffs, New Jersey.
Swarztrauber P. [1975]. Efficient Subprograms for the Solution of Elliptic Partial Differential Equations, Report TN/LA-109, National Center for Atmospheric Research, Boulder, Colorado.
Timoshenko S. [1956]. Strength of Materials, Part II, Van Nostrand, Princeton, NJ.
Varga R.S. [1962]. Matrix Iterative Analysis, Prentice-Hall, Englewood Cliffs, NJ.
Watson L.T. [1986]. Numerical linear algebra aspects of globally convergent homotory methods, SIAM Review 28, pp. 529–545.
Watson L.T., Billups S.C., Morgan A.P. [1987]. HOMPACK: A suite of codes for globally convergent homotopy algorithms, ACM Transactions on Mathematical Software 13, pp. 281–310.
Weaver H.J. [1983]. Applications of Discrete and Continuous Fourier Analysis, Wiley-Interscience, New York.
Wendroff B. [1966]. Theoretical Numerical Analysis, Academic Press, New York.
Whittaker E., Robinson G. [1924]. The Calculus of Observations, Blackie and Son, London. [Имеется перевод: Уиттекер Э.Т. Математическая обработка результатов наблюдений.– Л.–М.: Гостехиздат, 1933.]
Wilkinson J.H. [1963]. Rounding Errors in Algebraic Processes, Prentice-Hall, Englewood Cliffs, NJ.
Wilkinson J.H. [1967]. Two algorithms based on successive linear interpolation, Tech. Report STAN-CS-67-60, Computer Science Department, Stanford University, Stanford, California.

УКАЗАТЕЛЬ

Оглавление