

В.И. Струченков



МЕТОДЫ ОПТИМИЗАЦИИ

- *Линейное программирование*
- *Нелинейное программирование*
- *Динамическое программирование*

ЭКЗАМЕН



В.И. Струченков

МЕТОДЫ ОПТИМИЗАЦИИ

**ОСНОВЫ ТЕОРИИ, ЗАДАЧИ,
ОБУЧАЮЩИЕ
КОМПЬЮТЕРНЫЕ ПРОГРАММЫ**

Рекомендовано в качестве учебного пособия

**Издательство
«ЭКЗАМЕН»**

**МОСКВА
2005**

УДК 004(075.8)

ББК 302.973я73

С87

Автор — Валерий Иванович Струченков, д.т.н., профессор МИРЭА (ф-т кибернетики), специалист в области применения методов оптимизации в сложных системах управления и проектирования.

Струченков В.И.

С87 Методы оптимизации. Основы теории, задачи, обучающие компьютерные программы: Учебное пособие / В.И. Струченков. — М.: Издательство «Экзамен», 2005. — 256 с. (Серия «Учебное пособие для вузов»)

ISBN 5-472-00465-9

Цель настоящей работы — содействовать изучению и практическому применению современных методов решения задач оптимизации в различных областях практики. Изложены теоретические основы методов линейного, нелинейного и динамического программирования, приведены необходимые сведения о специально разработанных четырех обучающих компьютерных программах, а также конкретные примеры практического применения методов оптимизации.

Каждый раздел заканчивается контрольными вопросами и задачами; на большинство из них в приложениях даны ответы и решения. Детально разобраны примеры практических задач из различных областей, каждую из которых можно решать различными методами, и дан сопоставительный анализ этих вариантов.

Для студентов и аспирантов технических вузов, изучающих методы оптимизации, а также специалистов, сталкивающихся с проблемами выработки оптимальных решений в различных областях деятельности.

УДК 004(075.8)

ББК 302.973я73

Подписано в печать с диапозитивов 01.10.2004.

Формат 84х108/32. Гарнитура «Таймс». Бумага типографская.

Уч.-изд. л. 8,31. Усл. печ. л. 13,45. Тираж 3000 экз. Заказ № 2684.

ISBN 5-472-00465-9

© Струченков В.И., 2005

© Издательство «ЭКЗАМЕН», 2005

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	6
1. ЛИНЕЙНОЕ ПРОГРАММИРОВАНИЕ	12
1.1. Основные понятия и обозначения.....	12
1.2. Формулировка задачи линейного программирования	17
1.3. Необходимые сведения из линейной алгебры и математического анализа	19
1.4. Обучающая программа DANTZIG_1.exe	33
1.5. Основные формы записи задачи линейного программирования	36
1.6. Симплекс-метод.....	38
1.7. Двойственность в линейном программировании	50
1.8. Целочисленное линейное программирование	52
1.9. Практическое применение линейного программирования	54
1.10. Обучающая программа DANTZIG_2.exe	58
2. НЕЛИНЕЙНОЕ ПРОГРАММИРОВАНИЕ.....	64
2.1. Формулировка задач нелинейного программирования и их классификация	64
2.1. Дополнительные сведения из линейной алгебры и математического анализа	66
2.3. Методы безусловной оптимизации.....	79
2.3.1. Градиентные методы.....	83
2.3.2. Метод параллельных касательных	90
2.3.3. Метод сопряженных градиентов.....	91
2.3.4. Метод покоординатного спуска.....	94
2.3.5. О методах второго порядка	99
2.3.6. О методах прямого поиска	101
2.3.7. Методы одномерной минимизации	103

2.4. Задачи с линейными ограничениями	109
2.4.1. Задачи с ограничениями-равенствами.....	110
2.4.2. Задачи с ограничениями-неравенствами.....	111
2.4.2.1. Метод проекции градиента.....	113
2.4.2.2. Метод приведенного градиента	125
2.5. Задачи с нелинейными ограничениями	129
2.5.1. Методы штрафных функций	130
2.5.2. Методы барьерных функций	131
2.6. Построение начального приближения.....	135
2.7. Практическая реализация методов нелинейного программирования	136
2.8. Примеры задач, решаемых с применением методов нелинейного программирования	142
2.9. Обучающая компьютерная программа ROZEN.exe.....	155
3. ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ ...	160
3.1. Многоэтапные процессы принятия решений.....	160
3.2. Принцип оптимальности и уравнение Р. Беллмана	170
3.3. Область применения динамического программирования	177
3.4. Примеры задач, решаемых с помощью динамического программирования	183
3.5. Обучающая программа BELLMAN.exe.....	218
ЗАКЛЮЧЕНИЕ	226
ПРИЛОЖЕНИЯ.....	229
БИБЛИОГРАФИЧЕСКИЙ СПИСОК.....	254

В книге рассматриваются теоретические основы линейного, нелинейного и динамического программирования. По каждому из трех разделов приводятся контрольные вопросы и задачи, на большинство из них в приложениях даны ответы и решения. Разбираются также практические задачи из различных областей, решенные методами линейного, нелинейного и динамического программирования. Книга содержит сведения о специально разработанных обучающих компьютерных программах и рекомендации по их применению.

Используемый математический аппарат сведен к минимуму и поясняется в тексте, что обеспечивает понимание методов оптимизации при наличии математической подготовки в объеме программы обычного технического вуза.

В основу книги положены курс лекций автора в Московском институте радиотехники, электроники и автоматики (МИРЭА) и практический опыт разработки программных средств для решения задач оптимизации большой размерности в рамках САПР.

Книга может быть полезна студентам и аспирантам, изучающим методы оптимизации, а также специалистам, сталкивающимся с проблемами выработки оптимальных решений в различных областях деятельности.

Обучающие программы можно заказать по электронной почте (str1942@mail.ru) или по телефону (095) 930-19-44.

© В. И. Струченков, 2003

ВВЕДЕНИЕ

Математические методы оптимизации, соответствующие алгоритмы и компьютерные программы можно рассматривать как эффективный элемент наукоемких технологий, разработка которых в различных областях в настоящее время особенно актуальна. Оптимизационные методы используются в исследовании операций и системном анализе, в планировании производственной деятельности, в проектировании различных объектов, в военном деле и т.д.

Недостаточность классической теории оптимизации выявилась во второй трети XX в. при необходимости решать задачи с ограничениями в виде неравенств, особенно при большом числе переменных. Последнее обстоятельство (большая размерность задач) преодолено разработкой численных методов решения систем алгебраических уравнений и соответствующих компьютерных программ.

Задачи с ограничениями в виде неравенств, в которых равенство нулю частных производных вообще не является даже необходимым условием экстремума, потребовали разработки новой теории оптимизации — теории математического программирования. Эта теория дает совокупность методов решения задач поиска экстремума функции многих переменных (целевая функция) при наличии ограничений (равенств или неравенств) на искомые неизвестные. Как правило, это численные (итерационные) методы. Их практическая реализация осуществляется в соответствующих алгоритмах и компьютерных программах.

Классические задачи на безусловный экстремум (при отсутствии ограничений вообще) или при наличии только ограничений-равенств также могут решаться методами математического программирования (как частный случай). Отсюда следует теоретическая и практическая значимость этих методов.

Наиболее известными и простыми являются методы *линейного программирования*, используемые в научных исследованиях и практической деятельности значительно чаще, чем методы *нелинейного программирования* [9, 11, 12]. Но нелинейное программирование — это бурно развивающийся раздел современной теории оптимизации, созданный практически в последние 30–40 лет. Сравнительно редкое практическое применение методов нелинейного программирования объясняется именно этим обстоятельством (а не отсутствием реальных практических задач), так как необходимо значительное время для освоения новых теорий широким кругом практиков и реализации новых методов в конкретных алгоритмах и компьютерных программах. Более того, реализация методов нелинейного программирования для решения задач большой размерности требует мощных компьютеров, которые получили широкое распространение в нашей стране только в последние 10–15 лет.

Почти одновременно с линейным программированием (конец 40-х – начало 50-х годов прошлого века) был разработан метод динамического программирования. В «компьютерную эпоху» он получил широкое применение в самых различных областях практики. Этот мощный метод оптимизации далеко не универсальный. Известны безуспешные попытки его применения без должного анализа особенностей конкретной задачи оптимизации.

Автор не ставил своей целью изложение всех или хотя бы большинства математических методов оптимизации. Его

цель — помочь в освоении основных идей линейного, нелинейного и динамического программирования, на конкретных практических примерах из различных областей продемонстрировать эти идеи, соответствующие методы и алгоритмы, особенности их реализации в компьютерных программах.

Этой же цели служат и многочисленные контрольные вопросы и задачи по каждому разделу. Ответы и решения приведены в приложениях.

В большинстве источников методы нелинейного программирования излагаются с опорой на глубокие знания линейной алгебры и функционального анализа и доступны в основном специалистам-математикам. В то же время практическое применение современных методов оптимизации в значительной мере сдерживается отсутствием соответствующих знаний у инженеров, специалистов в прикладных областях.

Данная работа — это попытка преодолеть образовавшийся разрыв изложением основных идей, методов и алгоритмов с привлечением ограниченного числа математических понятий, которые также поясняются в тексте. Поэтому материал должен быть доступен при наличии математической подготовки в пределах программы технических вузов.

Вместо строгих доказательств известных в математике теорем, обосновывающих методы оптимизации, в книге изложены основные идеи каждого метода, соответствующие алгоритмы и даны расчетные и графические иллюстрации для задач малой размерности.

Без специальной математической подготовки попытка освоения и практического использования методов оптимизации имеет мало шансов на успех при опоре только на логико-математическое восприятие и абстрактное мышление. Поэтому и были разработаны специальные компьютерные

обучающие программы, дающие возможность образного восприятия основных идей и методов оптимизации и самостоятельной проверки своих знаний.

Программа DANTZIG_1.exe позволяет решать произвольную систему линейных неравенств с двумя неизвестными и получать графическое изображение не только окончательного решения, т.е. допустимой области, но и видеть процесс формирования этой области при вводе очередного ограничения. Поскольку с системой линейных неравенств приходится иметь дело не только в линейном программировании, но и при оптимизации нелинейных функций, образное восприятие различных ситуаций, возникающих при наличии различных линейных ограничений, имеет особое значение для понимания сущности широкого круга оптимизационных задач и методов их решения.

Программа DANTZIG_2.exe, полностью посвященная линейному программированию, демонстрирует различные формы записи задачи, излагает основные идеи и демонстрирует алгоритм симплекс-метода на конкретной задаче, задает большое число тестовых вопросов по линейному программированию, что позволяет пользователю оценить свои знания в данной области.

Программа ROZEN.exe, посвященная нелинейному программированию, излагает теоретические положения и иллюстрирует их графически различными примерами. Для контроля понимания теоретических основ пользователю предлагаются различные тестовые вопросы.

Главная особенность этой программы в том, что в ней реализованы алгоритмы методов наискорейшего спуска, покоординатного спуска и сопряженных градиентов для решения задачи минимизации квадратичной функции многих переменных. В результате пользователь при различных исходных данных может увидеть влияние на

процесс поиска минимума как размерности задачи, так и особенностей структуры целевой функции; можно проследить, как падает эффективность метода наискорейшего спуска при наличии сильной овражности. Есть возможность сравнить эффективность реализованных методов оптимизации при одних и тех же данных. Программа демонстрирует не только результат, но и процесс поиска минимума каждым из методов.

При изложении основных идей динамического программирования основное внимание уделяется принципу оптимальности Р. Беллмана и области его применимости. Рассмотрены различные виды целевых функций и систем ограничений, при наличии которых имеются особенности в применении динамического программирования. Для некоторых известных задач, решаемых с помощью динамического программирования, приведены новые, более эффективные алгоритмы. Критически оцениваются опыт применения динамического программирования в различных областях практики и некоторые устоявшиеся суждения на этот счет.

В обучающей программе BELLMAN.exe для демонстрации основных идей динамического программирования выбрана задача поиска кратчайшего пути на двумерной сетке. Программно реализованы два алгоритма: полный перебор всех возможных путей и динамическое программирование. Это позволяет реально на конкретных данных оценить эффективность алгоритма динамического программирования и увидеть, как растут вычислительные трудности решения задачи при росте размеров сетки, т.е. реально «почувствовать проклятие размерности».

Программа разъясняет принцип оптимальности Р. Беллмана и на конкретных примерах и тестовых вопросах показывает возможности метода и отсутствие его универсальности, зависимость его применимости от формализации понятия «состояние системы».

В основе этой книги лежат курс лекций автора в Московском институте радиотехники, электроники и автоматики (МИРЭА) и практический опыт разработки программных средств для решения задач большой размерности в рамках САПР.

В разработке компьютерных обучающих программ участвовали студенты МИРЭА А. Иванов, С. Устинов, С. Виноградов, Д. Однолько, К. Сараев. Автор выражает им свою благодарность.

1. ЛИНЕЙНОЕ ПРОГРАММИРОВАНИЕ

В данном разделе рассмотрены линейные задачи оптимизации, иначе говоря, задачи поиска минимума (или максимума) *линейной функции* многих переменных при наличии ограничений в виде *линейных* равенств и/или неравенств. Изложены необходимые математические положения, на основе которых исследована структура области допустимых решений, наличие или отсутствие решений, единственность решения. Рассмотрены конкретные примеры и задачи и методы их решения.

Приведены сведения об обучающих программах DANTZIG_1.exe и DANTZIG_2.exe и рекомендации по их применению. Проанализированы некоторые практические задачи, решаемые методами линейного программирования.

1.1. Основные понятия и обозначения

Набор неизвестных x_i ($i=1,2,\dots,n$) рассматривается как вектор (точка) *n -мерного арифметического евклидова пространства* (E_n), а сами неизвестные — это координаты вектора. Формально вектор может быть определен как упорядоченный набор чисел, причем и сами числа и их порядок в этом наборе существенны.

Евклидово пространство — это частный случай линейного пространства, которое в свою очередь есть множество элементов, на котором определены операции сложения и умножения на число (скаляр). Суммирование элементов евклидова пространства (векторов) и умножение их на чис-

ло сводится к суммированию и умножению на число соответствующих координат.

Векторы изображаются вертикальными столбцами чисел с первой компонентой в верхней позиции, т.е.:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix}$$

Иногда удобно использовать представление вектора как *строки*, т.е. горизонтального списка чисел $y = (y_1, y_2, \dots, y_n)$.

Переход от вектора-столбца \mathbf{x} к вектору-строке с теми же элементами обозначается $\mathbf{x}^T = (x_1, x_2, \dots, x_n)$. Аналогично осуществляется переход от вектора-строки к вектору-столбцу:

$$\mathbf{y}^T = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix}$$

Такая операция называется *транспонированием*.

Принадлежность вектора \mathbf{x} некоторому множеству M обозначается как:

$$\mathbf{x} \in M$$

Наш интерес к векторам объясняется тем, что задачи оптимизации часто сводятся к поиску нескольких (обычно большого числа) параметров (характеристик), которые определяют искомое оптимальное решение. Удобно объединить все неизвестные в один набор, т.е. считать, что неизвестным является вектор, удовлетворяющий некоторым условиям. Далее векто-

ры будем обозначать строчными буквами жирным шрифтом. Верхний индекс будет указывать номер вектора в некоторой совокупности векторов, а нижний индекс — указывать номер компоненты (координаты) вектора. Скаляры будем обозначать строчными буквам обычным шрифтом. Нижний индекс будет указывать номер скаляра в некоторой их совокупности. Координаты вектора — пример такой совокупности.

В евклидовом пространстве векторы можно умножать на числа и складывать, т.е. составлять *линейные комбинации векторов*. Линейная комбинация векторов $\mathbf{b}^1, \mathbf{b}^2, \dots, \mathbf{b}^r$ — это тоже вектор. Если его обозначить через \mathbf{p} , можно записать:

$$\mathbf{p} = k_1 \mathbf{b}^1 + k_2 \mathbf{b}^2 + \dots + k_r \mathbf{b}^r,$$

где k_1, k_2, \dots, k_r — некоторые числа, называемые коэффициентами линейной комбинации.

Прямоугольная таблица вещественных чисел называется *матрицей*. Сами числа называются элементами матрицы. Матрицы будем обозначать заглавными буквами и выделять жирным шрифтом. Элементы матриц будут обозначаться строчными буквами обычным шрифтом, имеющими двойной индекс. Так что b_{ij} — это элемент i -ой строки и j -ого столбца матрицы \mathbf{B} .

$$\mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \dots & \dots & \dots & \dots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix}$$

Строки матриц будем обозначать строчными буквами с нижними индексами и выделять жирным шрифтом.

Матрицы можно умножать на векторы, если число *столбцов* матрицы равно числу компонент вектора. Вектор можно рассматривать как матрицу, имеющую один столбец.

Результат умножения матрицы на вектор — это новый вектор, число координат которого равно числу *строк* матрицы. Этот вектор — линейная комбинация столбцов матрицы с коэффициентами, равными координатам исходного вектора, так что

$$\mathbf{B}\mathbf{x} = x_1\mathbf{b}^1 + x_2\mathbf{b}^2 + \dots + x_n\mathbf{b}^n. \quad (1.1)$$

Здесь \mathbf{b}^k ($k = 1, 2, \dots, n$) — k -й столбец матрицы \mathbf{B} .

Пример.

$$\mathbf{B} = \begin{vmatrix} 1 & 0 & 4 & 0 & 0 \\ 0 & 1 & 0 & 2 & 0 \end{vmatrix}$$

Умножение матрицы \mathbf{B} на произвольный вектор $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5)^T$ дает линейную комбинацию

$$\mathbf{B}\mathbf{x} = x_1 \begin{vmatrix} 1 \\ 0 \end{vmatrix} + x_2 \begin{vmatrix} 0 \\ 1 \end{vmatrix} + x_3 \begin{vmatrix} 4 \\ 0 \end{vmatrix} + x_4 \begin{vmatrix} 0 \\ 2 \end{vmatrix} + x_5 \begin{vmatrix} 0 \\ 0 \end{vmatrix}.$$

Если строки матрицы заменить ее столбцами, получится матрица, которая называется *транспонированной* по отношению к исходной.

Векторы называются *линейно независимыми*, если их линейная комбинация может быть равна нулю *только* при равенстве нулю всех ее коэффициентов. Максимальное число линейно независимых векторов называется *размерностью* пространства. А *любой* набор линейно независимых векторов, число которых равно размерности пространства, называется его *базисом*.

Сумма произведений соответственных координат двух векторов \mathbf{x} и \mathbf{y} называется их *скалярным произведением* и обозначается (\mathbf{x}, \mathbf{y}) .

$$(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{i=n} x_i y_i$$

Скалярное произведение — это число (скаляр); если оно равно нулю, то векторы называются *ортгональными*. Для этого, в отличие от умножения чисел, не обязательно, чтобы один из векторов имел только нулевые координаты. Если скалярное произведение положительно, векторы образуют острый угол.

Если векторы базиса попарно ортогональны, то это *ортгональный базис*. Базисов, в том числе ортогональных, при размерности пространства $n > 1$ всегда *бесконечно много*.

Отметим два свойства скалярного произведения, которые можно легко получить непосредственно из его определения:

- ◆ коммутативность: $(\mathbf{x}, \mathbf{y}) = (\mathbf{y}, \mathbf{x})$;
- ◆ дистрибутивность: $(\mathbf{x}, (\mathbf{y} + \mathbf{z})) = (\mathbf{x}, \mathbf{y}) + (\mathbf{x}, \mathbf{z})$.

Произведение матрицы на вектор можно понимать не только как линейную комбинацию столбцов матрицы с коэффициентами в виде компонент исходного вектора, но и как вектор, у которого каждая компонента есть произведение соответствующей строки матрицы (вектор-строка) на исходный вектор.

$$(\mathbf{Ax})_i = \mathbf{a}_i \mathbf{x} = (\mathbf{a}_i^T, \mathbf{x})$$

Здесь \mathbf{a}_i — i -ая вектор-строка матрицы \mathbf{A} .

Кроме того, нам потребуется еще тождество $(\mathbf{x}, \mathbf{Ay}) = (\mathbf{A}^T \mathbf{x}, \mathbf{y})$, где \mathbf{A}^T — это транспонированная матрица \mathbf{A} . Доказательство этого тождества приводится в приложении 1 (п. 12).

1.2. Формулировка задачи линейного программирования

В общем случае задачу линейного программирования можно записать в следующем виде.

Найти $\min \sum_{i=1}^{i=n} c_i x_i$ при ограничениях:

$$Ax = b$$

$$Dx \leq d$$

Здесь x — неизвестный вектор с координатами x_i ($i = 1, 2, \dots, n$), n — число неизвестных, c (c_1, c_2, \dots, c_n) — заданный вектор. Целевая функция (c, x) — скалярное произведение заданного вектора c на вектор неизвестных x . Предполагается, что c это ненулевой вектор, иначе задача теряет смысл.

Матрица A имеет не более n строк, а число строк матрицы D произвольно. Число столбцов в каждой из этих матриц равно n . Матрицы A и D и векторы b и d заданы. Возможны и ограничения по знаку $x_i \geq 0$. Ограничения-равенства могут отсутствовать, но *ограничения-неравенства обязательны*, так как при их отсутствии задача особого смысла не имеет. Действительно, если уравнения линейно независимы и их число равно числу неизвестных n , то такая система имеет единственное решение и нет задачи поиска минимума. Если же уравнений меньше, чем неизвестных, то можно из первого уравнения выразить x_1 через x_2, \dots, x_{n-1} и подставить во все остальные уравнения и в целевую функцию, затем аналогичным образом исключить x_2 и т.д. В итоге будет исключено столько неизвестных, сколько уравнений, а целевая функция останется линейной и будет зависеть только от оставшихся неизвестных. На эти оставшиеся неизвестные никаких ограничений нет, поэтому целевая функция неограничена и снизу и сверху, и задача поиска экстремума теряет смысл.

Чтобы в этом убедиться, достаточно положить все оставшиеся неизвестные, кроме одного, равными нулю, а этому неизвестному (коэффициент при нем не должен быть нулевым) придать сколь угодно большие положительные или отрицательные значения.

Пример.

Найти $\min (x_1 + 2x_3 + x_4 - x_5)$ при ограничениях:

$$\begin{cases} x_1 + 4x_3 = 2; \\ x_2 + 2x_4 = 3; \\ 2x_3 - x_5 \leq 1. \end{cases}$$

В данной задаче $n = 5$. Заданы: матрица

$$A = \begin{pmatrix} 1 & 0 & 4 & 0 & 0 \\ 0 & 1 & 0 & 2 & 0 \end{pmatrix},$$

матрица $D = (0, 0, 2, 0, -1)$, вектор $b = (2, 3)^T$, вектор $c = (1, 0, 2, 1, -1)^T$ и вектор d , имеющий только одну координату, равную 1.

Решением задачи (точкой минимума) называется такая точка (вектор), координаты которой удовлетворяют всем ограничениям; и при этом значение целевой функции минимально.

Поскольку задача на максимум сводится к задаче на минимум путем изменения знака целевой функции, приведенная формулировка задачи универсальна.

Система ограничений формирует *область допустимых решений* (ОДР), т.е. множество точек (векторов), координаты которых удовлетворяют *всем* ограничениям. Именно среди них нужно искать точку минимума. Если ограничения противоречивы, то таких точек нет, ОДР есть пустое множество и задача не имеет решений. Если ОДР не пустое множество, то это не означает, что задача имеет решение,

так как возможна ситуация, при которой *в пределах ОДР* целевая функция может неограниченно убывать. Это возможно только в том случае, когда сама ОДР неограничена. Возможна также ситуация, при которой существуют много точек допустимой области, в которых целевая функция (линейная форма) имеет одно и то же минимальное значение. В этом случае имеем много решений задачи.

Если наличие или отсутствие решений полностью определяется ОДР (в случае ее ограниченности), то количество точек минимума зависит и от ОДР, и от целевой функции (вектора c). Чтобы установить, как влияет ОДР на поиск решения задачи и что собой представляет ОДР в рассматриваемом случае линейных неравенств и равенств, нам потребуются некоторые основные понятия из линейной алгебры и математического анализа.

1.3. Необходимые сведения из линейной алгебры и математического анализа

Непустое подмножество M пространства E_n называется *линейным подпространством*, если любой его элемент можно умножать на скаляр, а два любых элемента складывать и при этом получится элемент из M . Отсюда следует: нулевой вектор принадлежит M , так как:

$$0 = x + (-x).$$

На плоскости ($n = 2$) все прямые, проходящие через начало координат, — это подпространства размерности единица, а все остальные прямые не являются подпространствами.

В трехмерном пространстве ($n = 3$) только плоскости и прямые, проходящие через начало координат, — это подпространства, соответственно двумерные и одномерные.

Если ко всем векторам подпространства M прибавить ненулевой фиксированный вектор x^0 , принадлежащий E_n ,

но не принадлежащий его подпространству M , то получим некоторое подмножество элементов из E_n . Но это подмножество уже не будет линейным подпространством, так как оно не содержит нуля.

Такое подмножество называется *линейным многообразием пространства E_n* .

Пример.

Прямая, проходящая через начало координат, является одномерным подпространством M пространства E_2 . Если сместить эту прямую на некоторый вектор x^0 (рис. 1), мы получим линейное многообразие. Заметим, что одно и то же линейное многообразие может быть получено в результате смещения подпространства M на различные векторы.

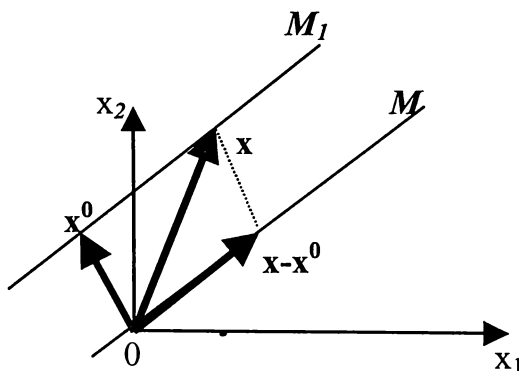


Рис. 1. Подпространство M и линейное многообразие M_1

Рассмотрим теперь, что представляет собой множество точек, удовлетворяющих отдельно взятому ограничению-неравенству.

Каждое неравенство системы ограничений можно рассматривать как условие того, чтобы скалярное произведе-

ние соответствующей строки матрицы ограничений на неизвестный вектор \mathbf{x} было меньше или равно заданному числу. Если система ограничений имеет вид $\mathbf{Ax} \leq \mathbf{b}$, то неравенство с номером i записывается как

$$(\mathbf{a}_i^T, \mathbf{x}) \leq b_i,$$

где \mathbf{a}_i это i -ая вектор-строка, а b_i — это правая часть неравенства (число). Уравнение $(\mathbf{a}_i^T, \mathbf{x}) = 0$ определяет подпространство \mathbf{M} в \mathbf{E}_n размерностью $n - 1$. Такое подпространство называется *гиперплоскостью*, т.е. это множество векторов, ортогональных вектору \mathbf{a}_i^T , который является *нормалью к подпространству*. Множество точек (векторов), удовлетворяющих условию $(\mathbf{a}_i^T, \mathbf{x}) = b_i$ ($b_i \neq 0$), — это линейное многообразие \mathbf{M}_i , т.е. подпространство \mathbf{M} , смещенное на некоторый вектор \mathbf{x}^0 , который при умножении на \mathbf{a}_i^T дает b_i . Действительно,

$$b_i = (\mathbf{a}_i^T, \mathbf{x}) = (\mathbf{a}_i^T, (\mathbf{x}^0 + (\mathbf{x} - \mathbf{x}^0))) = (\mathbf{a}_i^T, \mathbf{x}^0) + (\mathbf{a}_i^T, (\mathbf{x} - \mathbf{x}^0))$$

И если $(\mathbf{a}_i^T, \mathbf{x}^0) = b_i$, то $(\mathbf{a}_i^T, (\mathbf{x} - \mathbf{x}^0)) = 0$, т.е. вектор $\mathbf{x} - \mathbf{x}^0$ принадлежит гиперплоскости \mathbf{M} (рис. 1). Если вектор $(\mathbf{x} - \mathbf{x}^0)$ обозначить через \mathbf{y} , то:

$$\mathbf{x} = \mathbf{y} + \mathbf{x}^0,$$

Вектор \mathbf{y} принадлежит \mathbf{M} , а вектор \mathbf{x}^0 — нет.

Итак, мы установили, что каждое ограничение-равенство $(\mathbf{a}_i^T, \mathbf{x}) = b_i$ определяет линейное многообразие в \mathbf{E}_n (смещенную на некоторый вектор гиперплоскость). Это многообразие \mathbf{M}_i является *границей* интересующего нас множества точек, удовлетворяющих неравенству $(\mathbf{a}_i^T, \mathbf{x}) \leq b_i$. Граница делит все пространство \mathbf{E}_n на две части, поэтому каждое неравенство определяет *полупространство*.

В двумерном случае каждое ограничение-равенство $a_{i1}x_1 + a_{i2}x_2 = b_i$ определяет прямую, рассекающую плоскость на две полуплоскости.

В трехмерном случае каждое такое ограничение-равенство $a_{i1}x_1 + a_{i2}x_2 = b_i$ определяет плоскость (двумерное линейное многообразие), которая рассекает все пространство на два полупространства. Одно из них соответствует неравенству $a_{i1}x_1 + a_{i2}x_2 \leq b_i$, а другое — неравенству $a_{i1}x_1 + a_{i2}x_2 \geq b_i$. Эта плоскость параллельна оси Ox_3 .

Система из двух ограничений-равенств определяет пересечение двух линейных многообразий размерности $n - 1$, т.е. многообразие размерности $n - 2$ и т.д. А $n - 1$ равенство задает прямую (многообразие размерности 1) и, наконец, n равенств дают точку (многообразие размерности 0). Изложенное верно только для линейно независимых ограничений-равенств (строк матрицы ограничений).

Итак, в общем случае граница ОДР в задаче линейного программирования — это множество точек, принадлежащих линейным многообразиям различной размерности (от $n - 1$ до 0). Если допустимая область ограничена, то в трехмерном случае она представляет собой многогранник, а в двумерном — многоугольник.

В трехмерном случае граница допустимой области (ОДР) — это куски плоскостей (грани), отрезки прямых (ребра) и точки (вершины) многогранника.

В двумерном случае это отрезки прямых (стороны) и точки (вершины) многоугольника.

Пример.

Задана система ограничений

$$\begin{cases} x_2 + x_1 \leq 0; \\ x_2 - x_1 \leq 0; \\ x_2 \geq -1. \end{cases}$$

Соответствующая этой системе ограничений допустимая область показана на рис. 2.

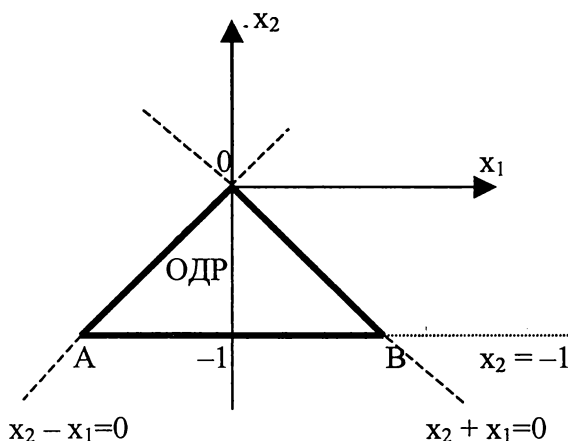


Рис. 2. Допустимая область АОВ

Ограничению $x_2 - x_1 \leq 0$ удовлетворяют точки полуплоскости вниз от прямой АО. Ограничению $x_2 + x_1 \leq 0$ удовлетворяют точки полуплоскости вниз от прямой ВО.

Эти ограничения выполняются совместно в неограниченном секторе АОВ. Далее, ограничение $x_2 \geq -1$ выполняется в полуплоскости вверх от прямой АВ.

Так как должны быть выполнены *все* три неравенства *совместно*, то допустимая область (ОДР) получается как *пересечение* сектора АОВ и этой полуплоскости. Это треугольник АОВ.

При большом числе ограничений-неравенств в двумерном случае можно получить многоугольник, число сторон которого может быть сколь угодно велико.

Однако не следует думать, что с увеличением числа ограничений-неравенств число сторон многоугольника обязательно увеличивается. Возможны случаи, в которых число неравенств сколь угодно велико, но допустимая область остается треугольником.

Допустимая область может быть неограниченной. Такой пример для двумерного случая приведен на рис. 3.

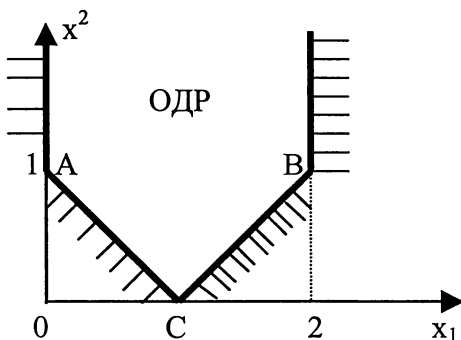


Рис. 3. Пример неограниченной ОДР

Система ограничений содержит пять неравенств:

$$\left\{ \begin{array}{l} x_1 \geq 0; \\ x_2 \geq 0; \\ x_1 \leq 2; \\ x_1 + x_2 \geq 1. \\ x_1 - x_2 \leq 1. \end{array} \right.$$

Очевидно, что x_2 может принимать сколь угодно большие значения и все неравенства будут выполнены.

Возникает вопрос: может ли при каких-либо коэффициентах в системе ограничений получиться невыпуклый многоугольник (в многомерном случае невыпуклый многогранник)? Или, например, область, являющаяся объединением двух треугольников, не имеющих общих точек (рис. 4)?

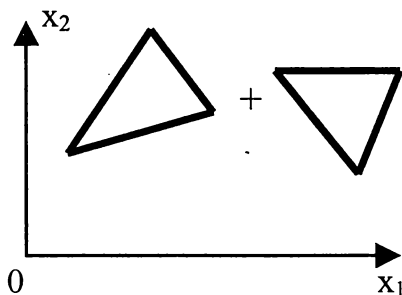


Рис. 4. Пример невозможной структуры ОДР

Для ответа на эти вопросы и окончательного решения вопроса о структуре ОДР нам понадобятся еще некоторые математические понятия.

Отрезок. Пусть x^1 и x^2 — две произвольные точки в E_n , а λ — число. $0 \leq \lambda \leq 1$. Множество точек $\lambda x^1 + (1 - \lambda)x^2$ называется отрезком, соединяющим точки x^1 и x^2 . При $\lambda = 0$ получаем точку x^1 , а при $\lambda = 1$ — точку x^2 . При других λ получаем внутренние (промежуточные) точки отрезка.

Выпуклое множество. Множество называется выпуклым, если вместе с любыми двумя его точками оно содержит и отрезок, их соединяющий.

Оказывается, исследуемое нами множество решений системы линейных неравенств (ОДР) — это *выпуклое множество при любых значениях элементов матрицы системы ограничений и векторе свободных членов*.

Действительно, возьмем произвольное i -е неравенство из системы $(a_i^T, x) \leq b_i$. Пусть оно выполнено в некоторых точках x^1 и x^2 , т.е. $(a_i^T, x^1) \leq b_i$ и $(a_i^T, x^2) \leq b_i$. Берем любую точку x^0 отрезка $[x^1, x^2]$. Ей соответствует некоторое λ ($0 \leq \lambda \leq 1$), так что $x^0 = \lambda x^1 + (1 - \lambda)x^2$.

Вычислим теперь (a_i^T, x^0) .

$$(\mathbf{a}_i^T, \mathbf{x}^0) = (\mathbf{a}_i^T, (\lambda \mathbf{x}^1 + (1-\lambda) \mathbf{x}^2)) = \lambda (\mathbf{a}_i^T, \mathbf{x}^1) + (1-\lambda) (\mathbf{a}_i^T, \mathbf{x}^2)$$

Но эта сумма не превосходит b_i , так как $0 \leq \lambda \leq 1$ и замена в ней каждого скалярного произведения $(\mathbf{a}_i^T, \mathbf{x}^1)$ и $(\mathbf{a}_i^T, \mathbf{x}^2)$ на b_i может только увеличить результат. После этой замены получим вместо суммы просто b_i , и поэтому $(\mathbf{a}_i, \mathbf{x}^0) \leq b_i$. Следовательно, любая точка \mathbf{x}^0 отрезка $[\mathbf{x}^1, \mathbf{x}^2]$ принадлежит множеству точек, удовлетворяющих неравенству $(\mathbf{a}_i, \mathbf{x}) \leq b_i$. Иначе говоря, это множество выпукло.

Если вместо ограничений-неравенств рассматривать ограничения-равенства, то тем же способом получим тот же результат, а именно: множество точек, удовлетворяющих равенству $(\mathbf{a}_i, \mathbf{x}) = b_i$ выпукло. Т.е. мы установили, что и полупространства и линейные многообразия — это выпуклые множества. Но решение системы линейных неравенств (или равенств) это *пересечение множеств*, соответствующих решениям отдельных неравенств (или равенств), так как эти неравенства (равенства) должны выполняться совместно. *Пересечение выпуклых множеств — это выпуклое множество* (см. приложение 1, п. 9), и, следовательно, допустимая область (ОДР) в задаче линейного программирования *выпукла*. Поэтому ОДР — это *замкнутый выпуклый многогранник (многоугольник)*. Она не может иметь вид, представленный на рис. 4. При $n = 2$ ОДР — выпуклый многоугольник (возможно, неограниченный) и *ничего более*.

Установленная структура ОДР имеет фундаментальное значение для решения задач линейного программирования.

При исследовании структуры ОДР нас не интересовала целевая функция и ее линейность не использовалась. Поэтому установленные свойства множества решений систем

линейных неравенств и/или равенств могут использоваться и при нелинейных целевых функциях.

Для анализа возможности решения задачи линейного программирования нам понадобится еще одно понятие.

Линии уровня целевой функции — это линии, на каждой из которых целевая функция постоянна. В линейном программировании целевая функция линейна и линии уровня — это параллельные прямые (в многомерном случае линейные многообразия, получаемые смещением гиперплоскости, соответствующей нулевому значению целевой функции). Каждой линии уровня соответствует свое значение целевой функции. В нашей задаче целевая функция — это скалярное произведение (c, x) . Задавая различные значения целевой функции, получаем уравнение для семейства линий уровня $(c, x) = \text{const}$. Мы уже рассматривали множества точек, удовлетворяющих таким уравнениям. При $\text{const} = 0$ это гиперплоскость (в двумерном случае это прямая, проходящая через начало координат, см. рис. 1), при других константах это линейные многообразия (в двумерном случае прямые). Вектор c ортогонален всем линиям уровня (линейным многообразиям). Все линии уровня параллельны между собой.

Геометрический смысл задачи линейного программирования состоит в том, чтобы, *не выходя за пределы допустимой области, найти точку на линии уровня с наименьшим значением* (в задаче на поиск максимума с наибольшим значением).

На рис. 5 представлена допустимая область $A0B$, соответствующая системе линейных неравенств:

$$\begin{cases} x_2 - x_1 \leq 0 \\ x_2 + x_1 \leq 0 \\ -x_2 \leq 1 \end{cases}$$

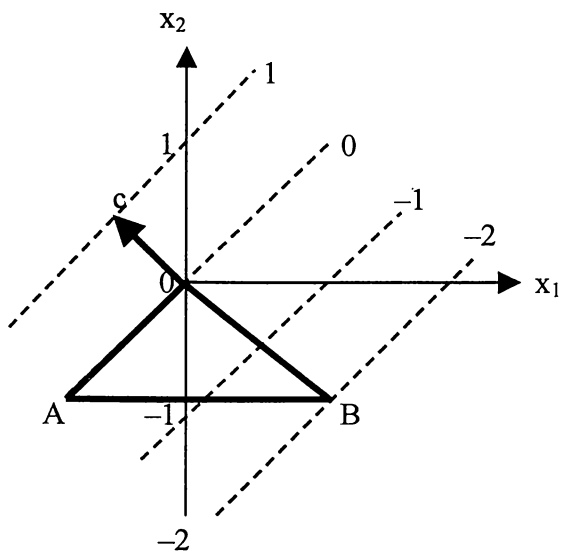


Рис. 5. Линии уровня

Если целевая функция $F(x_1, x_2) = x_2 - x_1$, то вектор c имеет координаты $(-1, 1)$ и показывает направление возрастания значений целевой функции. Линия уровня, соответствующая нулевому значению целевой функции, — это прямая $x_2 = x_1$. Различные линии уровня целевой функции это прямые параллельные биссектрисе первого координатного угла.

Если задача состоит в поиске точки минимума целевой функции, то это точка $B(1, -1)$, так как она является допустимой и принадлежит линии уровня с наименьшим значением минус 2.

Если же задача состоит в поиске точки максимума той же целевой функции на той же допустимой области, то таких точек бесконечно много (отрезок AO), но все они принадлежат линии уровня со значением нуль.

Если в качестве целевой функции взять $-F(x_1, x_2) = x_1 - x_2$, то точка минимума становится точкой максимума, а все точки максимума становятся точками минимума.

Из геометрических соображений понятно, что в двумерном случае на ограниченной ОДР задача линейного программирования имеет одно решение (и тогда это вершина многоугольника) или бесконечно много решений с одним и тем же значением целевой функции (в этом случае линии уровня параллельны стороне многоугольника, а вектор c ей ортогонален).

В многомерном случае имеет место аналогичная ситуация. Действительно, никакая внутренняя точка ОДР не может быть точкой экстремума линейной функции. В противном случае в этой точке должны быть равны нулю *все* ее частные производные (нулевой градиент). Но частные производные целевой функции — это коэффициенты линейной формы, и градиент — это вектор c , который не может быть *равным нулю, иначе целевая функция тождественный нуль*.

Итак, минимум лежит на границе, но в какой точке? Среди всех точек границы выделим *крайние* точки. Точка называется крайней точкой множества, если не существует отрезка, целиком принадлежащего множеству, для которого эта точка внутренняя. В задаче линейного программирования это вершины допустимой области (многогранника), т.е. точки, в которых ровно n ограничений выполнены как равенства (иначе говоря, это пересечение n граничных линейных многообразий). Из любой крайней точки по любому направлению в пределах допустимой области можно идти только в одну сторону, иначе она была бы внутренней точкой отрезка, принадлежащего допустимой области, т.е. не была бы крайней. Используем это свойство крайних точек и рассмотрим еще одно важное понятие, которое будет часто использоваться в дальнейшем.

Луч. Если в E_n заданы точка (вектор) x^0 и вектор p , то множество точек $x = x^0 + \lambda p$ (где $\lambda > 0$ скалярный параметр), называется *луч* из точки x^0 в направлении вектора p . При этом длина вектора λp называется *шагом* из точки x^0 в направлении вектора p . Если длина вектора p равна единице, то шаг равен λ . При $\lambda < 0$ можно считать, что сделан шаг в направлении минус p (рис. 6).

Напомним, что геометрически *все* векторы мы представляем стрелками, исходящими из начала координат, так как каждому вектору соответствует *одна* точка, являющаяся его концом.

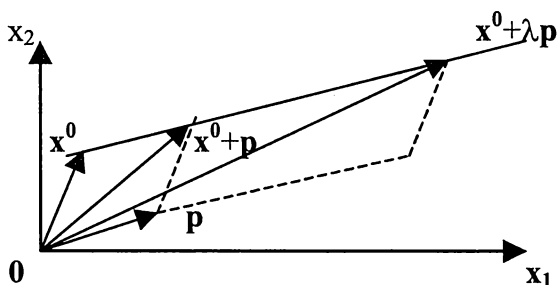


Рис. 6. К понятию луч

Предположим, что точка x^0 граничная (но не крайняя) точка ОДР. Вычислим в ней значение целевой функции (c, x^0) и рассмотрим, как ведет себя целевая функция, если мы будем двигаться из точки x^0 в некотором направлении p , оставаясь в пределах границы допустимой области, т.е. двигаться по допустимому лучу в пределах граничного многообразия M .

$(c, x) = (c, x^0 + \lambda p) = (c, x^0) + \lambda(c, p)$. Предположим сначала, что $(c, p) \neq 0$, т.е. что на граничном многообразии M существует хотя бы один луч, неортогональный вектору c . Тогда при $(c, p) > 0$ целевая функция возрастает при дви-

жении по лучу из точки \mathbf{x}^0 и убывает при движении в противоположном направлении, так как $(\mathbf{c}, \mathbf{x}^0)$ — это исходное значение целевой функции, а $\lambda(\mathbf{c}, \mathbf{p})$ — его приращение при шаге $\lambda \mathbf{p}$ по лучу. При $(\mathbf{c}, \mathbf{p}) < 0$, наоборот, в направлении \mathbf{p} целевая функция убывает, а в противоположном направлении она возрастает. Другими словами, если из граничной (но не крайней) точки \mathbf{x}^0 в пределах граничного многообразия \mathbf{M} существует *допустимый луч, неортогональный вектору \mathbf{c}* , то эта точка *не может быть точкой минимума (или максимума)*. Двигаясь из точки \mathbf{x}^0 в направлении убывания целевой функции, мы при некотором λ в соответствующей ему точке \mathbf{x}^1 достигнем новой границы. В противном случае ОДР неограничена и целевая функция может принимать сколь угодно малые значения (с учетом знака). Точка \mathbf{x}^1 принадлежит пересечению многообразия \mathbf{M} и нового граничного многообразия, т.е. она принадлежит многообразию *меньшей размерности, чем \mathbf{M}* . В трехмерном случае это соответствует движению по грани многогранника с выходом на пересечение другой грани, т.е. на ребро. Повторяя рассуждения для новой точки \mathbf{x}^1 мы установим или неограниченность ОДР и целевой функции (снизу), или не сможем найти допустимого луча \mathbf{p} , неортогонального вектору \mathbf{c} , или придем на многообразие еще меньшей размерности и так до размерности нуль, т.е. до вершины ОДР. Значит ли это, что вершина, в которой мы окажемся, и есть точка минимума? Вообще говоря, нет. А что означает отсутствие в пределах граничного многообразия луча, неортогонального вектору \mathbf{c} ? Только то, что вектор \mathbf{c} ортогонален этому многообразию. Так, на рис. 5 вектор \mathbf{c} ортогонален границе A_0 , а точка минимума B не принадлежит этой границе. Однако приведенные рассуждения позволяют утверждать, что *в задаче линейного программирования минимум (если он существует) достигается хотя бы в одной крайней точке (вершине) ОДР*.

Действительно, предположим, что это неверно, т.е. минимум существует, но достигается не в вершине, а в некоторой точке x^* . Поскольку, как уже отмечалось, он не может достигаться во внутренней точке, то точка минимума граничная (но не крайняя по предположению). Но мы установили, что это возможно только в случае ортогональности вектора s граничному многообразию, содержащему точку минимума x^* (иначе уйдем по лучу в направлении убывания целевой функции). Отсюда следует равенство значений целевой функции *во всех точках* этого многообразия (приращение целевой функции по любому лучу из точки x^* равно нулю). Но в каждом граничном многообразии есть вершина (при ограниченности ОДР) и мы показали как ее достигнуть, двигаясь из x^* по допустимому лучу. Это означает, что и в некоторой вершине целевая функция имеет то же значение, что и в точке минимума, что и доказывает приведенное утверждение. Итак, установлено: *решение задачи линейного программирования (если оно существует) достигается в вершине допустимой области. Это решение единственно или достигается во всех точках (в том числе и в вершинах) некоторого граничного линейного многообразия, ортогонального вектору s* . Поскольку градиент s ортогонален линиям (в многомерном случае плоскостям) уровня, то при отсутствии единственности решения задачи граничное линейное многообразие, на котором достигается минимум, параллельно линиям (плоскостям) уровня целевой функции. В любом случае решение следует искать *только в вершинах* ОДР, так как даже при отсутствии единственности решения во всех точках минимума, в том числе и в вершине, значения целевой функции равны. Тем самым установлена *особая роль вершин* (крайних точек) ОДР в линейном программировании. Доказательство того, что любая вершина ОДР является ее крайней точкой и обратно (любая крайняя точка ОДР — это ее вершина), приведено в приложении 1, п. 7.

Задача не имеет решения при отсутствии ОДР (противоречивые ограничения) или при неограниченности целевой функции на неограниченной ОДР.

1.4. Обучающая программа DANTZIG_1.exe

Программа DANTZIG_1.exe предназначена для изучения структуры области допустимых решений двумерной задачи линейного программирования, в которой все ограничения записаны в виде системы линейных неравенств. Предполагается, что каждое неравенство приведено к виду:

$$ax + by \leq c.$$

Если исходное неравенство содержит знак \geq , его нужно умножить на минус единицу. Таким образом, исходными данными для программы являются коэффициенты неравенств, которые вводятся последовательно (a,b,c). После ввода коэффициентов первого неравенства на экране показывается соответствующая ему полуплоскость и можно вводить коэффициенты для второго неравенства. После их ввода программа показывает пересечение соответствующих полуплоскостей, т.е. ОДР системы двух неравенств. Уже на этом этапе может оказаться, что ОДР пуста, например для системы

$$\begin{cases} x+y \leq 1; \\ -x-y \leq -2. \end{cases}$$

В подобных случаях программа отображает соответствующую ситуацию на экране и дает сообщение об отсутствии допустимых решений, после чего ввод данных для следующего неравенства смысла не имеет. После ввода данных для каждого очередного неравенства программа показывает на экране ОДР для уже введенной системы

(текущая ОДР). Пользователь видит, как очередное ограничение воздействует на ОДР, преобразуя ее или оставляя без изменений. Пользователь имеет возможность вводить практически любое число неравенств и наблюдать за ОДР или отказаться от ввода очередного неравенства. Можно отказаться от ошибочно введенного неравенства, т.е. вернуться на шаг назад.

После того как ввод неравенств закончен, программа анализирует ОДР. В случае если ОДР не ограничена, выдается соответствующее сообщение. Если ОДР ограничена и не пуста, программа выдает на экран ее графическое изображение. При этом можно получить и декартовы координаты вершин соответствующего многоугольника при его обходе по часовой стрелке.

Для удобства анализа ОДР пользователем программа позволяет менять масштаб изображения на экране по ходу ввода данных об очередном неравенстве.

Таким образом, программа DANTZIG_1.exe — это программа решения произвольной системы линейных неравенств с двумя неизвестными и графического отображения множества решений.

Задания

1. Решить с помощью программы DANTZIG_1.exe систему линейных неравенств:

$$x \geq 0; \quad x + y \geq 1;$$

$$y \geq 0; \quad x - y \leq 1.$$

$$x \leq 2;$$

2. Повторно задать программе все неравенства, кроме $y \geq 0$. Изменилась ли ОДР, и если нет, то почему?

3. Задать программе DANTZIG_1.exe систему

$$x \geq 0; y \geq 0; x \leq 2; x + y \leq 1; x - y \geq 1,$$

и установить, какой ОДР она соответствует.

4. Выяснить с помощью программы DANTZIG_1.exe как изменится ОДР, если знаки *всех* неравенств (например, в задании 1) изменить на противоположные, т.е. изменить знаки *всех* чисел, вводимых в программу.

Контрольные вопросы и задачи

1. Может ли ОДР в задаче линейного программирования состоять из одной-единственной точки? Если да, то привести пример.

Сопоставить свой ответ и результат выполнения задания 3.

2. Может ли в задаче линейного программирования ОДР состоять из двух и только из двух точек?

3. Пусть ОДР сформирована системой линейных неравенств и представляет собой многоугольник. Как изменится ОДР, если знаки *всех* неравенств изменить на противоположные, т.е. изменить знаки *всех* чисел, вводимых в программу? Можно ли утверждать, что новая ОДР представляет собой множество точек, внешних по отношению к прежней ОДР (многоугольнику), и потому неограничена? Сопоставить свой ответ и результат выполнения задания 4.

4. Доказать, что если x^0 граничная, но не крайняя точка ОДР, то существует луч из x^0 , по которому можно сделать шаг конечной длины в обоих направлениях, оставаясь на границе ОДР.

5. Доказать, что в задаче линейного программирования не может быть локальных минимумов (максимумов).

6. Может ли в двумерном случае ОДР представлять собой неограниченный сектор с центральным углом больше 180° ?

7. Доказать, что допустимая вершина (точка, удовлетворяющая ровно n ограничениям как равенствам) является крайней точкой ОДР и обратно, любая крайняя точка ОДР — это допустимая вершина.

8. Может ли число недопустимых вершин быть больше, чем число допустимых?

9. Доказать, что пересечение выпуклых множеств есть выпуклое множество.

10. В линейном программировании ОДР — это выпуклое множество. Является ли множество всех граничных точек ОДР выпуклым?

11. Верно ли, что при неограниченной ОДР задача линейного программирования не может иметь решения?

12. Доказать тождество $(x, Ay) = (A^T x, y)$.

1.5. Основные формы записи задачи линейного программирования

В задаче линейного программирования система ограничений может содержать и равенства и неравенства, т.е. может быть смешанной. Однако с помощью введения дополнительных неизвестных она всегда может быть преобразована к одной из основных форм. Этих форм две.

Первая форма содержит только ограничения в виде равенств и обязательно условие неотрицательности *всех* неизвестных.

Вторая форма содержит только ограничения-неравенства, в числе которых могут быть (а могут и не быть) условия неотрицательности неизвестных.

Первая форма задачи линейного программирования имеет следующий вид:

$$\text{Найти } \min \sum_{i=1}^{i=n} c_i x_i \text{ при ограничениях: } Ax = b \text{ и } x_i \geq 0.$$

Перейти от произвольной задачи линейного программирования к основной форме можно с помощью введения дополнительных переменных. При этом каждое неравенство становится равенством, т.е. вместо $(a_i^T, x) \leq b_i$ получаем $(a_i^T, x) + w_i = b_i$ и $w_i \geq 0$. Дополнительных неизвестных w_i столько сколько ограничений-неравенств. Далее, вместо неограниченных по знаку переменных x_i вводятся неотрицательные переменные u_i и v_i по формуле

$$x_i = u_i - v_i ,$$

Вместо каждого неограниченного по знаку неизвестного появляются два новых. При этом увеличение числа неизвестных равно числу неограниченных по знаку x_i . Естественно, что неограниченные по знаку x_i должны быть заменены на $u_i - v_i$ не только в системе ограничений, но и в целевой функции.

Вторая форма задачи линейного программирования имеет вид:

$$\text{Найти } \min \sum_{i=1}^{i=n} c_i x_i \text{ при ограничениях: } Ax \leq b.$$

К ней можно перейти путем замены каждого равенства двумя неравенствами противоположного знака. При этом увеличивается число ограничений, но остаются те же неизвестные.

Характерно, что в каждой из основных форм записи задачи линейного программирования присутствуют ограничения-неравенства, в частности условия неотрицательности неизвестных, т.е. неравенства вида $x_i \geq 0$ ($i = 1, \dots, n$).

Наличие ограничений в виде неравенств — это и есть та особенность задачи (не только в линейном случае), которая не позволяет применить для ее решения аппарат классической математики.

Переход от произвольной системы ограничений к одной из основных форм необходим потому, что задачи линейного программирования при большом числе неизвестных решаются на компьютерах, а алгоритмы их решения и соответствующие компьютерные программы как раз и требуют записи задачи в соответствующей форме.

1.6. Симплекс-метод

Поскольку точку минимума в задаче линейного программирования надо искать среди вершин ОДР, возникает идея перебрать по очереди все вершины, в каждой из них вычислить значение целевой функции и выбрать вершину, в которой оно минимально.

Этот метод полного перебора может быть применен только при малом числе неизвестных и ограничений. Фактически количество вершин равно количеству вариантов выбора n ограничений из общего числа m ограничений, так как в вершине ровно n ограничений активны, т.е. выполняются как равенства. При этом далеко не все вершины будут допустимыми (рис. 7).

Общее число вершин равно числу сочетаний при выборе из m по n , т.е.

$$m! / (n!(m-n)!).$$

При $m = 10$, $n = 5$ это число равно 252, при $m = 12$, $n = 6$ оно равно 924. При $m = 20$, $n = 10$ имеем уже 184 756 вершин. А при $m = 50$, а $n = 25$ это число примерно равно 10^{14} , что очень много.

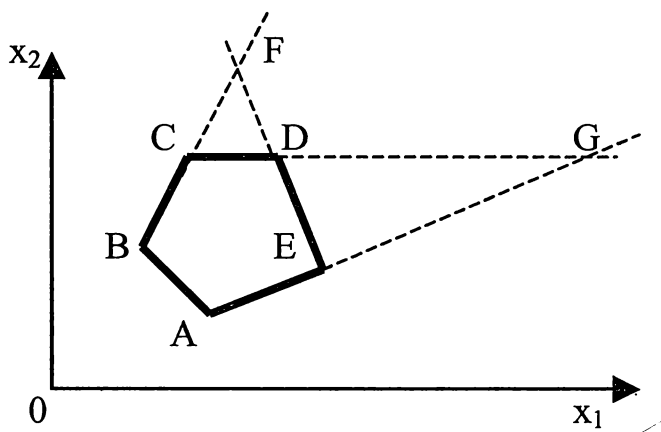


Рис. 7. Вершины A, B, C, D, E — допустимые,
F, G — не допустимые

В реальных практических задачах число переменных и ограничений может достигать сотен и даже тысяч. Поэтому необходимы более эффективные методы, чем полный перебор.

Такой метод предложил Дж.Б. Данциг в 1949 г. и назвал его симплекс-метод. Он доказал, что множество решений линейной системы неравенств (ОДР) — это многогранник. Отсюда и название метода, так как слово «симплекс» означает многогранник. Дж.Б. Данциг установил и отмеченную выше особую роль вершин многогранника для поиска решения.

Основная идея симплекс-метода: найти какую-либо допустимую вершину многогранника и от нее перейти в другую вершину с *меньшим* значением целевой функции. Поскольку число вершин конечно и; переходя на каждом шаге в вершину с меньшим значением целевой функции, мы не можем вернуться в уже пройденную вершину, то за

конечное число шагов вершина, в которой достигается минимум, будет найдена. При этом необходимо:

1. Уметь находить допустимую вершину (начальное приближение);
2. Знать, как перейти в вершину с меньшим значением целевой функции;
3. Иметь способ проверки не является ли достигнутая на очередном шаге вершина точкой минимума.
4. Уметь обнаруживать отсутствие решения, т.е. неограниченность целевой функции на ОДР снизу.

Данциг рассматривал задачу в первой основной форме, т.е.

$$\min (c, x)$$

при $Ax = b$ и $x \geq 0$.

Обозначим число переменных n , а число ограничений-равенств m . Таким образом, в матрице A m строк и n столбцов. Предположим, что строки матрицы A линейно независимы ($m < n$).

Нас интересуют вершины, т.е. точки, в которых ровно n ограничений выполнены как равенства. Но m равенств уже есть (система уравнений $Ax = b$), значит, в каждой вершине ровно $n - m$ переменных должны быть равны нулю (больше нигде взять недостающие $n - m$ равенств как из условий $x_i \geq 0$). *Оставшиеся m переменных должны быть строго положительными.*

Итак, должен быть некоторый набор из $n - m$ нулевых переменных (они называются свободными), а остальные m переменных (они называются базисными) можно найти из системы уравнений $Ax = b$ после подстановки в нее нулей вместо свободных переменных. Система имеет единственное решение в силу линейной независимости строк матрицы A . Если в результате решения системы мы получим только положительные числа, то это означает, что мы име-

ем допустимую вершину (допустимый базис). Если будет хотя бы одно отрицательное число, то это недопустимая вершина, так как условие неотрицательности всех переменных было изначально. Если же будет еще хотя бы один нуль, то число активных ограничений станет больше, чем n (m равенств-уравнений, плюс $n - m$ равных нулю свободных переменных и еще один нуль). Это так называемый случай вырожденности, который мы рассмотрим далее отдельно.

Симплекс-метод состоит в том, что сначала находится допустимый базис, затем производится замена одной из базисных переменных на одну из свободных, т.е. переход в другую вершину. При этой замене достигается уменьшение целевой функции. Алгоритм симплекс-метода подробно изложен в литературе [7, 8, 11, 19]. Здесь мы поясним его на конкретном примере преобразования симплексных таблиц.

Пример.

Найти $\max x_1 + 2x_2 + 2x_3 + x_4$ при ограничениях

$$\begin{cases} x_1 - x_3 + \frac{1}{2}x_4 = 1 \\ x_2 + x_3 - x_4 = 1 \\ x_i \geq 0 \quad (i = 1, 2, 3, 4) \end{cases}$$

Вместо задачи на максимум будем решать задачу на минимум. Для этого умножим целевую функцию на минус 1. В нашей задаче $n = 4$, $m = 2$, значит, должно быть две свободных и две базисных переменных. Если взять в качестве свободных переменных x_1 и x_4 и приравнять их нулю, то базисные будут равны $x_2 = 2$ и $x_3 = -1$, что соответствует вершине $x_1 = 0$, $x_2 = 2$, $x_3 = -1$, $x_4 = 0$. Но эта вершина недо-

пустимая, так как $x_3 < 0$. Если же взять в качестве свободных переменных x_3 и x_4 , то базисные переменные будут: $x_1 = 1$ и $x_2 = 1$, и получим допустимую вершину $x_1 = 1$, $x_2 = 1$, $x_3 = 0$, $x_4 = 0$. Это и есть начальное приближение. Чтобы перейти в следующую вершину с меньшим значением целевой функции, нужно выразить из исходной системы уравнений базисные переменные через свободные, подставить их выражения в целевую функцию и составить симплексную таблицу.

$$x_1 = 1 + x_3 - \frac{1}{2}x_4;$$

$$x_2 = 1 - x_3 + x_4.$$

Целевая функция $K(x_3, x_4) = -3 - x_3 - 2.5x_4$

Симплексная таблица имеет вид:

	1	x_3	x_4
x_1	1	1	-0,5
x_2	1	-1	1
K	-3	-1	-2,5

В таблице свободные переменные записаны в верхней строке, а базисные — в левом столбце. Двойная вертикальная черта означает знак равенства, так что каждая строка таблицы (кроме, разумеется, верхней) — это запись выражений базисных переменных и целевой функции (последняя строка) через свободные переменные. При этом первый столбец от вертикальной черты — это столбец свободных членов в соответствующих выражениях.

Займемся анализом симплексной таблицы. Мы хотим уменьшить значение целевой функции, которая теперь зависит только от x_3 и x_4 . В исходной вершине они нулевые и целевая функция равна -3. Свободные переменные (как и

базисные) не могут быть отрицательными по условию задачи, поэтому мы можем только увеличивать их, но при этом базисные переменные должны оставаться неотрицательными, а целевая функция уменьшаться. В нашем примере оба коэффициента целевой функции при свободных переменных отрицательны, и поэтому при увеличении свободных переменных она уменьшается. А что означало бы отсутствие отрицательных коэффициентов целевой функции при свободных переменных, т.е. отсутствие отрицательных чисел в последней строке (столбец свободных членов не учитывается)? Если все эти числа неотрицательны, то целевую функцию уменьшить нельзя, следовательно, минимум достигнут. Это и есть правило окончания счета: отсутствие отрицательных чисел в нижней строке симплексной таблицы (не считая столбец свободных членов).

В нашей задаче можно увеличить обе свободные переменные, но предпочтение отдадим x_4 , так как при ней коэффициент больше по абсолютной величине и, следовательно, при увеличении x_4 целевая функция уменьшится на большую величину, чем при таком же изменении x_3 . Итак, оставляем нулевое x_3 и начинаем увеличивать x_4 , т.е. в плоскости (x_3, x_4) из начала координат идем по оси x_4 . Целевая функция убывает, но *меняются и зависящие от x_4 базисные переменные*. Переменная x_2 увеличивается, так как коэффициент ее зависимости от x_4 положительный (плюс 1), но переменная x_1 , которая была равна 1, убывает, так как ее коэффициент отрицательный (-0.5). Это означает, что x_4 можно увеличивать только до тех пор, пока x_1 не станет равной нулю. Так как

$$x_1 = 1 + x_3 - \frac{1}{2}x_4 = 1 + 0 - \frac{1}{2}x_4,$$

то базисная переменная x_1 станет равной нулю, когда x_4 достигнет значения 2. При этом x_1 станет свободной переменной, а x_4 базисной. Целевая функция изменится на минус 5 и станет равной минус 8. Новое значение переменной x_2 равно 3, и новая вершина имеет координаты $(0, 3, 0, 2)$.

Теперь надо выразить новые базисные переменные x_2 и x_4 и целевую функцию через свободные переменные x_1 и x_3 и перейти к новой симплексной таблице. Выполнив соответствующие преобразования, получим

$$x_4 = 2 + 2x_3 - 2x_1;$$

$$x_2 = 3 + x_3 - 2x_1;$$

$$K(x_1, x_3) = -8 - 6x_3 + 5x_1.$$

Новая симплексная таблица, в которой переменные x_1 и x_4 поменялись местами, имеет следующий вид:

	1	x_3	x_1
x_4	2	2	-2
x_2	3	1	-2
K	-8	-6	5

В новой таблице в последней строке только один отрицательный коэффициент при свободных переменных (-6), но в соответствующем столбце (он называется разрешающий столбец) все числа положительны. Это означает, что переменную x_3 можно *неограниченно увеличивать*. При этом целевая функция будет неограниченно уменьшаться, базисные переменные x_4 и x_2 неограниченно увеличиваться, а свободная переменная x_1 останется равной нулю. Мы пришли к выводу, что задача не имеет решения, так как целевая функция неограничена из-за неограниченности ОДР, и даже нашли луч из точки $(0, 3, 0, 2)$ с координатами

(0, 1, 1, 2), в направлении которого можно двигаться сколь угодно долго, оставаясь в пределах ОДР.

Алгоритм симплекс-метода предполагает, что исходная допустимая вершина известна и включает следующие правила преобразования симплексных таблиц.

1. Выбираем наибольший по абсолютной величине отрицательный элемент в нижней строке коэффициентов при свободных переменных и тем самым разрешающий столбец. Если отрицательных коэффициентов нет, то минимум найден. Если все эти числа строго положительны, то этот минимум единственный, иначе бесконечное множество решений соответствует изменению свободной переменной, коэффициент при которой оказался нулем.
2. В разрешающем столбце находим отрицательные числа. Если их нет, то целевая функция неограниченно убывает при увеличении соответствующей свободной переменной и, следовательно, задача не имеет решения. Если отрицательные числа в разрешающем столбце встречаются в нескольких строках, то в каждой такой строке вычисляем частное от деления свободного члена на число в разрешающем столбце. Затем берем ту строку, для которой это частное минимально по абсолютной величине. Эта строка называется разрешающей, а соответствующий элемент разрешающего столбца называется опорным элементом. В нашем примере в первой симплексной таблице разрешающий столбец последний, а разрешающая строка первая, и опорный элемент равен -0.5 .
3. Элементы разрешающего столбца и разрешающей строки оставляем (пока) без изменения.
4. Все прочие элементы таблицы, включая нижнюю строку и столбец свободных членов, необходимо вычислить по правилу «прямоугольника», а именно: образовать прямоугольник, у которого перевычисляемый и опорный элементы составляют одну диагональ (назовем ее пер-

вой), а две вершины второй диагонали получаются при пересечении разрешающего столбца и строки с перевычисляемым элементом, а также разрешающей строки и столбца с перевычисляемым элементом. В нашем примере в первой симплексной таблице при перевычислении нового (промежуточного) значения целевой функции рассматривается прямоугольник

$$\begin{vmatrix} 1 & -0.5 \\ -3 & -2.5 \end{vmatrix}$$

В прямоугольнике берем разность произведений элементов первой и второй диагонали и результат записываем вместо перевычисляемого элемента. В нашем случае $1,5 - (-2,5) = 4$.

5. Разрешающая строка (кроме опорного элемента) меняет знак, опорный элемент заменяется на 1.
6. Вся таблица делится на опорный элемент (конечно, на его исходное значение, а не на 1).

В новой таблице необходимо поменять местами свободную переменную из разрешающего столбца и базисную из разрешающей строки.

Изложенные правила преобразования симплексных таблиц следуют непосредственно из порядка преобразования системы уравнений при замене одной базисной переменной на одну свободную.

Алгоритм симплекс-метода работает во всех случаях, кроме случаев вырожденности. Вырожденность возникает, когда кроме свободных оказывается равной нулю и хотя бы одна из базисных переменных, т.е. один из свободных членов становится нулем. Если этот член находится в разрешающей строке, то мы формально можем произвести преобразование симплексной таблицы, но получим то же значение целевой функции (по второй диагонали прямоуголь-

ника произведение равно нулю). Эта ситуация объясняется на рис. 8, где в вершине В допустимой области ABC пересекаются не две, а три граничные прямые

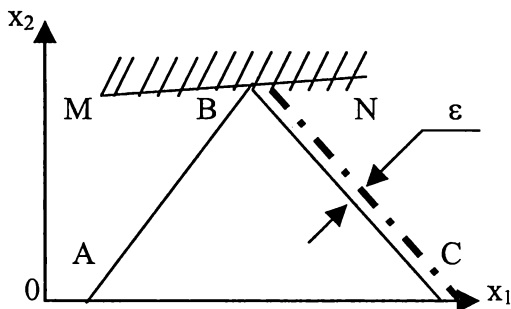


Рис. 8. Случай вырожденности

Преобразование базиса соответствует переходу от вершины, образованной пересечением прямых АВ и МN, к «новой» вершине, образованной пересечением АВ и ВС и т.д. В подобных ситуациях алгоритм симплекс-метода может заикли-ваться. Для устранения заикливания в практических реализа-циях алгоритма предпринимаются различные меры. Наиболее известным является ϵ -метод. Его идея состоит в незначитель-ном изменении свободного члена в одном из уравнений, соот-ветствующих «слипшимся вершинам»; на рис. 8 это соответ-ствует параллельному смещению прямой ВС.

Еще один полезный прием состоит в том, чтобы избе-жать попадания в вырожденные вершины, что, к сожале-нию, не всегда возможно. Если при выборе опорного эле-мента выясняется, что сразу два свободных члена станут равны нулю, то это и означает, что очередная вершина бу-дет вырожденной. Конечно, нельзя взять в качестве опорно-го элемента другой отрицательный элемент в том же столб-це, но можно поискать другой разрешающий столбец. Если

такой столбец существует, т.е. будет найдено другое отрицательное число в симплексной таблице, то целесообразно рассмотреть его в качестве разрешающего столбца.

Рассмотренный алгоритм симплекс-метода позволяет найти решение задачи или установить его отсутствие в случае неограниченности ОДР и целевой функции.

Ранее мы отмечали, что отсутствие решения возможно и при противоречивости ограничений (пустая ОДР). Как будет работать симплекс-метод в этой ситуации? Пока мы предполагали, что допустимая вершина (начальное приближение) нам известна и в рассмотренном примере ее просто «угадали». А как найти такую вершину при большом числе переменных? Изюминка симплекс-метода в том, что начальное приближение можно получить с его же помощью при решении вспомогательной задачи или обнаружить в процессе ее решения отсутствие допустимых решений исходной задачи.

В исходной задаче: найти

$$\min \sum_{i=1}^{i=n} C_i x_i$$

при ограничениях: $Ax = b$ и $x_i \geq 0$ ($i = 1, \dots, n$) ничто не мешает сделать все свободные члены системы уравнений (координаты вектора b) неотрицательными путем умножения при необходимости обеих частей соответствующих уравнений на минус 1.

Введем дополнительные переменные $z(z_1, z_2, \dots, z_m)$. Все координаты вектора z неотрицательны, $z_j \geq 0$ ($j = 1, 2, \dots, m$). Вспомогательная задача имеет вид: найти

$$\min \sum_{j=1}^{j=m} z_j$$

при ограничениях: $Ax + z = b$; $x_i \geq 0$ ($i = 1, \dots, n$) и $z_j \geq 0$ ($j = 1, 2, \dots, m$).

В новой задаче переменных стало $n + m$, ограничений-равенств m и ограничений-неравенств $n + m$. Изменилась и целевая функция, теперь она не содержит переменных x_i .

Смысл новой (вспомогательной) задачи состоит в том, что ее решение известно заранее, если существует решение исходной задачи. Это нулевой вектор z , так как меньше нуля сумма неотрицательных величин быть не может. Но если исходная задача не имеет решения, то это означает, что не существует такого вектора $x \geq 0$, который вместе с нулевым вектором z дает допустимую точку вспомогательной задачи. Таким образом, после решения вспомогательной задачи получаем набор из $n + m$ чисел — это n координат вектора x и m координат вектора z . Если все координаты вектора z в этом наборе равны нулю, то исходная задача разрешима и полученный вектор x является ее допустимой точкой, так как при этом выполнены все ограничения исходной задачи.

Более того, в этом наборе координат вектора x окажутся (при отсутствии вырожденности) $n - m$ нулей, так как мы получим вершину ОДР вспомогательной задачи, в которой $n + m$ переменных и, следовательно, $n + m$ активных ограничений, т.е. ограничений, выполняющихся как равенства. В системе уравнений m равенств плюс m нулевых z_j , так что до $n + m$ недостает ровно $n - m$ равенств, т.е. нулевых x_i . Это означает, что в результате решения вспомогательной задачи мы получаем не просто допустимую точку, а допустимую вершину ОДР исходной задачи (начальное приближение) или устанавливаем отсутствие допустимых решений исходной задачи вообще.

Вспомогательная задача характерна тем, что для нее начальное приближение (допустимая вершина ОДР) легко находится. Для этого все переменные x_i надо объявить свободными, а все z_j базисными. Это будет вершина ОДР вспомогательной задачи, так как число активных ограничений в ней равно числу переменных (m уравнений плюс n нулевых x_i). Базисные переменные крайне просто выража-

ются через свободные: $\mathbf{z} = \mathbf{b} - \mathbf{A}\mathbf{x}$. Чтобы точка ($\mathbf{x} = 0$, $\mathbf{z} = \mathbf{b}$) была *допустимой* вершиной нам и потребовалась неотрицательность координат вектора \mathbf{b} . Итак, первый столбец симплексной таблицы для вспомогательной задачи это вектор \mathbf{b} , а остальные — это столбцы матрицы \mathbf{A} с обратным знаком. Слева в таблице (базис) все z_j , наверху все x_i (свободные), а в нижней строке (коэффициенты целевой функции при свободных переменных) нужно записать суммы по столбцам, включая столбец свободных членов, так как во вспомогательной задаче целевая функция — это сумма z_j , выраженная через свободные переменные x_i . В процессе решения вспомогательной задачи переменные z_j становятся свободными (переходят на верх таблицы), а переменные x_i , наоборот, переходят в базис. Если в итоговой симплексной таблице хотя бы одна переменная из z_j «застрянет» в базисе, то исходная задача не разрешима. А если все z_j перейдут наверх таблицы, то нужно вычеркнуть из таблицы все столбцы, содержащие z_j . Наверху таблицы останется $n-m$ координат вектора \mathbf{x} (свободные переменные), а m координат будут базисными. Нижняя строка таблицы должна быть преобразована, так как в исходной задаче своя целевая функция (\mathbf{c}, \mathbf{x}) , которую нужно выразить через свободные переменные. Для этого следует использовать полученные выражения базисных переменных через свободные (строки симплекс-таблицы). В результате получим симплекс-таблицу для исходной задачи, которую далее можно решать симплекс-методом, как описано выше.

1.7. Двойственность в линейном программировании

Задачи линейного программирования обладают особым свойством двойственности, состоящим в том, что для каждой задачи можно по определенным правилам построить другую

задачу линейного программирования и, решив ее, найти экстремум исходной задачи (значение целевой функции).

Если *исходная* (прямая) задача записана в первой основной форме: найти

$$\min \sum_{i=1}^{i=n} c_i x_i \quad \text{при ограничениях: } \mathbf{Ax} = \mathbf{b} \text{ и } x_i \geq 0 \ (i = 1, \dots, n),$$

то *двойственная* задача имеет следующий вид:

$$\text{найти } \max \sum_{i=1}^{i=m} b_i y_i \quad \text{при ограничениях: } \mathbf{A}^T \mathbf{y} \leq \mathbf{c}.$$

Здесь m — число строк матрицы \mathbf{A} , а \mathbf{A}^T — транспонированная матрица \mathbf{A} . Заметим, что здесь ограничения $y_i \geq 0$ ($i = 1, \dots, m$) отсутствуют и двойственная задача представлена во второй основной форме. Вместо \min имеем \max , векторы \mathbf{b} и \mathbf{c} поменялись местами, уравнения стали неравенствами.

Если исходная задача имеет решение, то двойственная тоже имеет решение, и ее максимум численно равен минимуму исходной задачи. Число переменных в двойственной задаче равно числу ограничений-равенств исходной задачи, которое всегда меньше числа переменных исходной задачи. Число ограничений-неравенств двойственной задачи равно числу переменных исходной задачи, но, поскольку ограничения по знаку в двойственной задаче отсутствуют, то суммарно число ограничений в двойственной задаче меньше, чем в исходной.

Если исходная задача записана в виде: найти

$$\min \sum_{i=1}^{i=n} c_i x_i$$

при ограничениях: $\mathbf{Ax} \leq \mathbf{b}$ и $x_i \geq 0$ ($i = 1, \dots, n$),

то двойственная задача имеет вид: найти

$$\max \sum_{i=1}^{i=m} b_i y_i$$

при ограничениях: $\mathbf{A}^T \mathbf{y} \geq \mathbf{c}$ и $y_i \geq 0$ ($i = 1, \dots, m$).

Двойственность взаимна, т.е. полученную задачу можно рассматривать как исходную, а исходную как двойственную к ней. Если ставится цель найти экстремум целевой функции, а точка, в которой он достигается, не нужна, то переход к двойственной задаче в ряде случаев может быть оправдан.

Решив двойственную задачу, мы получаем экстремальное значение целевой функции, но не точку (значения переменных исходной задачи), в которой достигается экстремум.

1.8. Целочисленное линейное программирование

Многие практические задачи оптимизации сводятся к линейному программированию, но при дополнительном условии целочисленности всех или некоторых переменных. Примером может служить поиск оптимального распределения «штучного» ресурса, не допускающего деления на более мелкие части. Это может быть распределение дорогостоящих установок между потребителями с учетом их запросов и соответствующих затрат или планирование подачи транспортных средств и т.п.

Может показаться, что достаточно решить задачу, не учитывая требование целочисленности искомым переменных, а потом просто округлить каждое из полученных нецелых значений до ближайшего целого. Такое соображение обосновывается тем, что на практике сами исходные данные (элементы заданных матриц и векторов) не могут быть и постоянными и известными абсолютно точно, поэтому и решения можно искать приближенные. Это соображение не лишено оснований, но после округления можно получить решения далекие от оптимума, да и просто не удовлетворяющие ограничениям (рис. 9).

При округлении координат точки А, которая является решением задачи без учета требований целочисленности, получаем недопустимые решения. Оптимальной является точка В, а не ближайшие к точке А узлы сетки с целочисленными координатами.

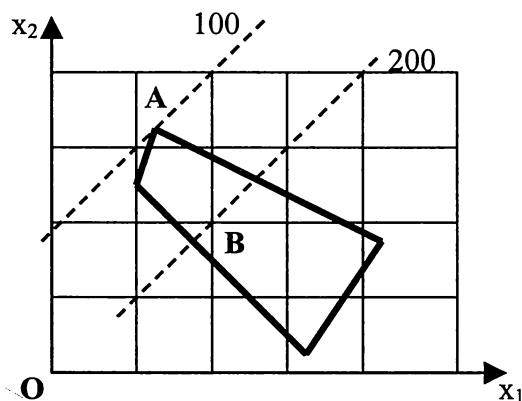


Рис. 9. Некорректность округления решений в задаче целочисленного линейного программирования

На практике часто встречаются задачи целочисленного программирования, в которых вообще переменные могут принимать только два значения: нуль или единица. В этих случаях к округлению результатов, полученных при решении задачи без учета требований целочисленности, прибегать не следует.

Существуют специальные методы решения целочисленных задач линейного программирования, которые позволяют за конечное число шагов получать точное решение (если задача имеет решение в целых числах). Здесь отметим алгоритм Гомори [1, 7, 19], который состоит в решении последовательности линейных задач, причем на каждом шаге после получения оптимальной точки с нецелыми координатами вводится дополнительное ограничение, отсекающее эту точку. Найденная точка оптимума становится недопустимой, и при дополнительном линейном ограничении определяется новая точка оптимума. И так до тех пор, пока очередная точка оптимума не будет удовлетворять всем ограничениям исходной целочисленной задачи.

1.9. Практическое применение линейного программирования

В настоящее время линейное программирование используется в самых различных областях (планирование, управление, военное дело, экономика и т.д.). Для некоторых задач, обладающих специфическими особенностями (например, транспортная задача), разработаны специальные методы решения, более эффективные, чем симплекс-метод [7, 19]. В качестве примера рассмотрим несколько задач, которые сводятся к линейному программированию.

1. *Задача выбора (назначения)*. Имеется n различного рода работ A_1, A_2, \dots, A_n и столько же механизмов (способов) их выполнения B_1, B_2, \dots, B_n . Каждый механизм с известной производительностью может выполнить любую работу, но будет выполнять только одну. Пусть заданы величины c_{ij} , которые являются оценками производительности (эффективности) механизма B_i при выполнении работы A_j . Вектор-строка $c_{i1}, c_{i2}, \dots, c_{in}$ характеризует качество i -го механизма. Естественно стремиться к такой расстановке механизмов, чтобы суммарный эффект (производительность) был максимален. Таким образом, для каждого механизма нужно выбрать работу (или для каждой работы механизм) с наибольшим суммарным эффектом.

Математически это означает следующее. Задана квадратная матрица

$$C = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \dots & \dots & \dots & \dots \\ c_{n1} & c_{n2} & \dots & c_{nn} \end{pmatrix}$$

Из каждой строки матрицы нужно выбрать один элемент так, чтобы из каждого столбца оказался выбранным тоже только один элемент, а сумма выбранных элементов была максимальной.

$$\sum_{k=1}^{k=n} c_{k j_k} \rightarrow \max.$$

При условии $j_k \neq j_m$ при $k \neq m$.

Здесь k — номер строки, а j_k — номер столбца выбранного элемента.

Если бы максимальные элементы в строках были в разных столбцах, то проблемы выбора не было бы. Но обычно это не так, т.е. для нескольких механизмов наиболее предпочтительна одна и та же работа или, иначе, для нескольких работ оказывается наиболее эффективен один и тот же механизм (способ выполнения работы). При больших значениях n задача не может быть решена методом полного перебора вариантов. Однако она успешно решается с помощью линейного программирования.

Введем переменные: $x_{ij} = 1$, если элемент c_{ij} выбран, и $x_{ij} = 0$, если этот элемент не выбран ($i, j = 1, 2, \dots, n$).

Целевая функция примет вид:

$$\sum_i \sum_j c_{ij} x_{ij} \rightarrow \max.$$

При условии $\sum_j x_{ij} = 1$ для всех $i = 1, 2, \dots, n$.

Это условие означает, что каждый механизм используется только на одной работе. Нужно еще условие:

$$\sum_i x_{ij} = 1$$

для всех $j = 1, \dots, n$.

Это условие означает, что каждую работу выполняет только один механизм. И конечно, нужны неравенства $0 \leq x_{ij} \leq 1$. Мы получили задачу целочисленного линейного программирования, в которой число переменных равно n^2 , а число ограничений-равенств равно $2n$ плюс двусторонние ограничения-неравенства на каждую переменную. Характерная особенность данной задачи в том, что о требованиях целочисленности можно забыть, они выполняются автоматически.

Если заданы не производительности механизмов при выполнении работ, а затраты (стоимости), в наших рассуждениях ничего не меняется, но вместо максимума нужен минимум. Наша задача записана в несколько необычной форме: вместо вектора коэффициентов целевой функции у нас матрица c_{ij} и вместо вектора неизвестных — матрица x_{ij} . Это так называемая двухиндексная задача. Конечно, можно упорядочить и коэффициенты и переменные в одномерные массивы, например по строкам, и тем самым перейти к прежней форме записи, но обычно это не делается и задача решается с учетом формы представления данных.

2. *Задача о защите поверхности.* Пусть имеется поверхность, разбитая на A_1, A_2, \dots, A_m — m элементов (или деталей одного изделия), которые нужно покрыть (например, для защиты от коррозии или проникающего излучения) одним из B_1, B_2, \dots, B_n материалов (способов). Один и тот же материал (способ) можно использовать для покрытия многих элементов, но на одном элементе используется только один материал (способ). Заметим, что вариант с использованием комбинации материалов на одном элементе можно рассматривать как новый материал (способ). Будем считать, что задана матрица затрат c_{ij} , в которой m строк и n столбцов. Возможно любое соотношение между m и n .

Задача аналогична предыдущей, но отсутствует условие равенства единице суммы элементов каждого столбца. Решение этой задачи очевидно, если один из способов оказывается наиболее приемлемым среди всех других способов *одновременно для всех* элементов поверхности.

Для разных элементов (деталей) могут оказаться наиболее приемлемыми разные способы защиты. Кроме того, задача может осложняться, если каждый способ не дает абсолютную защиту и с течением времени эксплуатационные качества изделия меняются (например, растет вероятность выхода из строя). Если заданы для каждого элемента при каждом способе его защиты коэффициенты p_{ij} , учитывающие это обстоятельство, то появляется дополнительное ограничение на максимальную величину p_{ij_k} , где j_k — тот способ, который будет выбран для i -го изделия. Это условие $\max p_{ijk} \leq p$, где величина p задана, означает, что в каждой строке матрицы неизвестных x_{ij} *останутся только те, для которых это условие выполнено*, так как с самого начала можно отбросить те способы защиты конкретного элемента, которые не соответствуют этому требованию. Выясняется, что такое дополнительное ограничение только упрощает задачу.

Но возможны и более сложные случаи. Например, ограничен *не максимальный* показатель потерь по каждому элементу, *а сумма* таких показателей. Возникает дополнительная связь между переменными, которая может быть записана в виде:

$$\sum_i \sum_j d_{ij} x_{ij} \leq D_{\max},$$

где D_{\max} и все d_{ij} заданные величины.

Это снова задача целочисленного линейного программирования.

3. *Задача о размещении оборудования.* Предположим, что есть n объектов, которые нужно разместить на некотором транспортном средстве (например, приборы на самолете, на космическом корабле, грузы на товарном поезде, автомобиле и т.д.) Все объекты разместить нельзя из-за ограниченности грузоподъемности, габаритов или других ресурсов. Каждый объект с номером i имеет известный вес, габарит или иной показатель a_{ij} , по каждому из которых и ставится суммарное ограничение b_j . Кроме того, известна оценка полезности каждого объекта. Задача состоит в том, чтобы не превышая имеющиеся ресурсы (грузоподъемность, объем и т.д.), разместить такие объекты, чтобы их суммарная полезность была максимальна. Эта задача также сводится к линейному программированию.

Пусть $x_i = 1$, если i -ый объект будет размещен и 0 в противном случае, $i = 1, 2, \dots, n$. Пусть далее c_i — мера полезности i -го объекта. Задача запишется в следующем виде. Найти:

$$\min \sum_{i=1}^{i=n} c_i x_i \quad \text{при ограничениях:} \quad \sum_{i=1}^{i=n} a_{ij} x_i \leq b_j$$

($j = 1, \dots, m$). Таких неравенств столько, сколько показателей, по которым имеются ограничения. Вместо условия равенства x_i нулю или единице приходится ставить ограничения $0 \leq x_i \leq 1$ ($i = 1, 2, \dots, n$).

Это тоже задача целочисленного линейного программирования, но автоматически целочисленные решения могут и не получаться. Округление получающихся дробей имеет мало смысла не только из-за потери оптимальности, но и из-за нарушения ограничений

1.10. Обучающая программа DANTZIG_2.exe

Программа предназначена для помощи в освоении основных идей линейного программирования. Она дает две

формы записи задачи и объясняет, как от произвольной смешанной системы перейти к каждой из форм. Затем приводится несколько контрольных вопросов на понимание того, как именно осуществляется этот переход и как при этом меняется число переменных и ограничений.

Затем программа переходит к выяснению свойств множества решений системы линейных неравенств, т.е. ОДР.

Программа начинает с вопроса, что является решением одного отдельного неравенства, а затем переходит к системе неравенств. В заключение этого раздела она задает контрольные вопросы, в частности предъявляет различные варианты решений систем линейных неравенств (т.е. варианты допустимых областей) и спрашивает, возможны они или нет. Для иллюстрации различных ситуаций, возникающих при решении систем линейных неравенств, рассматривается двумерная задача, записанная во второй форме. Приводятся конкретные примеры задач и их геометрическая интерпретация (ОДР плюс линии уровня целевой функции). Рассматриваются случаи отсутствия решений, единственного решения и бесконечного множества решений, ограниченной и неограниченной ОДР. При этом приводятся примеры наличия и отсутствия решений при одной и той же неограниченной ОДР, но при различных целевых функциях. Для пояснения приводится следующий пример. Найти

$$\min x + y$$

при

$$x \geq 0; \quad -2x + y \leq 1;$$

$$y \geq 0; \quad 0.5x - y \leq 0.5.$$

Соответствующая ОДР не ограничена (рис. 10), но точка минимума (0, 0) и минимум целевой функции тоже нуль.

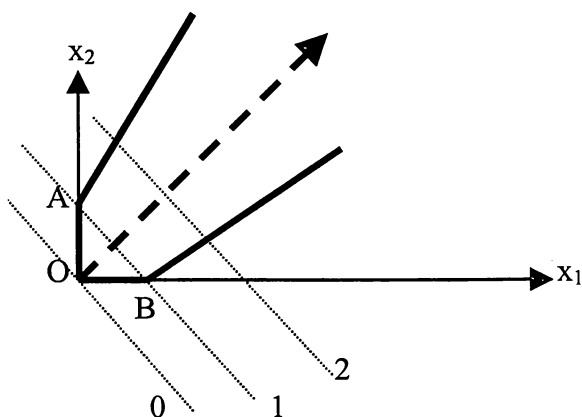


Рис. 10. Пример наличия решения при неограниченной ОДР

В направлении, показанном пунктирной стрелкой, целевая функция неограниченно возрастает. Но если рассматривать ту же задачу, но изменить знак целевой функции, то она в этом направлении неограниченно убывает и решение отсутствует.

Далее программа разъясняет сущность симплекс-метода. Она решает конкретную задачу, начиная с приведения исходной системы к первой основной форме. Затем находит начальное приближение, дает симплексную таблицу, показывает опорный элемент и следующую симплексную таблицу и так до определения точки минимума. Программа не только указывает эту точку и значение целевой функции в ней, но и показывает геометрически соответствие очередной допустимой вершины и симплексной таблицы. Пользователь имеет возможность увидеть весь процесс перехода от одной допустимой вершины к другой и соответствующие изменения целевой функции и симплексной таблицы.

Программа задает контрольные вопросы, касающиеся сущности симплекс-метода и ситуаций, возникающих в линейном программировании. На каждый вопрос предлагается несколько вариантов ответа, причем только один из них программа считает правильным. Пользователь имеет возможность выбрать любой из предложенных вариантов ответа. Если программа сообщает, что выбран неверный ответ, то она дает пользователю возможность выбрать другой вариант ответа. Таким образом, при желании пользователь всегда сможет узнать правильный ответ. Рекомендуется не только дойти до правильных ответов на все вопросы, предлагаемые программой, но и найти объяснение, почему именно этот вариант ответа программа считает правильным.

Задания

1. Взять конкретную задачу линейного программирования, которую рассматривает обучающая программа, и решить ее самостоятельно с помощью симплекс-метода. Затем сопоставить симплексные таблицы с их аналогами из обучающей программы (должно быть совпадение и таблиц и результата).

Рекомендация — в случае несовпадения таблиц или результата не спешить с выводами об ошибке в обучающей программе, а проверить свои расчеты.

2. Построить и решить двойственную задачу к той задаче, которую демонстрирует обучающая программа DANTZIG_2.exe. Сравнить результаты.

3. Решить геометрически и с помощью симплекс-метода задачу. Найти

$$\min(-x_1 - 2x_2 - 2x_3 - x_4)$$

при ограничениях:

$$\begin{cases} x_1 - x_3 + \frac{1}{2}x_4 = 1; \\ x_2 + x_3 - x_4 = 1; \\ x_i \geq 0; (i = 1, 2, 3, 4) \end{cases}$$

приняв свободными переменные x_1 и x_3 и исходную вершину $(0, 3, 0, 2)$. Сравнить с решением примера, приведенным выше (разд. 1.6).

Контрольные вопросы и задачи

1. В смешанной системе ограничений было n переменных, m ограничений-равенств и p ограничений-неравенств. При этом переменные не были ограничены по знаку. Эту систему привели к первой основной форме. Сколько стало переменных и ограничений-равенств?

2. Тот же вопрос, но при условии, что исходную систему привели ко второй основной форме.

3. Доказать, что число возможных вариантов выбора в задаче о расстановке механизмов равно $n!$ и вычислить его при $n = 10$.

4. В симплексной таблице в нижней строке есть коэффициенты целевой функции -2 и -1 . В столбце над -1 нет отрицательных чисел, а в столбце над -2 есть. Как решать эту задачу дальше? Какой столбец должен быть разрешающим?

5. Тот же вопрос, но вместо числа -1 стоит 0 .

6. Возможна неоднозначность при выборе разрешающего столбца симплексной таблицы и опорного элемента при уже выбранном разрешающем столбце. К чему может привести эта неоднозначность?

7. Если в нижней строке симплексной таблицы два отрицательных числа, то возможна ли ситуация, при которой выбор меньшего по абсолютной величине числа и соответствующего опорного элемента дает большее по абсолютной величине изменение целевой функции (при переходе в следующую вершину), чем выбор «самого отрицательного» числа?

8. Если ОДР имеет вырожденные вершины, ни одна из которых не является точкой оптимума, то можно ли при соответствующем выборе разрешающих столбцов в симплексных таблицах избежать попадания в эти вершины?

9. При попадании в вырожденную вершину может ли симплекс-алгоритм «не заметить» этого и покинуть ее, или заикливание неизбежно?

10. Можно ли утверждать, что в задаче целочисленного линейного программирования решение, получаемое без учета требования целочисленности, отклоняется от оптимума по каждой координате не более чем на 2?

2. НЕЛИНЕЙНОЕ ПРОГРАММИРОВАНИЕ

В данном разделе рассматриваются нелинейные задачи оптимизации, т.е. задачи поиска минимума (или максимума), в которых или целевая функция многих переменных, или ограничения или и то и другое одновременно *нелинейны*.

Показаны принципиальные отличия нелинейных задач от линейных и дополнительные трудности их решения.

Излагаются необходимые математические положения, на основе которых рассматриваются методы решения некоторых классов нелинейных задач. Приводятся конкретные примеры и задачи и методы их решения.

Кроме того, приводятся сведения об обучающей программе ROZEN.exe, рекомендации по ее применению, а также практические задачи, решаемые с помощью методов нелинейного программирования.

2.1. Формулировка задач нелинейного программирования и их классификация

В общем случае задача нелинейного программирования может быть записана в следующем виде. Найти

$$\min F(\mathbf{x})$$

при ограничениях:

$$c_j(\mathbf{x}) = 0, j = 1, 2, \dots, m';$$

$$c_j(\mathbf{x}) \leq 0, j = m'+1, m'+2, \dots, m',$$

где $x(x_1, x_2, \dots, x_n)$ — вектор из E_n , $F(x)$ — заданная целевая функция, минимум которой предстоит найти, а $c_j(x)$ — заданные скалярные функции, определяющие ограничения на область поиска. Дополнительно может присутствовать требование целочисленности всех или некоторых координат вектора x .

Эта формулировка универсальна в том смысле, что любая задача математического программирования может быть представлена в таком виде. В частности, при условии линейности всех функций получаем задачу линейного программирования. Поскольку в этой формулировке никаких ограничений на вид или свойства функций задачи $F(x)$ и c_j не накладывается, то разнообразие функций и их свойств предопределяет и разнообразие задач нелинейного программирования. Эти функции могут быть или не быть непрерывными, иметь или не иметь частные производные, быть или не быть выпуклыми, одноэкстремальными и т.д.

Универсального метода — такого как симплекс-метод для линейных задач — для решения нелинейных задач нет. С другой стороны, нет оснований считать каждую задачу уникальной, так как далеко не каждое изменение в формулировке задачи требует пересмотра метода ее решения. Другими словами, можно выделить классы задач и в пределах каждого класса — свои наиболее подходящие методы решения.

Классификация задач проводится по различным признакам.

1. Наличие явных аналитических выражений функций задачи или только возможности их вычисления при конкретных значениях переменных путем решения дополнительных подзадач.
2. Непрерывность функций и их производных (условия гладкости).

3. Наличие локальных экстремумов (многоэкстремальность).
4. Размерность задачи. От нее зависит, сколько памяти и времени вычислений требуется для решения задачи тем или иным методом. Как правило, методы, эффективные для задач с небольшим числом переменных, не пригодны при числе переменных в сотни и тысячи.

Дополнительно часто имеют значение «природа» задачи и внешние факторы. Например, некоторые ограничения должны быть выполнены с высокой точностью, а некоторые могут выполняться с существенной допустимой погрешностью (невязкой). Если результаты решения нужны как второстепенные данные для более общей задачи, то может оказаться бессмысленным поиск решения с максимальной точностью. Наконец, может требоваться только приближенное значение минимума, а сама точка, в которой он достигается, или вообще не нужна, или может быть найдена со значительными отклонениями по некоторым переменным. Подобные ситуации возникают при оптимизации параметров отдельных подсистем в больших системах.

Важное значение имеет конкретный вид функций. Например, система линейных ограничений может иметь структуру или соответствовать матрице, имеющей много нулевых элементов, что позволяет создать наиболее эффективные алгоритмы как в рамках известных методов, так и новые методы оптимизации.

2.1. Дополнительные сведения из линейной алгебры и математического анализа

Выпуклые функции. Функция $F(x)$, определенная на выпуклом множестве S , называется выпуклой, если для

любых $x^1 \in S$ и $x^2 \in S$ выполняется неравенство $F(\lambda x^1 + (1 - \lambda)x^2) \leq \lambda F(x^1) + (1 - \lambda)F(x^2)$ для всех $0 \leq \lambda \leq 1$. Все точки $\lambda x^1 + (1 - \lambda)x^2$ при $0 \leq \lambda \leq 1$ принадлежат S , так как S является выпуклым множеством. В левой части неравенства стоит значение функции в некоторой точке отрезка $[x^1, x^2]$, определяемой λ , а в правой части с тем же λ берется точка на отрезке $[F(x^1), F(x^2)]$. Если в определении выпуклой функции заменить знак \leq на $<$, то получим определение строго выпуклой функции.

С геометрической точки зрения выпуклость функции означает, что между любыми двумя точками на графике функции ее график лежит ниже хорды, соединяющей эти точки (рис. 11).

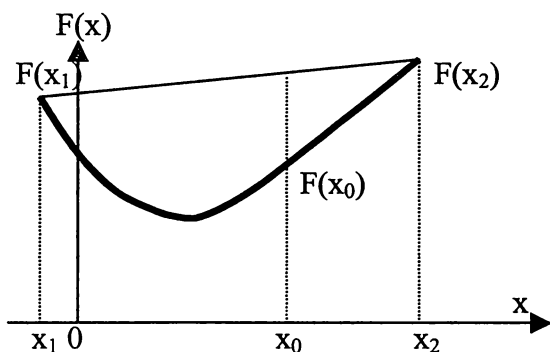


Рис. 11. Выпуклая функция

Если функция имеет производную, то можно сказать, что на отрезке выпуклости хорда лежит выше касательной. Заметим, что функция может быть выпукла не во всей своей области определения, а только на ее подмножестве. Так, функция $\sin(x)$ выпуклая на $[\pi, 2\pi]$, но не на $[0, 2\pi]$.

Выпуклые функции в нелинейном программировании играют особую роль, так как локальный минимум строго выпуклой функции является глобальным минимумом. А значения нестрого выпуклой функции во всех точках минимума равны. Таким образом, достаточно найти одну точку минимума выпуклой функции, так как другой точки с меньшим значением целевой функции не существует.

При этом допустимая область должна быть выпуклым множеством, так как только для таких областей определено понятие выпуклой функции.

Если допустимая область не является выпуклым множеством, то может существовать отличный от глобального локальный минимум даже для линейной целевой функции (рис. 12).

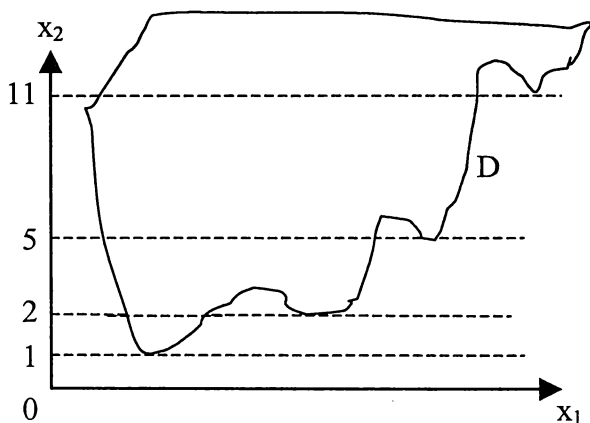


Рис. 12. Глобальный (1) и локальные (2, 5, 11) минимумы

Допустимая область D невыпуклая, и линейная целевая функция $F(x_1, x_2) = x_2$ имеет на ней локальные минимумы.

При линейных ограничениях выпуклость допустимой области гарантирована и наличие или отсутствие локальных

экстремумов зависит только от выпуклости целевой функции. При нелинейных ограничениях, как показано на рис.12, все может быть значительно сложнее. Нелинейные ограничения могут приводить к тому, что допустимая область может быть несвязной и состоять из нескольких областей. Примером может служить система неравенств:

$$\begin{aligned}(x_1 - 1)x_2 &\leq 1, & x_1 &\geq 0, \\ x_1 + x_2 &\geq 4, & x_2 &\geq 0.\end{aligned}$$

В этой системе второе неравенство линейное и определяет полуплоскость вверх от прямой АВ на рис. 13, что совместно с первым неравенством делает недопустимым сегмент, ограниченный пунктиром.

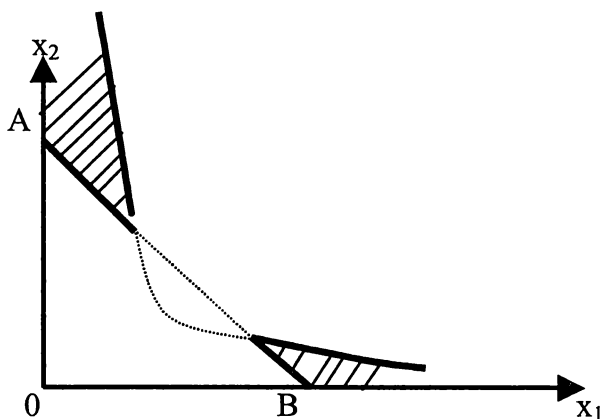


Рис. 13. Несвязная допустимая область при нелинейных ограничениях

Однако не следует думать, что нелинейные ограничения обязательно приводят к «плохим» допустимым областям.

Все решает конкретный вид соответствующих функций, задающих ограничения.

В дальнейшем мы будем рассматривать только достаточно гладкие функции, поэтому отметим еще два свойства выпуклых гладких функций:

1. Для функции одной переменной:

- ♦ если на некотором интервале функция выпукла и дифференцируема, то ее производная на этом интервале не убывает.

Для функции многих переменных:

- ♦ если функция $F(\mathbf{x})$ выпукла и вектор — градиент $\mathbf{g}(\mathbf{x})$ существует, то $F(\mathbf{x} + \mathbf{h}) \geq F(\mathbf{x}) + (\mathbf{g}(\mathbf{x}), \mathbf{h})$, т.е. приращение функции не меньше, чем скалярное произведение градиента на приращение аргумента.

2. Для функции одной переменной:

- ♦ если функция обладает непрерывной второй производной на некотором интервале, то для выпуклости функции на этом интервале необходимо и достаточно, чтобы вторая производная была неотрицательной в каждой точке интервала.

Для функции многих переменных вместо неотрицательности второй производной имеем положительную полуопределенность матрицы вторых частных производных, которая называется матрицей Гессе.

Ряд Тэйлора и квадратичные формы. Из курса математического анализа известно, что равенство нулю первой производной является необходимым, но недостаточным условием экстремума функции одной переменной. Например, в точке $x = 0$ производная $F(x) = x^3$ равна нулю, но это не точка экстремума, а точка перегиба. Достаточное условие экстремума формулируется с использованием второй производной. Для

функции многих переменных аналогичную роль играют матрицы Гессе. Это можно видеть при рассмотрении разложения функции в ряд Тэйлора в точке \mathbf{x}^0 .

$$F(x_1, x_2, \dots, x_n) = F(x_1^0, x_2^0, \dots, x_n^0) + \sum_i \frac{\partial F}{\partial x_i} (x_i - x_i^0) \Big|_{x^0} + \\ + 0,5 \sum_i \sum_j \frac{\partial^2 F}{\partial x_i \partial x_j} \Big|_{x^0} (x_i - x_i^0) (x_j - x_j^0) + o(r^2). \quad (2.1)$$

В этой формуле последнее слагаемое это остаточный член, который имеет более высокий порядок малости, чем квадрат расстояния между точками \mathbf{x} и \mathbf{x}^0 , т.е. $((\mathbf{x} - \mathbf{x}^0), (\mathbf{x} - \mathbf{x}^0))$

Если обозначить вектор-градиент в точке \mathbf{x}^0 через $\mathbf{g}(\mathbf{x}^0)$, а матрицу вторых частных производных в той же точке через $\mathbf{H}(\mathbf{x}^0)$, то эту формулу можно записать с использованием скалярного произведения так:

$$F(\mathbf{x}) = F(\mathbf{x}^0) + (\mathbf{g}(\mathbf{x}^0), (\mathbf{x} - \mathbf{x}^0)) + \\ + 0,5((\mathbf{x} - \mathbf{x}^0), \mathbf{H}(\mathbf{x}^0)(\mathbf{x} - \mathbf{x}^0)) + o(r^2).$$

Матрица Гессе всегда симметрическая, так как при взятии второй частной производной не имеет значение порядок дифференцирования.

Необходимое условие экстремума состоит в равенстве нулю всех первых частных производных, т.е. равенстве нулю градиента $\mathbf{g}(\mathbf{x}^0) = 0$. Иначе второе слагаемое (линейная часть приращения функции) меняет знак при изменении знака приращения аргумента, а в точке экстремума знак приращения функции сохраняется для всех точек в достаточно малой окрестности точки \mathbf{x}^0 . Итак, пусть $\mathbf{g}(\mathbf{x}^0) = 0$. Тогда второй член определяет является ли точка \mathbf{x}^0 точкой локального максимума или минимума или не является ни тем ни другим. Этот второй член разложения функции в ряд Тэйлора представляет собой так называемую квадратичную форму.

Квадратичная форма — это функция многих переменных, которая является суммой слагаемых второй степени относительно независимых переменных $\sum_i \sum_j q_{ij} x_i x_j$.

Всегда можно сделать $q_{ij} = q_{ji}$, не меняя суммы. С использованием скалярного произведения квадратичная форма записывается как:

$$(Q\mathbf{x}, \mathbf{x}).$$

Часто рассматривают при этом еще и слагаемые первой степени, т. е. линейную форму (\mathbf{c}, \mathbf{x}) .

Квадратичные формы играют особую роль в нелинейном программировании, так как часто целевая функция является квадратичной формой «в чистом виде». Для прочих функций их приближенное представление в виде квадратичной формы позволяет построить эффективные алгоритмы поиска экстремума.

Запись с использованием скалярного произведения становится очевидной, если вынести x_i из внутренней суммы.

$$\sum_i \sum_j q_{ij} x_i x_j = \sum_i x_i \sum_j q_{ij} x_j$$

Теперь внутренняя сумма при каждом i — это произведение i -ой строки матрицы Q с элементами q_{ij} на вектор \mathbf{x} , т.е. i -ая компонента вектора $Q\mathbf{x}$, а внешняя сумма дает произведение соответственных компонент векторов \mathbf{x} и $Q\mathbf{x}$, т.е. их скалярное произведение $(Q\mathbf{x}, \mathbf{x})$.

Пример.

$2x_1^2 + 5x_1x_2 + x_2x_1 + x_2^2$ квадратичная форма, матрица которой $Q = \begin{pmatrix} 2 & 3 \\ 3 & 1 \end{pmatrix}$. Вектор $Q\mathbf{x} = \begin{pmatrix} 2x_1 + 3x_2 \\ 3x_1 + x_2 \end{pmatrix}$ и его скалярное произведение на вектор \mathbf{x} равно $(2x_1 + 3x_2)x_1 + (3x_1 + x_2)x_2$. Но это и есть исходное выражение.

Градиент квадратичной формы (Qx, x) это вектор $g = 2Qx$. Для удобства квадратичную форму записывают в виде $1/2(Qx, x)$ и тогда градиент $g = Qx$. При наличии линейных слагаемых квадратичная функция имеет вид $1/2(Qx, x) + (c, x)$. А ее градиент равен $g = Qx + c$.

Квадратичная форма (Qx, x) называется *положительно определенной*, если $(Qx, x) > 0$ при любом $x \neq 0$. Если вместо знака $>$ стоит знак \geq , то квадратичная форма (и соответственно матрица Q) называется *положительно полуопределенной*. Аналогично определяются отрицательно определенные и полуопределенные формы. Если возможны и положительные и отрицательные значения квадратичной формы, то она называется *знаконеопределенной*.

Поведение квадратичной формы определяется собственными числами матрицы Q . Напомним, что любой вектор $f \neq 0$ называется собственным вектором матрицы Q , если выполнено условие $Qf = \lambda f$. Скаляр λ называется собственным числом. Из собственных векторов матрицы Q можно построить ортонормальный базис, к которому можно перейти с помощью линейного преобразования, т.е. осуществить замену переменных. В этом новом базисе квадратичная форма превращается в сумму квадратов новых переменных с коэффициентами, равными собственным числам.

$$(Qx, x) = \sum_i \lambda_i y_i^2. \text{ Здесь } y_i \text{ координаты вектора } x \text{ в ор-}$$

тонормальном базисе, составленном из собственных векторов матрицы Q . Отсюда непосредственно следует, что для положительной определенности матрицы необходима и достаточна положительность ее собственных чисел. В двумерном случае положительно определенной форме соответствует поверхность, которая называется эллиптический параболоид. Линии уровня этой поверхности представляют собой концентрические эллипсы. Если вспомнить каноническое уравнение эллипса $x_1^2/a_1^2 + x_2^2/a_2^2 = 1$, где a_1 и a_2 его полуоси, и переписать выражение для квадратичной формы

в виде $(\mathbf{Q}\mathbf{x}, \mathbf{x}) = \sum y_i^2 / (1/\lambda_i)$, то становится ясно, что

длины полуосей эллипса (в трехмерном случае эллипсоида и т.д.) *обратно пропорциональны квадратным корням из собственных чисел*. Если одно собственное число *много меньше* всех остальных, то ему соответствует полуось, длина которой *много больше* длин других полуосей, т.е. эллипс (эллипсоид) будет иметь сильно вытянутую форму, что имеет существенное значение при решении задач поиска минимума.

Из представления квадратичной формы в виде суммы квадратов следуют и условия экстремума произвольной функции многих переменных, имеющей непрерывные вторые частные производные в окрестности точки \mathbf{x}^0 , в которой градиент $\mathbf{g}(\mathbf{x}^0) = 0$. Такая точка называется стационарной (или критической). Поскольку следующий член разложения функции в ряд Тэйлора (см. формулы 2.1 и 2.2) представляет собой квадратичную форму с матрицей Гессе, то все определяется знаками собственных чисел этой матрицы.

Если матрица Гессе, вычисленная в критической точке \mathbf{x}^0 положительно определенная, то имеет место минимум, если отрицательно определенная — то максимум. Если матрица Гессе *знаконеопределенная*, то это ни минимум, ни максимум, а седловая точка. При *полуопределенности* матрицы Гессе выводов об экстремуме в точке \mathbf{x}^0 без дополнительных исследований сделать нельзя.

Итак, если нет ограничений, то для нахождения минимума надо решить систему уравнений, которая получается приравнованием нулю градиента целевой функции. Если целевая функция квадратичная, то эта система линейна. Среди найденных критических точек могут быть точки минимума, поэтому в каждой из них надо вычислить матрицу Гессе и установить, является ли она положительно или отрицательно определенной, или выполнить другие исследования поведения целевой функции в окрестности каждой критической точки.

При решении практических задач часто бывает известно, что целевая функция не имеет максимумов (или минимумов). Например, если задача состоит в минимизации затрат (финансов, материалов, времени и т.д.) и требуется найти оптимальное решение (план, проект, конструкцию, распределение ресурсов и т.д.), то максимального значения или вообще не существует, или очевидно, что найденная критическая точка не является точкой максимума. Аналогично в задачах на максимум (прибыли, производительности и т.д.) критические точки, как правило, не могут быть точками минимума. Более того, часто (к сожалению, не всегда) целевые функции обладают свойством выпуклости, так что не может быть и седловых точек, локальный минимум является глобальным и найденная критическая точка дополнительно не анализируется. Вторые частные производные или не вычисляются вообще, или используются для поиска точки оптимума, но не для анализа критических точек. Сказанное не означает, что в практических задачах минимизации опасность остановки процесса поиска в седловых точках или в точках локального минимума полностью исключена. Речь идет о том, что знание практического смысла задачи и свойств ее математической модели часто помогает установить без дополнительных вычислений, чем на самом деле является точка, в которой градиент равен нулю.

В задачах с большим числом неизвестных приравнение нулю частных производных приводит к системам большого числа уравнений, решение которых само по себе является вычислительной проблемой. Обычно решение задачи получается в результате итерационного процесса, в котором многократно вычисляются значения функции, градиента и матрицы Гессе. В этой связи методы оптимизации подразделяются на методы *нулевого порядка* — в них используются *только* значения целевой функции, методы *первого порядка*, в которых используются значения градиента, и методы *второго порядка*, в которых кроме градиента используется и матрица Гессе.

При наличии ограничений точка безусловного минимума, получаемая приравнением нулю градиента, может оказаться и, как правило, оказывается недопустимой. Поэтому вместо градиента используются его ортогональные проекции на различные подпространства.

Проекция вектора на подпространство. Пусть в евклидовом пространстве E_n задано подпространство M и вектор f , не принадлежащий M . Вектор p называется проекцией вектора f на подпространство M , если выполняются два условия:

1. Вектор p принадлежит M ,
2. Вектор $f - p$ ортогонален *любому* вектору из M , т.е. является нормалью к M .

Тем самым речь идет о разложении вектора на проекцию и нормаль.

Пример.

В E_2 задана прямая $x_2 = 2x_1$ и вектор f с координатами $(1, 3)$ (рис. 14). Найти проекцию f на M .

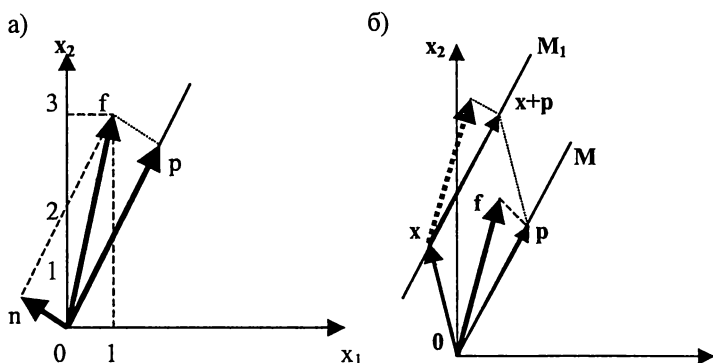


Рис. 14. Проекция на подпространство (а)
и линейное многообразие (б)

Координаты неизвестного вектора \mathbf{p} обозначим p_1 и p_2 . Тогда из первого условия следует, что $p_2 = 2p_1$, а из второго, что вектор с координатами $1 - p_1$ и $3 - p_2$ ортогонален вектору с координатами $(1, 2)$, который принадлежит нашему подпространству, т.е. прямой $x_2 = 2x_1$. Условие равенства нулю скалярного произведения этих векторов дает $1 - p_1 + 2(3 - 2p_1) = 0$. Отсюда $p_1 = 1.4$, и $p_2 = 2.8$. А нормаль \mathbf{n} имеет координаты -0.4 и 0.2 .

Если \mathbf{x} принадлежит не подпространству \mathbf{M} , а линейному многообразию \mathbf{M}_1 , полученному смещением \mathbf{M} на некоторый вектор (рис. 14б), и \mathbf{p} проекция заданного вектора \mathbf{f} на подпространство \mathbf{M} , то вектор $\mathbf{x} + \mathbf{p}$ и вообще луч $\mathbf{x} + \lambda \mathbf{p}$ принадлежат \mathbf{M}_1 . В этом смысле часто говорят о проекции на многообразие. Далее при рассмотрении методов решения задач с линейными ограничениями мы будем иметь дело с граничными линейными многообразиями \mathbf{M}_1 , которые соответствуют множеству решений системы линейных уравнений $\mathbf{Ax} = \mathbf{b}$, в которой число уравнений m меньше числа неизвестных n . Это ребра, грани и т.д. многомерного многогранника, которым является ОДР.

В двумерном случае это одно уравнение с двумя неизвестными $a_{11}x_1 + a_{12}x_2 = b_1$, которому соответствует прямая, а в многомерном случае каждая такая система определяет линейное многообразие, размерность которого равна $n - m$. Нам потребуются направления (лучи) из точки \mathbf{x} , принадлежащей \mathbf{M}_1 , в направлении проекции заданного вектора \mathbf{f} на подпространство \mathbf{M} , т.е. на множество точек \mathbf{x} , удовлетворяющих системе $\mathbf{Ax} = \mathbf{0}$, а не $\mathbf{Ax} = \mathbf{b}$ (рис. 14а).

Рассмотрим вопрос, как построить проекцию \mathbf{p} заданного вектора \mathbf{f} на подпространство $\mathbf{Ax} = \mathbf{0}$. Такое подпространство называется *нуль-пространство матрицы A*.

По определению проекции (пункт 1) $\mathbf{Ap} = \mathbf{0}$ и вектор $\mathbf{f} - \mathbf{p}$ ортогонален любому вектору, принадлежащему подпространству (пункт 2). Левую часть каждого из уравнений

системы $Ax = 0$ можно рассматривать как скалярное произведение соответствующей строки матрицы A (если эту строку рассматривать как вектор-столбец, т.е. транспонировать) на вектор x , и это скалярное произведение равно нулю. Значит, каждая строка матрицы A , после транспонирования дает вектор, ортогональный любому вектору x подпространства $Ax = 0$, как и вектор $f - p$. Это означает, что вектор $f - p$ можно представить как линейную комбинацию векторов, образованных из строк матрицы A транспонированием. Другими словами, мы рассматриваем не только подпространство M , определяемое системой $Ax = 0$, (его размерность $n - m$), но и подпространство, в котором базис составляют нормали, т.е. столбцы матрицы A^T (его размерность m и оно является ортогональным дополнением к M).

Если из коэффициентов разложения $f - p$ по базису, состоящему из нормалей, составить вектор u (он имеет m компонент), то можно записать

$$A^T u = f - p. \quad (2.3)$$

Здесь A^T — это транспонированная матрица A (см. раздел 1, формулу. 1.1) Полученное равенство умножаем на матрицу A слева. Получаем

$$AA^T u = A(f - p).$$

Отсюда, используя $Ap = 0$, имеем

$$u = (AA^T)^{-1} Af. \quad (2.4)$$

Определив u , найдем и интересующую нас проекцию p , подставляя u из (2.4) в формулу (2.3).

$$A^T (AA^T)^{-1} Af = f - p.$$

И наконец,

$$\mathbf{p} = \left(\mathbf{E} - \mathbf{A}^T (\mathbf{A} \mathbf{A}^T)^{-1} \mathbf{A} \right) \mathbf{f}. \quad (2.5)$$

Здесь \mathbf{E} — единичная матрица ($n \times n$). Полученная формула может быть переписана в виде $\mathbf{p} = \mathbf{P} \mathbf{f}$, где $\mathbf{P} = \mathbf{E} - \mathbf{A}^T (\mathbf{A} \mathbf{A}^T)^{-1} \mathbf{A}$ называется матрицей проектирования. Итак, чтобы построить проекцию произвольного вектора на нуль-пространство матрицы \mathbf{A} , нужно вычислить матрицу проектирования. В общем случае это требует вычисления обратной матрицы (решения системы линейных уравнений) размера $m \times m$. Отметим, что кроме проекции мы получили и коэффициенты разложения нормали $\mathbf{f} - \mathbf{p}$ по базису, составленному из строк \mathbf{A} (формула 2.4). Эти результаты впервые получил Розен (Rozen J.B.) в 1960 г. [1, 5]. Мы будем использовать их в дальнейшем при решении задач с ограничениями. Формула (2.5) устрасяет объемом вычислений, необходимых для построения проекции. Однако дело значительно упрощается при наличии структурных особенностей матрицы \mathbf{A} , что позволяет в ряде случаев построить проекцию без операций умножения и обращения матриц.

2.3. Методы безусловной оптимизации

Это методы поиска минимума функции многих переменных при отсутствии ограничений. Есть по меньшей мере три причины для их изучения.

1. Важно освоить итерационные методы решения задач без ограничений, прежде чем перейти к методам решения более сложных задач с ограничениями, так как эти методы имеют много общего.

2. Задачи оптимизации с ограничениями часто решаются путем преобразования к задачам без ограничений.
3. Некоторые практически важные задачи, например решение систем большого числа уравнений, сводятся к задачам безусловной оптимизации.

Итерационные методы минимизации функции $F(x)$ состоят в построении последовательности векторов, т.е. точек x^0, x^1, \dots, x^k , таких что $F(x^0) > F(x^1) > \dots > F(x^k) > \dots$. Любой такой метод называется методом *спуска*. Естественно, должна быть обеспечена сходимость получаемой последовательности точек к точке минимума, т.е. рассматриваются методы, позволяющие получить точку минимума за конечное число шагов или приблизиться к ней достаточно близко при соответствующем числе шагов. К сожалению, здесь нельзя употребить слова «сколь угодно близко при неограниченном увеличении числа шагов». Дело в том, что *теоретически* все сходящиеся методы этим свойством обладают, но *практически* близость к минимуму в задачах большой размерности ограничивается ошибками вычислений, которые обусловлены погрешностью представления чисел в компьютере. Поэтому необходимо вести вычисления с удвоенной точностью, выбирая соответствующие типы данных при разработке конкретных компьютерных программ.

Для построения итерационной последовательности необходимо выбрать начальное приближение x^0 . В задачах с ограничениями это должна быть допустимая точка, а в задачах без ограничений — теоретически любая точка. Однако целесообразно использовать всю имеющуюся информацию о поведении целевой функции $F(x)$, чтобы выбрать x^0 поближе к точке минимума.

После того как начальное приближение получено, прежде чем перейти к следующей точке, нужно принять два решения:

1. Выбрать направление, по которому пойдём из x^0 в точку с меньшим значением целевой функции (направление спуска);
2. Определить величину шага по направлению спуска.

При любом методе спуска итерационная последовательность точек $\{\mathbf{x}^k\}$ может быть записана в виде:

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \lambda_k \mathbf{p}_k, (k = 0, 1, \dots),$$

где вектор \mathbf{p}^k — это направление, а $\lambda_k \|\mathbf{p}^k\|$ — величина шага. Если $\|\mathbf{p}^k\| = 1$, т.е. длина вектора спуска равна единице, то величина шага равна λ_k . Отметим, что иногда λ_k называют шагом независимо от длины вектора \mathbf{p}^k [12]. Здесь $\|\mathbf{p}^k\| = \sqrt{(\mathbf{p}^k, \mathbf{p}^k)}$ — длина вектора \mathbf{p}^k .

Методы спуска отличаются процедурами построения вектора \mathbf{p}^k и вычисления λ_k . При этом могут использоваться одна (последняя) или две последние (или более) из уже полученных точек.

Для задач безусловной минимизации любое направление является *возможным* (никакие ограничения не мешают), но далеко не все направления приемлемы. Нас могут интересовать только направления, обеспечивающие убывание целевой функции, хотя бы при достаточно малом шаге. Предполагая непрерывность первых частных производных целевой функции и используя ее разложение в ряд Тэйлора в произвольной точке \mathbf{x} , получим $F(\mathbf{x} + \lambda \mathbf{p}) \approx F(\mathbf{x}) + \lambda(\mathbf{g}, \mathbf{p})$. Здесь \mathbf{g} — градиент функции, вычисленный в точке \mathbf{x} . Отсюда следует, что приращение функции $F(\mathbf{x} + \lambda \mathbf{p}) - F(\mathbf{x}) < 0$ при *отрицательном* скалярном произведении (\mathbf{g}, \mathbf{p}) . Итак, *направление спуска должно составлять острый угол с антиградиентом*. Этот вывод справедлив и для задач с ограничениями, но там еще дополнительно требуется, чтобы при достаточно малом шаге не нарушалось ни одно из ограничений.

Все такие направления называются *подходящими* (приемлемыми). Методы спуска, позволяющие получать последовательно подходящие направления, Зойтендейк (Zoutendijk G.) назвал *методами возможных направлений* [6].

Это широкий класс методов, многие из которых имеют собственные названия. Это могут быть методы нулевого порядка (прямые методы), методы первого порядка (градиентные методы) и методы второго порядка, в которых используются вторые частные производные целевой функции. Производные могут вычисляться аналитически или численно через конечные разности значений функции в последовательно вычисляемых точках. Все зависит от конкретной задачи и той информации, которая доступна для использования при оптимизации.

Вообще говоря, методы второго порядка при наименьшем числе шагов приводят к точкам, достаточно близким к точкам минимума. Однако это не означает, что они являются наиболее эффективными с точки зрения затрат машинного времени. Целевая функция может быть такой сложной, что вычисление ее вторых частных производных требует слишком много времени и оказывается лучше сделать несколько малых шагов (в смысле уменьшения целевой функции), но при малом объеме вычислений, чем один большой шаг при большом объеме вычислений.

При большой размерности задач имеет значение и объем требуемой оперативной памяти. До недавнего времени это требование было очень важным, так как работа с матрицей Гессе размером 1000×1000 была невозможна вообще. Существенное значение имеет и тот факт, что различные методы обладают и различной чувствительностью к влиянию ошибок округления чисел в компьютере.

Таким образом, невозможно выделить какой-то один метод, пригодный в любом случае. Для отыскания метода, наиболее пригодного для оптимизации конкретной функции, могут оказаться необходимыми рациональный подход, опыт, а может быть, и предварительные исследования.

Методы первого и второго порядка вырабатывают очередную итерационную точку по следующему правилу:

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \lambda^k \mathbf{A}^k \mathbf{g}^k. \quad (2.6)$$

Другими словами, из очередной точки идем по антиградиенту $-\mathbf{g}^k$ (в этом случае матрица \mathbf{A}_k равна единичной на всех итерациях) или преобразуем антиградиент с помощью некоторой матрицы \mathbf{A}_k , которая может быть постоянной или вычисляться по ранее полученным точкам и градиентам.

2.3.1. Градиентные методы

Градиентным методом называется метод, по которому на каждом шаге очередная точка определяется по формуле

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \lambda_k \mathbf{g}^k, \quad (2.7)$$

т.е. направление спуска на каждой итерации - это антиградиент, вычисленный в текущей точке \mathbf{x}^k .

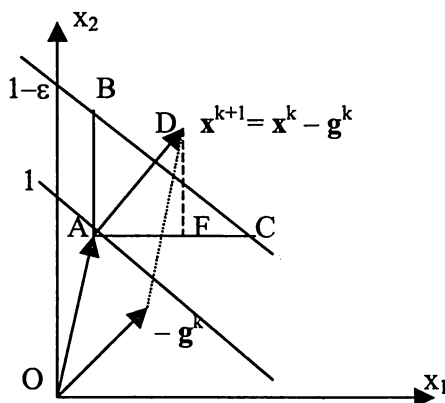


Рис. 15. Линии уровня и антиградиент

На рис. 15 текущему вектору \mathbf{x}^k соответствует точка А. Примем за единицу значение целевой функции в этой точке и рассмотрим линию уровня со значением $1 - \varepsilon$, где ε достаточно малая величина. При достаточно малом ε эти линии уровня будем считать параллельными. Переход на линию уровня с меньшим значением при изменении только x_1 дает точку С, а при изменении только x_2 точку В. Обозначим расстояние АС через Δ_1 , а АВ через Δ_2 . Тогда для частных производных имеем *приближенные* равенства: $\partial F / \partial x_1 = -\varepsilon / \Delta_1$ и $\partial F / \partial x_2 = -\varepsilon / \Delta_2$.

Далее, пусть точка D соответствует вектору $\mathbf{x}^{k+1} = \mathbf{x}^k - \mathbf{g}^k$. Вектор \overrightarrow{AD} имеет те же длину и направление, что и антиградиент $-\mathbf{g}^k$, координаты которого и есть частные производные (со знаком минус). Их отношение равно Δ_2 / Δ_1 . Но $AB / AC = \Delta_2 / \Delta_1$. Значит, треугольники ABC и ADF подобны и *антиградиент ортогонален линии уровня*. Он показывает направление *наискорейшего убывания* функции в данной точке. При движении из точки А во всех прочих направлениях до пересечения линии уровня, которой соответствует уменьшение целевой функции на ε , пришлось бы сделать больший шаг. *Градиент меняется при переходе из точки в точку, следовательно, меняется и направление наискорейшего убывания функции*. Антиградиентное направление не является наилучшим из возможных направлений. Всегда существует направление из текущей точки непосредственно в точку минимума, но мы его не знаем.

Итак, мы нашли одно из подходящих направлений. Это антиградиент. Какой шаг нужно сделать в этом направлении? Другими словами, как определить величину λ_k в формуле (2.7)? Можно без дополнительных вычислений взять для λ_k некоторое относительно малое значение и, используя формулу (2.7), перейти в новую точку \mathbf{x}^{k+1} . Если сделан слишком малый шаг, то практически в том же направлении придется делать еще один или несколько шагов. Если шаг

большой, то вместо уменьшения можно получить увеличение целевой функции, если пройден ее минимум в направлении антиградиента (рис. 16).

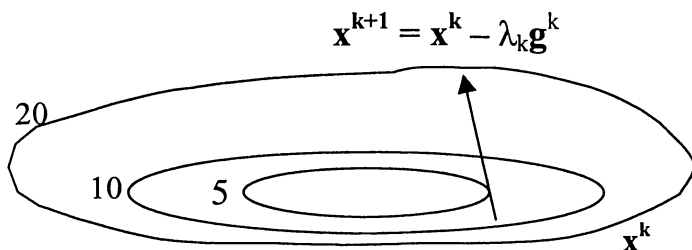


Рис. 16. Пример неправильного выбора шага

Во избежание подобных неприятностей лучше в направлении антиградиента идти *до точки минимума*. В двумерном случае это соответствует касанию некоторой линии уровня. При выбранном направлении для выбора шага остается *только один* неизвестный параметр λ_k . При движении по выбранному направлению значение целевой функции полностью определяется этим параметром и возникает задача поиска такого λ_k , при котором значение функции минимально. Это задача *одномерной* оптимизации.

Если целевая функция $K(x)$ является квадратической

$$K = \frac{1}{2}(Qx, x) + (c, x),$$

то для оптимального шага можно получить конкретную расчетную формулу. Опуская номер итерации k и используя свойства скалярного произведения, запишем функцию, минимум которой по λ нам предстоит найти:

$$\begin{aligned}
K &= \frac{1}{2} (Q(x - \lambda g), (x - \lambda g)) + (c, (x - \lambda g)) = \\
&= \frac{1}{2} (Qx, x) - \lambda (Qx, g) + \frac{1}{2} \lambda^2 (Qg, g) + (c, x) - \lambda (c, g) = \\
&= \frac{1}{2} (Qx, x) - \lambda (Qx + c, g) + \frac{1}{2} \lambda^2 (Qg, g) + (c, x) = \\
&= \frac{1}{2} (Qx, x) - \lambda (g, g) + \frac{1}{2} \lambda^2 (Qg, g) + (c, x).
\end{aligned}$$

Получили квадратный трехчлен относительно неизвестной λ . В преобразованиях использовано свойство симметрических матриц и скалярного произведения, а именно $(Qx, g) = (Qg, x)$, а также формула для градиента квадратичной функции $g = Qx + c$. В полученном выражении неизвестна только λ . Приравнявая нулю производную по λ , находим оптимальное значение λ_{opt} и, следовательно, шаг до точки минимума целевой функции в направлении $-g$.

$$\lambda_{opt} = \frac{(g, g)}{(Qg, g)} \quad (2.8)$$

Если целевая функция не квадратическая, то формулу для вычисления шага до точки минимума по антиградиенту удастся получить далеко не всегда и приходится решать задачу одномерной оптимизации каким-либо численным методом. Далее этот вопрос рассмотрен отдельно. Конечно, минимум по направлению не обязан быть минимумом целевой функции вообще, если наше направление не указывает на точку минимума.

Градиентный метод, в котором на каждой итерации используется шаг до точки минимума в направлении антиградиента, называется *методом наискорейшего спуска*.

Название наискорейший спуск не должно вводить в заблуждение, так как оно вовсе не означает, что метод позволяет найти минимум за наименьшее число шагов по сравнению с другими методами.

Траектория спуска в этом методе носит зигзагообразный характер и градиенты в любых двух последовательных точках ортогональны (рис. 17).

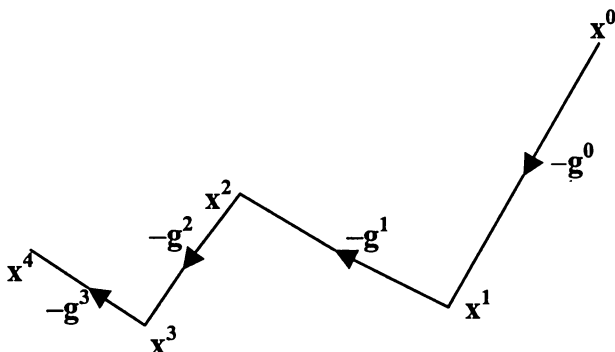


Рис. 17. Зигзагообразная траектория наискорейшего спуска

Ортогональность градиентов обусловлена тем, что в направлении антиградиента мы идем до точки минимума. Если бы новый антиградиент, вычисленный в этой точке, не был ортогонален направлению спуска (старому антиградиенту), то движение можно было бы продолжить или пойти в обратном направлении. Как следует из разложения функции в ряд Тэйлора, линейная часть приращения функции $F(x)$ при смещении на $\lambda \mathbf{p}$, где \mathbf{p} произвольный вектор, равна $\Delta F(x) = \lambda(\mathbf{g}, \mathbf{p})$, где \mathbf{g} — градиент в данной точке. Рассматривая любую точку, в которой мы могли остановиться при движении из точки x^k по антиградиенту ($\mathbf{p} = -\mathbf{g}^k$), приходим к выводу, что, если линейная часть приращения функции

равна нулю, то градиент \mathbf{g} , вычисленный в этой точке, ортогонален направлению \mathbf{p} . Но в точке минимума по направлению $-\mathbf{g}^k$ линейная часть приращения гладкой функции равна нулю, следовательно, антиградиент в этой точке ортогонален направлению спуска, т.е. старому антиградиенту $-\mathbf{g}^k$. Соответствующая ситуация для двумерного случая представлена на рис. 18.

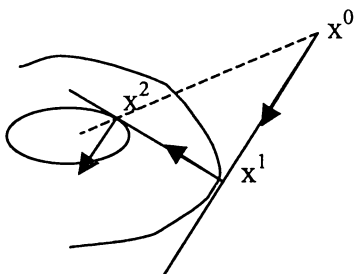


Рис. 18. Наискорейший спуск в двумерном случае

Если линии уровня сильно вытянуты в каком-то направлении, то по нормали к этому направлению целевая функция меняется значительно быстрее, чем вдоль направления. Такой характер целевой функции называется *овражным*. Дно оврага к тому же может быть искривлено (рис. 19). При наличии сильной овражности траектория наискорейшего спуска совершает зигзаги, шаги становятся все меньше и меньше и процесс практически останавливается вдали от точки минимума \mathbf{x}^* из-за малых шагов. Так, в примере на рис. 19, идя по антиградиенту из точки \mathbf{x}^0 , мы «проскакиваем» линию дна оврага и оказываемся в точке \mathbf{x}^1 , которая лежит на его противоположном склоне. Далее перемещаемся зигзагами с одного склона на другой, вместо того чтобы идти по дну оврага.

В многомерном случае возможно и наличие многомерных оврагов.

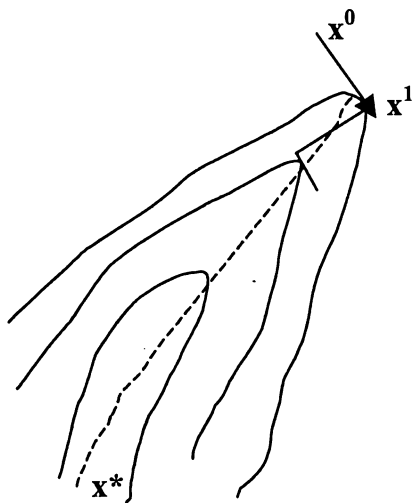


Рис. 19. Отсутствие сходимости к точке минимума в овраге

Если целевая функция представляет собой квадратичную положительно определенную форму, то в двумерном случае линии уровня — это концентрические эллипсы. Однако и здесь возможна овражная ситуация, если собственные числа матрицы квадратичной формы сильно отличаются друг от друга, т.е. мало отношение минимального собственного числа λ_{\min} к максимальному λ_{\max} . В теории квадратического программирования доказывается, что для метода наискорейшего спуска справедливо неравенство [1]:

$$\frac{F(x^{k+1})}{F(x^k)} \leq \frac{(\lambda_{\min}/\lambda_{\max} - 1)^2}{(\lambda_{\min}/\lambda_{\max} + 1)^2}. \quad (2.9)$$

Отсюда видно, что при очень малом отношении $\lambda_{\min}/\lambda_{\max}$ отношение значений целевой функции на двух последовательных итерациях близко к единице, т.е. возможна очень медленная сходимость метода наискорейшего спуска даже для квадратичных функций.

Критерием прекращения процесса вычислений является равенство нулю всех компонент градиента. Поскольку точного равенства нулю достичь не удастся, приходится ограничиваться условием малости суммы квадратов компонент градиента, т.е. $(g, g) < \varepsilon$, где ε — заданная точность расчетов. Если задать слишком малое значение ε , то при овражном характере целевой функции можно и не дожидаться окончания счета (это так называемое алгоритмическое заикливание). Поэтому нужно проверять и условие малости шага и прерывать итерационный процесс, как только шаги (или изменение функции) станут достаточно малыми.

2.3.2. Метод параллельных касательных

В двумерном случае (см. рис. 17) из ортогональности последовательных градиентов следует параллельность касательных к линиям уровня в точках, взятых через одну. Если линии уровня — концентрические эллипсы, то линия, соединяющая две точки, в которых касательные параллельны, проходит через центр. Это означает, что в двумерном случае при минимизации квадратичной формы из точки x^2 надо идти не по антиградиенту, а по направлению их x^0 в x^2 . Следующая точка оказывается последней, так как она является точкой минимума. Эта же идея ускорения сходимости алгоритма наискорейшего спуска может использоваться и в многомерном случае и не только для квадратичных форм. Итак, алгоритм состоит в следующем:

1. Из точки \mathbf{x}^0 делаем два шага как в методе наискорейшего спуска (рис. 17) и получаем точку \mathbf{x}^2 .
2. Из точки \mathbf{x}^2 идем не по антиградиенту, а по направлению $\mathbf{x}^2 - \mathbf{x}^0$ (рис. 18).
3. Находим \mathbf{x}^3 как точку минимума функции в этом направлении и вычисляем антиградиент в ней.
4. Проверяем условия окончания счета и, если они не выполнены, повторяем пункт 1, используя \mathbf{x}^3 вместо \mathbf{x}^0 .

В методе наискорейшего спуска информация об уже пройденных точках не хранилась и никак не использовалась. А в методе параллельных касательных надо запоминать пройденные точки, так как они используются для выбора направления спуска. Эта идея использования информации, полученной в процессе поиска минимума, для ускорения сходимости лежит в основе многих методов оптимизации.

2.3.3. Метод сопряженных градиентов

Идея этого метода в том, чтобы на каждом шаге в качестве направления спуска использовать не антиградиент, а его линейную комбинацию с прежним направлением спуска. Если обозначить через \mathbf{p}^k направление спуска на k -ой итерации, то последовательность векторов \mathbf{p}^k строится следующим образом:

1. $\mathbf{p}^0 = -\mathbf{g}^0$, т.е. первый шаг делаем по антиградиенту,
2. $\mathbf{p}^{k+1} = -\mathbf{g}^{k+1} + \beta_k \mathbf{p}^k$, где $\beta_k = (\mathbf{g}^{k+1}, \mathbf{g}^{k+1}) / (\mathbf{g}^k, \mathbf{g}^k)$.

Это означает, что

$$\beta_0 = (\mathbf{g}^1, \mathbf{g}^1) / (\mathbf{g}^0, \mathbf{g}^0).$$

Другими словами, сделав из начальной точки один шаг по методу наискорейшего спуска, надо вычислить антиградиент в новой точке. Затем взять отношение квадратов длин нового и старого градиентов (это и есть β_0), умножить старый антиградиент на это число и результат сложить с новым антиградиентом. Это и будет направление спуска $\mathbf{p}^1 = -\mathbf{g}^1 - \beta_0 \mathbf{g}^0$, которое мы будем использовать вместо антиградиента $-\mathbf{g}^1$. В направлении \mathbf{p}^1 нужно дойти до точки минимума, в ней вычислить антиградиент $-\mathbf{g}^2$, затем $\beta_1 = (\mathbf{g}^2, \mathbf{g}^2) / (\mathbf{g}^1, \mathbf{g}^1)$ и $\mathbf{p}^2 = -\mathbf{g}^2 + \beta_1 \mathbf{p}^1$ и т.д. Заметим, что при вычислении β_k используются только градиенты, а для вычисления нового направления нужно помнить и использовать старое направление, а не старый антиградиент. В методе сопряженных градиентов новое и старое направления неортогональны. В случае минимизации положительно определенной квадратичной формы с матрицей \mathbf{Q} все направления спуска \mathbf{p}^i оказываются \mathbf{Q} ортогональными, т.е. удовлетворяют условию $(\mathbf{p}^i, \mathbf{Q}\mathbf{p}^j) = 0$ при любом $i \neq j$. Такие векторы называются сопряженными, из чего и происходит название метода. Этот метод обладает замечательным свойством: если целевая функция от n переменных представляет собой квадратичную форму, то алгоритм сопряженных градиентов дает точку минимума не более чем за n шагов. Это доказано в теории квадратического программирования [1, 5, 8].

Казалось бы, что при использовании данного метода сложнее определить величину $\lambda_{\text{орт}}$, которая нужна при поиске шага до точки минимума в направлении, которое теперь не совпадает с антиградиентным. Однако это не так. При минимизации квадратичной формы с матрицей \mathbf{Q} , независимо от того как получено направление спуска \mathbf{p} , для $\lambda_{\text{орт}}$ справедлива формула, аналогичная формуле (2.8) для метода наискорейшего спуска.

$$\lambda_{opt} = -\frac{(\mathbf{p}, \mathbf{g})}{(\mathbf{Qp}, \mathbf{p})}. \quad (2.10)$$

Появление знака минус объясняется тем, что \mathbf{g} — это градиент, а не антиградиент. Эту формулу приходится применять на каждой итерации. Алгоритм сопряженных градиентов аналогичен алгоритму наискорейшего спуска. Отличие только в выборе направления. Но именно это позволяет, по крайней мере теоретически, найти решение квадратичной задачи за *конечное* число шагов, равное числу переменных. Практически же в овражных случаях из-за неточности вычисления шага, обусловленной машинной погрешностью представления чисел, требуется несколько больше шагов. Во избежание накопления ошибок вычислений рекомендуется после n шагов (если решение не получено ранее) сделать обновление, т.е. начать снова с антиградиентного направления и далее по сопряженным градиентам. Легко видеть, что этот метод требует несколько больше оперативной памяти, чем метод наискорейшего спуска, но его сходимость существенно лучше, особенно для квадратичных задач.

При решении не квадратичных задач безусловной оптимизации также можно использовать метод сопряженных градиентов. Но при этом, как и в методе наискорейшего спуска, для поиска λ_{opt} нужно использовать какой-либо численный метод. Естественно, что в общем случае метод перестает быть конечным. Сходимость метода в общем случае зависит от свойств матриц Гессе минимизируемой функции. Для квадратичной формы матрица Гессе постоянна. Если матрица Гессе минимизируемой функции положительно определенная и меняется медленно (вторые частные производные почти постоянны или, что то же самое, третьи производные почти нули), то метод сопряженных градиен-

тов можно успешно применять и для минимизации таких функций.

В любом случае трудно представить задачу, где этот метод будет работать хуже, чем наискорейший спуск.

2.3.4. Метод покоординатного спуска

Рассмотрим еще один простой метод безусловной оптимизации, идея которого состоит в том, чтобы свести оптимизацию в многомерном пространстве к многократно повторяемой одномерной оптимизации. Метод состоит в следующем:

1. В начальной точке x^0 фиксируем все координаты, кроме x_1 .
2. Определяем такое значение x_1 , при котором целевая функция достигает минимума. Это одномерная задача, так как все переменные, кроме x_1 , фиксированы.
3. Фиксируем найденную координату x_1 и все остальные, кроме x_2 .
4. Определяем x_2 из условия минимума целевой функции и т.д.? пока не переберем все координаты.
5. Проверяем условия окончания счета и, если они не выполнены, возвращаемся к пункту 1, приняв полученную точку за x^0 . В качестве условия прекращения счета можно взять малое изменение функции после поочередного изменения всех координат или малые изменения *всех координат*.

Геометрический смысл метода состоит в движении в направлениях, параллельных координатным осям (рис. 20).

При движении вдоль координатной оси идем до точки минимума, т.е. касания некоторой линии (в многомерном случае поверхности) уровня. Ясно, что число шагов зависит от начального приближения и, самое главное, от *ориентации* линий уровня относительно координатных осей. Так,

в задаче минимизации квадратичной формы при ориентации осей эллипсов (в двумерном случае) вдоль координатных осей достаточно однократного изменения всех координат. В овражных случаях при произвольной ориентации оврага относительно координатных осей даже в квадратичной задаче метод работает плохо, тем более в общем случае при искривленном дне оврага.

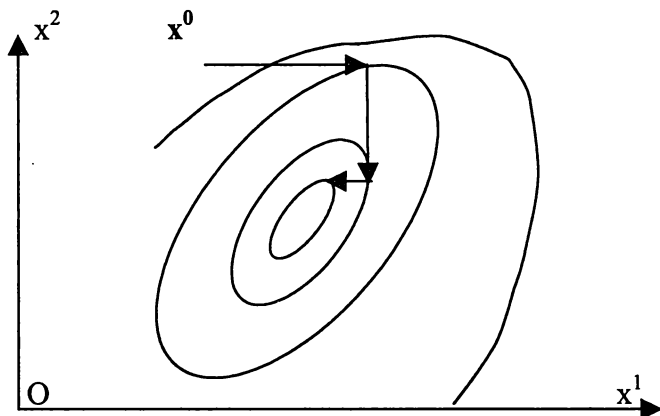


Рис. 20. Покоординатный спуск

Отметим, что при наличии простых ограничений на отдельные переменные вида $a_i \leq x_i \leq b_i$ дополнительных сложностей не возникает. Нужно просто при изменении соответствующей координаты не нарушать это ограничение. Если точка минимума выходит за пределы ограничивающего отрезка $[a_i, b_i]$, нужно поставить ее на соответствующую границу отрезка.

В квадратичной задаче точка минимума при изменении очередной координаты вычисляется аналитически следующим образом. Целевая функция $1/2 (Qx, x) + (c, x)$ при изменении k -ой координаты вектора x становится функцией только x_k , так все остальные координаты фиксированы.

В точке минимума этой функции ее производная по x_k равна нулю. Но эта производная есть k -ая компонента градиента $\mathbf{g}_k = (\mathbf{Q}\mathbf{x})_k + c_k = 0$. Получаем

$$\sum_j q_{kj} x_j + c_k = 0.$$

Сумма — это произведение k -ой строки матрицы квадратичной формы на вектор \mathbf{x} . В этой сумме неизвестна только x_k , для которой получаем:

$$x_k = -1/q_{kk} \left(c_k + \sum' q_{kj} x_j \right).$$

В последней сумме суммирование идет по всем j от 1 до n , кроме $j = k$.

Пример.

В качестве иллюстрации методов безусловной оптимизации рассмотрим целевую функцию

$$F(\mathbf{x}) = x_1^2 + x_1 x_2 + x_2^2 = \frac{1}{2} (\mathbf{Q}\mathbf{x}, \mathbf{x}),$$

где матрица $\mathbf{Q} = \begin{vmatrix} 2 & 1 \\ 1 & 2 \end{vmatrix}$.

Собственные числа матрицы \mathbf{Q} равны 3 и 1, а ортонормальные собственные векторы $\mathbf{f}_1 = (1/\sqrt{2}, 1/\sqrt{2})^T$ и $\mathbf{f}_2 = (1/\sqrt{2}, -1/\sqrt{2})^T$ соответственно. Новые переменные $y_1 = 1/\sqrt{2}(x_1 + x_2)$ и $y_2 = 1/\sqrt{2}(x_1 - x_2)$ и

$$F(\mathbf{x}) = \frac{1}{2} (3y_1^2 + y_2^2) = \frac{3}{4} (x_1 + x_2)^2 + \frac{1}{4} (x_1 - x_2)^2.$$

Линии уровня целевой функции — эллипсы с центром в начале координат, так как отсутствует линейная часть (рис. 21).

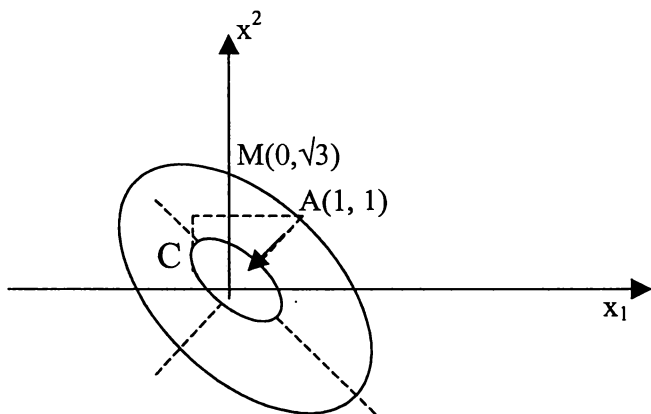


Рис. 21. Пример покоординатного спуска

Пусть задана начальная точка A с координатами $(1, 1)$. Ей соответствует значение целевой функции равное 3.

Первый шаг. Фиксируем $x_2 = 1$ и ищем минимум функции

$$\frac{3}{4}(x_1 + 1)^2 + \frac{1}{4}(x_1 - 1)^2,$$

который достигается в точке $x_1 = -1/2$ и равен $3/4$.

Второй шаг. Фиксируем $x_1 = -1/2$ и ищем минимум функции

$$\frac{3}{4}\left(x_2 - \frac{1}{2}\right)^2 + \frac{1}{4}\left(x_2 + \frac{1}{2}\right)^2,$$

который достигается в точке $x_2 = 1/4$ и равен $3/16$. Получаем точку $C(-1/2, 1/4)$.

Третий шаг. Фиксируем $x_2 = 1/4$ и находим $x_1 = -1/8$.

Четвертый шаг. Фиксируем $x_1 = -1/8$ и находим $x_2 = -1/16$. При этом достигли точки $D(-1/8, 1/16)$ и значения целевой функции $3/256$. Заметим, что если бы у нас было ограничение $x_1 \geq 0$, то второй шаг был бы последним. Так что не всегда ограничение — это усложнение задачи, бывает и наоборот.

Алгоритм наискорейшего спуска в той же задаче из той же начальной точки $(1, 1)$ дает минимум за одну итерацию. Это очевидно, так как антиградиент в этой точке имеет координаты $(-3, -3)$ и при $\lambda = 1/3$ мы получаем точку минимума $(0, 0)$.

С другой стороны, если взять за начальное приближение точку $M(0, \sqrt{3})$, которая лежит на той же линии уровня, то покоординатный спуск дает минимум за один шаг, если угадать, что первой надо изменять x_2 , а не x_1 . А наискорейший спуск из этой точки, в которой антиградиент $-g^0$ имеет компоненты $(-\sqrt{3}, -2\sqrt{3})$, дает за одну итерацию точку $x^1(-5/14\sqrt{3}, 4/14\sqrt{3})$, в которой целевая функция имеет значение $9/28$.

Если сделать еще один шаг по антиградиенту $-g^1$ с координатами $(6/14\sqrt{3}, -3/14\sqrt{3})$, то получим (при $\lambda = 5/6$) точку $(0, 3/28\sqrt{3})$ и значение целевой функции в ней $27/784$. Это означает, что после двух итераций по одной из координат отклонение от точки минимума составляет примерно 0.17 , а по целевой функции — примерно 0.034 .

Теперь сделаем второй шаг не по антиградиенту, а по сопряженному направлению p^1 . Для этого вычисляем

$$(g^1, g^1) = (36 + 9) \frac{3}{14^2} = \frac{135}{14^2};$$

$$(\mathbf{g}^0, \mathbf{g}^0) = 3 + 12 = 15;$$

$$\text{и } \beta_0 = (\mathbf{g}^1, \mathbf{g}^1) / (\mathbf{g}^0, \mathbf{g}^0) = \frac{9}{14^2}.$$

Далее $\mathbf{p}^1 = -\mathbf{g}^1 - \beta_0 \mathbf{g}^0$.

Этот вектор имеет координаты: $\left(75\sqrt{3}/14^2, -60\sqrt{3}/14^2\right)$.

Идем из точки \mathbf{x}^1 по направлению \mathbf{p}^1 , т.е. по лучу $\mathbf{x}^1 + \lambda \mathbf{p}^1$. Точки на этом луче имеют координаты $\sqrt{3}/14(-5 + \lambda 75/14)$ и $\sqrt{3}/14(4 - \lambda 60/14)$. При $\lambda = 14/15$ обе координаты обращаются в нуль. Это означает, что направление \mathbf{p}^1 ведет в точку минимума $(0, 0)$.

Важно отметить, что метод сопряженных градиентов при минимизации квадратичной формы дает точку минимума не более чем за n шагов (у нас $n = 2$) независимо от выбора начальной точки.

На данном примере мы убедились в том, что число шагов и при покоординатном спуске и при наискорейшем спуске существенно зависит от выбора начального приближения, а при покоординатном спуске имеет значение еще и порядок нумерации координат. В квадратичных задачах безусловной оптимизации неоспоримое преимущество имеет метод сопряженных градиентов и его следует использовать вместо метода наискорейшего спуска.

2.3.5. О методах второго порядка

Методы второго порядка предусматривают использование вторых частных производных целевой функции. Поскольку для квадратичной задачи целевая функция $1/2(\mathbf{Q}\mathbf{x}, \mathbf{x}) + (\mathbf{c}, \mathbf{x})$, необходимое условие экстремума дает

$$\mathbf{g} = \mathbf{Q}\mathbf{x} + \mathbf{c} = 0.$$

Получаем критическую точку $\mathbf{x}^* = -\mathbf{Q}^{-1} \mathbf{c}$.

Если исходная точка \mathbf{x}^0 и градиент в ней $\mathbf{g}^0 = \mathbf{Q}\mathbf{x}^0 + \mathbf{c}$, то выражение для \mathbf{x}^* можно записать так:

$$\mathbf{x}^* = -\mathbf{Q}^{-1}(\mathbf{g}^0 - \mathbf{Q}\mathbf{x}^0) = \mathbf{x}^0 - \mathbf{Q}^{-1}\mathbf{g}^0.$$

Другими словами, можно найти критическую точку за один шаг из любой точки. Но идти надо не по антиградиенту, а сначала умножить антиградиент на \mathbf{Q}^{-1} (см. формулу 2.6). При положительно определенной квадратичной форме критическая точка будет точкой минимума. Но для этого надо знать обратную матрицу \mathbf{Q}^{-1} или (что равносильно) решить систему линейных уравнений $\mathbf{Q}\mathbf{x} + \mathbf{c} = 0$. При большой размерности задачи это требует большого объема вычислений (порядка n^3 умножений).

Если целевая функция $F(\mathbf{x})$ не квадратичная, то она аппроксимируется квадратичной путем разложения в ряд Тейлора до членов второго порядка включительно. При этом роль матрицы \mathbf{Q} играет матрица вторых производных \mathbf{H} (матрица Гессе). Но в этом случае одного шага до точки минимума на луче $\mathbf{x}^0 - \lambda\mathbf{H}^{-1}\mathbf{g}^0$ недостаточно. В квадратичной задаче минимум достигался при $\lambda = 1$. В общем случае при $\lambda = 1$ получаем точку минимума не исходной функции, а аппроксимирующей ее квадратичной формы с матрицей Гессе, что не одно и то же. Поэтому $\lambda = 1$ следует рассматривать лишь как приближенное значение и искать точное значение путем решения задачи одномерной оптимизации функции $F(\mathbf{x})$ на луче $\mathbf{x}^0 - \lambda\mathbf{H}^{-1}\mathbf{g}^0$, рассматривая ее как функцию единственного параметра λ . С задачей одномерной оптимизации мы сталкиваемся всякий раз при поиске шага до точки минимума, когда *направление спуска известно*. Эта задача рассмотрена в разделе 2.3.7.

В полученной точке минимума по направлению приходится снова вычислять градиент и матрицу, обратную к матрице Гессе и т.д.

Сложность еще и в том, что в общем случае, помимо непрерывности вторых частных производных для сходимости метода, требуется еще и хорошее начальное приближение к точке минимума [5, 16].

Вместо вычисления вторых производных и обращения матрицы Гессе был предложен метод *переменной метрики*, или так называемый DFP-метод (Дэвидсон – Флетчер – Пауэлл). Идея метода в том, чтобы в процессе спуска получать все более точные приближения к обратной матрице Гессе, используя градиенты функции в уже пройденных итерационных точках [1, 5, 17, 18]. Метод требует существенно большего объема памяти, но и сходится быстрее, чем метод сопряженных градиентов, не говоря уже о наискорейшем спуске [1].

2.3.6. О методах прямого поиска

Методами прямого поиска называются методы, в которых используются только значения функции и не используются ее производные (градиент и матрица Гессе). Таким образом, они относятся к методам нулевого порядка. Если известно градиентное направление, то относительно любого направления можно установить, является ли оно улучшающим (по знаку его скалярного произведения на градиент). Если же неизвестно градиентное направление, то относительно любого направления ничего не ясно, если пробный шаг по нему был не удачным, так как всегда можно предположить, что этот шаг был велик. Другими словами, если есть возможность вычисления градиента, то этой возможностью пренебрегать не следует, так как методы прямого поиска, как правило, менее эффективны, чем методы первого и второго порядка. Однако в некоторых случаях методы первого и второ-

го порядка применить не удастся, например из-за разрывности функций и производных. Более того, возможны задачи, в которых целевая функция в явном виде не задана и для вычисления ее значений приходится решать дополнительно уравнения, в том числе и дифференциальные, или системы уравнений, относящихся к различным подсистемам некоторой системы. В этих случаях не только аналитическое, но даже численное определение производных оказывается очень сложным и даже невозможным.

Другим случаем, когда методы прямого поиска могут конкурировать с градиентными, является случай наличия локальных минимумов и несвязных допустимых областей (см. рис. 13).

Различные алгоритмы прямого поиска основаны на идее обследования окрестности выбранной точки путем пробных шагов в нескольких направлениях. Это могут быть, например, координатные направления. При этом последовательно меняется по одной координате исходной точки. После того как найдено приемлемое направление, в этом направлении делаются постепенно увеличивающиеся шаги, пока целевая функция в этом направлении не перестанет убывать. Если из некоторой точки уже не удастся сделать шаг большой длины в данном направлении, то уменьшают шаг. Если шаг становится малым, но в данном направлении уменьшить целевую функцию не удастся, то работа с направлением закончена и нужно найти другое направление. Для этого снова обследуется окрестность полученной точки и т.д. Фаза обследования может выполняться различным образом, поэтому и методов прямого поиска много [1]. Без существенного усложнения фазы обследования окрестности промежуточных точек эти методы так же, как и градиентные, могут «застевать» в точках локальных минимумов и в оврагах.

Другая идея прямого поиска состоит в отказе от последовательного поиска, т.е. от вычисления новых точек по ранее полученным значениям функции, ее градиента или матриц Гессе. Это идея так называемого *случайного поиска* [1]. Существует много алгоритмов случайного поиска, но все они так или иначе основаны на случайном выборе точек из допустимой области. Если нет никакой информации об улучшающих направлениях и никакая часть допустимой области не имеет преимущества, то принимается равная вероятность, иначе могут быть выделены более вероятные направления (например, в алгоритмах с самообучением). Случайный поиск, как и прочие прямые методы, чаще всего не имеющие математического обоснования, можно применять в особо сложных случаях («не от хорошей жизни»), к которым, помимо уже описанных, можно отнести задачи с дискретными переменными и смешанные задачи с непрерывными и дискретными переменными.

При наличии ограничений общего вида для всех прямых методов, в том числе и, может быть даже особенно, для случайного поиска, проблемой является получение не то что подходящего, а просто допустимого направления или точки допустимой области.

2.3.7. Методы одномерной минимизации

Существует много методов поиска минимума функции одной переменной на заданном отрезке. Как правило, функция предполагается унимодальной, т.е. имеющей один минимум. Для гладких унимодальных функций точка минимума совпадает с точкой, в которой производная функции равна нулю. Поэтому вместо точки минимума функции $f(x)$ можно искать корень уравнения $f'(x) = 0$.

Мы пришли к задаче одномерной оптимизации при поиске оптимального шага из заданной точки \mathbf{x}^k по заданному направлению \mathbf{p}^k . В искомой точке скалярное произведение антиградиента $-\mathbf{g}$ на вектор \mathbf{p}^k равно нулю, поэтому наша задача состоит в поиске нуля функции $(\mathbf{g}, \mathbf{p}^k)$. Здесь вектор \mathbf{p}^k известен, а вектор \mathbf{g} меняется при движении по лучу $\mathbf{x}^{k+1} = \mathbf{x}^k + \lambda \mathbf{p}^k$. Наша цель найти λ и тем самым \mathbf{x}^{k+1} , при котором $(\mathbf{g}, \mathbf{p}^k) = 0$. В исходной точке \mathbf{x}^k антиградиент и вектор \mathbf{p}^k составляют острый угол, т.е. $(\mathbf{g}, \mathbf{p}^k) < 0$, иначе \mathbf{p}^k — неподходящее направление (см разд. 2.3). Чтобы ограничить интервал поиска, нужно найти точку, в которой $(\mathbf{g}, \mathbf{p}^k) > 0$. Это можно сделать, используя пробные значения λ и вычисляя $(\mathbf{g}, \mathbf{p}^k)$ в пробных точках \mathbf{x}^{k+1} . На первой итерации ($k = 0$) λ приходится брать произвольно, а на всех остальных можно взять для начала оптимальное значение λ из предыдущей итерации. Если при первом пробном шаге скалярное произведение отрицательно, необходимо увеличить шаг и так до тех пор, пока не получим положительное скалярное произведение, т.е. не «перешагнем» через точку минимума. Увеличивать λ можно по любому закону, например умножением его на 2. Если вдруг окажется в некоторой точке, что скалярное произведение $(\mathbf{g}, \mathbf{p}^k) \approx 0$, то эту точку нужно принять за результат поиска. В противном случае имеем интервал между предпоследним и последним значением λ . Если в первой пробной точке $(\mathbf{g}, \mathbf{p}^k) > 0$, то левая граница интервала есть точка нуль. Определив интервал, мы можем использовать любой из методов сокращения его длины [1, 5]. Например, можно разделить его пополам, вычислить в средней точке скалярное произведение $(\mathbf{g}, \mathbf{p}^k)$ и оставить тот из двух интервалов, на котором оно меняет знак. Такую процедуру можно повторять до тех пор, пока длина интервала неопределенности не станет меньше требуемой точности поиска λ . В итоге можно принять за результат середину

интервала неопределенности, но лучше взять меньшее значение λ_k во избежание «проскакивания» через точку минимума и дополнительной опасности зигзагов. Описанный алгоритм поиска нуля функции известен со времен Древней Греции и называется *дихотомия*. Существует много более сложных алгоритмов, в том числе использующих не только знаки функции, нуль которой мы ищем (у нас это $(\mathbf{g}, \mathbf{p}^k)$), но и ее значения в уже исследованных точках [5].

Может оказаться, что проще вычислять не значение градиента в промежуточных точках на луче, а непосредственно значение целевой функции. Тогда не нужно сводить одномерную задачу минимизации к поиску нуля скалярного произведения градиента и вектора спуска, а решать ее непосредственно.

Пусть мы с помощью пробных значений λ и вычисления значений функции при каждом из них, до тех пор пока не получим ее возрастания, определили интервал поиска $[\lambda_1, \lambda_2]$ (рис. 22). Вычисленные точки помечены значком \times .

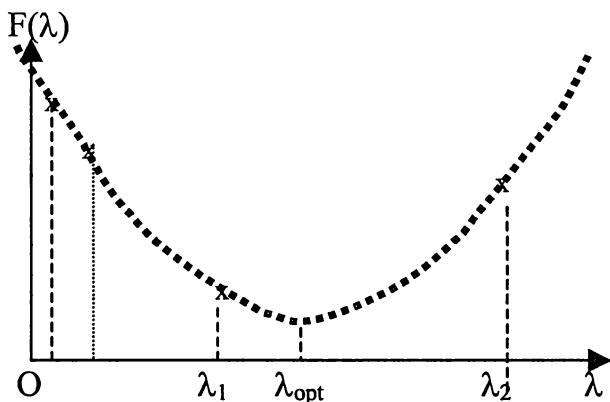


Рис. 22. Определение начального интервала (λ_1, λ_2) и λ_{opt}

Теперь можно по последним трем точкам построить параболу второй степени, найти ее точку минимума и принять эту точку за результат. Но можно с помощью вычисления значения функции в дополнительных точках получить более надежное решение.

Например, интервал (λ_1, λ_2) можно разбить на три части по методу золотого сечения [1] (рис. 23).

Для этого вставляются две точки λ_3 и λ_4 , так что

$$(\lambda_4 - \lambda_1) / (\lambda_2 - \lambda_1) = \tau \approx 0,6180,$$

$$\text{а } (\lambda_3 - \lambda_1) / (\lambda_2 - \lambda_1) = 1 - \tau \approx 0,3820,$$

где τ корень уравнения $\tau^2 + \tau - 1 = 0$.

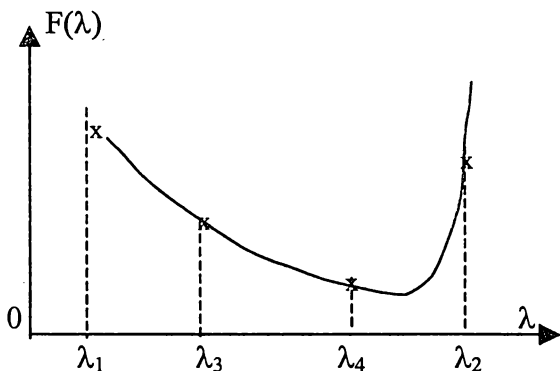


Рис. 23. Определение λ_{opt} по методу золотого сечения

Поскольку функция унимодальна, то один из крайних интервалов — (λ_1, λ_3) или (λ_4, λ_2) — можно отбросить, так что в средней из оставшихся точек значение функции меньше, чем в крайних. В примере на рис. 23 можно отбро-

силь интервал (λ_1, λ_3) . Примечательно, что средняя точка делит оставшийся интервал на части, длины которых относятся как $\tau / (1 - \tau)$. Поэтому далее нужно добавить не две, а только одну точку, разбив большую из частей так, чтобы из трех образовавшихся интервалов два крайних имели равную длину. Добавление точек заканчивается, когда длина интервала неопределенности становится меньше требуемой точности поиска точки минимума по направлению. Вместо метода золотого сечения можно использовать разбиение с помощью чисел Фибоначчи [1] или какой-либо другой алгоритм [1, 5], от этого зависит время расчета и сложность программы, но не конечный результат оптимизации. Заранее предсказать, какой метод (например, полиномиальная интерполяция или поиск с помощью чисел Фибоначчи) даст лучший результат, крайне сложно, так как важное значение имеют свойства конкретной целевой функции.

Контрольные вопросы и задачи

1. Является ли выпуклой функция $y = x^3 - x$ на отрезках $[-1, 1]$ и $[0, 1]$? Найти ее минимум на $[0, 1]$.

2. Можно ли считать линейную функцию выпуклой?

3. Исследовать квадратичную форму $x_1^2 + 3x_1x_2 + x_2^2$. Найти собственные числа и собственные векторы. Привести квадратичную форму к сумме квадратов и установить, является ли она знакоопределенной. Является ли точка $(0, 0)$ критической, точкой минимума или максимума?

4. Построить линии уровня функции

$$F(x_1, x_2) = x_1^2 + 3x_1x_2 + x_2^2.$$

5. Какой из векторов с координатами $(1, -2)$, или $(-2, 1)$, или $(2, -1)$ задает направление спуска для $F(x_1, x_2) = x_1^2 + 3x_1x_2 + x_2^2$ из точки $(1, 1)$?

6. Можно ли утверждать, что если некоторое направление \mathbf{p} не является направлением спуска, то минус \mathbf{p} обязательно будет направлением спуска?

7. Градиент функции двух переменных в некоторой точке не равен нулю. Может ли быть в этой точке два не подходящих (не улучшающих) направления?

8. Вывести формулу (2.10) на основе вывода формулы (2.8).

9. Формула (2.9) устанавливает максимальную величину отношения значений целевой функции на двух последовательных итерациях наискорейшего спуска. Чем меньше это отношение, тем лучше. А чему соответствует его минимальное значение в заданной точке? Какому направлению?

10. Можно ли утверждать, что существует бесконечно много точек, из которых наискорейший спуск *за один шаг* дает точку минимума положительно определенной квадратичной формы?

11. Имеет ли значение порядок нумерации переменных для метода наискорейшего спуска?

12. В методе золотого сечения интервал неопределенности с каждой новой точкой составляет $\tau \approx 0,618$ от имевшегося, а в методе половинного деления (дихотомия) ровно 0,5 от имевшегося. Можно ли всегда использовать дихотомию вместо золотого сечения для ускорения сходимости?

13. Убедиться в том, что в рассмотренном выше примере минимизации $F(\mathbf{x}) = x_1^2 + x_1x_2 + x_2^2$ антиградиент и сопряженный с ним вектор \mathbf{pQ} ортогональны, т.е. $(\mathbf{g}, \mathbf{Qp}) = 0$.

14. Известно, что собственные векторы положительно определенной матрицы, соответствующие различным собственным числам, ортогональны. Можно ли считать их \mathbf{Q} ортогональными, т.е. сопряженными относительно матрицы \mathbf{Q} ?

2.4. Задачи с линейными ограничениями

Ограничения в задачах оптимизации являются отражением реальных ситуаций, имеющих место в технических, экономических, социальных и др. сложных системах. Ограничены различные виды ресурсов, наличествуют связи, зависимости между теми или иными переменными, описывающими реальные процессы и т.д. Часто приходится вводить ограничения даже там, где, казалось бы, они не нужны по смыслу задачи. Например, при поиске оптимального распределения ресурсов, естественно, не должны получаться отрицательные величины. Но, если не поставить соответствующего ограничения, то при решении задачи оптимизации они вполне могут быть получены. Нельзя забывать о том, что оптимизация выполняется *формально в рамках математической модели*.

Линейные ограничения — это частный случай ограничений. Но многие задачи сводятся к задачам оптимизации именно с линейной системой ограничений. Относительно допустимой области при наличии линейной системы ограничений справедливо все, что сказано об ОДР в разделе «Линейное программирование». Однако наличие нелинейной целевой функции приводит к тому, что теория линейного программирования уже не применима. В частности, точкой минимума может быть и внутренняя точка допустимой области, а крайние точки (вершины многогранника) уже не играют той исключительной роли, как в линейных задачах. Минимум может достигаться во внутренней или в граничной точке, как в вершине, так и на граничных линейных многообразиях различной размерности.

Если целевая функция выпукла на допустимой области, то точка локального минимума является и точкой глобального минимума, или, как часто говорят, локальных мини-

мумов нет. Однако в этой точке градиент целевой функции может быть и не равным нулю, если точка не является внутренней (рис. 24). Поэтому изменяются и необходимые условия экстремума по сравнению с задачами без ограничений.

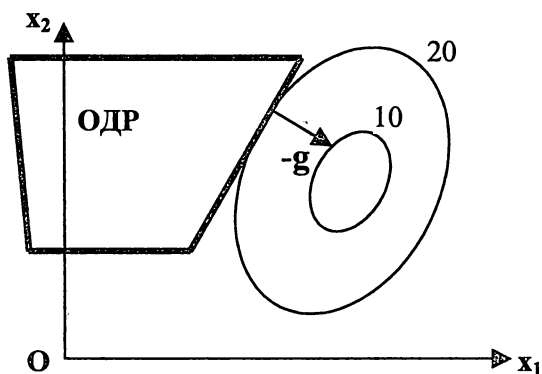


Рис. 24. В точке минимума антиградиент не нуль

2.4.1. Задачи с ограничениями-равенствами

Если линейная система ограничений содержит только ограничения-равенства, то задача записывается в следующем виде.

Найти: $\min F(\mathbf{x})$ при $A\mathbf{x} = \mathbf{b}$, где A — заданная матрица ($m \times n$), \mathbf{b} — заданный вектор, а \mathbf{x} — неизвестный вектор с координатами x_i ($i = 1, 2, \dots, n$). Относительно $F(\mathbf{x})$ сохраним все те же предположения гладкости, что и в задачах безусловной оптимизации. Число строк матрицы A меньше числа ее столбцов ($m < n$) и строки предполагаются линейно независимыми, так как их линейная зависимость означала бы либо несовместность системы (отсутствие решений),

либо наличие лишних ограничений, которые можно отбросить, не изменив решение задачи.

Рассматриваемая нами задача может быть сведена к задаче безусловной минимизации с помощью множителей Лагранжа. При этом ограничения исчезают, но для каждого из них вводится неизвестный множитель Лагранжа λ_i ($i = 1, 2, \dots, m$) и целевая функция приобретает вид:

$$\sum_i \lambda_i ((\mathbf{a}_i^T, \mathbf{x}) - b_i).$$

Здесь \mathbf{a}_i — i -ая строка матрицы \mathbf{A} , так что $((\mathbf{a}_i^T, \mathbf{x}) - b_i)$ — это фактически левая часть i -ого ограничения-равенства, если систему переписать в виде $\mathbf{Ax} - \mathbf{b} = \mathbf{0}$. Конечно, как и при линейной целевой функции, можно было бы попытаться исключить m переменных из системы и подставить их выражения через оставшиеся переменные в целевую функцию. При этом уменьшилась бы размерность задачи и исчезли бы ограничения. Но для этого требуется большой объем вычислений, и нелинейная целевая функция не становится проще.

Введение множителей Лагранжа увеличивает число переменных (их стало $n + m$), но дает возможность применить любой из методов безусловной минимизации. Заметим, что равенство нулю производных новой целевой функции по λ_i ($i = 1, 2, \dots, m$) соответствует ограничениям-равенствам, так что получаемая в результате безусловной минимизации точка минимума является допустимой. Но в этой точке сумма, добавленная к $F(\mathbf{x})$, равна нулю, что и означает, что эта точка является решением исходной задачи.

2.4.2. Задачи с ограничениями-неравенствами

Если в системе ограничений есть неравенства, то ее можно представить в виде $\mathbf{Ax} \leq \mathbf{b}$. При этом число строк

матрицы A может быть и меньше и больше числа ее столбцов. Но мы будем рассматривать случай ограниченной ОДР, и поэтому для нас $m > n$.

В точке минимума x^* некоторые ограничения могут выполняться как строгие неравенства (относительно них точка x^* является внутренней); такие ограничения называются *неактивными*. А некоторые ограничения могут выполняться как равенства (относительно них точка x^* является граничной); такие ограничения называются *активными*. Во внутренних точках ОДР все ограничения неактивны, а каждому граничному линейному многообразию соответствует свой набор активных ограничений. Вспомним, что в вершинах многогранника в отсутствии вырожденности активно ровно n ограничений (но в каждой вершине это свой набор ограничений).

Если бы было известно, какие ограничения будут активны в точке минимума, то можно было бы отбросить все прочие ограничения, активные ограничения сделать равенствами и свести все к безусловной оптимизации и решать задачу как и ранее. Но обычно активный набор в точке минимума неизвестен, а попытка перебора всех возможных комбинаций ограничений еще более неуместна, чем перебор вершин многогранника при попытке решить задачу линейного программирования. Тем более что перебирать пришлось бы не только вершины, но и многообразия различной размерности (ребра, грани и т.д.). Далее будем именовать эти граничные множества для краткости гранями различной размерности, так что ребро — это грань размерности 1, а вершина — грань размерности 0.

Существует много методов решения задач с линейными ограничениями. Они являются обобщением соответствующих методов безусловной оптимизации. В частности, как и для задач без ограничений, существуют методы первого и второго порядка. Мы рассмотрим только некоторые методы первого порядка.

2.4.2.1. Метод проекции градиента

Предположим, что нам известна некоторая внутренняя точка допустимой области x^0 . В этой точке нет активных ограничений и ничто не мешает нам выполнить шаг по антиградиенту до точки минимума. Если эта точка опять окажется внутренней, то из нее снова можно сделать шаг по антиградиенту, как в методе наискорейшего спуска, или по сопряженному с антиградиентом направлению, как в методе сопряженных градиентов и т.д. Очевидно, что возможно два исхода такого процесса.

1. Мы остановимся вблизи точки безусловного минимума целевой функции, так и не встретив границу. Это означает, что минимум достигается *внутри допустимой области и все ограничения просто лишние*.

2. На каком-то из шагов точка минимума в направлении спуска не удовлетворяет одному или нескольким ограничениям, т.е. лежит за пределами ОДР. Это означает, что был сделан слишком большой шаг из последней внутренней точки и шаг нужно было делать не до точки минимума, а до встречи с границей (рис. 25).

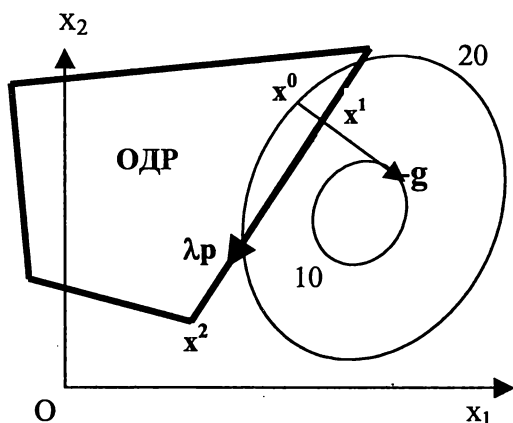


Рис. 25. К вычислению шага

Встретив границу, мы не можем далее не считаться с ограничениями и будем действовать в рамках идей рассмотренного выше метода возможных направлений. Теперь возможными являются направления, которые при достаточно малом шаге не выводят за пределы допустимой области. Среди них будем искать *подходящие* направления, т.е. такие направления, которые дают при соответствующем выборе шага уменьшение целевой функции. Мы уже знаем, что целевая функция уменьшается в заданном направлении \mathbf{p} , если оно составляет острый угол с антиградиентом $-\mathbf{g}$, т.е. $(\mathbf{g}, \mathbf{p}) < 0$. Одно из подходящих направлений — это проекция антиградиента на ту грань (точнее, на соответствующее подпространство), которую мы встретили при движении из внутренней точки, если, конечно, при этом проекция не равна нулевому вектору, что будет рассмотрено отдельно. При любом наборе активных ограничений проекцию можно вычислить по формуле (2.5), подставив в нее вместо матрицы \mathbf{A} матрицу, составленную из строк, соответствующих активным ограничениям, а вместо вектора \mathbf{f} антиградиент $-\mathbf{g}$. В итоге получим

$$\mathbf{p} = -(\mathbf{E} - \mathbf{A}^T (\mathbf{A} \mathbf{A}^T)^{-1} \mathbf{A}) \mathbf{g} = -\mathbf{P} \mathbf{g}, \quad (2.11)$$

где \mathbf{P} — матрица проектирования. Направление \mathbf{p} является не только допустимым (мы идем по нему вдоль границы, не нарушая ограничений), но и подходящим, так как $(-\mathbf{g}, \mathbf{p}) = (\mathbf{p} + \mathbf{n}, \mathbf{p}) = (\mathbf{p}, \mathbf{p}) > 0$. Таким образом, можно идти в направлении проекции антиградиента и уменьшить значение целевой функции при правильном выборе шага. При выборе шага мы не должны выйти за пределы допустимой области. Поэтому нужно прежде всего найти такой *максимальный шаг* (соответственно λ_{\max}), при котором *все* ограничения выполнены. Это соответствует *минимальному* шагу, при котором луч $\mathbf{x}^1 + \lambda \mathbf{p}$ встречает границу (рис. 25,

точка \mathbf{x}^2). Для определения такого λ надо поочередно подставить $\mathbf{x}^1 + \lambda \mathbf{p}$ вместо \mathbf{x} во все неактивные ограничения, из каждого уравнения определить λ и из всех найденных положительных значений взять наименьшее. Например, i -ое ограничение $(\mathbf{a}_i^T, \mathbf{x}) \leq b_i$ в точке \mathbf{x}^1 неактивно. Оно станет активно при $(\mathbf{a}_i^T, \mathbf{x}^1 + \lambda_i \mathbf{p}) = b_i$, т.е. при

$$\lambda_i = (b_i - (\mathbf{a}_i^T, \mathbf{x}^1)) / ((\mathbf{a}_i^T, \mathbf{p})). \quad (2.12)$$

Если получили $\lambda_i < 0$, то это означает, что в данном направлении мы соответствующую границу не встретим. Поэтому начинать надо со знаменателя и при $(\mathbf{a}_i^T, \mathbf{p}) \leq 0$ λ_i вычислять вообще не надо. Пройдя по всем неактивным ограничениям, найдем минимальное λ , при котором встречаем границу (рис. 25, точка \mathbf{x}^2). Это и будет максимальный шаг из точки \mathbf{x}^1 . Но этот шаг может быть больше, чем до точки минимума в направлении \mathbf{p} (рис. 25), и тогда его нельзя использовать. Другими словами, из двух значений шага — *до границы и до точки минимума* — надо взять *наименьший*. Сначала надо выяснить, действительно ли шаг до границы больше, чем для точки минимума. Для этого не надо определять шаг до точки минимума, так как мы рассматриваем выпуклую целевую функцию. Достаточно выяснить знак скалярного произведения вектора спуска \mathbf{p} и нового антиградиента в точке \mathbf{x}^2 . Если это скалярное произведение неотрицательно, то нужно перейти в точку \mathbf{x}^2 , а если отрицательно, то нужно искать точку минимума целевой функции на отрезке $[\mathbf{x}^1, \mathbf{x}^2]$. Возникает в точности та же задача одномерной оптимизации, что и при решении задач без ограничений (см. раздел 2.3.7). Здесь даже задача несколько проще, так как интервал поиска известен (по λ это $0 < \lambda < \lambda_{\max}$).

В новой точке вычисляем новый антиградиент и его проекцию и т.д. Этот процесс остановится, когда проекция

градиента на очередную грань станет равна нулю. Но это еще не означает, что получено решение задачи, т.е. найден минимум функции при ограничениях в виде неравенств. На каждом шаге число активных ограничений могло только увеличиваться (при шаге до очередной границы) и мы могли попасть в вершину ОДР. В этом случае проекция антиградиента (на точку!) равна нулю, но минимум может быть совсем в другой точке и при другом наборе активных ограничений. На рис. 26 представлен случай, когда процесс останавливается в точке x^1 (вершине), в которой активны ограничения, не имеющие отношения к активному набору в точке минимума.

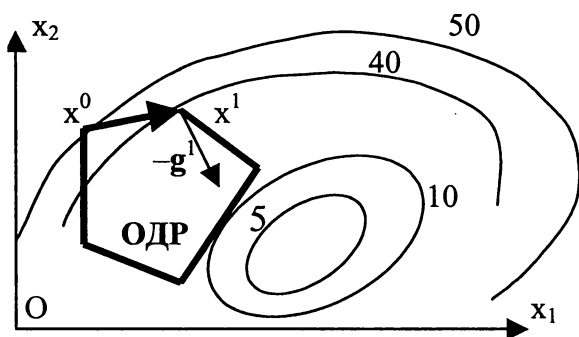


Рис. 26. Нулевая проекция антиградиента в «чужой» вершине

Таким образом, равенство нулю проекции градиента — это *необходимое, но не достаточное* условие минимума. Нужно еще, чтобы в активном наборе были представлены ограничения, активные в неизвестной нам точке минимума. Другими словами, необходимо выйти на нужную грань, не застревая в посторонних точках. Активный набор надо не только пополнять, но и исключать из него ограничения, мешающие дальнейшему движению к точке минимума.

Ограничение может быть исключено из активного набора, если проекция антиградиента на грань, определяемую оставшимися в активном наборе ограничениями, не нарушает отброшенное ограничение. На рис. 26 представлен случай, когда можно отбросить ограничение (ребро $x^0 x^1$), спроектировать антиградиент на оставшееся ребро и продолжить процесс. Возможны ситуации, при которых есть выбор и можно отбросить то или другое из активных ограничений. Но возможны ситуации, при которых ни одно ограничение нельзя исключить из активного набора, так как проекция антиградиента, построенная «без его участия», ведет за пределы допустимой области. На рис. 27 показан именно такой случай. Если проектировать антиградиент $-g$ только на AC , то нарушается отброшенное ограничение (AB) , и наоборот.

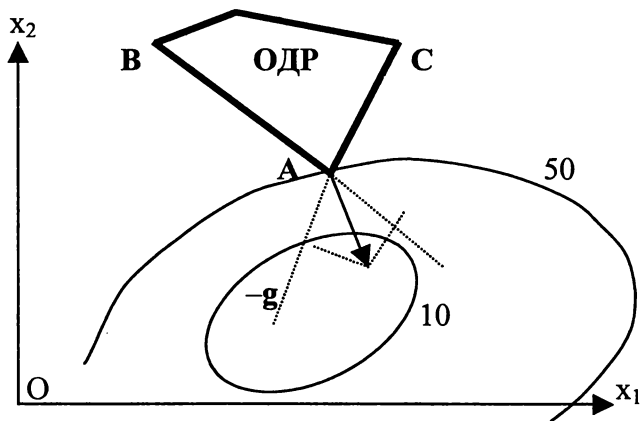


Рис. 27. Невозможность исключения ограничений

Если проекция градиента равна нулю и ни одно ограничение нельзя исключить из активного набора, то минимум достигнут.

Это и есть необходимое и достаточное условие минимума. Для проверки возможности исключения ограничений из активного набора нет необходимости исключать их по очереди, затем строить проекцию \mathbf{p} и проверять, не нарушается ли отброшенное ограничение. Вместо этого нужно разложить вектор $-\mathbf{g} - \mathbf{p}$ по базису, составленному из нормалей к гиперплоскостям, соответствующим активным ограничениям, и проанализировать коэффициенты этого разложения. Формула для коэффициентов (2.4) приведена в разделе 2. В ней надо вместо произвольного вектора \mathbf{f} подставить антиградиент $-\mathbf{g}$. Получим $\mathbf{u} = -(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{A}\mathbf{g}$. Если все коэффициенты (компоненты вектора \mathbf{u}) неотрицательны, то ни одно ограничение исключить нельзя. Если какой-либо коэффициент отрицательный, то соответствующее ему ограничение можно исключить. Обычно исключают ограничение, которому соответствует наибольший по абсолютной величине отрицательный коэффициент. Если несколько коэффициентов отрицательны, то это не означает, что можно отбросить сразу несколько ограничений. Это может привести к тому, что построенная проекция ведет за пределы допустимой области, т.е. из отрицательных коэффициентов можно выбрать *любой, но не более одного*.

Чтобы исключить ограничение из активного набора, не обязательно ждать, когда проекция станет равной нулю. Это можно сделать на любом шаге. Однако существует опасность «зигзагов», когда отброшенное ограничение снова попадает в активный набор, затем исключается и снова попадает и т.д. При этом нарушается сходимость алгоритма [5, 18]. Для борьбы с «зигзагами» рекомендуется допускать повторное исключение ограничений только после заданного числа итераций.

В целом алгоритм метода проекции градиента состоит из следующих пунктов.

1. Построение допустимого начального приближения x^0 .
2. Вычисление антиградиента $-g$.
3. Формирование активного набора и построение проекции антиградиента p . Здесь возможно исключение одного из ограничений и повторное вычисление проекции антиградиента.
4. Проверка условий окончания счета. Если они выполнены (проекция равна нулевому вектору и ни одно активное ограничение нельзя исключить), решение получено, иначе выполняем следующий пункт.
5. Поиск шага по направлению проекции антиградиента как минимального из шагов до границы и до точки минимума.
6. Переход в новую точку. Далее к пункту 3, если антиградиент в новой точке уже вычислялся, иначе к пункту 2.

В качестве начального приближения может использоваться любая точка допустимой области.

Существуют возможности ускорения сходимости за счет использования более эффективных алгоритмов. Описанный алгоритм при поиске минимума на одной грани (при неизменном активном наборе) ведет себя как наискорейший спуск. Целесообразно использовать вместо наискорейшего спуска метод сопряженных градиентов. В роли антиградиентов выступают их проекции. Естественно, что при каждом изменении активного набора следующий шаг приходится делать по проекции антиградиента. Поэтому ограничения лучше исключать, когда проекция антиградиента станет равна нулю.

Примеры построения проекции.

1. Если все активные ограничения простые, т.е. имеют вид

$$c_i \leq x_i \leq d_i,$$

то каждое из них может изменить только соответствующую компоненту проектируемого вектора. Эта компонента не изменяется, если она не нарушает ограничение, в противном случае она обнуляется.

2. Пусть есть одно активное ограничение на сумму неизвестных, т.е. $x_1 + x_2 + \dots + x_n \leq b$. Нужно спроектировать вектор f на гиперплоскость $x_1 + x_2 + \dots + x_n = 0$. Конечно, это можно сделать с помощью формулы (2.5), но можно и проще. Представим f в виде $f = p + n$, где p искомая проекция, а n — нормаль. Нормаль к гиперплоскости, т.е. транспонированная строка коэффициентов левой части неравенства, имеет все компоненты равные единице. Значит, у вектора n все компоненты равны между собой. Обозначим их через d . Тогда для любой компоненты проекции имеем $p_i = f_i - d$. Но сумма этих компонент должна быть равна нулю, так как проекция принадлежит подпространству, на которое мы проектируем. Отсюда $0 = \sum_i p_i = \sum_i (f_i - d)$.

Т.е. d равно среднему арифметическому компонент проектируемого вектора, а компоненты проекции получаются путем вычитания из соответствующих компонент проектируемого вектора этого среднего арифметического.

3. Та же задача, но при каждом неизвестном вместо единицы свой коэффициент α_i , так что подпространство задается условием $\sum_i \alpha_i x_i = 0$. Теперь компоненты нормали проектируемого вектора пропорциональны α_i с неизвестным коэффициентом d . Т.е. $p_i = f_i - d\alpha_i$, а $\sum_i \alpha_i p_i = 0$, так как p — проекция на подпространство. Далее,

$$0 = \sum_i \alpha_i p_i = \sum_i \alpha_i (f_i - d\alpha_i).$$

Отсюда находим $d = \left(\sum_i \alpha_i f_i \right) / \sum_i \alpha_i^2$ и затем вычисляем все $p_i = f_i - d \alpha_i$.

4. Система активных ограничений имеет вид: $x_{i+1} - x_i \leq b_i$ ($i = 1, 2, \dots, n-1$), что соответствует двухдиагональной матрице A , в которой $n-1$ строка и n столбцов. Это означает, что размерность нуль-пространства матрицы A

$$A = \begin{bmatrix} -1 & 1 & & & \\ & -1 & 1 & & 0 \\ 0 & & & & \\ & & & & \\ & & & -1 & 1 \end{bmatrix},$$

т.е. того подпространства $Ax = 0$, на которое мы проектируем вектор f , равна единице (прямая). У любого вектора этого подпространства все компоненты равны между собой. Следовательно, и у вектора p все компоненты равны некоторому p_0 . Но вектор $f - p$ должен быть ортогонален любому вектору с равными между собой компонентами, например вектору, у которого все компоненты равны единице. Значит, $\sum_i (f_i - p_0) = 0$ и $p_0 = \left(\sum_i f_i \right) / n$. Получается, что все компоненты проекции равны среднему арифметическому компонент проектируемого вектора, а для получения компоненты нормали нужно вычесть из соответствующей компоненты проектируемого вектора это среднее арифметическое. Если сравнить этот результат с примером 2, то увидим, что проекция и нормаль поменялись местами. Это не удивительно, так как соответствующие подпространства, на которые мы проектируем в этих примерах, ортогональны.

5. Система активных ограничений имеет вид:

$$(x_{i+2} - x_{i+1})/s_{i+1} - (x_{i+1} - x_i)/s_i \leq b_i, (i = 1, 2, \dots, n-2),$$

что соответствует трехдиагональной матрице A , в которой $n-2$ строки и n столбцов. Все числа s_i заданы.

$$A = \begin{pmatrix} 1/s_1 & -(1/s_1 + 1/s_2) & 1/s_2 & & & \\ & 1/s_2 & -(1/s_2 + 1/s_3) & 1/s_3 & 0 & \\ 0 & & & & & \\ & & & & & \\ & & & & & \\ & & & & 1/s_{n-2} & -(1/s_{n-1} + 1/s_{n-2}) & 1/s_{n-1} \end{pmatrix}.$$

Нуль-пространство матрицы A в данной задаче имеет размерность два (плоскость). Мы можем указать на нем базис, т.е. два линейно независимых вектора. Первый из них тот же, что и в предыдущем случае. Это вектор, у которого все компоненты равны между собой. Например, все по единице. Обозначим его c^1 . Второй базисный вектор имеет компоненты, которые есть последовательные суммы массива чисел s_i , а именно:

$$(0, s_1, s_1 + s_2, s_1 + s_2 + s_3, \dots, s_1 + s_2 + \dots + s_{n-1}).$$

Его обозначим c^2 .

В том, что этот вектор удовлетворяет системе $Ax = 0$ можно убедиться непосредственно, подставляя его компоненты в эту систему. Легко убедиться в линейной независимости векторов c^1 и c^2 .

Пусть, далее, искомым вектор p , т.е. проекция f на нуль-пространство матрицы A , имеет в этом базисе координаты y_1 и y_2 .

Это означает $p = y_1 c^1 + y_2 c^2$, и если мы найдем y_1 и y_2 , то легко найти и искомый вектор p .

Вектор $f - p$ ортогонален произвольному вектору нуль-пространства матрицы A , в том числе и его базисным векторам c^1 и c^2 . Значит,

$$(f - y_1 c^1 - y_2 c^2, c^1) = 0 \quad y_1 (c^1, c^1) + y_2 (c^1, c^2) = (f, c^1)$$

\rightarrow

$$(f - y_1 c^1 - y_2 c^2, c^2) = 0 \quad y_1 (c^1, c^2) + y_2 (c^2, c^2) = (f, c^2)$$

Получили систему двух линейных алгебраических уравнений с неизвестными y_1 и y_2 , в которой все коэффициенты известны.

$$(c^1, c^1) = n; \quad (c^1, c^2) = s_1 (n-1) + s_2 (n-2) + \dots + s_{n-1};$$

$$(c^2, c^2) = s_1^2 + (s_1 + s_2)^2 + \dots + (s_1^2 + s_2^2 + \dots + s_{n-1}^2);$$

$$(f, c^1) = f_1 + f_2 + \dots + f_n;$$

$$(f, c^2) = f_2 s_1 + f_3 (s_1 + s_2) + \dots + f_n (s_1 + s_2 + \dots + s_{n-1}).$$

Из этой системы определяем y_1 и y_2 и далее все компоненты вектора p .

Заметим, что нам пришлось решать систему двух уравнений с двумя неизвестными. А если пользоваться формулой (2.5) для вычисления проекции, то придется решать систему из $n - 2$ уравнений. При размерности задачи n порядка сотен и более это обстоятельство весьма существенно.

Данный пример допускает обобщение на случай произвольной матрицы, для которой в ее нуль-пространстве можно «угадать» базис.

Итак, пусть мы знаем этот базис, и его столбцы составляют матрицу C , а координаты неизвестного вектора p в этом базисе составляют вектор y . Это означает $p = Cy$ (см. раздел 1.1, формула (1.1)). Далее, $f - p$ (нормаль) ортогонален *любому* вектору p^1 в нуль-пространстве. Но этот любой вектор можно разложить по базису, как мы сделали это с вектором p . Естественно, что при этом будут получены другие координаты: вместо y какой-то вектор y^1 . Итак, $p^1 = Cy^1$ и так как p^1 произвольный вектор, то и его координаты (вектор y^1) произвольны. Запишем условие ортогональности $(f - p, p^1) = 0$. Или $(f - Cy, Cy^1) = 0$. Иначе это можно записать так $(f, Cy^1) - (Cy, Cy^1) = 0$. Далее, используя свойства скалярного произведения (см. раздел 1.1), освобождаем y^1 и получаем

$$(C^T f, y^1) - (C^T Cy, y^1) = 0$$

или

$$(C^T f - C^T Cy, y^1) = 0.$$

Скалярное произведение двух векторов, из которых один произвольный (y^1), равно нулю. Это возможно, когда другой вектор нулевой.

$$C^T f - C^T Cy = 0 .$$

Отсюда $y = (C^T C)^{-1} C^T f$ и $p = C(C^T C)^{-1} C^T f$.

При условии

$$n - m < m, \text{ т.е. } m > n/2,$$

где m и n соответственно число строк и столбцов матрицы активных ограничений, эта формула для построения

проекция требует решать системы с меньшим числом уравнений, чем при использовании (2.5).

Отметим, что если бы мы знали не просто базис в подпространстве, на которое проектируем, а *ортонормальный базис*, то $\mathbf{C}^T\mathbf{C}$ была бы единичной и для определения координат проекции проектируемый вектор нужно было бы умножить на $\mathbf{C}\mathbf{C}^T$ (слева), т.е. в ортонормальном базисе матрица проектирования — это просто $\mathbf{C}\mathbf{C}^T$, так как в этом случае $\mathbf{p} = \mathbf{C}\mathbf{C}^T\mathbf{f}$. При вычислении проекции антиградиента вместо \mathbf{f} надо брать $-\mathbf{g}$.

При решении практических задач часто приходится иметь дело с системами ограничений, имеющими структурные особенности. Их анализ может «подсказать» базисные векторы, т.е. решения однородной системы, и существенно упростить вычисления. Знание базиса позволяет при $m > n/2$ не только уменьшить размерность систем линейных уравнений, которые приходится решать на *каждой итерации* алгоритма проекции градиента, но и вообще *определять направление спуска без решения систем линейных уравнений при любом соотношении m и n* . Это направление мы получим в следующем разделе.

2.4.2.2. Метод приведенного градиента

Зададимся вопросом: что, если забыть, что базис не ортонормальный, и при вычислении проекции антиградиента $-\mathbf{g}$ вместо $\mathbf{p} = -\mathbf{C}(\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\mathbf{g}$ брать просто $\mathbf{p}^* = -\mathbf{C}\mathbf{C}^T\mathbf{g}$? К чему приведет такая ошибка? Понятно, что \mathbf{p}^* вычислить просто. Не нужно даже вычислять произведение матриц $\mathbf{C}\mathbf{C}^T$, а просто сначала вычислить $\mathbf{C}^T\mathbf{g}$, а потом умножить \mathbf{C} на результат. Достаточно хранить только базисные векторы.

Но имеет ли смысл такое упрощение сложной формулы для вычисления проекции \mathbf{p} ? При неортонормальном

базисе это упрощение не будет давать проекцию. Это очевидно. Тем не менее рассмотрим какой вектор \mathbf{p}^* мы получили бы.

Во-первых, это допустимое направление, так как вектор представлен в виде $\mathbf{C}\mathbf{y}^*$, где $\mathbf{y}^* = -\mathbf{C}^T\mathbf{g}$ это вектор коэффициентов разложения \mathbf{p}^* по базисным векторам.

Во-вторых, и это самое интересное, \mathbf{p}^* это подходящее направление, т.е. направление уменьшения целевой функции, так как скалярное произведение \mathbf{p}^* на антиградиент положительно. Действительно,

$$(\mathbf{p}^*, -\mathbf{g}) = (-\mathbf{C}\mathbf{C}^T\mathbf{g}, -\mathbf{g}) = (\mathbf{C}^T\mathbf{g}, \mathbf{C}^T\mathbf{g}) > 0.$$

Вектор $\mathbf{C}\mathbf{C}^T\mathbf{g}$, где \mathbf{g} градиент, называется *приведенный градиент*, а \mathbf{p}^* можно было бы назвать приведенный антиградиент.

Мы можем использовать \mathbf{p}^* как *направление спуска* вместо «правильной» проекции антиградиента. Но для реализации алгоритма проекции градиента с \mathbf{p}^* вместо \mathbf{p} необходимо еще иметь правило исключения ограничений из активного набора. В методе проекции градиента для этого вычисляются коэффициенты разложения (вектор \mathbf{u}) разности антиградиента и его проекции (т.е. нормали) по базису, составленному из строк матрицы активных ограничений. Для этого была получена формула $\mathbf{u} = -(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{A}\mathbf{g}$. Эта формула опять-таки требует вычисления $(\mathbf{A}\mathbf{A}^T)^{-1}$ (решения системы линейных уравнений). При известном базисе в нуль-пространстве матрицы \mathbf{A} нам удалось упростить вычисление направления спуска, но этого мало, так как надо еще упростить вычисление вектора \mathbf{u} или найти другой способ выявления возможности исключения ограничений из активного набора. Если мы знаем проекцию, то можно вычислить нормаль $\mathbf{n} = -\mathbf{g} - \mathbf{p}$ и записать $\mathbf{A}^T\mathbf{u} = \mathbf{n}$. При наличии структурных

особенностей матрицы A , которые позволили нам построить базис в подпространстве $Ax = 0$, часто удается и легко найти вектор u . Так, в примере 4 с двухдиагональной матрицей системы $x_{i+1} - x_i \leq b_i$ ($i = 1, 2, \dots, n-1$) получаем $u_1 = -n_1$, $u_2 = -n_1 - n_2$ и т.д. Не возникает проблем и с трехдиагональной матрицей примера 5, так как там тоже первое уравнение содержит только u_1 (сразу получаем $u_1 = s_1 n_1$), второе уравнение содержит только u_1 и u_2 , но u_1 уже знаем и т.д. Но это все просто при известной нормали, которую вычислили через проекцию. У нас же есть не проекция, а приведенный антиградиент p^* . Вектор $n^* = -g - p^*$ не является нормалью и не может быть представлен в виде $A^T u^*$. Так что при наличии приведенного антиградиента мы не можем использовать формулу (2.4) для решения вопроса об исключении ограничений из числа активных. Приходится искать другие пути.

Один из таких путей состоит в построении m векторов, каждый из которых нарушает одно и только одно ограничение из активного набора. Например, для i -ого ограничения построен такой вектор b^i . Если скалярные произведения (a_i^T, b^i) и (b^i, g) имеют одинаковые знаки, то это ограничение можно исключить из активного набора [15]. Здесь a_i — i -ая строка матрицы активных ограничений, а g — градиент целевой функции. Это правило надо применять ко всем векторам, каждый из которых нарушает одно и только одно ограничение. Как только будет найдено ограничение, которое можно исключить, дальнейший поиск можно прекратить, добавить вектор b^i в матрицу C , перевычислить приведенный антиградиент и продолжить процесс оптимизации. Характерно, что перевычисление приведенного антиградиента не требует работы с матрицами и сводится к вычитанию из каждой j -ой компоненты имеющегося приведенного антигради-

ента величины b_j^i (\mathbf{b}^i, \mathbf{g}). Это не сложно, тем более что (\mathbf{b}^i, \mathbf{g}) уже вычислен при анализе возможности исключения ограничения из числа активных.

Важно отметить, что при таком построении дополнительных базисных векторов имеется возможность исключать не одно, а сразу несколько ограничений из активного набора [15]. Если принять меры по предотвращению «зигзагов» (см. разд. 2.4.2.1), это открывает возможность быстрее выйти на нужную грань, т.е. сформировать нужный набор активных ограничений, ускорить сходимость и сократить время счета.

Пример.

Рассмотрим снова пример 4 с двухдиагональной матрицей системы $x_{i+1} - x_i \leq b_i$ ($i = 1, 2, \dots, n-1$). В данном случае размерность нуль-пространства равна единице и проекция антиградиента и приведенный антиградиент задают одно и то же направление, хотя и отличаются множителем $1/n$. Единственный базисный вектор имел все компоненты равные единице. Базисный вектор, нарушающий только ограничение $x_{k+1} - x_k$, если оно было активно, имеет все нулевые компоненты с 1-ой до k -ой включительно, а остальные по единице. Его скалярное произведение на антиградиент равно сумме компонент антиградиента с $k+1$ -ой по последнюю, и если оно положительно, то ограничение можно исключить.

Выбор вектора, нарушающего только одно ограничение, *неоднозначен*. Для определенности возьмем первое ограничение. Если выбрать дополнительный вектор так, как мы только что рассмотрели, то его первая компонента равна нулю, а все остальные по единице. Ограничение можно исключить, если *сумма всех компонент градиента, кроме первой, положительна*. Теперь возьмем вектор,

у которого первая компонента равна минус единице, а остальные нули. Очевидно, что он тоже нарушает только первое ограничение. Его скалярное произведение на градиент дает первую компоненту градиента с обратным знаком, и это произведение должно быть положительно. Только тогда первое ограничение можно отбросить. Получается, что все решает знак первой компоненты g_1 , а не знак суммы остальных компонент. На самом деле здесь нет противоречия, так как, отбрасывая ограничение, мы включаем в базис *разные векторы и по-разному будем пересчитывать приведенный антиградиент*. Включая один дополнительный вектор, мы можем отбросить ограничение, а включая другой вектор, нарушающий только это же ограничение, может быть, и нет. По этой же причине, используя метод приведенного градиента, нельзя для решения вопроса об исключении ограничений использовать знаки компонент вектора u , получаемых по формуле (2.4), так как мы не собираемся строить новую проекцию градиента после исключения соответствующего ограничения из активного набора.

2.5. Задачи с нелинейными ограничениями

Этот класс задач значительно сложнее задач с линейными ограничениями. Методы их решения основываются на идеях сведения задачи к безусловной оптимизации или к локальной аппроксимации нелинейных функций линейными для поиска направления спуска с последующим возвратом в допустимую область. Есть методы, в которых все ограничения делятся на две группы: линейные и нелинейные. Ограничения второй группы — учитываются преобразованием целевой функции, а ограничения первой группы как описано в разделе 2.4.

2.5.1. Методы штрафных функций

Совокупность методов, получивших это название, может применяться при решении задач с ограничениями как в виде равенств, так и в виде неравенств [1, 5, 18]. Эти ограничения могут быть и нелинейными. Сохраняются только требования гладкости. Основная идея — свести задачу с ограничениями к задаче (чаще к последовательности задач) без ограничений. Делается это с помощью различных модификаций целевой функции и дальнейшего применения различных методов безусловной оптимизации.

Общая формулировка задачи следующая. Найти $\min F(x)$ при ограничениях

$$c_j(x) \leq 0, (j = 1, 2, \dots, m),$$

где все или часть функций-ограничений $c_j(x)$ могут быть нелинейны.

Идея метода штрафных функций, называемого также методом *внешней точки*, состоит в таком *преобразовании целевой функции*, чтобы в допустимой области ее значения не изменились, а за пределами допустимой области были очень велики по сравнению с исходной $F(x)$. Тогда минимум новой функции и минимум исходной будут близки. Такое преобразование целевой функции может выполняться различными способами. Один из обычных приемов состоит в том, что к исходной $F(x)$ прибавляется сумма штрафов $g_j(x)$ за нарушение ограничений. Если в текущей точке x j -ое ограничение выполнено (т.е. $c_j(x) \leq 0$), то штраф равен нулю, иначе, чем больше нарушение, тем больше штраф.

Часто принимают в качестве штрафных функции

$$g_j(x) = \begin{cases} 0, & \text{при } c_j(x) \leq 0; \\ c_j^2(x) & \text{при } c_j(x) > 0. \end{cases}$$

Такая функция представляет собой квадрат невязки в j -ом ограничении, а общий штраф равен сумме штрафов. Так что преобразованная функция имеет вид

$$\Phi(\mathbf{x}) = F(\mathbf{x}) + k_n \sum_{j=1}^m g_j(\mathbf{x}). \quad (2.13)$$

Далее решается задача поиска безусловного минимума $\Phi(\mathbf{x})$. Если взять коэффициент k_n очень большим, то теоретически минимумы $\Phi(\mathbf{x})$ и $F(\mathbf{x})$ близки. Но этого делать нельзя, так как введение штрафа порождает овраг. В направлениях вдоль границ ОДР функция меняется медленно, но при малом изменении невязок в ограничениях функция меняется очень сильно. В такой ситуации методы безусловной оптимизации работают плохо. Поэтому сначала решается задача с малым значением коэффициента штрафа (например, $k_1 = 1$), полученное решение используется как начальное приближение для следующей задачи с $k_2 = 2$, затем решается задача с $k_3 = 4$ и так далее при все большем значении k . Фактически решается последовательность задач. Процесс останавливается, когда ограничения выполнены с требуемой точностью. Последняя точка минимума считается решением исходной задачи.

Если в исходной системе были ограничения-равенства, то штрафуются как положительные, так и отрицательные значения $c_j(\mathbf{x})$, т.е. $g_j(\mathbf{x}) = c_j^2(\mathbf{x})$.

Естественно, что в самом лучшем случае метод может дать точку, близкую к одному из локальных минимумов $F(\mathbf{x})$.

2.5.2. Методы барьерных функций

Недостаток метода внешней точки в том, что обычно не удается с абсолютной точностью выполнить все огра-

ничения. От этого недостатка свободны методы *внутренней* точки, иначе называемые методами *барьерных* функций. В этих методах также строится последовательность точек минимума некоторых функций, которая сходится к точке минимума исходной целевой функции, но не извне, а изнутри ОДР. Добавки к исходной целевой функции служат барьерами, удерживающими точку внутри допустимой области. В качестве барьерной может использоваться функция

$$S(\mathbf{x}) = \sum_{j=1}^m 1/c_j(\mathbf{x}),$$

и в целом преобразованная функция имеет вид

$$\Phi(\mathbf{x}) = F(\mathbf{x}) - k_n S(\mathbf{x}) = F(\mathbf{x}) - k_n \sum_{j=1}^m 1/c_j(\mathbf{x}). \quad (2.14)$$

Начальная точка должна быть внутренней точкой ОДР, где все $c_j(\mathbf{x}) < 0$. Когда она приближается к границе, к функции $F(\mathbf{x})$ прибавляется большая положительная величина. Последовательность коэффициентов k_n в методе барьерных функций стремится не к бесконечности, а к нулю, начиная с достаточно больших положительных значений.

Метод барьерных функций применяется при решении задач, в которых целевые функции за пределами допустимой области не определены или их трудно вычислить. Кроме того, он предпочтителен, когда не нужна высокая точность достижения минимума, но важно выдержать ограничения.

Одним из эффективных методов решения задач с нелинейными ограничениями является разработанный в 70-х гг. XX в. метод модифицированной функции Ла-

гранжа (МФЛ). Метод базируется на той же идее штрафных функций, но вместо одного параметра k_n в формуле (2.13) вводится несколько параметров, управляя которыми по ходу решения задачи можно улучшить сходимость. Существует несколько вариантов метода, отличающихся видом штрафной функции и методами изменения ее параметров. При решении практических задач хорошие результаты были получены по алгоритму Пауэлла (Powell M.J.D.), в котором для поиска минимума исходной целевой функции $F(x)$ при ограничениях $c_j(x) \leq 0$ используется функция

$$\Phi(x, \sigma, \theta) = F(x) + \frac{1}{2} \sum_{j=1}^m \sigma_j (c_j(x) + \theta_j)_+^2.$$

Векторы σ и θ имеют по m компонент (по числу ограничений) и представляют собой набор параметров штрафной функции, которая отличается от рассмотренной выше тем, что вместо одного параметра теперь два параметра на каждое ограничение. Знак $+$ означает, что в сумму включаются только те слагаемые, для которых $c_j(x) + \theta_j > 0$. Здесь $\theta_j > 0$ — «запас» в j -ом ограничении, т.е. штрафуются не только настоящее нарушение при $c_j(x) > 0$, но и $c_j(x) > -\theta_j$.

Если в системе ограничений были равенства, то соответствующие им слагаемые всегда присутствуют в сумме. При $\theta_j = 0$ и $\sigma_j = k$ ($j = 1, 2, \dots, m$) штрафная функция та же, что в (2.13). Недостаток (2.13) в том, что ее вторые производные по x разрывны на границе допустимой области, причем разрывы тем больше, чем больше k , который приходится увеличивать в каждом новом итерационном цикле безусловной минимизации. Иное дело, когда σ_j постоянны, а варьируются только θ_j . В этом случае поверхности разрывов вторых производных удалены от точек безусловно-

го минимума, определяемых при решении задач в каждом цикле оптимизации [5,18].

Начальные значения параметров $\theta_j > 0$ и $\sigma_j > 0$ выбирают, ориентируясь на смысл и важность соответствующих ограничений и величины невязок в ограничениях в точке начального приближения. Затем решается задача на безусловный минимум $\Phi(\mathbf{x}, \boldsymbol{\sigma}, \boldsymbol{\theta})$. Наилучшие результаты для задач с нелинейными ограничениями были получены при использовании методов второго порядка с последовательным формированием обратной матрицы Гессе, т.е. DFP-процедуры (см. раздел 2.3.5). При этом на начальных этапах не требуется высокая точность поиска безусловного минимума. После того как задача на безусловный минимум решена, проверяются ограничения и, если есть нарушения, меняются параметры $\boldsymbol{\sigma}$ и $\boldsymbol{\theta}$. При этом используется правило: если j -ое ограничение выполнено с «запасом», т.е. $c_j(\mathbf{x}) > -\theta_j$, то новое значение $\theta_j^1 = 0$, а если нет, то $\theta_j^1 = \theta_j + c_j(\mathbf{x})$. И так по всем ограничениям.

Для пересчета σ_j действует другое правило: если в j -ом ограничении в результате решения задачи на безусловный минимум невязка уменьшается быстро, то σ_j не меняется, а если медленно, то увеличивается. Обычно используются такие константы: если невязка уменьшилась меньше, чем в четыре раза, то соответствующее σ_j умножается на 10 и при этом θ_j делится на 10.

После пересчета параметров повторяется процесс безусловной минимизации или, как говорят, делается очередная внешняя итерация. Счет прекращается в следующих случаях:

1. Получено решение с приемлемыми невязками. При этом можно для контроля сделать еще одну внешнюю итерацию и убедиться, что решение практически не изменилось.
2. После исчерпания заданного лимита внешних итераций решение не получено. При этом есть все основа-

ния усомниться в совместности системы ограничений и, следовательно, в существовании решения исходной задачи.

2.6. Построение начального приближения

Все рассмотренные нами методы итерационные. Для них нужно начальное приближение, чтобы построить последовательность точек, сходящуюся к минимуму. Разные методы предъявляют и разные требования к начальному приближению. В методах безусловной оптимизации, как правило, можно взять любое начальное приближение, если известно, что нет локальных минимумов. В противном случае можно получить решение далеко от точки глобального минимума. Если известно и число локальных минимумов и их приближенные значения, то задачу следует решать многократно, начиная из точек, близких к очередному локальному минимуму

В задачах с ограничениями, если мы не можем по смыслу задачи указать хотя бы одну допустимую точку, можно прибегнуть к решению вспомогательной задачи. Вспомним, что именно так строилось начальное приближение в задаче линейного программирования (см. раздел 1.6). Кстати, при линейных ограничениях можно использовать в точности тот же прием и получить вершину допустимой области или установить несовместность системы ограничений.

Для построения начального приближения можно использовать и метод штрафных функций, исключив из рассмотрения исходную целевую функцию и минимизируя просто сумму квадратов невязок в ограничениях.

$$\sum_{j=1}^m g_j(x) \rightarrow \min$$

Для этого задачу достаточно решить один раз. При этом можно получить не только граничную, но и внутреннюю точку, если ужесточить ограничения, приняв вместо $c_j(x) \leq 0$ $c_j(x) \leq -\varepsilon$.

На практике встречаются задачи, в которых все ограничения можно разделить на две группы, так что для части ограничений (группа 1) легко указать допустимую точку, но она не удовлетворяет другим ограничениям (группа 2). Тогда в штрафную функцию можно включить квадраты невязок только в ограничениях группы 2, взять начальное приближение, удовлетворяющее всем остальным ограничениям, и решать задачу минимизации штрафной функции при ограничениях группы 1. Решение этой вспомогательной задачи будет начальным приближением для исходной задачи. Такой прием оказывается эффективным, если мы умеем хорошо работать с ограничениями группы 1, например, при линейных ограничениях можем легко вычислять соответствующие проекции.

2.7. Практическая реализация методов нелинейного программирования

Для решения практической задачи оптимизации создается ее математическая модель. При этом стремятся обеспечить ее максимальное приближение к действительности, а вопрос о том, как будут проводиться оптимизационные расчеты, откладывается до момента, когда модель построена. При этом часто оказывается, что среди известных алгоритмов оптимизации нет подходящего, который позволил бы эффективно решить полученную математическую задачу. С другой стороны, желание облегчить поиск оптимума не может оправдать чрезмерные упрощения модели. В особенности это касается стремления путем линеаризации све-

сти все к линейному программированию как наиболее изученному классу задач оптимизации. При этом часто приходится для обеспечения приемлемой точности линеаризации резко увеличивать размерность задачи. При решении вопроса о допустимых упрощениях целевой функции и ограничений необходимо учитывать точность и достоверность исходных данных.

Прежде чем приступить к решению задачи при уже построенной математической модели, эту модель нужно проанализировать с точки зрения организации будущего вычислительного процесса и по возможности предотвратить его осложнения. Например, целевая функция выражена квадратичной формой с диагональной матрицей:

$$F(\mathbf{x}) = \sum_{i=1}^n k_i x_i^2, \quad k_i > 0, \quad i = 1, 2, \dots, n.$$

Если численные значения k_i существенно различны, то поверхности уровня функции $F(\mathbf{x})$ вытянуты по тем координатным осям, которым соответствуют малые значения k_i . Возникает овражность, существенно замедляющая сходимость многих алгоритмов оптимизации. В данном случае этой овражности легко избежать с помощью замены переменных $y_i = k_i^{1/2} x_i$. После этой замены целевая функция превращается в сумму квадратов, а поверхности уровня в сферы. При наличии ограничений в них тоже необходимо произвести замену переменных.

Для функций произвольного вида вместо k_i можно взять приближенные значения вторых производных $\partial^2 F(\mathbf{x}) / \partial x_i^2$, если эти производные легко доступны для вычисления и не сильно меняются в допустимой области. При этом не получим сферических поверхностей, но существенно уменьшим овражность.

Масштабирование заменой переменных с вычислительной точки зрения имеет своей целью добиться, чтобы в области поиска приходилось иметь дело с переменными, абсолютные величины которых имеют одинаковые порядки. Если в области поиска модули переменных существенно меняются, то такая замена ненадежна.

Если для каждой переменной известен диапазон изменения, например в явном виде заданы ограничения $a_i \leq x_i \leq b_i$, то имеет смысл, как говорят, «отмасштабировать и центрировать переменные». Это означает переход к новым переменным y_i по формуле

$$y_i = 2x_i / (b_i - a_i) - (a_i + b_i) / (b_i - a_i).$$

После этой замены все новые переменные y_i окажутся на отрезке $[-1, 1]$. Иногда такое преобразование называют переходом к безразмерным центрированным величинам. После оптимизации по y_i исходные переменные x_i определяются с помощью обратного преобразования:

$$x_i = \frac{1}{2}(b_i - a_i)y_i + \frac{1}{2}(b_i + a_i).$$

Или в матричном виде $\mathbf{x} = \mathbf{D}\mathbf{y} + \mathbf{c}$, где \mathbf{D} — диагональная матрица с элементами $d_{ii} = 1/2 (b_i - a_i)$, а \mathbf{c} — вектор с компонентами $c_i = 1/2 (b_i + a_i)$. Если градиент по старым переменным x_i обозначить как обычно \mathbf{g} , то градиент по новым переменным $\mathbf{g}_y = \mathbf{D}\mathbf{g}$.

Существенные сложности при решении задач оптимизации возникают при наличии негладких функций. Разработанные в недавнем прошлом специальные методы негладкой оптимизации достаточно сложны. В то же время в практических целях функции с разрывными производными часто могут с успехом быть заменены функциями, имеющими

непрерывные производные. Например, непрерывные кусочно-линейные функции, графиком которых являются ломаные линии, с помощью параболической аппроксимации в окрестности точек разрыва производной могут быть сколь угодно точно заменены функциями, имеющими непрерывные производные (рис. 28).

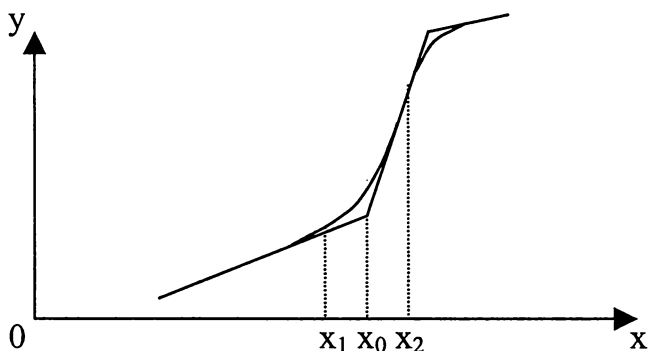


Рис. 28. Параболическая аппроксимация

Если два смежных звена ломаной имеют уравнения $y = k_1x + d_1$ и $y = k_2x + d_2$, то в окрестности точки пересечения в образовавшийся угол можно вписать параболу $y = ax^2 + bx + c$. Неизвестные коэффициенты a, b, c можно найти из условия касания параболы и звеньев ломаной в точках x_1 и x_2 . Получаем систему из четырех уравнений относительно пяти неизвестных a, b, c, x_1, x_2 .

$$\begin{cases} ax_1^2 + bx_1 + c = k_1x_1 + d_1 \\ ax_2^2 + bx_2 + c = k_2x_2 + d_2 \\ 2ax_1 + b = k_1 \\ 2ax_2 + b = k_2 \end{cases}$$

Если обозначить через δ длину интервала аппроксимации, то $x_2 = x_1 + \delta$ и, решая систему уравнений, выразим все неизвестные через δ . Получим

$$\begin{cases} a = (k_2 - k_1)/(2\delta); \\ x_1 = (d_1 - d_2)/(k_2 - k_1) - \delta/2; \\ b = (k_1 x_2 - k_2 x_1)/\delta; \\ c = k_1 x_1 + d_1 - a x_1^2 - b x_1. \end{cases}$$

Аналогично можно аппроксимировать и кусочно-квадратическую функцию, вписав в окрестности точки пересечения двух парабол третью параболу, сохраняя при этом непрерывность первой производной. Длину интервала аппроксимации не следует сразу брать очень малой величиной. Лучше ее уменьшать в процессе счета, разбивая его на ряд этапов. Дело в том, что при малых δ получаются большие значения параметра a , т.е. большие скачки второй производной при переходе с прямой на параболу или с одной параболы на другую, что замедляет сходимость градиентных методов оптимизации. Это постепенное повышение точности аппроксимации аналогично поэтапному увеличению параметра штрафной функции k_n в формуле (2.13).

В программных реализациях стремятся избежать работы с величинами, численные значения которых различаются на много порядков, чтобы уменьшить влияние ошибок представления чисел в компьютере. Поэтому, в частности, формула (2.13) для метода штрафных функций переписывается в виде:

$$\Phi(\mathbf{x}) = F(\mathbf{x})/k_n + \sum_{j=1}^m g_j(\mathbf{x})$$

и вблизи точки безусловного минимума при больших k_n все слагаемые стремятся к нулю.

Существенных упрощений процесса оптимизации в ряде случаев можно добиться преобразованием целевой функции и ограничений при введении дополнительных переменных. Наиболее наглядно это проявляется в задаче минимизации суммы модулей линейных функций при линейных ограничениях. Задача имеет вид: найти $\min \sum |\eta_i(\mathbf{x})|$ при ограничениях $\mathbf{Ax} \leq \mathbf{b}$. Здесь

$$\eta_i(\mathbf{x}) = c_{i1}x_1 + c_{i2}x_2 + \dots + c_{in}x_n + c_{i0}, \quad (i = 1, 2, \dots, m).$$

Дополнительные переменные y_i вводятся так: $|\eta_i(\mathbf{x})| \leq y_i$. При этом появляются $2m$ дополнительных линейных неравенств:

$$y_i + \eta_i(\mathbf{x}) \geq 0, \text{ и } y_i - \eta_i(\mathbf{x}) \geq 0$$

относительно n исходных и m дополнительных переменных, но целевая функция приобретает простой вид $\sum y_i$, так что исходная задача с негладкими функциями сводится к задаче линейного программирования.

В практических задачах могут встречаться требования минимизации не суммы, а максимальной по абсолютной величине функции, т.е. $\min_x \max_i |\eta_i(\mathbf{x})|$ при тех же ограничениях. Это задача чебышевского приближения системы линейных функций, названная по имени великого русского математика П.Л. Чебышева. В этом случае вводится только одна дополнительная переменная y . Но число дополнительных ограничений $y + \eta_i(\mathbf{x}) \geq 0$ и $y - \eta_i(\mathbf{x}) \geq 0$ равно $2m$. Снова получается задача линейного программирования с целевой функцией просто y .

2.8. Примеры задач, решаемых с применением методов нелинейного программирования

1. Решение совместных систем линейных алгебраических уравнений ($n \times n$). При большом числе уравнений решение системы организуется в виде итерационного процесса. Если записать систему как $Ax - b = 0$, то поиск ее решения эквивалентен поиску точки минимума в задаче:

$$(Ax - b, Ax - b) \rightarrow \min.$$

Это задача квадратического программирования без ограничений с матрицей квадратичной формы $Q = A^T A$ и линейной формой $-2(b, Ax)$.

В целом задача сводится к минимизации $1/2 (Qx, x) - (A^T b, x)$, что может быть выполнено одним из методов безусловной минимизации, например, методом сопряженных градиентов.

Если дополнительно известно, что исходная матрица A — симметричная положительно определенная, то целевая функция может быть записана в виде $1/2 (Ax, x) - (b, x)$, так как точка минимума этой функции x^* удовлетворяет условию $Ax^* - b = 0$. Это есть условие равенства нулю градиента квадратичной формы.

2. Решение произвольных систем алгебраических уравнений.

$$f_i(x_1, x_2, \dots, x_n) = 0, (i = 1, 2, \dots, m).$$

Целевую функцию запишем в виде:

$$F(x) = \sum_{i=1}^m (f_i(x))^2.$$

Если исходная система имеет единственное решение, то целевая функция имеет единственный минимум, равный нулю.

3. *Задачи уравнивания.* Предположим, мы хотим получить в эксперименте значения некоторых величин x_i ($i = 1, 2, \dots, n$). Непосредственно выполнить измерения нужных величин не удастся, и вместо этого измеряются другие величины y_j ($j = 1, 2, \dots, m$), связанные с исходными линейными зависимостями $y = Ax + b$. Если измерения абсолютно точные, то достаточно замерить столько величин, сколько неизвестных ($m = n$), и решить полученную систему уравнений для определения x . Но измерения не бывают абсолютно точными, поэтому для уменьшения влияния погрешностей измерений их число $m > n$ и в системе больше уравнений, чем неизвестных. При любых значениях переменных x_i правые части системы $Ax + b - y = 0$ не могут одновременно обращаться в нули. Возникают отклонения (невязки), которые различны при различных значениях x_i ($i = 1, 2, \dots, n$). Искомое решение (вектор x) в некотором смысле должно соответствовать малым невязкам, так как получить все невязки равные нулю невозможно.

Если за критерий малости невязок брать сумму квадратов невязок, то задача уравнивания сводится к задаче квадратического программирования: $(Ax + b - y, Ax + b - y) \rightarrow \min$. Поскольку матрица A и векторы b и y известны, то можно считать, что известна и матрица квадратичной формы $Q = A^T A$ и вектор линейной формы $2A^T(b - y)$. В итоге получаем задачу безусловной минимизации функции:

$$F(x) = 1/2(A^T A x, x) + (A^T(b - y), x).$$

Постоянные слагаемые, как не влияющие на поиск минимума, исключены.

Если измерения не являются равноточными, то невязки имеют разный вес. Более точные измерения должны иметь больший вес. Таким образом, каждая строка матрицы A умножается на свой весовой коэффициент, после чего преобразованная матрица A участвует в формировании матрицы квадратичной формы. Возможно наличие таких равенств, которые должны выполняться точно. Например, сумма углов треугольника должна быть равна 180. Такое условие нет смысла включать в систему уравнений. Оно должно рассматриваться как ограничение. При наличии таких условий задача превращается в задачу квадратического программирования с ограничениями.

Можно искать решение из условия минимума не суммы квадратов, а суммы модулей невязок. Тогда эта задача сводится к задаче линейного программирования (см. раздел 2.7). Можно минимизировать и максимальную невязку, т.е. искать чебышевское приближение. Снова будет получена задача линейного программирования (см. раздел 2.7).

4. *Задача о защите поверхности.* Вернемся к рассмотренной ранее задаче о покрытии каждого из m элементов поверхности одним из n материалов (см. раздел 1.9, задача 2). Будем считать заданными матрицу затрат с элементами c_{ij} и матрицу потерь d_{ij} , а также допустимые суммарные потери d . Мы не будем вводить целочисленные переменные x_{ij} , как в разделе 1.9, и сводить задачу к линейному программированию, а постараемся уменьшить размерность задачи.

Для каждого элемента расположим материалы в порядке убывания их стоимости (c_i) и, следовательно, возрастания потерь (d_i) от недостаточной защиты поверхности (рис. 29). Получим m графиков, на каждом n точек. Если большей цене соответствуют большие потери, то соответствующий способ явно непригоден для данного элемента.

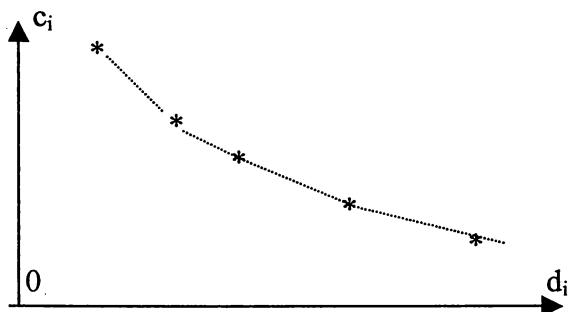


Рис. 29. Ценовой график для i -го элемента

Забудем о том, что у нас только несколько способов покрытия, и будем рассматривать зависимость $c_i(d_i)$ как кусочно-линейную. Это монотонно убывающая функция.

В качестве неизвестных выберем d_i ($i = 1, 2, \dots, m$), т.е. потери на i -ом элементе. Для каждого элемента известны пределы изменения d_i , что дает простые ограничения $d_i^1 \leq d_i \leq d_i^2$. Дополнительно имеем ограничение на суммарные потери $\sum_i d_i \leq d$. Других ограничений нет.

Целевая функция задачи $\sum_i c_i(d_i)$.

Здесь $i = 1, 2, \dots, m$ — номер элемента, а $c_i(d_i)$ — затраты при защите его избранным способом. Получили задачу нелинейного программирования, в которой не $m \times n$, а только m переменных (по числу элементов). Простой вид ограничений позволяет легко вычислять проекцию градиента и решать вопрос об исключении ограничений из активного набора (см. раздел 2.4.2.1, примеры 1, 2 построения проекции). Целевая функция не имеет локальных минимумов, и в зависимости от конкретных данных с ней можно работать по-разному. Во-первых, можно рассматривать ее как кусочно-линейную и с помощью параболической аппроксимации

сколь угодно точно приблизить все $c_i(d_i)$ -функциями, имеющими непрерывные производные (см. раздел 2.7, рис. 28). Во-вторых, можно подобрать гладкие функции, аппроксимирующие $c_i(d_i)$ сначала на всем интервале $d_i^1 \leq d_i \leq d_i^2$, а затем, по мере приближения к минимуму, на все более малых интервалах.

Существенным недостатком такой модели является то, что она не учитывает дискретность переменных. Но результат минимизации может быть хорошим начальным приближением для применения методов, учитывающих дискретность. Этот результат ни в чем не уступает решению задачи с помощью линейного программирования без учета целочисленности.

5. *Проектирование оптимальных трасс линейных сооружений.* Линейные сооружения (железные и автомобильные дороги, каналы, магистральные трубопроводы, траншеи для водо- и газоснабжения и др.) — это такие сооружения, положение которых на местности определяется *линией (трассой)*, которая совпадает с осью сооружения. Трасса — это трехмерная кривая, удовлетворяющая целому ряду ограничений. Традиционно эта трехмерная кривая представляется в виде двух плоских кривых: план и продольный профиль (рис. 30). *План* — это проекция трассы на координатную плоскость $ХОУ$, а *продольный профиль* — это зависимость аппликаты z от длины дуги в плане. Продольный профиль получается при развертке на плоскость цилиндрической вертикальной поверхности, проходящей через трассу. У этой поверхности образующая параллельна оси OZ , а направляющая — это план трассы. Вертикальная цилиндрическая поверхность пересекает поверхность земли, и при развертке на плоскость линия пересечения превращается в продольный профиль земли по трассе. Таким образом, получается профиль земли и профиль трассы или исходный и проектный профиль соответственно. Задача проектирования оптимальной трассы превращается в две взаимосвязанные задачи: проектирование плана и проектирование продольного профиля. Обычно ва-

рианты плана трассы намечаются вручную, по каждому из них строят продольный профиль земли и по нему проектируют продольный профиль трассы. На положение трассы влияют рельеф земли, геологические, гидрологические, климатические и другие условия. Оптимальному варианту должны соответствовать минимальные затраты на строительство и последующую эксплуатацию сооружения.

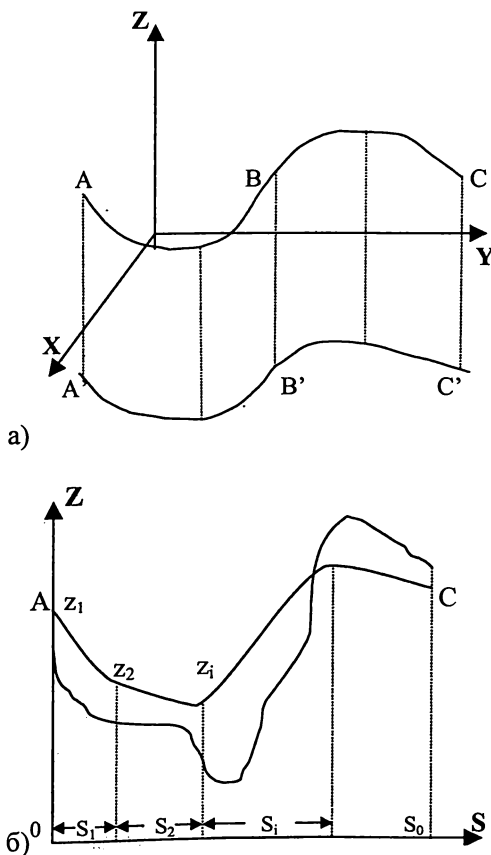


Рис. 30. Представление трассы: а) план, б) продольный профиль

В зависимости от вида сооружения предъявляются различные требования к плану и профилю его трассы. Применительно к одному из самых сложных линейных сооружений — трассе железной дороги эти требования включают следующее.

В плане. Элементами плана трассы могут быть отрезки прямых и окружностей, сопрягаемые клотоидами, с тем чтобы обеспечить непрерывность не только первых, но и вторых производных (точнее, кривизны трассы) для плавности и безопасности движения поездов. При этом длины элементов должны быть не менее некоторых величин, а радиусы кривых и параметры клотоид тоже ограничены. Соответствующие ограничения на план трассы выражаются нелинейными неравенствами относительно переменных, определяющих план. При этом в каждом конкретном случае, вообще говоря, неизвестно число элементов плана, т.е. размерность задачи.

В продольном профиле. Элементы продольного профиля прямые, так что проектная линия ломаная, на элементы которой также накладываются ограничения. Если обозначить профиль земли $H(s)$, а проектную линию $Z(s)$, то в первом приближении задача состоит в следующем. По заданной $H(s)$ найти такую ломаную $Z(s)$, чтобы она удовлетворяла всем ограничениям, и был

$$\min \int_0^{S_0} F(H(s) - Z(s)) ds. \quad (2.16)$$

где S_0 — длина трассы в плане, а функция F моделирует затраты на элементе длины.

Эта задача поиска одной функции по заданной другой функции есть задача вариационного исчисления, но не математического программирования. Однако она сводится

к задаче нелинейного программирования, обладающей интересными особенностями не зависимо от конкретного вида функции F .

Поскольку число элементов исходной ломаной неизвестно, то приходится считать, что переломы профиля земли и проектной линии (т.е. профиля трассы) имеют одни и те же абсциссы (S_i). Профиль земли всегда представлен в виде ломаной с неравномерным шагом, и такое допущение позволяет фиксировать количество элементов n (размерность задачи) и длины s_i элементов (в плане). При этом получится ломаная с большим, чем нужно, числом элементов, но из-за многочисленных ограничений ее отклонения от окончательной $Z(s)$ невелики. Идея в том, чтобы найти эту ломаную путем решения задачи оптимизации, затем преобразовать ее в ломаную с элементами, длина которых не менее допустимой, определив тем самым реальную размерность задачи и начальное приближение, и на последнем этапе выполнить оптимизацию при всех ограничениях и необходимых уточнениях целевой функции. Такой многоэтапный процесс с уточнением математической модели и ее параметров является обычным для решения сложных проектных задач творческого характера.

Зная число и длины элементов искомой ломаной, можно аналитически выразить все ограничения на $Z(s)$, если принять в качестве неизвестных z_i ($i = 1, 2, \dots, n$) ее ординаты в точках перелома. Эти ограничения делятся на три группы.

1. На ординаты в отдельных точках $z_i \leq z_i^{\max}$ или $z_i \geq z_i^{\min}$.

2. На уклоны элементов профиля

$$a_i \leq (z_{i+1} - z_i) / s_i \leq b_i \quad (i = 1, 2, \dots, n-1).$$

Здесь s_i — длины элементов. Это ограничение является дискретным аналогом ограничения на первую производную.

3. На разности уклонов смежных элементов:

$$c_i \leq (z_{i+2} - z_{i+1})/s_{i+1} - (z_{i+1} - z_i)/s_i \leq d_i.$$

Это ограничение является дискретным аналогом ограничения на кривизну в профиле.

Интеграл в формуле (2.16) превращается в сумму интегралов по элементам профиля, и, поскольку на каждом i -ом элементе аргумент подинтегральной функции F полностью определяется переменными z_i и z_{i+1} , появляется возможность вычисления производных целевой функции по неизвестным z_i , т.е. градиента.

Система ограничений имеет четко выраженную структуру. Именно эта структура рассмотрена нами в разделе 2.4.2.1 (примеры 1, 4, 5). Там были даны формулы для вычисления проекции на соответствующие подпространства, определяемые активными ограничениями. Реально возможны и комбинации активных ограничений из этих трех групп. Например, на участке, где активны ограничения группы 2 (трасса идет предельным уклоном), одновременно активно одно ограничение группы 1 (нельзя изменить соответствующее z_i). Становится очевидным, что все компоненты проекции антиградиента для этого участка должны быть равны нулю. Действительно, ограничения по уклону (группа 2) делают все компоненты проекции равными друг другу (сдвиг), а высотное ограничение (группа 1) требует равенства соответствующей компоненты нулю (иначе изменится z_i и ограничение или будет нарушено, или перестанет быть активным). Значит, и все компоненты проекции на этом участке равны нулю.

Далее, наличие одного активного ограничения группы 2 на участке, где активны все ограничения группы 3, приводит к тому, что все компоненты проекций на этом участке должны быть равны между собой. Только при этом условии

сохранят активность все ограничения. Выясняется, что ограничения группы 2 для сохранения активности (т.е. для построения проекции) допускают только изменение всех переменных на одну и ту же величину (сдвиг). Только при этом условии уклоны всех элементов остаются предельными. А ограничения на разность уклонов (группа 3), кроме сдвига, допускают еще и изменение всех уклонов на одну и ту же величину (поворот). Только при этих трансформациях проектной линии сохраняется разность уклонов. Именно эти изменения проектной линии при соответствующей комбинации активных ограничений находят отражение в структуре базисных векторов (см. раздел 2.8).

Возможны и более сложные комбинации активных ограничений. Например, два или более участков, на каждом из которых активны ограничения по разности уклонов (группа 3), стыкуются в точках, в которых разность уклонов не достигает предельного значения (точки С и D на рис. 31).

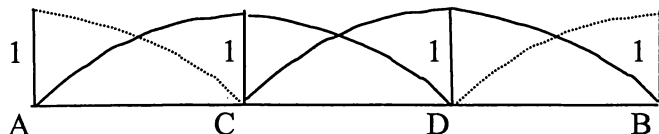


Рис. 31. Построение базисных векторов при трех участках предельной разности уклонов

Независимо от длины каждого из участков предельной разности уклонов, т.е. от числа переменных, входящих в соответствующую систему активных ограничений, размерность нуль-пространства матрицы активных ограничений, т.е. число базисных векторов, равно числу таких участков плюс 1 или, иначе, числу свободных точек, включая крайние. Матрица активных ограничений в данном случае состоит из

трех блоков (по числу участков), каждый из которых трех-диагональный (см. раздел 2.4.2.1, пример 5) Схематически структура матрицы представлена на рис. 32. Элементы матрицы обозначены точками, а блоки прямоугольниками. Все прочие элементы равны нулю. В пределах каждого блока при переходе к следующей строке происходит сдвиг положения ненулевых элементов вправо на 1, а при переходе к новому блоку — на 2.

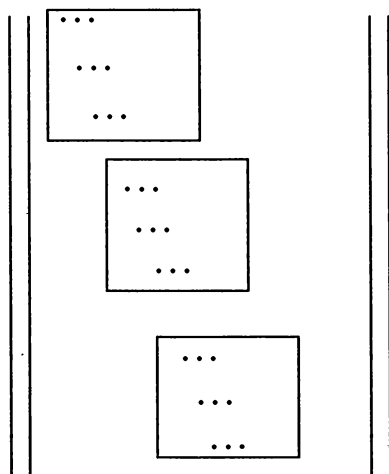


Рис. 32. Структура матрицы активных ограничений

Базисные векторы будем выбирать так, чтобы каждый из них имел как можно больше нулевых компонент. Другими словами, будем стараться сделать так, чтобы движение вдоль базисного вектора затрагивало один или два участка.

Первый базисный вектор соответствует повороту участка АС с центром поворота в точке С (рис. 31). Зная длины всех входящих в участок АС элементов, легко вычислить угол поворота (приращение уклонов всех эле-

ментов), при котором смещение точки А равно 1. Изменением длин элементов при изменении их уклонов можно пренебречь из-за малости уклонов. Считается, что длина элементов в профиле и плане совпадают.

Пронумеруем длины элементов, входящих в рассматриваемый участок, начиная от центра поворота (s_1, s_2, \dots, s_k). Приращение уклонов обозначим через δ . Тогда смещения концов элементов на данном участке, начиная от центра поворота, будут следующими:

$$0, \delta s_1, \delta(s_1 + s_2), \delta(s_1 + s_2 + s_3), \delta(s_1 + s_2 + \dots + s_k).$$

Так как смещение конечной точки участка поворота равно 1, то

$$\delta = 1 / (s_1 + s_2 + \dots + s_k).$$

Базисный вектор построен.

Второй базисный вектор соответствует такой комбинации двух поворотов с центрами в точках А и D, при которой смещение в точке С равно 1. Третий базисный вектор соответствует аналогичной комбинации поворотов с центрами в точках С и В и единичным смещением в точке D. Наконец, четвертый базисный вектор соответствует повороту с центром в точке В и единичным смещением в точке D. Знание базисных векторов позволяет построить проекцию градиента, решая систему уравнений малой размерности (по числу базисных векторов). В нашем примере участок может охватывать несколько десятков элементов, а базисных векторов всего четыре. Более того, знание базисных векторов позволяет вычислить приведенный градиент и вообще обойтись без решения системы линейных уравнений для поиска направления спуска и исключения ограничений из активного набора (см. раздел 2.4.2.2).

Структурные особенности системы ограничений позволили еще в 70-х гг. создать и реализовать на маломощных, с современных позиций, ЭВМ, таких как Минск-32 и ЕС-1022, эффективный алгоритм решения задачи об оптимальном профиле дороги при числе переменных $n = 200$ и числе ограничений более 800. На современных компьютерах (Pentium 2 с тактовой частотой 200 мегагерц) соответствующие программы решают задачу об оптимальном профиле дороги при числе переменных $n = 1000$ (больше в реальных задачах не требуется) и соответственно числе ограничений более 4000 за несколько минут.

Аналогично были решены задачи проектирования продольного профиля других линейных объектов (автомобильные дороги, трубопроводы, оросительные каналы и др.). Подсистема проектирования продольного профиля была встроена в САПР линейных сооружений, что позволило автоматизировать целый комплекс проектных задач, а не только проектирования плана и профиля.

Оптимизация трассы как пространственной кривой, т.е. совместное проектирование плана и профиля, осложняется нелинейностью системы ограничений. Однако смысл этих ограничений тот же. Это позволило и в нелинейном случае построить такие трансформации трассы, которые сохраняют активность ограничений (сдвиги и повороты в плане и в профиле) и построить траекторию спуска на поверхности, соответствующей целевой функции, не выходя за пределы допустимой области. Мы не будем углубляться здесь в детали метода оптимизации трассы.

Данный пример свидетельствует: если математическая модель правильно отражает реальность, то физический смысл задачи может подсказать существенные упрощения алгоритмов оптимизации и создать программное обеспечение, оставаясь в рамках математически обоснованных методов.

2.9. Обучающая компьютерная программа ROZEN.exe

Обучающая программа предназначена для оказания помощи в освоении основных идей нелинейного программирования и может рассматриваться как вспомогательное средство для изучения методов нелинейного программирования. Основная идея — дать образное представление об основных положениях теории нелинейного программирования и методах решения соответствующих задач.

Программа разъясняет общую постановку задачи и дает классификацию задач нелинейного программирования по различным признакам:

- ◆ по наличию производных;
- ◆ по виду целевой функции;
- ◆ по наличию локальных экстремумов.

Далее программа переходит к определению понятия «выпуклая функция» и предлагает набор тестовых вопросов о связи выпуклости целевой функции и допустимой области с наличием локальных минимумов. На различные варианты даются примеры взаимного положения линий уровня целевой функции и допустимой области.

Обсуждается вопрос о возможности положения точки минимума внутри и на границе допустимой области и графически показываются соответствующие варианты. Тем самым подчеркивается существенное отличие от линейного программирования.

Далее программа разъясняет смысл методов покоординатного спуска, наискорейшего спуска и сопряженных градиентов.

Для иллюстрации эффективности методов в различных условиях и зависимости ее от свойств целевой функции программа предлагает задать данные для минимизации квадратичной формы с трехдиагональной матрицей.

$$F(x) = (x_1 + x_2)^2 / a_1 + (x_1 - x_2)^2 / a_2 + (x_2 + x_3)^2 / a_3 + \\ + (x_2 - x_3)^2 / a_4 + \dots + (x_{n-1} + x_n)^2 / a_{2n-3} + \\ + (x_{n-1} - x_n)^2 / a_{2n-2} + c_1 x_1 + c_2 x_2 + \dots + c_n x_n$$

При различных положительных величинах $a_1, a_2, \dots, a_{2n-2}$ можно смоделировать различные свойства целевой функции, включая овражную ситуацию.

Обучающая программа позволяет выполнить оптимизацию из различных начальных точек при различных значениях входных данных по различным алгоритмам. Это дает возможность реально представить, как влияет число переменных на время счета при оптимизации различными методами, как влияет овражность целевой функции, близость начального приближения к оптимуму, наличие линейной формы и т.д. Для двумерного случая программа демонстрирует графически процесс оптимизации. Для многомерного случая можно проследить процесс оптимизации по выдаваемым программой данным о значении целевой функции, норме (длине) градиента и величине шага.

Задание

1. С помощью программы **ROZEN.exe** решить задачу квадратического программирования при $a_1 = a_2 = \dots = a_{2n-2} = 1$ и $b_1 = b_2 = \dots = b_n = 0$ из различных начальных точек по методу наискорейшего спуска. Как зависит число итераций от размерности задачи и от начального приближения? Результат объяснить.

2. С помощью программы **ROZEN.exe** решить ту же задачу квадратического программирования при различных значениях a_i и $b_1 = b_2 = \dots = b_n = 0$ из различных начальных точек по методу наискорейшего спуска. Например, задать

все a_i , кроме a_{2n-2} , по единице и $a_{2n-2} = 100$, а затем и 10 000. Как изменилось число итераций и почему?

3. Повторить решение задач пунктов 1 и 2, но по методу сопряженных градиентов. Сопоставить результаты и число итераций.

4. Исследовать экспериментально, как влияет наличие линейной формы на результат и время счета.

Контрольные вопросы и задачи

1. Доказать, что локальный минимум строго выпуклой функции является ее глобальным минимумом, а значения нестрого выпуклой функции во всех точках минимума равны.

2. Можно ли использовать метод покоординатного спуска при наличии линейных ограничений общего вида? Если нет, то почему? Привести пример.

3. Если в задаче с линейными ограничениями-неравенствами с помощью дополнительных переменных преобразовать все неравенства в равенства, то можно ли ее решить с помощью множителей Лагранжа?

4. Не подозревая о том, что решением задач с линейными ограничениями-неравенствами является внутренняя точка допустимой области, за начальное приближение приняли вершину ОДР и стали искать минимум по методу проекции градиента. Будет ли в этом случае получено правильное решение, или процесс остановится на границе ОДР?

5. Можно ли использовать метод проекции градиента, если все ограничения записаны в виде линейных равенств?

6. Можно ли использовать метод проекции градиента, если система ограничений включает линейные равенства и неравенства (смешанная система)?

7. Нужно ли для применения метода проекции градиента приводить смешанную систему линейных ограничений к одной из основных форм?

8. Можно ли использовать метод сопряженных градиентов, если все ограничения записаны в виде линейных равенств?

9. Построить базис в нуль-пространстве матрицы системы трех активных ограничений:

$$\begin{cases} x_2 - x_1 \leq b_1; \\ x_3 - x_2 \leq b_2; \\ x_5 - x_4 \leq b_3. \end{cases}$$

10. Спроектировать на подпространство предыдущей задачи вектор-антиградиент с координатами (1, 2, -6, 4, 6).

11. Построить нормаль и определить, какие ограничения можно исключить из активного набора, пересчитать проекцию антиградиента.

12. Найти приведенный антиградиент для той же системы и определить, какие ограничения можно исключить из активного набора. Пересчитать приведенный антиградиент.

13. Решить задачи 9, 10, 11, 12 для системы двух ограничений

$$\begin{cases} x_3 - 2x_2 + x_1 = b_1 \\ x_5 - 2x_4 + x_3 = b_2 \end{cases}$$

14. Как влияют свободные члены системы активных ограничений на построение проекции градиента и приведенного антиградиента? Ответ объяснить.

15. В E_3 вычислить матрицу проектирования на подпространство $x_3 = 0$.

16. Какие собственные числа имеет матрица проектирования?

17. Если P — матрица проектирования на некоторое подпространство, то может ли матрица $2P$ быть матрицей проектирования на какое-то подпространство?

3. ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

В данном разделе рассмотрены задачи оптимизации, возникающие в многоэтапных (многошаговых) процессах принятия решений, изложен принцип оптимальности Р. Беллмана и проанализированы условия его применимости. Приведены примеры и практические задачи, решаемые по методу динамического программирования, в том числе и некоторые задачи, рассматривавшиеся в предыдущих разделах. Обсуждены сравнительные возможности их решения методами линейного, нелинейного и динамического программирования. Даны рекомендации по использованию обучающей компьютерной программы BELLMAN.exe, в которой реализован алгоритм динамического программирования применительно к задаче поиска наивыгоднейшего пути на двумерной сетке.

3.1. Многоэтапные процессы принятия решений

Динамическое программирование — это особый метод, наиболее эффективный при решении задач, распадающихся на ряд последовательных этапов (шагов), таких как планирование производства и инвестиций на ряд временных интервалов (лет, кварталов, месяцев), последовательность тестовых испытаний при контроле аппаратуры, поиск оптимальной траектории движения и др. В любом случае речь идет о процессах, в которых окончательное решение (план, проект, последовательность управляющих воз-

действий, обеспечивающая нужную траекторию и т.д.) вырабатывается последовательно (по шагам), причем на каждом шаге приходится решать однотипные задачи, которые существенно проще, чем решение исходной задачи в целом. В этом и состоит основная идея метода: свести решение одной сложной задачи к решению множества однотипных, иногда совсем простых задач, например выборка чисел из массива, суммирование и сравнение результатов.

Возникновение динамического программирования связано с исследованием многошаговых процессов, возникающих в теории создания запасов. Основная идея, которая привела к созданию вычислительного метода, была сформулирована в начале 50-х гг. прошлого века Р. Беллманом (R. Bellman), сделавшим самый большой вклад в развитие метода, который он назвал «динамическое программирование» [2], но который чаще называют просто методом Беллмана.

Метод не является универсальным в отличие, например, от симплекс-метода Данцига для решения задач линейного программирования. Его реализация для каждого конкретного многошагового процесса принятия решений, как правило, требует разработки нового алгоритма. Многие практические задачи, которые можно решать с помощью этого метода, были рассмотрены его автором совместно с Дрейфусом (S. Dreyfus) [3].

Первоначально метод предлагался для решения сравнительно узкого класса задач, возникающих в процессах, которые развиваются во времени. Отсюда и название: динамическое программирование, причем слово программирование скорее означало планирование, чем разработку компьютерных программ. Фактически этот метод, как и симплекс-метод Данцига, был разработан на заре «компьютерной эры», т.е. до массового применения

вычислительной техники. Тогда слово «programming» на русский язык переводилось как планирование, и метод Р. Беллмана первоначально назывался динамическое планирование.

После появления первых работ Р. Беллмана выяснилось, что для многих задач, которые не являются многоэтапными в явном виде, например рассмотренная нами в разделе 1 задача оптимального размещения оборудования и вообще многие задачи оптимизации, эту многоэтапность можно организовать искусственно и применить метод Беллмана.

Рассмотрим сначала простую задачу поиска оптимального пути на двумерной прямоугольной сетке, в которой разрешены переходы из одного узла в другой только по горизонтали (вправо) или по вертикали (вверх).

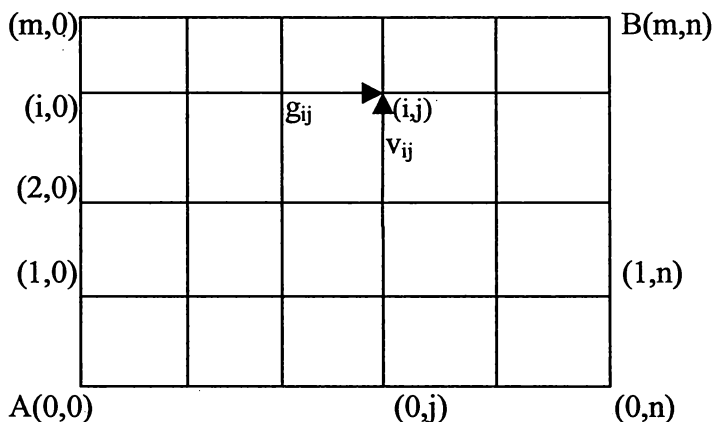


Рис. 33. Задача поиска оптимального пути.

Заданы затраты на каждый из возможных переходов и требуется найти путь с минимальными суммарными затратами из левого нижнего угла сетки (рис. 33, точка А)

в правый верхний угол (точка В). Такой путь называется оптимальным.

Узлы сетки пронумерованы так, как показано на рис. 33, где m и n задают соответственно вертикальный и горизонтальный размеры сетки.

Затраты на переход в узел i, j по горизонтали (из узла $i, j - 1$) составляют g_{ij} , а по вертикали (из узла $i - 1, j$) — v_{ij} . В точке А соответствующие величины равны нулю. Таким образом, исходными данными в этой задаче являются: m, n и все шаговые затраты g_{ij} и v_{ij} ($i = 0, 1, 2, \dots, m$; $j = 0, 1, 2, \dots, n$). Всего $n(m + 1)$ чисел g_{ij} и $m(n + 1)$ чисел v_{ij} , т.е. всего $2mn + m + n$ переходов и соответствующих им затрат.

При небольших размерах сетки можно попытаться решить задачу методом полного перебора вариантов возможных путей из точки А в точку В. Однако эта идея абсолютно бесперспективна уже при величинах m и n порядка 10 из-за резкого роста числа вариантов возможных путей из точки А в точку В с увеличением размеров сетки. Действительно, каждому варианту пути из точки А в точку В соответствует ровно m шагов по вертикали и ровно n шагов по горизонтали, но последовательность этих шагов для каждого варианта своя. Если шаг по горизонтали поставить в соответствие 0, а шаг по вертикали 1, то очевидно, что вариант пути — это выбор размещения m единиц по $m + n$ возможным местам (оставшиеся n мест займут нули). Для размещения первой единицы имеется $m + n$ возможностей, для второй $m + n - 1$ возможностей и т.д. В итоге получаем формулу для числа вариантов пути K :

$$K = \frac{(m + n)!}{m!n!}.$$

Уже при $m = n = 10$ $K = 184\,756$.

Следующая идея состоит в том, чтобы из точки А идти в том направлении, которое требует минимальных затрат на *первом шаге* (первый ход), не думая о затратах на последующих шагах, и *так в каждой точке*. Т.е. рассматривать только затраты на данном шаге и выбирать тот переход, для которого *на данном шаге* затраты минимальны. Легко убедиться в ошибочности этой идеи даже при $m = n = 1$ (рис. 34).

Действительно, если первый шаг выбрать по вертикали в точку С (затраты 1 против 5), то в итоге после второго шага получим суммарные затраты равные 101, а при выборе на первом шаге «неоптимального» решения (точка D) суммарные затраты равны всего лишь 6.

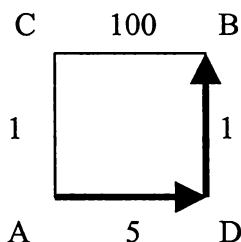


Рис. 34 Оптимальный путь ADB, а не ACB

Приходим к выводу о необходимости корректного метода решения задачи, к сожалению, более сложного, чем этот.

Поиск оптимального пути можно рассматривать как многошаговый процесс. На первом шаге находимся в точке А и имеем две возможности: пойти вверх или направо. Сделать выбор нельзя, так как нужно учитывать последствия этого выбора. При любом выборе, попадем мы в точку (0,1) или (1,0), встанет та же задача выбора из двух воз-

возможностей и опять выбор сделать нельзя и т.д. Однако существуют точки, находясь в которых мы не имеем выбора. Эти точки С и D находятся в одном шаге от последней точки В и имеют координаты $(m, n - 1)$ и $(m - 1, n)$ (рис. 35).

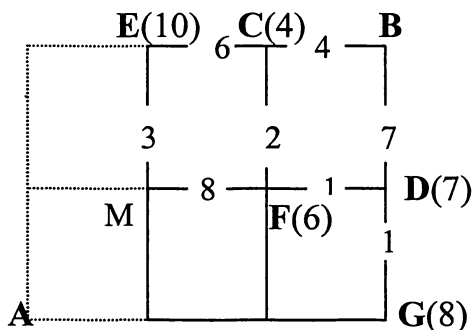


Рис. 35. Выбор на последних шагах

Для каждой из этих точек запомним затраты на оставшийся путь и рассмотрим предпоследний шаг. В двух шагах от финиша мы можем быть в точках Е, F или G. В точках Е и G выбора нет, и мы просто запомним для каждой из них *суммарные* затраты на весь оставшийся путь. На рис. 35 это 10 для точки Е и 8 для точки G. А что делать, если мы в двух шагах от финиша окажемся в точке F? Ответ кажется простым: надо идти в точку С, а не в точку D, так как, несмотря на то что на предпоследнем шаге затраты больше (2 против 1), с учетом затрат на оставшийся путь (4 против 7) суммарные затраты на путь до точки В окажутся меньше (6 против 8). Конечно, из точки F надо идти в точку С, но при одном непеременимом условии: ничто не может помешать нам это сделать, нет никакой связи с тем, как мы попали в данную точку или,

как говорят, нет «предыстории». В нашей задаче такой связи нет, но в более сложных задачах она вполне возможна. Например, могло быть задано дополнительное условие: суммарное количество изменений направления (поворотов) не больше заданного числа. Тогда мы, во-первых, должны знать сколько было сделано поворотов до попадания в точку F и каким образом (по горизонтали или по вертикали) мы попали в эту точку, т.е. должны знать предысторию. Может оказаться, что мы попали в точку F по горизонтали и уже исчерпали заданный «лимит» поворотов, тогда переход из точки F в точку C просто невозможен, так как это уже лишний поворот. Получается, что сделать оптимальный выбор в точке F нам может помешать предыстория.

Продолжим поиск оптимального пути в нашей задаче, в которой таких осложнений нет, и предыстория не имеет значения. В точке F мы запомним затраты на весь оставшийся путь при условии, что выбран оптимальный вариант: переход в точку C. Сделав еще шаг назад, т.е. оказавшись за три шага до финиша, мы увидим, что ситуация полностью аналогична предыдущей. Выбора или нет (точки на крайней верхней или крайней правой стороне сетки), или есть две возможности, но для *каждой из них уже известны последствия выбора*. Так, оказавшись в точке M, мы выберем не точку F, для которой затраты до конца пути равны 6, а казавшуюся бесперспективной точку E. Для нее эти затраты равны 10, но *суммарные затраты на весь оставшийся путь* меньше (13 против 14). При этом выборе мы решаем совсем простую задачу: суммируем затраты на каждый из возможных переходов на данном шаге (в точку E или в точку F) *с уже известными* затратами на дальнейший путь по *оптимальному для выбранной точки варианту*. Поступая аналогичным образом, мы рано или поздно в своем обратном движении

придем в начальную точку А (рис. 36). Но при этом уже будут известны последствия для каждого из вариантов выбора (пойти по вертикали в точку К или по горизонтали в точку L), так как для каждой из них уже вычислены и записаны затраты на весь оставшийся путь до точки В. Эта ситуация аналогична той, что представлена на рис. 34.

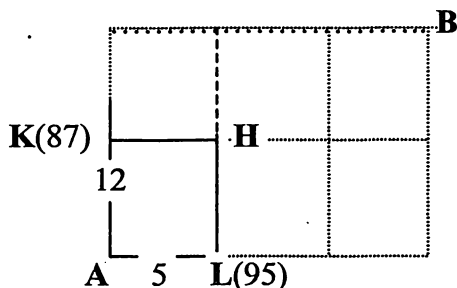


Рис. 36. Выбор на первом шаге

Теперь ничто не мешает нам сделать выбор, куда идти из точки А. Просуммируем затраты из А в К с тем, что записано для точки К на весь оставшийся путь от К до В. Затем просуммируем затраты из А в L с тем, что записано для L на путь из L в В, выберем наименьшую из сумм, которая и будет равна суммарным затратам по оптимальному пути. В примере на рис. 36 получаем для точки К 99, а для точки L 100, и, следовательно, теперь ясно, что идти надо в точку К. Но нам нужны не только эти минимальные из всех возможных затраты, но и сам оптимальный путь. Пока мы знаем, куда идти из точки А на первом шаге. А дальше? Дальше знаем только затраты на весь оставшийся путь. Чтобы не оказаться в такой ситуации неопределенности, при записи затрат на оставшийся путь в каждой из промежуточных

точек (С, D, E, F, G, M, и т.д. рис. 35) нужно записывать и сделанный выбор: куда идти из этих точек. Когда из точки А выберем точку К или L, *в любой из них* уже будет записано, куда идти (по вертикали или по горизонтали, т.е. 1 или 0) и т. д. Обратным разворотом мы дойдем до точки В и восстановим оптимальный путь.

Заметим, что если при сравнении вариантов на любом из этапов попадутся два варианта с равными затратами, то можно выбрать любой из них. Это означает, что оптимальный путь может быть не единственным.

В данной задаче мы исследовали многошаговый процесс от последнего шага к первому. Но ничто не мешает развернуть его и рассмотреть шаги в прямом направлении от точки А к точке В, ничего не меняя в методе Р. Беллмана по существу. Именно это предложили в начале 60-х гг., лет через 10 после появления работ Р. Беллмана, сотрудники Института кибернетики АН УССР применительно к решению задачи об оптимальном профиле новой железной дороги [10] и назвали этот метод методом последовательного анализа вариантов.

Применительно к рассматриваемой нами задаче поиска оптимального пути на двумерной сетке алгоритм выглядит следующим образом.

1. Из точки А делаем шаг в каждом из возможных направлений, запоминаем в точках К и L затраты и направление, по которому пришли в эту точку (0 — по горизонтали и 1 по вертикали). Соответственно, для точки К запомним 12 и 1, а для точки L 5 и 0.

2. На втором и всех последующих шагах, кроме последнего, если в точку ведет один путь (шаги по левой и нижней сторонам сетки), то просто запоминаем суммарные затраты на путь от начала и направление откуда пришли, а если в точке сходятся два варианта (на рис. 36 это точка Н, а на

рис. 35 точки М, F и др.), то сравниваем две возможности: прийти в эту точку по горизонтали или по вертикали. Для каждой из них вычисляем суммарные затраты на путь от начала, выбираем тот вариант, для которого эти суммарные затраты минимальны, запоминаем их и соответствующее им направление. Таким образом, происходит отбраковка вариантов, сходящихся в точке: *вариант с наибольшими затратами отбрасывается и все его продолжения далее не анализируются*. Естественно, это можно сделать только в таких простых задачах «без предыстории», как наша, когда совпадают множества возможных продолжений сравниваемых вариантов.

3. На последнем шаге в точку В ведут два направления, и по каждому из них все известно: для каждой точки (на рис. 35 это точки С и D) записаны затраты на весь путь от начала и направление откуда пришли в эту точку. Снова суммируем затраты по вариантам, выбираем наименьшие, а затем обратным разворотом восстанавливаем оптимальный путь.

В реальных задачах многоэтапные процессы могут иметь не один исход (точку финиша) или начало (точку старта), а несколько. Однако это не мешает применить метод Р. Беллмана. Во-первых, для упрощения алгоритма всегда можно ввести еще один фиктивный этап с нулевыми затратами на нем. Например, ввести единственную фиктивную точку финиша, которая достигается из реальных точек финиша без дополнительных затрат. Во-вторых, можно просто сравнить варианты финиша в каждой из точек, если их несколько. Аналогично можно поступить и при наличии нескольких точек старта, но в этом случае необходимо либо добавить фиктивную точку старта, из которой затраты на первый шаг равны нулю, либо сразу «тащить» много вариантов.

3.2. Принцип оптимальности и уравнение Р. Беллмана

Принцип оптимальности Р. Беллмана сформулирован для широкого круга задач управления, распределения ресурсов, проектирования и др. и вообще задач принятия решений в многошаговых (многоэтапных) процессах. Если задано начальное (точка А) и конечное (точка В) состояния некоторой системы и переход из начального в конечное состояние осуществляется в несколько промежуточных этапов, на каждом из которых система может находиться в одном из различных состояний, и каждому переходу на каждом этапе можно сопоставить некоторые затраты, то задача состоит в том, чтобы выбрать такой путь (т.е. все промежуточные состояния), при котором некоторая количественная оценка этого пути достигает минимума.

Предположим для начала, что количественная оценка пути — это суммарные затраты (сумма затрат по этапам, т.е. сумма шаговых затрат). Пусть на первом этапе из начального состояния (точка А) можно перейти в некоторое конечное число состояний (условно это четыре точки на вертикали 1, рис. 37) с соответствующими затратами. Далее, аналогично, из состояний первого этапа (результатов первого шага) можно перейти в конечное число состояний (три точки на вертикали 2) и т.д., вплоть до конечного этапа, на котором из всех возможных состояний возможен переход только в конечное состояние (точка В).

В конкретной задаче может быть так, что для каждого или некоторых промежуточных состояний (например, точка С) дальнейшие переходы зависят от того, каким образом система была переведена в это состояние (например, переход СD возможен, если пришли в точку С из точки М, и невозможен, если пришли в точку С из точки N). Но может быть и так, что *для всех* промежуточных состояний даль-

нейшие переходы *никак не зависят* от того как система попала в это состояние.

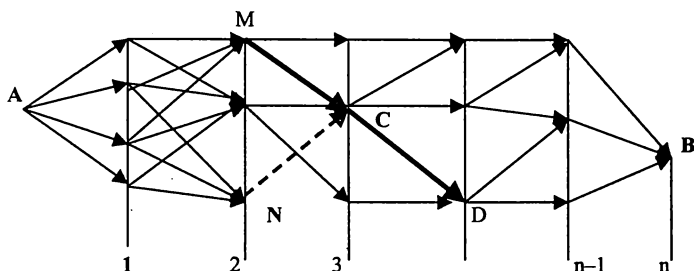


Рис. 37. Иллюстрация метода динамического программирования

В первом случае говорят о «задачах с предысторией», а во втором случае — об отсутствии предыстории, точнее, о том, что предыстория не имеет значения для дальнейшего поведения системы. Принцип Р. Беллмана гласит: *«Если в каждом из состояний дальнейшее поведение системы не зависит от того, как она попала в это состояние, то дальнейшая траектория должна быть оптимальной».*

Под траекторией понимается последовательность состояний, в которых находится система, начиная с первого и до последнего. Если состояние системы определяется координатами некоторой точки (например, центра тяжести движущегося объекта) на плоскости или в обычном трехмерном пространстве, то это понятие совпадает с привычным понятием траектории. В других задачах состояние системы характеризуется набором чисел, который можно считать координатами вектора в некотором многомерном пространстве и, следовательно, можно говорить о траектории в этом пространстве.

Приведенная формулировка принципа оптимальности означает, что речь идет *только о системах «без предистории»*. В этом случае из каждого промежуточного состояния *можно отдельно, независимо от пройденных этапов*, решать задачу поиска оптимального пути в конечное состояние. Или, что фактически то же самое, сравнивать и отбраковывать варианты достижения *любого промежуточного состояния* из начальной точки и *оставлять только один вариант*.

В отличие от рассмотренной нами простой задачи поиска оптимального пути на двумерной сетке в сложных задачах каждое промежуточное состояние может определяться не двумя, а многими координатами, т.е. каждому состоянию может соответствовать вектор многомерного пространства. От того, как формально определено понятие «состояние», зависит не только формализации понятия «переход в другое состояние», но и, как мы увидим в дальнейшем, сама возможность применения принципа Р. Беллмана.

Итак, пусть в каждом из состояний, в котором может находиться система в начале очередного этапа, известны все возможные воздействия на нее. И для каждого такого воздействия известны его последствия, т.е. состояние, в которое перейдет система, и затраты на этот переход. Воздействие на i -ом шаге обозначим через x_i . Эти воздействия часто называют шаговыми управлениями [4]. Если число этапов обозначить через n , то задача состоит в поиске последовательности шаговых управлений, т.е. вектора $x(x_1, x_2, \dots, x_n)$. Поскольку начальное состояние системы задано, то последовательность шаговых управлений однозначно определяет последовательность переходов системы из одного состояния в другое, т.е. последовательность состояний или траекторию движения (в широком смысле слова). В сложных задачах x_1, x_2, \dots, x_n не обязательно числа. Это могут быть векторы или функции.

Обозначим затраты на i -ом этапе через z_i . Требуется найти такую последовательность шаговых управлений x_i , при которой суммарные затраты минимальны.

$$Z = \sum_{i=1}^n z_i \rightarrow \min$$

Ту последовательность шаговых управлений, при которой достигается минимум, будем называть оптимальным управлением и обозначать через x^* ($x_1^*, x_2^*, \dots, x_n^*$). Соответствующие ей минимальные по всем возможным последовательностям x затраты $Z^* = \min Z(x)$. [4]

Затраты на i -ом шаге (этапе) зависят не только от x_i , но и от состояния S , в котором была система до воздействия x_i , т.е. фактически от всех предшествующих шаговых управлений. Состояние S' , в которое перейдет система, зависит только от S и x_i , *если нет влияния предыстории*. Формально это можно записать в виде $S' = f(S, x_i)$, где f заданная функция, так как известны последствия воздействия x_i на систему, находящуюся в состоянии S . В соответствии с принципом оптимальности x_i надо выбирать так, чтобы суммарные затраты на все последующие этапы были минимальны. Эти суммарные затраты зависят от состояния S и складываются из затрат на i -ом шаге $z_i(S, x_i)$ и на всех последующих шагах. Суммарные затраты на все шаги (этапы), начиная с i -го и до конца, обозначим Z_i . Тогда можно записать $Z_i = z_i + z_{i+1}$ и оптимальные управления надо на каждом шаге выбирать так, чтобы

$$Z_i(S) = \min_{x_i} \{ z_i(S, x_i) + Z_{i+1}(f(S, x_i)) \}. \quad (3.1)$$

Это и есть основное рекуррентное уравнение динамического программирования, выражающее затраты на все оставшиеся этапы из любого состояния S через затраты на данном z_i и на всех последующих шагах $Z_{i+1}(S')$. Только на

последнем шаге из любого состояния S можно легко найти оптимальный переход в конечное состояние. Если конечное состояние единственное, то и для каждого из состояний S на последнем шаге этот переход (т.е. шаговое управление x_n) единственное. В общем случае для последнего шага уравнение (3.1) приобретает вид:

$$Z_n(S) = \min_{x_n} z_n(S, x_n).$$

так как $n + 1$ шага просто нет. Это означает, что для каждого из возможных состояний на последнем шаге мы определяем и запоминаем затраты на этот последний шаг. Если конечных состояний несколько (рис. 38), то на последнем шаге придется сравнивать варианты переходов из каждого состояния $1, 2, \dots, k$ в конечные состояния B_1, B_2, \dots, B_m , т.е. шаговые управления x_n , обеспечивающие переходы $1B_1, 1B_2, \dots, 1B_m$, и запоминать наилучшее управление и соответствующие ему минимальные затраты на последний шаг для состояния 1, затем все переходы из состояния 2 и т.д. В результате для каждого состояния S на последнем шаге станут известны затраты $Z_n(S)$ и управление x_n , обеспечивающее оптимальный переход в конечное состояние.

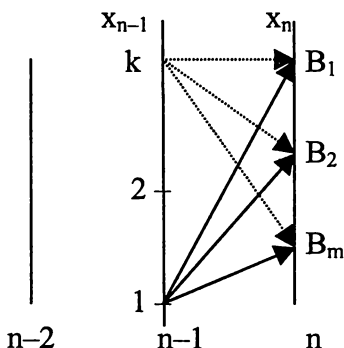


Рис. 38. Последний шаг при нескольких конечных состояниях

Запишем теперь уравнение для предпоследнего шага $i = n - 1$.

$$Z_{n-1}(S) = \min_{x_{n-1}} \{ z_{n-1}(S, X_{n-1}) + Z_n(f(S, X_{n-1})) \} \quad (3.2)$$

Здесь $Z_n(f(S, x_{n-1})) = Z_n(S')$ уже известны, а найти надо x_{n-1} для каждого состояния S на $n-1$ шаге. Это задача полностью аналогична той, что мы решали для последнего шага. Последовательно рассматривая шаги от $n-1$ до 2, определим для каждого из состояний наименьшие затраты на все последующие этапы и управления (переходы на следующий этап). На первом шаге теперь известны последствия любого из возможных переходов (управлений), что и дает возможность сделать окончательный выбор управления x_1 , т.е. перехода из точки A .

Если начальных состояний (точек A) несколько, то каждое из них рассматривается отдельно, выбирается наилучший переход (управление) и определяются соответствующие минимальные затраты. Затем сравниваются начальные состояния и выбирается то из них, для которого суммарные затраты минимальны. Таким образом, при любом числе начальных состояний становятся известными и суммарные минимальные затраты на все этапы. Затем обратным разворотом можем найти оптимальную последовательность шаговых управлений (переходов в следующее состояние), если мы запоминали для каждого промежуточного состояния оптимальный переход из него.

Отметим, что поиск оптимального перехода из заданного состояния на i -ом шаге (в том числе и на последнем) может быть сложнее, чем выборка чисел из заданного массива, суммирование и сравнение. В сложных задачах использование принципа оптимальности Р. Беллмана может приводить к разностным, дифференциальным и др. уравнениям.

Часто принцип оптимальности Р. Беллмана формулируют следующим образом: «Оптимальная траектория состоит из оптимальных частей». Эта формулировка относится не только к траекторным задачам, но ко всем задачам, решаемым с помощью метода динамического программирования, если под траекторией понимать последовательность переходов из одного состояния в другое в результате решений (управлений), принимаемых на последовательных этапах. Если под частью траектории понимать любую последовательность состояний, в которой начальное и конечное состояния фиксированы (аналог кривой, соединяющей две фиксированные точки), то для оптимальной траектории в задачах «без предыстории» это безусловно так. Но это не означает, что для двух состояний (скажем, на этапах p и q рис. 39 состояния C и D), которые не содержатся в оптимальной последовательности состояний, т.е. не принадлежат оптимальной траектории, не может быть последовательности переходов с меньшими затратами, чем для оптимальной последовательности на тех же промежуточных этапах с p -го по q -ый. На рис. 39 для части $C^* D^*$ оптимальной траектории (показана стрелками) на этапах с p -го по q -ый затраты больше, чем для части неоптимальной траектории CD на тех же этапах, но в целом суммарные затраты на все этапы для оптимальной траектории меньше.

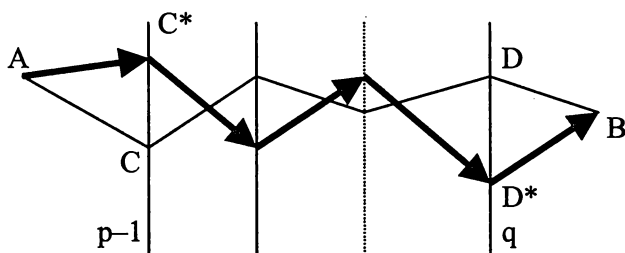


Рис. 39. Сравнение траекторий

Например, для оптимальной траектории на участке $AC^* D^*$ затраты равны 5, на участке $C^* D^*$ 50, на $D^* B$ 4, всего 59. А на неоптимальной траектории на участке AC затраты 10, на CD 45 (против 50 для $C^* D^*$) и на DB 10, всего 65 против 59.

Управления x_i в оптимальной последовательности управлений являются *условно оптимальными*, так как они *выбирались для конкретных состояний* с учетом последствий (затрат на все последующие этапы).

Еще раз подчеркнем, что принцип оптимальности Р. Беллмана и, соответственно, уравнение 3.1 не универсальны, и рассмотрим условия их применимости.

3.3. Область применения динамического программирования

Общее условие применимости принципа оптимальности выражается в требовании отсутствия влияния «предыстории». Именно поэтому формулировка этого принципа, приведенная в разделе 3.2, начинается со слова «если» (в отличие от ряда других работ; см., например, [4], с. 109 и последующие переиздания).

Установить возможность применения принципа Р. Беллмана — значит доказать отсутствие влияния «предыстории». Это влияние может возникать при наличии более сложных целевых функций, чем сумма затрат. Может оказаться, что даже если удастся разбить задачу на ряд последовательных этапов, значение целевой функции можно вычислить только при полностью известной траектории (полной последовательности переходов системы от начального состояния к конечному). Например, пусть целевая функция по-прежнему представляет собой сумму затрат на последовательных этапах. Это может быть

строительство некоторого крупного объекта. Существуют различные варианты организации строительства, и требуется найти вариант с наименьшими затратами. Затраты на отдельных этапах включают в себя стоимость различного рода используемых ресурсов, но не только это. Некоторые ресурсы (например, материалы) могут быть ограниченными и взаимозаменяемыми, но неравноценными и по расходу, и по цене, и по эффективности использования на различных этапах. При сравнении вариантов планируемого строительства на отдельных этапах (например, на последнем, см. рис. 38) нельзя ориентироваться на самый выгодный набор ресурсов, так как к этому этапу их может и не остаться и придется использовать заменители. Другими словами, на последнем этапе мы, не зная общей потребности в отдельных видах ресурсов, просто не можем вычислить затраты на этот этап для различных состояний. Аналогично, при движении из начальной точки в конечную и сравнении промежуточных вариантов нельзя ориентироваться на использование самых выгодных ресурсов до их исчерпания, так как неизвестна потребность в различных ресурсах на дальнейших этапах из-за неопределенности плана строительства в целом и нехватка некоторых ресурсов в дальнейшем может привести к значительно большим затратам, чем выигрыш от их использования на начальных этапах.

Фактически это обстоятельство мы уже отмечали при рассмотрении задачи поиска оптимального пути на двумерной сетке при дополнительном ограничении на число поворотов. Это число поворотов и можно считать ограниченным ресурсом, так что его необдуманный расход на начальных этапах в дальнейшем не позволит выйти на оптимальную траекторию.

Особенности целевой функции, даже при отсутствии каких-либо ограничений, могут «создавать предысторию» и

тем самым осложнять или вообще делать невозможным применение принципа Р. Беллмана. Например, целевая функция представляет собой сумму произведений затрат на двух последних этапах:

$$z_1 z_2 + z_2 z_3 + \dots + z_{n-1} z_n \rightarrow \min. \quad (3.3)$$

Сравнивая варианты (например, варианты достижения узла на двумерной сетке) после первых двух этапов и отдавая предпочтение варианту с меньшим произведением затрат $z_1 z_2$, мы никак не учитываем, что важно не только это произведение, но и конкретно z_2 , так как от z_2 зависит следующее слагаемое. Другими словами, важно не только значение целевой функции, но и как оно получено (предыстория). Может оказаться, что выгоднее оставить вариант с большим значением $z_1 z_2$, но с меньшим значением z_2 .

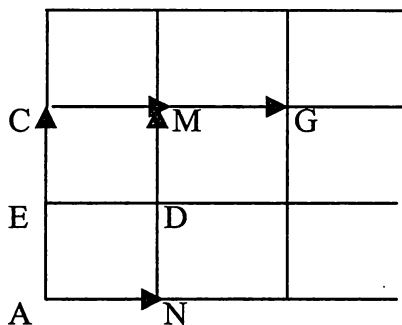


Рис. 40. Сравнение вариантов не «в точку», а «в отрезок»

Отметим, что во многих задачах влияние предыстории можно преодолеть за счет усложнения формализации понятия «состояние». Так, если в задаче поиска оптимального

пути на двумерной сетке считать состоянием не узел, а два последних пройденных узла, т.е. отрезок, их соединяющий, то сравнимыми становятся варианты, у которых общим является не один последний узел, а два, т.е. варианты, имеющие общий последний отрезок (рис. 40).

Варианты, сходящиеся в точке D (AED и AND), теперь сравнивать нельзя, так как они соответствуют разным состояниям. Нужно сравнивать AEDM и ANDM и оставить лучший из них.

Это сравнение корректно и при новой целевой функции (3.3), и при наличии ограничений на число поворотов, так как оно у них одно и то же.

Лучший из оставшихся вариант нельзя сравнивать с вариантом AECM. К нему нужно добавить общий отрезок, например MG, и сравнивать с AECMG по целевой функции (3.3). Но при наличии ограничений на число поворотов эти варианты не сравнимы, так как AECMG имеет один поворот, а все остальные больше. Нужно ввести еще одну «координату» — число поворотов и усложнить тем самым формализацию понятия «состояние». Чем больше чисел (координат) нужно для задания конкретного состояния, тем больше объем вычислений и требуемой памяти, так как сравниваются варианты достижения одного и того же промежуточного состояния и число таких вариантов резко возрастает с ростом числа координат.

На конкретных примерах мы убедились в том, что применению принципа оптимальности Р. Беллмана может мешать не только наличие целевой функции более сложной, чем сумма пошаговых затрат, но и наличие ограничений. В этой связи трудно согласиться с утверждением о том, что ограничения «не страшны» для этого метода. Например, в книге Е.С. Вентцель «Исследование операций. Задачи, принципы, методология» (см. [4], с. 107 и все переиздания) утверждается: «Метод динамического про-

граммирования является очень мощным и плодотворным методом оптимизации управления; ему не страшны ни целочисленность решения, ни нелинейность целевой функции, *ни вид ограничений, накладываемых на решение*». Это утверждение, безусловно, верно, если ограничения накладываются на каждую или некоторые переменные отдельно. Такие ограничения могут даже упростить поиск решения. Но если ограничения даже очень простого вида (например, линейные) связывают несколько переменных, то каждое новое ограничение увеличивает число координат, задающих состояние системы (их называют еще параметры состояния). А при числе параметров состояния больше трех и большом числе возможных вариантов перехода из каждого промежуточного состояния вычислительные трудности при реализации метода динамического программирования могут оказаться непреодолимыми даже для сверхмощных современных компьютеров. Вот почему *вид ограничений, накладываемых на искомое решение, крайне важен*.

Что касается целевой функции, то и здесь важна ее структура. Мы видели, что даже сравнительно простая функция в виде суммы произведений затрат на двух последних шагах (3.3) требует увеличения числа параметров состояния. Существенным является не то, что целевая функция нелинейна, а возможность ее вычисления по отдельным шагам по одним и тем же правилам. Если этой возможности нет и целевая функция может быть вычислена *только после того как сформирована вся траектория*, т.е. последовательность состояний, то метод динамического программирования применить не удастся. Конечно, для линейной функции такая возможность пошаговых вычислений очевидна. И для суммы нелинейных функций, в которой каждое слагаемое зависит только от одной переменной, дополнительных сложностей не возникает, так как нет связи

между переменными. Но если в целевую функцию включитьлагаемое, зависящее нелинейным образом от двух переменных (например, первой и последней), то все осложняется. Такая нелинейность целевой функции весьма существенна для применимости метода динамического программирования. Следовательно, цитированное выше утверждение о том, что «не страшно» для метода динамического программирования, требует существенного уточнения и в части влияния нелинейности, и в части влияния вида ограничений.

Очень важным достоинством метода динамического программирования является его нечувствительность к отсутствию непрерывности производных целевой функции в задачах с непрерывными переменными.

Часто говорят и о нечувствительности этого метода к наличию локальных экстремумов. В дискретном случае это действительно так, и это очень важно. Если же дискретность вводится искусственно, то здесь могут возникнуть осложнения, так как теоретически при любой конечной величине дискрета можно привести пример многоэкстремальной функции, при минимизации которой по методу динамического программирования получаем неоптимальное решение (см. приложение 5, рис. 51).

Чтобы охарактеризовать тип задач, к которым можно применить метод динамического программирования, отметим следующее:

1. Должна быть возможность интерпретации задачи как многошагового процесса принятия решений, в котором решение, принимаемое на каждом шаге, состоит в выборе одного или нескольких чисел (управляющих переменных, определяющих однозначно переход в следующее состояние). В сложных случаях управляющие переменные могут быть векторами или функциями.

2. Задача должна быть определена для любого числа шагов и иметь структуру, не зависящую от числа шагов.
3. При рассмотрении задачи, состоящей из k шагов, должно быть задано некоторое множество параметров, описывающих состояние системы (параметры состояния). Это же множество параметров должно описывать состояние системы независимо от количества шагов.
4. Выбор управления (перехода) на каждом из шагов и состояний не должен зависеть от того, как система попала в это состояние, т.е. не должна влиять предыстория.

3.4. Примеры задач, решаемых с помощью динамического программирования

1. *Задача об инвестициях.* Пусть имеется n инвестиционных проектов $\Pi_1, \Pi_2, \dots, \Pi_n$ и капитал K , который можно вложить в эти проекты. Для каждого проекта ожидаемый доход D_i ($i = 1, 2, \dots, n$) зависит от вложенного в него капитала x_i , и функции $D_i(x)$ заданы.

Требуется найти такое распределение капитала, при котором суммарный доход максимален.

Эту задачу можно попытаться решить методами классического анализа. Пусть переменными являются x_i ($i = 1, 2, \dots, n$). Тогда требуется найти такие значения переменных, при которых

$$\sum_{i=1}^n D_i(x_i) \rightarrow \max$$

при условии $\sum_{i=1}^n x_i = K$.

Можно выразить x_1 через все остальные переменные ($x_1 = K - x_2 - x_3 - \dots - x_n$), подставить это выражение в целевую функцию и решать задачу безусловной максимизации полученной функции от $n-1$ переменных.

Можно и не исключать переменную x_1 , а ввести множитель Лагранжа λ и решать задачу на безусловный максимум функции от x_i ($i = 1, 2, \dots, n$) и λ .

$$\sum_{i=1}^n D_i(x_i) + \lambda \left(\sum_{i=1}^n x_i - K \right) \rightarrow \max$$

Эти пути малоперспективны, так как частные производные могут быть отличны от нуля, например если некоторые функции $D_i(x)$ линейны; максимум может достигаться в точках, не удовлетворяющих условию $0 \leq x_i \leq K$; функции $D_i(x)$ вообще могут не иметь производных в отдельных точках. Из-за необходимости дополнительно учитывать условия $0 \leq x_i \leq K$ задача не сводится к решению системы уравнений, получаемой приравнованием нулю частных производных даже при гладких функциях $D_i(x)$, а применению различных методов спуска может помешать разрывность производных.

В данной задаче, казалось бы, ничто не напоминает о динамическом программировании: нет многоэтапного процесса, нет дискретности состояний и т.д. Тем не менее эта задача может успешно решаться методом динамического программирования, несмотря на наличие ограничения, связывающего все переменные. Более того, это ограничение можно записать и в виде неравенства:

$$\sum_{i=1}^n x_i \leq K.$$

Этапы введем искусственно. Будем считать, что на первом этапе решается вопрос об инвестициях в первый проект, на втором этапе — во второй и, наконец, на n -ом этапе — в последний. При этом какой проект считать первым, а какой последним, не имеет никакого значения. Далее, введем дискретность, т.е. минимальную сумму, на которую могут отличаться инвестиции, и эту сумму примем за единицу (например, будем распределять в тысячах или миллионах рублей). «Состоянием системы» будем считать *количество уже распределенного капитала*.

На первом этапе система находится в состоянии нуль. Обозначим число дискретов, содержащихся в K , через M . На первом этапе решается судьба проекта Π_1 , в который может быть вложено $0, 1, 2, \dots, m$ единиц. Это и будут состояния, в которые может быть переведена система после первого этапа. Каждому из них соответствует свое значение целевой функции, т.е. доход, полученный от первого проекта $D_1(x_1)$. Далее, состояние m означает, что весь капитал вложен в первый проект и последовательность состояний (траектория) будет m, m, \dots, m . Соответственно распределение капитала между проектами $m, 0, \dots, 0$. Состояние 0 после первого этапа означает, что в первый проект капитал не вкладывается и из этого состояния можно перейти в любое из $0, 1, 2, \dots, m$ состояний следующего этапа. Состояние $p < m$ после первого этапа означает, что осталось $m - p$ состояний, и, следовательно, из него можно перейти в любое из $p, p+1, \dots, m$ состояний на следующий этап. После второго этапа в каждое из состояний можно попасть разными путями и возможности дальнейшего распределения никак не зависят от того, как система попала в данное состояние. Поэтому можно сравнивать варианты и в каждом из промежуточных состояний оставлять только один из них. Поясним сказанное простым численным примером, который носит условный характер, чтобы избежать громоздких вычислений.

Пусть имеется $n = 4$ инвестиционных проекта. $K = 4$ и дискрет равен 1 (например, распределяем 4 млн руб. с дискретом 1 млн руб.). Функции дохода имеют следующий вид:

$$D_1(x)=x, D_2(x)=\begin{cases} 2x & \text{при } 0 \leq x \leq 3 \\ 6 & \text{при } x > 3 \end{cases}$$

$$D_3(x) = \begin{cases} x^2 & \text{при } 0 \leq x \leq 2 \\ 4 & \text{при } x > 2 \end{cases}$$

$$\text{и } D_4(x) = \frac{x^2}{2}$$

Составим таблицу возможных инвестиций в каждый из проектов и соответствующих доходов.

Таблица 1

Инвестиции	Доходы			
	D_1	D_2	D_3	D_4
0	0	0	0	0
1	1	2	1	0.5
2	2	4	4	2
3	3	6	4	4.5
4	4	6	4	8

Начальное состояние (точка О) на рис. 41 соответствует нулевым инвестициям. После первого этапа инвестиции могут составить 0, 1, 2, 3, 4. Соответственно, имеем пять точек на 1-й вертикали. В скобках указан полученный доход D_1 , который на первом этапе, т.е. от первого проекта, равен инвестициям.

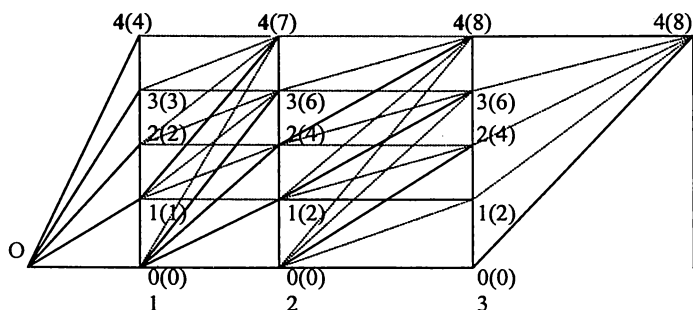


Рис. 41. Решение задачи об инвестициях

Далее, *суммарные инвестиции* в первый и второй проект, т.е. состояния системы на втором этапе, снова могут составить те же величины 0, 1, 2, 3, 4. Но им будет соответствовать разный суммарный доход в зависимости от того, как они распределяются между первым и вторым проектами. Например, 3 млн руб. можно распределить между первым и вторым проектами 4 способами: $0 + 3$ с доходом 6, или $1 + 2$ с доходом 5, или $2 + 1$ с доходом 4, или $3 + 0$ с доходом 3. Варианты распределения соответствуют линиям, сходящимся в точке 3 на второй вертикали. Для этого состояния 3 на втором этапе выбираем вариант $0 + 3$ с наибольшим доходом 6, запоминаем это число и номер состояния на предыдущем этапе (0), которому оно соответствует. Аналогично поступаем для всех состояний второго этапа: запоминаем наибольший доход из возможных при соответствующих инвестициях и состояние предыдущего этапа. На рис. 41 на вертикали 2 для каждого суммарного значения инвестиций наибольший суммарный доход указан в скобках, а вариант, при котором он достигается, показан сплошной линией. Переходя к третьему этапу, мы оказываемся в той же ситуации: известны все состояния предыдущего этапа и все

возможные переходы в каждое из 5 возможных состояний. Характерно, что некоторые варианты могут оказаться равноценными, тогда оставляем любой из них. Например, в состояние 2 можно перейти из состояний 0, 1 или 2 предыдущего этапа, т.е. вкладывая в третий проект 2, 1 или 0 с суммарным доходом 4, 3 или 4 соответственно. Переходы 0,2 и 2,2 равноценны. Аналогичная ситуация возникает и для состояния 3 с переходами 1,3 и 3,3 и суммарным доходом 6.

Переходя к последнему этапу, заметим, что нас может интересовать только состояние 4, т.е. когда весь капитал вложен, иначе суммарный доход не будет максимальным. Оказывается, что два перехода (0, 4) и (4, 4) равноценны и дают одинаковый суммарный доход, равный 8. Им соответствуют два равноценных распределения инвестиций $0 + 0 + 0 + 4$ и $0 + 2 + 2 + 0$.

В этом примере отсутствие производных функций $D_2(x)$ при $x = 3$ и $D_3(x)$ при $x = 2$ никак не сказывается на решении задачи.

Рассмотренная задача допускает обобщение на случай максимизации суммы функций, каждая из которых зависит от одной переменной, при наличии одного линейного ограничения в виде неравенства с положительными коэффициентами и целочисленности всех переменных $x_i \geq 0$.

$$\sum_{i=1}^n f_i(x_i) \rightarrow \max$$

$$\text{при } \sum_{i=1}^n a_i x_i \leq b, \quad a_i > 0, \quad x_i \geq 0 \quad (i = 1, 2, \dots, n).$$

В этой задаче ресурс — это слагаемое в последней сумме, относящееся к x_i . Суммарный ресурс ограничен величиной b .

Первый этап — выбор x_1 из интервала $0 \leq x_1 \leq b/a_1$, второй этап — выбор x_2 и так до x_n . Условие $a_i > 0$ потребовалось для ограничения интервала изменения x_i . Состояние системы на любом этапе можно задать количеством распределенного ресурса (или оставшегося ресурса, что не имеет принципиального значения). Задача решается аналогично рассмотренной нами. Характерно, что при наличии двух линейных ограничений того же вида задача может быть решена аналогично, но приходится рассматривать два вида ресурсов и, соответственно, требуется не одна, а две переменных, определяющих состояние системы. Это расход (или остаток) ресурсов каждого вида. Теоретически, и при наличии большего числа ограничений того же вида задача может быть решена с помощью динамического программирования, но только теоретически. Дело в том, что число параметров состояния равно числу ограничений (видов распределяемых ресурсов) и при числе ограничений три и более вычислительные трудности могут стать непреодолимыми [16]. В этом случае методы нелинейного программирования оказываются предпочтительнее, хотя и могут дать только приближенное решение при наличии требований целочисленности.

2. *Задача об управлении ресурсами.* Пусть требуется спланировать деятельность двух предприятий Π_1 и Π_2 в течение n лет. Средства X_1 и X_2 , вложенные в каждое предприятие в начале года, в конце года приносят доход, который делится на две части. Первая часть в предприятия не вкладывается (будем считать, что это расходы на нужды предприятия, не связанные с дальнейшим производством, и условно назовем их накоплениями), а вторая часть дохода от каждого предприятия *суммируется и заново распределяется* между предприятиями на их дальнейшую деятельность.

В начале планируемого периода имеется K_1 средств, все они вкладываются в предприятия (в том или ином соотношении). В начале второго года распределяется только та часть полученного дохода, которая вкладывается в предприятия, *дополнительных средств не поступает* и так в начале каждого следующего года. Доход, полученный после n -го года, также делится на две части по тем же правилам, что и в предыдущие годы. Заданы функции, которые позволяют для каждого предприятия в зависимости от вложенных в него средств (X_1 и X_2 соответственно) вычислить и часть дохода, идущего на накопление ($d_1(X_1)$ и $d_2(X_2)$ соответственно), и часть дохода, направляемую на распределение в начале следующего года ($p_1(X_1)$ и $p_2(X_2)$ соответственно).

Требуется найти такой план распределения средств между предприятиями, чтобы к концу n -го года суммарные накопления были максимальны. Другими словами, максимизируются средства, полученные от предприятий за n лет и им не возвращенные, при заданных единовременных вложениях в начале первого года K_1 .

Рассмотрим конкретный пример.

Пусть $n = 5$, $K_1 = 10$,

$$\begin{aligned} d_1(x) &= x^2, & p_1(x) &= 0.8x, \\ d_2(x) &= 2x^2, & p_2(x) &= 0.2x. \end{aligned} \quad (3.4)$$

Итак, планируемый период составляет 5 лет, начальные распределяемые средства равны 10 единиц и в начале каждого следующего года распределяется $0.8X_1 + 0.2X_2$, где X_1 и X_2 соответственно средства, вложенные в предприятия в истекшем году. Накопления за истекший год составляют:

$$d_1(X_1) + d_2(X_2) = X_1^2 + 2X_2^2$$

и их сумма за пять лет должна быть максимальна.

Задача естественным образом распадается на пять этапов по числу планируемых лет. На каждом этапе достаточно определить только X_1 , т. е. средства, вкладываемые в первое предприятие, так как сумма распределяемых средств известна.

«Состоянием системы» на каждом из этапов будем считать количество распределяемых средств. На первом этапе состояние задано и определяется числом 10. Обозначим через K_i средства, распределяемые на i -ом шаге, т.е. в начале i -ого года, через x_i средства, вложенные в первое предприятие в i -ом году (во второе предприятие вкладывается $K_i - x_i$), а через D_i — накопления, которые получаются от использования K_i в i -ом году. В наших обозначениях на первом этапе $X_1 = x_1$, $X_2 = 10 - x_1$, на втором этапе $X_1 = x_2$, $X_2 = K_2 - x_2$, ..., на пятом этапе $X_1 = x_5$, $X_2 = K_5 - x_5$.

Состояние, в которое перейдет система после первого этапа, определяется как $0.8x_1 + 0.2(10 - x_1)$. И управлений x_1 и, соответственно, состояний *бесконечно много*. Казалось бы нужно ввести дискретность по x_1 и перейти к конечному числу управлений и состояний, но *это не обязательно*.

Рассмотрим последний, пятый год. Накопления D_5 зависят от распределяемых средств K_5 и от того, как они распределяются, т.е. от x_5 . Используя (3.4), можно записать:

$$D_5(K_5, x_5) = x_5^2 + 2(K_5 - x_5)^2 \dots$$

Величина K_5 , характеризующая состояние системы на пятом этапе, неизвестна. Далее, x_5 — любое число на отрезке $0 \leq x_5 \leq K_5$. Выбор x_5 не зависит от того, как было получено состояние K_5 (не влияет предыстория), и при заданном K_5 x_5 надо выбрать так, чтобы накопления

$D_5(K_5, x_5)$ были максимальны. Итак, нужен максимум по x_5 функции

$$x_5^2 + 2(K_5 - x_5)^2$$

при условии $0 \leq x_5 \leq K_5$.

Максимизируемая функция относительно x_5 при любом K_5 представляет собой квадратный трехчлен с положительным первым коэффициентом (при x_5^2). Ее максимум достигается в одной из крайних точек отрезка $[0, K_5]$ (а не при равенстве нулю производной по x_5 , где функция имеет минимум). При $x_5 = 0$ значение функции равно $2K_5^2$, а при $x_5 = K_5$ только K_5^2 . Отсюда следует, что надо взять $x_5 = 0$ для любого состояния K_5 .

Переходим к четвертому этапу (году). Состояние системы определяется величиной K_4 — распределяемыми средствами на четвертом этапе. Из этого состояния нужно дойти до конца и, в частности перейти на пятый этап, по оптимальной траектории, т.е. так, чтобы были максимальны *суммарные накопления* на оставшихся (четвертый + пятый) этапах. Эти накопления составят $D_4(K_4, x_4) + D_5(K_5, x_5) = D_4(K_4, x_4) + 2K_5^2$, так как оптимальную величину $x_5 = 0$, *не зависящую от x_4* , мы знаем. Ей соответствует $D_5 = 2K_5^2$. Переход от четвертого этапа к пятому определяется величиной x_4 — вложениями в первое предприятие на четвертом году. Во второе предприятие будет вложено $K_4 - x_4$, и для величины K_5 в соответствии с (3.4) получим:

$$K_5 = 0.8x_4 + 0.2(K_4 - x_4) = 0.6x_4 + 0.2K_4$$

При любом K_4 величину x_4 надо искать из условия:

$$x_4^2 + 2(K_4 - x_4)^2 + 2(0.6x_4 + 0.2K_4)^2 \rightarrow \max$$

при $0 \leq x_4 \leq K_4$.

Это соответствует принципу оптимальности и представляет собой запись рекуррентного уравнения Р. Беллмана для нашей задачи применительно к четвертому этапу. Максимум этой функции может достигаться только в крайних точках отрезка $[0, K_4]$. При $x_4 = 0$ имеем $2.08 K_4^2$, а при $x_4 = K_4$ соответствующие суммарные накопления на двух последних этапах составят $2.28 K_4^2$. Поэтому выбираем $x_4 = K_4$ при любом K_4 и переходим к третьему этапу (году). Уравнение Р. Беллмана примет вид:

$$x_3^2 + 2(K_3 - x_3)^2 + 2.28 K_4^2 \rightarrow \max$$

при $0 \leq x_3 \leq K_3$.

$$K_4 = 0.8x_3 + 0.2(K_3 - x_3) = 0.6x_3 + 0.2K_3$$

и, следовательно, x_3 должна быть точкой максимума для

$$x_3^2 + 2(K_3 - x_3)^2 + 2.28(0.6x_3 + 0.2K_3)^2$$

при $0 \leq x_3 \leq K_3$.

Аналогично четвертому этапу исследуем только значения $x_3 = 0$ и $x_3 = K_3$.

При $x_3 = 0$ имеем $2.0912 K_3^2$, а при $x_3 = K_3$ соответственно $2.4592 K_3^2$. Выбираем $x_3 = K_3$ и при этом на трех последних этапах суммарные накопления будут составлять $2.4592 K_3^2$.

На втором этапе нужно найти x_2 из условия

$$x_2^2 + 2(K_2 - x_2)^2 + 2.4592 K_3^2 \rightarrow \max$$

при $0 \leq x_2 \leq K_2$.

$$\text{Здесь } K_3 = 0.8x_2 + 0.2(K_2 - x_2) = 0.6x_2 + 0.2K_2$$

и нам нужен максимум

$$x_2^2 + 2(K_2 - x_2)^2 + 2.4592 (0.6x_2 + 0.2K_2)^2$$

при $0 \leq x_2 \leq K_2$.

Максимум достигается при $x_2 = K_2$ и примерно равен $2.57492 K_2^2$.

Наконец, на первом шаге x_1 надо выбрать из условия максимума

$$x_1^2 + 2(K_1 - x_1)^2 + 2.5749(0.6x_1 + 0.2K_1)^2$$

при $0 \leq x_1 \leq K_1$.

Но K_1 задано и равно 10. Поэтому нужен максимум

$$x_1^2 + 2(10 - x_1)^2 + 2.5749(0.6x_1 + 2)^2$$

при $0 \leq x_1 \leq 10$.

Этот максимум достигается при $x_1 = 10$. В результате суммарные накопления при оптимальном управлении ресурсами составят примерно 265.8.

При этом первое предприятие получит по годам 10, 8, 6.4, 5.12, 0, а второе предприятие получит 0, 0, 0, 0, 4.096.

Точно накопления составят $100 + 64 + 6.4^2 + 5.12^2 + 2(4.096)^2$.

Полученный результат нельзя считать неожиданным. Действительно, при *равных вложениях* первое предприятие дает вдвое меньшие накопления, но вчетверо большие распределяемые средства (см. (3.4)). Естественно, что ему отдается приоритет в первые годы деятельности, чтобы обес-

печить развитие производства и максимальные суммарные накопления за пять лет. В последний год нас интересуют только накопления, а не дальнейшая деятельность предприятий, поэтому все распределяемые средства отданы второму предприятию. Не случайно функции $p_1(x)$ и $p_2(x)$ никак не влияют на поиск величины x_5 .

Может показаться, что такое распределение сохранится при неизменных функциях $d_1(x) = x^2$, $d_2(x) = 2x^2$ и при изменении коэффициентов в $p_1(x)$, $p_2(x)$, сохраняющем условие $p_1(x) > p_2(x)$. Однако это не так.

Сохраняя $d_1(x) = x^2$, $d_2(x) = 2x^2$, вместо $p_1(x) = 0.8x$, $p_2(x) = 0.2x$ в (3.4) возьмем $p_1(x) = 0.75x$, $p_2(x) = 0.3x$ и тот же начальный капитал K_1 . Очевидно, $x_5 = 0$. Для четвертого этапа имеем теперь

$$x_4^2 + 2(K_4 - x_4)^2 + 2(0.45x_4 + 0.3K_4)^2 \rightarrow \max$$

при $0 \leq x_4 \leq K_4$, т.к. на пятом году накопления равны $2K_5^2$, а $K_5 = 0.75x_4 + 0.3(K_4 - x_4)$

При $x_4 = 0$ суммарные накопления за два последних этапа составят $2.18 K_4^2$, а при $x_4 = K_4$ только $2.125 K_4^2$ и поэтому выбирать нужно $x_4 = 0$, а не $x_4 = K_4$, как было при коэффициентах 0.8 и 0.2.

Данный пример с точки зрения экономики, конечно, носит искусственный характер и приведен для того, чтобы продемонстрировать применение метода динамического программирования не при дискретном, а при непрерывном изменении параметров состояния системы и сведении рекуррентного уравнения Р. Беллмана к задаче поиска максимума квадратичной функции. Можно усложнить пример, задавая средства, направляемые на производство в долях от полученного дохода, а не от вложенных средств на предыдущем этапе, но при этом получаются более сложные уравнения.

3. Проектирование трасс линейных сооружений.

Одной из первых задач оптимального проектирования, которую в нашей стране предлагалось решать с помощью метода динамического программирования, была задача поиска оптимальной трассы новой железной дороги, соединяющей две заданные точки. Эти предложения относятся к началу 60-х гг. XX в. и связаны с применением уже упоминавшегося метода последовательного анализа вариантов [10], который принципиально ничем не отличается от метода динамического программирования. Задача поиска оптимальной трассы дороги, рассмотренная нами в разделе 2.8, состоит в поиске трехмерной кривой и не сводится к поиску оптимального пути на двумерной сетке.

Основной фактор, влияющий на поиск решения, — это рельеф местности. Попытки игнорировать или как-то усреднить рельеф по отдельным участкам местности, с тем чтобы свести задачу поиска пространственной кривой (трассы) к поиску ее проекции на горизонтальную плоскость (плана), не привели, да и не могли привести к практическим результатам. В этой связи приводимые в ряде работ (см. [4], с. 88–98) примеры поиска оптимальной трассы как оптимального пути на двумерной сетке следует рассматривать как учебные, не имеющие никакого отношения к реальной проектной задаче.

Проектирование трассы как пространственной кривой можно рассматривать как поиск оптимального пути на трехмерной сетке. Но при этом надо учитывать ограничения на ее элементы в плане и в профиле, которые серьезно осложняют поиск решения (см. раздел 2.8). По этой причине метод динамического программирования применялся для проектирования оптимального продольного профиля при *заданном положении трассы в плане*, т.е. для поиска плоской кривой. Эта задача может рас-

смагиваться как аппроксимация заданной ломаной линии (профиль земли) другой ломаной линией (проектный профиль), удовлетворяющей ряду ограничений (см. раздел 2.8). Если целевая функция может вычисляться последовательно по элементам (до того как построена вся ломаная), то задача распадается на ряд этапов. Очередной этап — это поиск очередного звена ломаной. В этом случае задача об оптимальном профиле дороги действительно сводится к поиску оптимального пути на двумерной сетке, располагаемой относительно профиля земли (на рис. 42 он показан жирной линией).

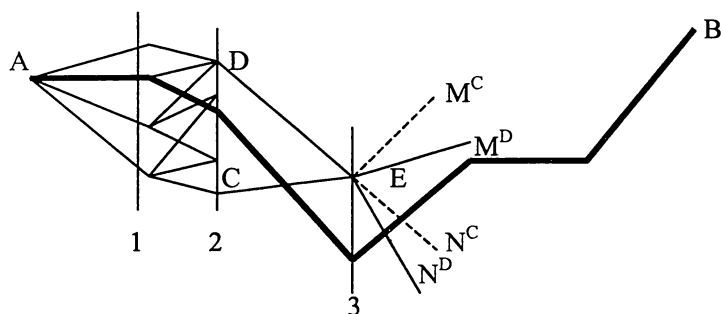


Рис. 42. Проектирование продольного профиля дороги

Искусственно вводится дискретность по вертикали, причем из-за наличия ограничений по уклонам и разностям уклонов смежных элементов точки на вертикалях должны располагаться достаточно часто. В данной задаче 0,01 м — это значимая величина.

Итак, «состояние системы» на конкретном этапе соответствует точке на нужной вертикали, а «управление» или «шаговый переход» — это отрезок, соединяющий две точки на смежных вертикалях. Естественно, рассматриваются только допустимые по ограничениям точки

на вертикалях и их соединения. Начиная со второй вертикали, при достижении любого из состояний несколькими путями они сравниваются, и остается путь (вариант) с меньшим значением целевой функции (сравнение «в точку»). Именно этот алгоритм был реализован еще в 60-х гг. [10]. Если число точек на каждой вертикали равно k , а число проектируемых элементов равно n , то всего приходилось вычислять целевую функцию примерно на nk^2 элементах, примерно столько же раз проверять выполнение ограничений и запоминать примерно nk чисел, что не так уж и много для реальных задач. При разработке алгоритма отмечалось, что он не вполне соответствует принципу оптимальности Р. Беллмана. Действительно, при наличии ограничений на разность уклонов варианты, сходящиеся в точке, сравнивать нельзя, так как возможные продолжения отброшенного варианта могут не содержаться среди возможных продолжений оставшегося. На рис. 42 варианты, сходящиеся в точке Е, имеют разные возможности для дальнейшего участия в построении проектной линии. Продолжения варианта СЕ содержатся в секторе M^CEN^C (показан пунктирной линией), а продолжения варианта DE содержатся в секторе M^DEN^D (показан сплошной линией). Эти сектора не только не совпадают, но в реальных задачах могут вообще не иметь ничего общего (не пересекаться) при малых допустимых значениях разности уклонов смежных элементов. Так, если один вариант приходит в общую точку снизу (уклон имеет знак плюс), а другой сверху (уклон имеет знак минус), то разность их уклонов может превышать удвоенную допустимую разность уклонов смежных элементов. Положительный уклон и на следующем элементе останется положительным, а отрицательный останется отрицательным. Получается, что для дальнейшего имеет значение не только «состояние системы», но и то, как оно

достигнуто, т.е. «предыстория». Реализация алгоритма последовательного анализа вариантов в строгом соответствии с методом динамического программирования требует другой формализации понятия «состояние системы» и, соответственно, другого правила сравнения и отбраковки вариантов. Если считать состоянием не точку, т.е. возможную вершину искомой ломаной, а элемент ломаной, то сравнимыми становятся только варианты, у которых последний элемент общий (см. рис. 40 — сравнение «в отрезок»). При этом число проверяемых связей на выполнение ограничений по разности уклонов составляет примерно nk^3 вместо nk^2 . А количество запоминаемых чисел также возрастает примерно в k раз. Естественно, можно хранить значения целевой функции только для состояний (отрезков) последнего из пройденных этапов, но для восстановления траектории при обратном развороте связи с состояниями на предыдущих этапах надо хранить *все*.

Из-за большого числа запоминаемых данных даже при упрощенном правиле отбраковки вариантов («сравнение в точку») задачу приходилось решать в несколько этапов. На первом этапе использовали сетку с крупными ячейками, но относительно малым числом точек на каждой вертикали, а на втором этапе разбивали более мелкую сетку вокруг полученного решения. Это еще один эвристический прием, который не гарантирует нахождение оптимума [16, с. 380].

Реально число дискретов на вертикали порядка нескольких десятков. Например, при ширине зоны поиска 5 м и дискретности поиска 0,1 м (что довольно грубо) уже получаем $k = 50$. Поэтому переход к «сравнению в отрезок» требовал существенно более мощных ЭВМ, чем те, что были доступны в 60-х гг. Это и объясняет попытку реализации некорректного алгоритма. Предполагалось, что

при этом могут получаться решения, незначительно отклоняющиеся от оптимальных. Но реально оказалось все значительно хуже, так как в условиях пересеченного рельефа при наличии высотных ограничений алгоритм с упрощенным правилом отбраковки вариантов вообще останавливался из-за «вырождения» вариантов. Это происходило из-за того, что при отбраковке вариантов оставались элементы с такими уклонами, что для выхода в конечную точку на заданный уклон или на фиксированную точку «не хватало места для разворота и набора высоты», т.е. ограничения по разности уклонов не позволяли продолжить процесс построения элементов.

Этот пример показателен тем, что различного рода «эвристические» поправки к математическим методам оптимизации могут приводить не просто к некоторым отклонениям от оптимума, но и вообще к неработоспособным алгоритмам и программам.

В дальнейшем (к 1975 г.) с ростом мощности доступных ЭВМ разработчиками метода последовательного анализа был реализован метод динамического программирования в «чистом» виде («сравнение в отрезок») и создан работоспособный комплекс программ. Но к тому времени выяснилось, что в условиях пересеченного рельефа исходное предположение о том, что целевую функцию можно вычислять по элементам, не имея всей проектной линии, чаще всего не выполняется. Дело в том, что при строительстве дорог насыпи сооружаются из грунта выемок и, если этого грунта недостаточно, используется привозной грунт. Строительные затраты существенным образом зависят не только от объемов работ, но и от соотношения объемов насыпей и выемок, грунт которых можно использовать. Но это соотношение меняется при вариациях проектной линии. Оптимум чаще всего соответствует балансу грунтов. При оптимизации не

объемов работ, а стоимостей нужно задать единичные затраты (на 1 м^3), которые различны при различном соотношении объемов грунта в насыпях и в выемках. Оказалось, что при задании единичных затрат в расчете на использование грунта выемок для сооружения насыпей, как правило, получается вариант с недостатком грунта в выемках, и наоборот. Получается, что для вычисления значения целевой функции нужно иметь проектную линию *полностью*, вычислить соответствующие ей объемы работ и только потом — строительные затраты. Это означает неприменимость динамического программирования при решении задачи с учетом данного фактора, который никак нельзя считать второстепенным. Именно поэтому задача была решена методами нелинейного программирования, которые на каждой итерации имеют дело с проектной линией как с единым целым [15].

Сказанное не означает неприменимость динамического программирования к проектированию продольного профиля других линейных сооружений. Например, в проектировании трасс трубопроводов и вообще траншей различного назначения система ограничений практически та же, что и в проектировании дорог, но нет дополнительной взаимосвязи элементов, так как нет балансировки грунтов.

Метод динамического программирования был успешно реализован и при проектировании реконструкции железных дорог.

4. *Задача о защите поверхности.* Для этой задачи мы уже построили две математические модели и с их помощью свели ее к задаче целочисленного линейного программирования (раздел 1.9) с большим числом переменных, а затем к задаче нелинейного программирования при резком сокращении числа переменных (раздел 2.8). В последнем случае в рассмотрение были введены цено-

вые графики (рис. 29), каждый из которых представляет собой зависимость стоимости защиты конкретного элемента поверхности c_i от допускаемых потерь d_i , возникающих от недостаточно полной защиты элемента. Графики были представлены как монотонно убывающие кусочно-линейные функции (ломанные линии). Фактически же нас должны интересовать только узловые точки графиков (вершины ломаных). Каждый узел соответствует одному из способов защиты данного элемента поверхности (детали изделия). Задача состоит в том, чтобы на каждой графике взять один узел (вершину ломаной) так, чтобы сумма ординат взятых узлов была минимальна (минимум стоимости защиты всей поверхности), а сумма абсцисс не превышала заданной величины, т.е. максимально допустимых суммарных потерь.

Эту задачу будем рассматривать как задачу распределения ограниченного ресурса (D) и попытаемся ее решить с помощью метода динамического программирования. Ресурсом будем считать допускаемые потери от не полной защиты. Этапы принятия решений соответствуют выбору способа защиты соответствующего элемента, причем как обычно нумерация элементов существенного значения не имеет. На каждом из этапов «система» может находиться в нескольких состояниях. Каждое состояние соответствует уже распределенному ресурсу, т.е. уже допущенным суммарным потерям от неполной защиты всех рассмотренных элементов. Каждому состоянию соответствует и свое значение целевой функции, т.е. суммарная стоимость защиты уже рассмотренных элементов. Начальное состояние можно задать одной точкой, которой соответствуют нулевые потери (ресурс не использован) и нулевые затраты. Состояния на каждом последующем этапе можно изобразить точками на вертикальных прямых (рис. 43).

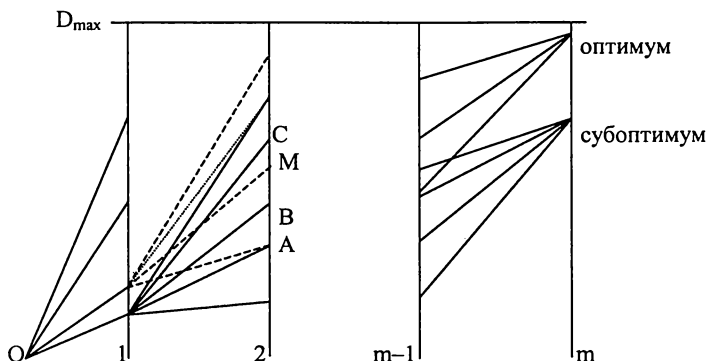


Рис. 43. Схема выбора оптимального способа защиты

На первой вертикали нижняя точка соответствует способу защиты первого элемента, при котором потери (использованный ресурс) минимальны, а верхняя точка соответствует способу, ведущему к максимальным потерям. При переходе снизу вверх от одной точки к другой потери растут, а соответствующая им стоимость защиты уменьшается. Если это не так и существует способ защиты первого элемента, который стоит дороже, но дает потери (требует ресурс) больше, чем один из рассмотренных, то такой способ защиты *данного элемента* должен быть исключен из рассмотрения как неконкурентный, и соответствующая ему точка не получает своего места на первой вертикали.

Далее, рассматриваем все возможности защиты второго элемента в комбинации с уже предполагаемыми способами защиты первого элемента, т.е. начинаем «заселять» вторую вертикаль. Для начала считаем, что выбрана нижняя точка на первой вертикали (самый дорогой способ защиты первого элемента), и рассматриваем в комбинации с ним все возможные способы защиты второго элемента. Получаем несколько точек на второй вертикали (состояний системы на

втором этапе). Каждой точке соответствуют *суммарные для двух элементов* потери и стоимость при соответствующих способах их защиты. Причем, как и на первой вертикали, при переходе снизу вверх потери растут, а стоимость (затраты) падает. Если для некоторой комбинации суммарные потери превышают допустимое значение, то такая комбинация исключается и соответствующая ей точка на вертикаль не помещается. Этого правила будем придерживаться при рассмотрении всех возможных комбинаций на всех последующих этапах.

Переходим к рассмотрению всех возможных способов защиты второго элемента в комбинации не с первым, а со вторым способом защиты первого элемента (пунктирные линии на рис. 43). Может оказаться, что суммарные потери при одной из таких комбинаций совпадают с потерями для одной из уже рассматривавшихся комбинаций (рис. 43, точка А на вертикали 2). Это означает, что одного и того же состояния на втором этапе система может достичь *разными путями*. Дальнейшее поведение системы при переходе из этого состояния не зависит от того, как она попала в это состояние, так как дальнейшее поведение — это в нашей задаче есть распределение оставшегося ресурса (допустимых потерь) при выборе способов защиты оставшихся элементов. На этот дальнейший выбор никак не может повлиять то, как были выбраны способы защиты первых двух элементов, если они привели к одним и тем же потерям, т.е. привели систему в одно и то же состояние. В полном соответствии с принципом оптимальности Р. Беллмана из нескольких путей, приводящих систему в одно и то же состояние, в данной задаче можно оставить только один путь, для которого суммарные затраты (целевая функция) минимальны. Для каждой точки на вертикали 2 сделаем такой выбор и запомним соответствующую связь с вертикалью 1.

А что делать, если новая точка не совпадает ни с одной из уже имеющихся на вертикали 2 (на рис. 43 точка М между точками В и С)? Казалось бы, получено новое состояние системы на втором этапе, которое нельзя сравнивать ни с одним из имеющихся, и число таких состояний должно резко возрастать при переходе от этапа к этапу. Как следствие, редкие случаи отбраковки вариантов должны привести к неэффективности алгоритма динамического программирования уже при числе элементов и способов их защиты в несколько десятков. Но это не так.

Дело в том, что в данной задаче можно *сравнивать и отбраковывать состояния системы* на каждом из этапов, а не только пути достижения фиксированного состояния. Другими словами, новая точка на вертикали 2 может не получить своего места или заменить одну (или даже несколько) точек, если она и не совпадает ни с одной из них. Рассматриваемой новой точке М на вертикали 2 соответствуют потери большие, чем точке В, но меньшие, чем точке С. Если при этом суммарные затраты, соответствующие точке М, больше, чем затраты, соответствующие точке В, то это означает, что состояние М проигрывает состоянию В «по всем показателям». Возникает вопрос: а нужно ли вообще рассматривать такое состояние, может ли оно попасть в оптимальную траекторию или, другими словами, может ли комбинация способов защиты двух элементов, соответствующая точке М, конкурировать с другой комбинацией, соответствующей точке В? Очевидно, нет, и вот почему. Возьмем оптимальный путь из точки М до конца, т.е. оптимальное распределение оставшегося ресурса при условии нахождения в точке М. Иначе говоря, рассмотрим оптимальный вариант защиты оставшихся элементов при условии, что для первых двух элементов выбраны способы защиты такие, что суммарные потери соответствуют точке М. Но

ничто не мешает взять точно такие же способы защиты оставшихся элементов при условии, что на первых двух элементах выбраны способы защиты, соответствующие попаданию в точку В. При этом и суммарные потери, и затраты на оставшихся элементах одни и те же что для точки М, что для точки В. Но на первых двух элементах путь в точку В имеет преимущество (и потери меньше, и затраты меньше), поэтому в такой ситуации нет смысла запоминать точку М.

Предположим, что суммарные затраты на первых двух элементах для точки М меньше, чем для точки В ($Z_M < Z_B$). Это означает, что точку М нельзя исключить из дальнейшего рассмотрения. Но может оказаться, что эти затраты меньше и соответствующих затрат для точки С ($Z_M \leq Z_C$). Тогда точка С не может конкурировать с точкой М (она хуже по всем показателям, как это было ранее при сравнении точек М и В). Это означает, что точка С должна быть исключена, а точка М оставлена. Более того, может оказаться, что точка М «убивает» еще одну или несколько точек, которые находятся выше точки С на вертикали 2 и соответствуют затратам большим, чем для точки М. Все такие точки также исключаются из дальнейшего рассмотрения. Итак, на вертикали 2 при всех возможных переходах из всех точек вертикали 1 остаются только точки, для которых при движении снизу вверх (в направлении возрастания суммарных допущенных потерь) убывают суммарные затраты на защиту уже рассмотренных элементов.

Аналогично поступаем при переходе на вертикаль 3, а затем и на все последующие вертикали. Это правило, конечно, сокращает и число состояний на каждом этапе, и число запоминаемых связей. Однако в общем случае оно не спасает от резкого роста числа состояний уже при числе элементов (этапов) в несколько десятков. Это означает, что на интервале допустимых потерь (от нуля до D_{\max}) на-

капливается большое число точек. Если это число достигает 1000 и более, то возникает вопрос: а нужно ли хранить столько точек, велика ли разница между соответствующими им состояниями? Как только эта разница в потерях (расстояние на вертикали) между соседними точками становится существенно меньше погрешности определения этих потерь, из двух смежных точек можно оставить одну.

При достижении последней вертикали, т.е. при завершении рассмотрения всех способов защиты последнего элемента в комбинации с уже рассмотренными способами защиты всех остальных элементов на последней вертикали будут помещены все возможные точки (состояния). Каждая из них не нарушает ограничение на суммарные потери и соответствует суммарным потерям и суммарным затратам при соответствующей комбинации способов защиты элементов. При этом верхней точке соответствуют максимальные (но допустимые) суммарные потери и минимальные суммарные затраты. Эта точка и дает решение задачи. Вторая сверху точка на последней вертикали дает меньшие потери, но большие затраты, чем оптимальная (верхняя) точка. Это так называемое субоптимальное решение. С учетом всякого рода дополнительных соображений это решение может быть признано лучшим, чем формальный оптимум. Например, разница в затратах незначительна, а в потерях весьма существенна, так что для оптимальной точки эти суммарные потери достигают допустимого максимума, а для субоптимальной точки имеется «запас прочности».

Рассмотрим, при каких исходных значениях числа элементов m и числа способов их защиты n может быть решена задача на современных доступных компьютерах с точки зрения требуемой памяти.

Для каждой точки на каждой вертикали надо хранить 3 числа: суммарные потери на пройденных этапах, суммарные затраты и связь с предыдущей вертикалью (для восста-

новления траектории при обратном развороте), т.е. всего порядка $3km$ чисел. Здесь k — число точек на вертикали. На первой вертикали n точек, а далее это число растет и определяется не только числом n , но и конкретными данными по затратам и потерям для различных элементов и способов их защиты. Будем рассматривать наихудший случай и примем число k равным максимальному числу точек на вертикали, определяемому требуемой точностью расчетов. Пусть $k = 1000$. Тогда даже при $m = 1000$ требования к оперативной памяти отнюдь не чрезмерные (не более 12 мегабайт), так что с этой точки зрения любое из чисел m и k можно увеличить на порядок. Численные расчеты для задач, в которых m было несколько десятков и $k = 1000$, показали высокую эффективность описанного алгоритма и с точки зрения времени счета.

5. Задача о размещении оборудования.

Мы уже рассматривали эту задачу в разделе 1.9 и свели ее к задаче целочисленного линейного программирования. Задача заключалась в выборе тех объектов, суммарная полезность (стоимость) которых максимальна, а некоторая ограничивающая характеристика (объем, вес и др.) не превышает заданного числа или заданных чисел, если имеют место ограничения по нескольким показателям. Каждый из ограничивающих показателей можно рассматривать как ресурс. Если есть только один ограничивающий показатель (ресурс), например объем, то эта задача с формальной точки зрения ничем не отличается от рассмотренной нами задачи защиты поверхности. Обе задачи представляют собой поиск оптимального распределения ограниченного ресурса. То обстоятельство, что в данной задаче целевая функция должна достигать максимума (суммарная полезность), а не минимума (суммарные затраты), значения не имеет. Более того, с вычислительной точки зрения эта задача даже проще рассмотренной выше задачи о защите поверхности, так как

из каждого состояния есть только два возможных перехода. Первый из них соответствует решению не брать рассматриваемый объект. При этом уже использованный ресурс остается неизменным. Второй переход соответствует решению взять рассматриваемый объект, при этом использованный ресурс увеличивается на соответствующую данному объекту величину. Если полученная сумма не превосходит максимально допустимый ресурс, получаем новое «состояние системы».

Если число ограничивающих показателей (видов ресурсов) больше одного, то требуется не один, а несколько параметров, определяющих «состояние системы», но принципиально в алгоритме оптимизации ничего не изменяется. Однако с ростом числа параметров (видов ресурсов) резко растут вычислительные трудности.

Отметим одно важное обстоятельство: для решения данной задачи возможна такая же модификация алгоритма динамического программирования, как и использованная нами при решении задачи о защите поверхности. Речь идет о возможности существенного сокращения числа рассматриваемых вариантов за счет отбраковки бесперспективных состояний на промежуточных этапах, а не только бесперспективных вариантов достижения промежуточных состояний. Эту модификацию алгоритма рассмотрим на конкретной задаче загрузки машины, решение которой в течение многих лет приводится в различных источниках как пример применения метода динамического программирования. Конкретные данные и решение по исходному алгоритму возьмем из книги Е.С. Вентцель «Исследование операций. Задачи, принципы методология» [см. [4], с. 104–106 и все переиздания].

Задача состоит в следующем. Имеется автомашина грузоподъемностью $Q = 35$ единиц веса и шесть предметов, веса и стоимости которых указаны в таблице 3.2.

Таблица 3.2

Предмет Π_i	Π_1	Π_2	Π_3	Π_4	Π_5	Π_6
Вес q_i	4	7	11	12	16	20
Стоимость c_i	7	10	15	20	27	34

Суммарный вес предметов превышает грузоподъемность машины, и поэтому требуется эту грузоподъемность использовать оптимальным образом, т.е. взять такие предметы, *суммарный вес которых не превышает $Q = 35$, а суммарная стоимость максимальна.*

Согласно [4], с. 104, рассматривается шесть шагов (этапов), на каждом из которых принимается решение, брать соответствующий предмет в машину или не брать. Номер шага соответствует номеру предмета в табл. 3.2. Управление на каждом шаге равно единице, если мы берем данный предмет, и нулю — если не берем. На каждом шаге всего лишь два возможных управления.

Состояние системы S перед очередным шагом характеризуется весом, который еще остался в нашем распоряжении до полной загрузки машины после того, как предыдущие шаги выполнены, т.е. решен вопрос брать или не брать предметы с меньшими номерами и какие-то из них погружены в машину. Для каждого состояния S предлагается найти $W_i(S)$ — суммарную максимальную стоимость предметов, которыми можно «догрузить» машину при данном значении S , и положить $x_i(S) = 1$, если мы берем данный (i -ый) предмет, и $x_i(S) = 0$, если не берем. Величины $x_i(S)$ ($i = 1, 2, \dots, 6$) — это условные оптимальные шаговые управления. Их последовательность определяет набор взятых предметов. Они называются условными, так как зависят от состояния S .

В соответствии с решением, изложенным в [4, с. 105–106], рассматриваются только целые значения и для S получается 36 значений (от 0 до 35 включительно). Выбор цело-

численных значений очевиден, а вот рассмотрение такого большого числа состояний ничем не диктуется и не является необходимым, если рассматривать процесс начиная не с последнего, а с первого шага. Вместо этого в [4] рассматривается «классический» алгоритм (от конца к началу), т.е. вначале рассматриваются все переходы на шестом шаге в конечное состояние. Можно, конечно, просчитать все возможные комбинации на первых пяти шагах и выявить возможные состояния системы на шестом шаге (т.е. сколько могло остаться ресурса грузоподъемности после того, как определились варианты решений относительно первых пяти предметов). В данной задаче таких состояний 32, а допустимых по грузоподъемности и того меньше. Но при большом числе шагов этот перебор просто нецелесообразен. Поэтому и рассматриваются все 36 состояний перед шестым шагом как возможные. Первые 20 из них (осталось от 0 до 19 единиц грузоподъемности) не позволяют взять шестой предмет, вес которого 20 единиц. Им соответствует $x_6 = 0$, $W_6 = 0$. Остальным 16 состояниям соответствует $x_6 = 1$, $W_6 = 34$, так как стоимость шестого предмета равна 34. Переходя к пятому шагу, снова вынуждены рассматривать 36 возможных состояний и для каждого из них определять оптимальное шаговое управление. Для состояний с S от 0 по 15 включительно выбора нет, так как вес пятого предмета 16, а осталось меньше. Для состояний с S с 16 по 19 включительно можно взять пятый предмет или не брать. Если не брать, на шестой все равно не хватит грузоподъемности (нужно не менее 20). Поэтому для этих состояний $x_5 = 1$, $W_5 = 27$. Для состояний с S от 20 по 35 есть выбор: брать пятый (следовательно, не брать шестой) или не брать пятый, но взять шестой предмет. Для этих состояний $x_5 = 0$, $W_5 = 34$ ($x_6 = 1$). Аналогично приходится поступать вплоть до первого шага. В целом процесс поиска оптимального варианта загрузки машины представлен в табл. 3.3.

Таблица 3.3

S	i=6		i=5		i=4		i=3		i=2		i=1	
	x_i	W_i	x_i	W_i	x_i	W_i	x_i	W_i	x_i	W_i	x_i	W_i
0	<u>0</u>	0	0	0	0	0	0	0	0	0		
1	0	0	0	0	0	0	0	0	0	0		
2	0	0	0	0	0	0	0	0	0	0		
3	0	0	0	0	0	0	0	0	0	0		
4	0	0	0	0	0	0	0	0	0	0		
5	0	0	0	0	0	0	0	0	0	0		
6	0	0	0	0	0	0	0	0	0	0		
7	0	0	0	0	0	0	0	0	1	10		
8	0	0	0	0	0	0	0	0	1	10		
9	0	0	0	0	0	0	0	0	1	10		
10	0	0	0	0	0	0	0	0	1	10		
11	0	0	0	0	0	0	1	15	0	15		
12	0	0	0	0	1	20	0	20	0	20		
13	0	0	0	0	1	20	0	20	0	20		
14	0	0	0	0	1	20	0	20	0	20		
15	0	0	0	0	1	20	0	20	0	20		
16	0	0	<u>1</u>	27	0	27	0	27	0	27		
17	0	0	1	27	0	27	0	27	0	27		
18	0	0	1	27	0	27	0	27	0	27		
19	0	0	1	27	0	27	0	27	1	30		
20	1	34	0	34	0	34	0	34	0	34		
21	1	34	0	34	0	34	0	34	0	34		
22	1	34	0	34	0	34	0	34	0	34		
23	1	34	0	34	0	34	1	35	1	37		
24	1	34	0	34	0	34	1	35	1	37		
25	1	34	0	34	0	34	1	35	1	37		
26	1	34	0	34	0	34	1	35	1	37		
27	1	34	0	34	0	34	1	42	1	44		
28	1	34	0	34	<u>1</u>	47	<u>0</u>	47	0	47		
29	1	34	0	34	1	47	0	47	0	47		

Табл. 3.3 (продолжение)

S	i=6		i=5		i=4		i=3		i=2		i=1	
	x_i	W_i	x_i	W_i	x_i	W_i	x_i	W_i	x_i	W_i	x_i	W_i
30	1	34	0	34	1	47	0	47	0	47		
31	1	34	0	34	1	47	1	49	0	49		
32	1	34	0	34	1	54	0	54	0	54		
33	1	34	0	34	1	54	0	54	0	54		
34	1	34	0	34	1	54	0	54	0	54		
35	1	34	0	34	1	54	0	54	<u>1</u>	57	<u>0</u>	<u>57</u>

В каждом состоянии мы запоминаем свой оптимальный выбор (для пятого состояния это нуль), что дает возможность установить, в какое состояние мы переходим на этом шаге (сохраняется или уменьшается остаток грузоподъемности). Кроме того, запоминаем и максимальную стоимость предметов, которые можно погрузить при оптимальном использовании имеющейся в нашем распоряжении грузоподъемности (на пятом шаге это $W_5 = 34$).

Оптимальное решение (выделенные значения x_i в табл. 3.3) состоит в том, чтобы взять второй, четвертый и пятый предмет. При этом получается максимальная стоимость 57 единиц.

Характерно, что даже на втором шаге при такой формализации задачи рассматриваются все те же 36 состояний, хотя очевидно, что на первые два предмета не только нельзя полностью истратить весь ресурс грузоподъемности, но в любом случае останется 24 единицы. Поэтому большинство состояний (соответственно, обилие нулей в табл. 3.3) вообще можно было бы не рассматривать.

Рассмотрим теперь другую модель той же задачи и другой алгоритм. Прежде всего отметим, что большему весу соответствует большая стоимость каждого предмета. Однако даже если бы оказалось, что это не так и для некото-

рого из предметов P_1 нашелся бы другой предмет P_2 с меньшим весом и большей стоимостью, то это означало бы только то, что P_2 имеет преимущество и в оптимальном наборе не может быть P_1 без P_2 . Но исключить из рассмотрения P_1 нельзя, так как они оба могут попасть в оптимальный набор, если среди других предметов не все заведомо выгоднее, чем P_1 . В этом смысле примечание «заранее ясно, что загружать машину предметами большого веса и малой стоимости нецелесообразно» [4, с. 105] требует уточнения.

Каждый предмет существует в одном экземпляре, и условия выбора в этой задаче несколько отличаются от условий выбора способов в задаче о защите поверхности.

Итак, будем решать задачу в прямом направлении и рассмотрим те же шаги. На первом шаге решаем вопрос о первом предмете, на втором — о втором и т.д. *Состоянием системы на каждом шаге* будем считать ресурс уже использованной грузоподъемности, т.е. суммарный вес уже погруженных предметов. Перед первым шагом состояние системы характеризуется числом нуль. После этого шага система может оказаться в одном из двух состояний (а не из 36). Первое состояние характеризуется по-прежнему числом нуль (первый предмет не взяли), а второе состояние характеризуется числом 4 (взяли первый предмет). Им соответствуют стоимости 0 и 7. После второго шага (когда решили вопрос о втором предмете весом 7 и стоимостью 4) система может оказаться в одном из четырех состояний 0(0), 4(7), 7(10) и 11(17). Первая цифра — использованный ресурс, а вторая — суммарная стоимость. Эти состояния означают соответственно для первых двух предметов: не брать ни одного из них, взять только первый, взять только второй, взять оба. После третьего шага могут остаться те же состояния, если не брать третий предмет весом 11 и стоимостью 15. Но могут возникнуть и

новые, если берем третий предмет. Например, первые два предмета не брали, а третий решили взять. Тогда система после третьего шага перейдет в состояние 11(15). А если взять два первых, а третий не брать, то система перейдет в состояние 11(17). Это одно и то же состояние, но достигается оно разными путями (разная стоимость). В соответствии с принципом оптимальности путь 0, 0, 1 (брать только третий) можно далее не рассматривать и ни в какие дальнейшие комбинации не включать. Поэтому после третьего шага останется семь состояний, а не восемь: 0(0), 4(7), 7(10), 11(17), 15(22), 18(25), 22(32). Эти же состояния получим после четвертого шага, если не возьмем четвертый предмет весом 12 и стоимостью 20.

Теперь рассмотрим те состояния после четвертого шага, в которые можно перейти из состояний после третьего шага, если взять четвертый предмет. Из состояния 0(0) перейти можно в состояние 12(20). Оно попадает между состояниями 11(17) и 15(22) и логично занимает свое место (больше истраченный ресурс — больше и суммарная стоимость). Но вот из состояния 4(7) попадаем в состояние 16(27). Оно попадает между возможными состояниями 15(22), 18(25). Вот тут начинается самое интересное. Судьбу четырех первых предметов можно решить так, чтобы при общем весе $16 (4 + 0 + 0 + 12)$ получить суммарную стоимость 27 $(7 + 0 + 0 + 20)$. Спрашивается, следует ли далее рассматривать состояние 18(25), которое соответствует для этих же четырех предметов распределению ресурса $(0 + 7 + 11 + 0)$ со стоимостями $(0 + 10 + 15 + 0)$? Состояние 18(25) бесперспективно и его можно исключить из дальнейшего рассмотрения вообще. Действительно, истратив всего 16 единиц грузоподъемности вместо 18, всегда проще размещать оставшиеся предметы. Здесь мы *сравниваем* не варианты (пути) достижения конкретного состояния, а *сами состояния*, устанавливаем бесперспек-

тивность состояния 18(25) и исключаем его вместе со всеми его продолжениями.

В итоге после четвертого шага получаем 13, а не 16 состояний: 0(0), 4(7), 7(10), 11(17), 12(20), 15(22), 16(27), 19(30), 22(32), 23(37), 27(42), 30(45) и 34(52).

На пятом шаге решается вопрос: брать или не брать P_5 весом 16 и стоимостью 27 единиц. Если не брать, то перейдем в те же 13 состояний. А если брать, то получим несколько новых.

Рассуждая как и ранее, установим бесперспективность состояний 22(32) (так как появится состояние 20(34)), а также состояний 27(42), 30(45) и 34(52). Некоторые переходы оказываются недопустимыми из-за превышения лимита грузоподъемности ($Q = 35$). В итоге после пятого шага останется для дальнейшего анализа только 15 (а не 32 и не 36) состояний 0(0), 4(7), 7(10), 11(17), 12(20), 15(22), 16(27), 19(30), 20(34), 23(37), 27(44), 28(47), 31(49), 32(54) и 35(57).

Осталось решить, брать или не брать шестой предмет весом 20 и стоимостью 34 единицы. Если не брать, то состояние 35(57) окажется наилучшим, а если брать, то можно получить 35(56). Все остальное или превышает в сумме 35 по весу, или дает меньше по стоимости. Результат: максимальная стоимость составляет 57. Если мы запоминали для каждого состояния связь с предыдущим шагом (брали или не брали соответствующий предмет), то обратным разворотом восстанавливаем, какие предметы были взяты (второй, четвертый и пятый). Отметим, что ни на одном из шагов мы не оставляли более 15 состояний. В целом процесс выбора оптимального варианта представлен на рис. 44. Пунктиром показаны бесперспективные переходы. Оптимальный вариант показан жирной линией.

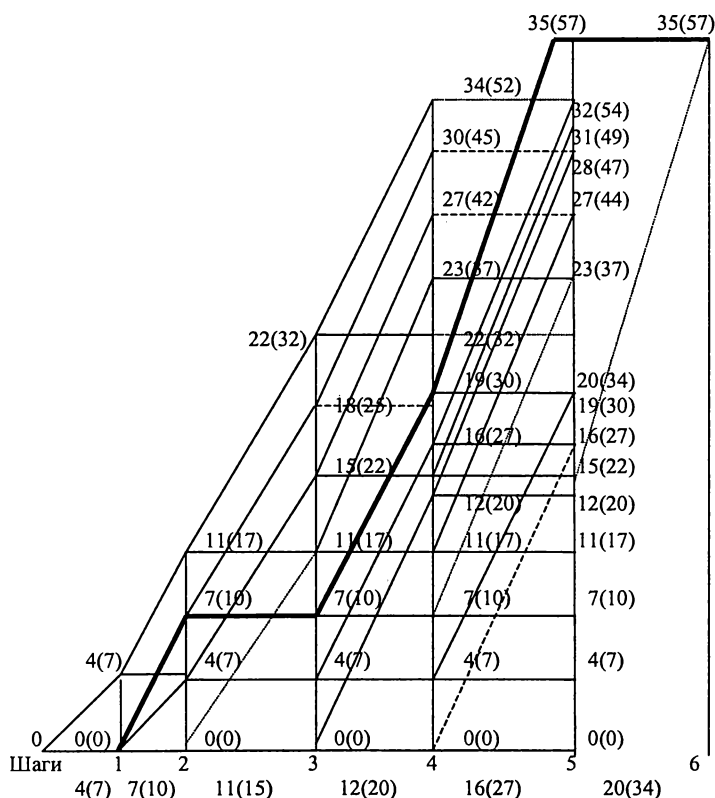


Рис. 44. Схема поиска оптимального варианта

Как отмечено в [4], для динамического программирования, в отличие от полного перебора вариантов, увеличение числа шагов не страшно: оно приводит только к пропорциональному возрастанию объема расчетов. Это верно при фиксированном числе состояний на каждом шаге. Значительно важнее сократить число этих состояний, что и демонстрирует модифицированный нами алгоритм. Если в данной конкретной задаче можно обойтись и без такого снижения числа состояний, так как имеется только два воз-

можных перехода из каждого состояния, то в более сложных задачах, таких как задача о защите поверхности, это обстоятельство весьма существенно.

Данный пример показателен в нескольких отношениях.

Во-первых, формализация понятия состояния может быть разной. Нет смысла вводить состояния, которые в принципе не достижимы. Так и число состояний и сами состояния на каждом этапе совсем не обязательно определять заранее. В [4] это число состояний определяется дискретностью-поиска и равно числу дискретов в общей грузоподъемности Q , что *больше, чем общее число вариантов при полном переборе на пяти шагах*. При увеличении Q число состояний при такой формализации растет (при том же дискрете), а число вариантов при полном переборе остается неизменным. Это число равно 2^N , где N — число предметов. Получается, что при такой формализации задачи метод динамического программирования не имеет преимущества перед полным перебором. Но при другой формализации задачи все встает на свои места и эффективность алгоритма динамического программирования несомненна.

Во-вторых, существенное значение для разработки алгоритма поиска оптимального решения имеет выбор направления: от начала к концу или наоборот.

В-третьих, специфические особенности этой (и не только этой!) задачи позволяют модифицировать метод динамического программирования и отбраковывать не только варианты достижения промежуточных состояний, но и сами бесперспективные промежуточные состояния. К сожалению, это удастся далеко не всегда.

3.5. Обучающая программа `BELLMAN.exe`

Для освоения метода динамического программирования, иллюстрации его возможностей и условий применимости

разработана специальная обучающая компьютерная программа BELLMAN.exe.

Обучение ведется на конкретной задаче поиска оптимального пути на двумерной сетке, которую мы рассматривали выше. Эта задача может служить моделью многих практических задач оптимизации.

Программа позволяет выполнить расчет и найти оптимальный путь при различных размерах сетки. Для иллюстрации эффективности метода динамического программирования в программе реализован также и метод полного перебора вариантов.

Для наглядности реализация алгоритмов полного перебора и динамического программирования сопровождается графическим изображением процесса поиска оптимального варианта.

Компьютерная программа дает возможность не только понять принцип оптимальности Р. Беллмана, уяснить преимущества метода динамического программирования по сравнению с полным перебором вариантов, но и проверить свои знания в данной области.

Программа не предъявляет высоких требований к объему оперативной памяти компьютера и его быстродействию и может использоваться практически на любой персональной ЭВМ.

В принципе никаких специальных знаний, выходящих за пределы программы средней школы, для знакомства с динамическим программированием непосредственно с помощью обучающей программы не требуется.

Вопросы, которые программа задает пользователю, относятся к пониманию сущности принципа оптимальности и области применения динамического программирования. Ответы непосредственно на эти конкретные вопросы вряд ли можно найти в литературных источниках, но, получив с помощью программы информацию о своих ошибках и в

конечном итоге правильный ответ, пользователь всегда сможет вникнуть в существо дела.

Программа может быть полезна также преподавателю как для иллюстрации возможностей динамического программирования, так и для проверки понимания студентами особенностей метода, условий его применимости и характеристики области использования. Она дает пользователю возможность реально представить сложность решения практических задач большой размерности, в конкретных расчетах ощутить «проклятие размерности».

Программа предлагает несколько тестов, фиксирует правильные и неправильные ответы, дает возможность проверить знания динамического программирования, поразмыслить над ошибками и закрепить полученные знания.

Программа последовательно предъявляет пользователю изображения и текст на экране (странице). Если не требуется детальный анализ страницы, то переход к следующей странице происходит автоматически. В противном случае программа ждет вмешательства пользователя. Если страница содержит вопрос к пользователю и варианты ответа на него, то в случае правильного выбора варианта ответа выдается сообщение «Правильный ответ» и программа переходит к следующей странице без вмешательства пользователя. Если выбран неправильный вариант ответа, то после сообщения «Неправильный ответ» появляется запрос «Хотите повторить?» и меню «Да», «Нет». При выборе «Да» появляется страница с тем вопросом, на который был дан неверный ответ, а при ответе «Нет» появляется следующая страница. Если на поставленный вопрос (например, о применимости метода динамического программирования в конкретных условиях) возможен ответ только «Да» или «Нет», то при любом ответе появляется следующая страница, но при этом фиксируется наличие или отсутствие ошибки. Для особо важных вопросов из этого

правила сделано исключение, и даже при наличии только двух вариантов ответа после ошибочного выбора варианта есть возможность вернуться к вопросу. Это сделано с целью предоставить возможность пользователю внимательно изучить вопрос.

Вначале программа предлагает пользователю убедиться в бесперспективности метода полного перебора при поиске оптимального пути на двумерной сетке. Она формулирует задачу, задает несколько простых вопросов на понимание ее условия и особенностей и предлагает пользователю вопрос о количестве путей из точки А в точку В, т.е. о числе вариантов, из которых предстоит сделать выбор при полном переборе вариантов. Это число определяется размерами сетки m и n . Пользователю предъявляются несколько формул для расчета числа вариантов, в частности mn , $(m+n)!/(m!n!)$, $(m+n)!/(m!n!)$ и др. Для правильного ответа на этот вопрос нужно уяснить, что каждому варианту пути из точки А в точку В соответствует ровно m шагов по вертикали и ровно n шагов по горизонтали, но последовательность этих шагов для каждого пути своя. Если шагу по горизонтали соответствует 0, а шагу по вертикали 1, то очевидно, что вариант пути — это выбор размещения m единиц по $m+n$ местам (оставшиеся n мест займут нули). Для размещения первой единицы имеется $m+n$ возможностей, для второй $m+n-1$ и т.д. В итоге получаем формулу с факториалами $(m+n)!/(m!n!)$.

Чтобы показать, что стоит за этими факториалами, программа предлагает примерно оценить число вариантов при различных значениях m и n и дает различные варианты ответа. Убедившись в том, что уже при значениях этих чисел 10 и более, число вариантов составляет сотни тысяч, пользователь может задать размеры сетки (например 5×5) и посмотреть как идет процесс перебора вариантов и сколько времени он занимает (шаговые затраты задавать не нужно,

они выбираются программой случайным образом и показываются на экране вместе с сеткой).

Если расчет занимает много времени (а при размерах сетки 10×10 и более это так и есть), можно прервать расчет нажатием любой клавиши. Как при завершении расчета, так и при его прерывании на экране показывается красным цветом достигнутое решение (при окончании расчета — это оптимальный путь) и суммарная стоимость. Пользователю предоставляется возможность повторить расчет при других m и n .

Рекомендуется сначала выполнить расчет при малых m и n (не более 5), получить решение и убедиться в том, что оно действительно оптимально (для этого на экране есть все необходимое), а затем повторить расчет при больших m и n (например, при $m = n = 10$), если хватит терпения, дождаться окончания расчета или прервать его. В любом случае программа сохранит исходные данные для того, чтобы потом заново выполнить этот же расчет по методу динамического программирования.

Поскольку перебор вариантов упорядочен и текущий вариант пути выделен (зеленым цветом), посмотрев на экран, можно представить, много ли времени займет расчет.

Начальный вариант соответствует последовательности из m единиц и n нулей, при этом на экране зеленой линией обведены левая вертикальная и верхняя горизонтальная границы сетки. Например, для сетки 4×4 начальная последовательность 11110000.

Первая серия вариантов соответствует перемещению последней единицы вправо, т.е. зеленая линия имеет один уступ, который смещается вправо (рис. 45)

В течение обработки всей первой серии вариантов вертикальная зеленая линия совпадает с левой вертикальной границей сетки.

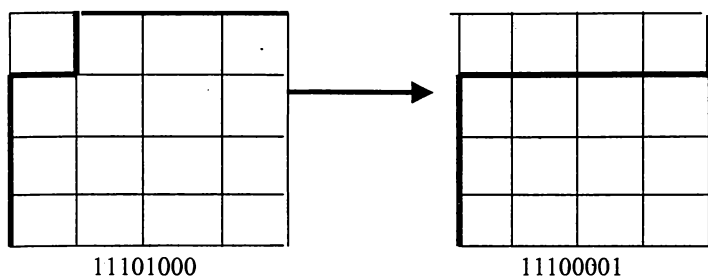


Рис. 45. Первая серия вариантов

Вторая серия вариантов соответствует переходу от последовательности 11011000 к последовательности 11010001. Третья серия — это переход от 11001100 к 11001001 и т.д. Левая зеленая вертикальная линия укорачивается только после перебора всех вариантов, не требующих ее изменения. Если на экране присутствует вертикальная зеленая линия где-то в левой части сетки, то до конца расчета еще далеко, так как в конце рассматриваются последовательности от 00011110 до 00001111 (рис. 46), т.е. вертикальная часть зеленой линии присутствует только на предпоследней и последней вертикали сетки.

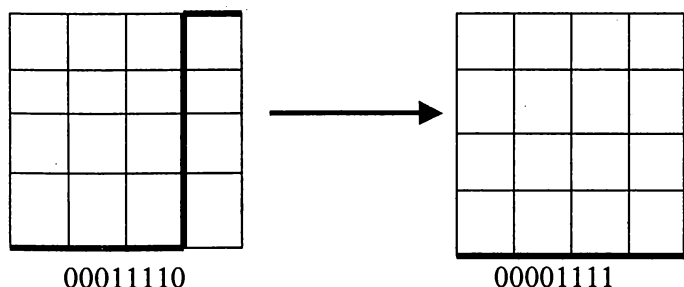


Рис. 46. Последняя серия вариантов

После завершения (или прерывания) расчета на экране оптимальный вариант (или последний из рассмотренных) показывается красным цветом и приводится величина соответствующих суммарных затрат.

Для иллюстрации резкого роста числа вариантов пути в зависимости от размеров сетки на экран выдается график зависимости числа путей от размеров сетки (принята квадратная сетка, чтобы число путей было функцией одной переменной). Для наглядности график представлен сплошной линией, хотя фактически состоит из отдельных точек, так как числа m и n целые. Кроме того, дается таблица соответствующих характерных точек графика.

Сделав вывод о нецелесообразности использования метода полного перебора, программа задает вопрос о применимости простой идеи, а именно: из каждого узла идти в соседний с наименьшими затратами, не думая о последствиях, т.е. смотреть только на один шаг вперед. Эта идея отвергается.

Далее разъясняется принцип оптимальности Р. Беллмана и соответствующий алгоритм решения данной задачи.

Программа позволяет пользователю прямым расчетом практически убедиться в преимуществах метода динамического программирования по сравнению с методом полного перебора вариантов.

После демонстрации работы метода динамического программирования программа предлагает несколько вопросов, позволяющих выявить степень понимания пользователем сущности метода и его особенностей. Большинство вопросов относится к условиям применимости метода динамического программирования.

В итоге программа сообщает количество верных и неверных ответов суммарно по всем вопросам с учетом всех попыток ответа.

Контрольные вопросы

1. В задаче поиска оптимального пути на двумерной сетке в качестве целевой функции приняли сумму шаговых затрат плюс количество изменений направления (поворотов), умноженное на заданное число, т.е. штраф за повороты. Можно ли использовать для поиска оптимального пути при такой целевой функции метод динамического программирования, и если да, то как?

2. Если в задаче поиска оптимального пути на двумерной сетке ограничено число поворотов, а варианты, сходящиеся в одном узле, считаются сравнимыми, то возможна ли ситуация «вырождения вариантов», т.е. остановка процесса поиска из-за исчерпания лимита поворотов?

3. Если для решения некоторой вариационной задачи поиска оптимальной плоской кривой (экстремали) используется модель поиска оптимального пути на двумерной сетке, то можно ли утверждать, что отклонение от оптимума (максимальная разность ординат), обусловленное искусственным введением дискретности, не превышает шага сетки?

4. Если для решения задачи поиска оптимальной кривой искусственно вводится дискретность и разбивается сначала крупная, а затем мелкая сетка, то как могут повлиять на выбор шага сетки свойства целевой функции, наличие и конкретный вид ограничений на искомую кривую?

5. Можно ли утверждать, что наличие локальных экстремумов не существенно при использовании метода динамического программирования?

6. В задаче поиска оптимального пути на двумерной сетке в качестве целевой функции приняли не сумму шаговых затрат, а логарифм (натуральный) этой суммы. Можно ли решить задачу поиска оптимального пути с такой целевой функцией по методу динамического программирования?

ЗАКЛЮЧЕНИЕ

Рассмотренные нами методы далеко не исчерпывают все многообразие методов оптимизации. Изложены лишь основные идеи и методы.

Предсказать заранее, какой метод окажется наиболее эффективным в конкретном случае, далеко не всегда возможно. В практической работе возникают различные задачи, но самое главное состоит в понимании смысла задачи, умении увидеть ее особенности и использовать их при создании математической модели, выборе метода оптимизации и разработке конкретного алгоритма и программы.

Одна и та же практическая задача может быть решена различными методами. Мы это видели на примере задачи о защите поверхности, для которой были построены математические модели различного вида, позволяющие использовать различные алгоритмы оптимизации. Общие рекомендации на этот счет включают следующее:

- ♦ максимально использовать возможности динамического программирования, если по смыслу задачи неизвестные могут принимать дискретные значения, так как эффективных методов решения задач целочисленного программирования большой размерности (особенно нелинейных) нет;
- ♦ если анализ показывает наличие локальных экстремумов, то динамическое программирование может оказаться наиболее эффективным, даже если дискретность придется вводить искусственно;

- ✦ при использовании динамического программирования в каждом конкретном случае особое внимание следует обращать на корректность применения принципа оптимальности и выбор способа формализации понятия «состояние системы»;
- ✦ при использовании методов нелинейного программирования выбор переменных и формализация целевой функции и ограничений (т.е. математическая модель задачи) также имеют особое значение. Здесь какие-либо рекомендации дать крайне сложно. Даже очевидное стремление максимально уменьшить размерность задачи не всегда оправдано;
- ✦ в рамках одной и той же математической модели могут использоваться различные методы нелинейного программирования. Современные компьютеры дают возможность применения алгоритмов, требующих большой памяти, но обладающих высокой скоростью сходимости. Это касается прежде всего методов оптимизации второго порядка. Однако эти методы, как правило, требуют хорошего начального приближения. В этой связи методы первого порядка, в частности метод сопряженных градиентов, могут оказаться предпочтительнее. Возможна и комбинация различных методов. Все зависит от конкретной задачи.

Опыт решения задач оптимизации большой размерности, особенно таких, в которых оптимальность решения трудно установить даже после того, как оно получено, убедительно свидетельствует о нецелесообразности использования различного рода «эвристических» алгоритмов. Такого рода алгоритмы иногда могут работать, но чаще всего пренебрежение математическим обоснованием алгоритма оптимизации не проходит бесследно. Математическая теория оптимизации бурно развивается и, прежде чем изобретать

очередной «эвристический» алгоритм, следует убедиться в том, что эта теория бессильна в данном конкретном случае, т.е. детально проанализировать особенности задачи, ее возможные математические модели и методы решения.

В анализе математической модели и выборе метода оптимизации большую помощь может оказать образное представление задачи и алгоритма ее решения в двумерном случае. На это и нацелены обучающие программы, которые планируется развивать и совершенствовать.

ПРИЛОЖЕНИЯ

Приложение 1

Ответы и решения к разделам 1.2–1.3

2. Нет. Это не выпуклое множество.

4. Если x^0 граничная, но не крайняя точка ОДР, то она является внутренней точкой некоторого отрезка $[x^1, x^2]$, *целиком содержащегося* в ОДР (иначе она крайняя). Этот отрезок определяет луч из x^0 . Но должны ли концы отрезка x^1 и x^2 принадлежать границе или один из них может быть внутренней точкой ОДР? Предположим, что, например x^1 , внутренняя точка и проведем луч из x^1 в x^0 . Тогда точка x^2 должна лежать вне отрезка $[x^1, x^0]$ на этом луче, т.е. за пределами ОДР. Значит, отрезок полностью принадлежит границе.

5. По определению локальный минимум — это точка, в некоторой окрестности которой значение функции по крайней мере не меньше, чем в данной точке. Мы доказали, что в задаче линейного программирования точка минимума x^0 — это вершина или любая точка некоторого линейного многообразия (ребра, грани и т.д. ОДР). В последнем случае во всех точках минимума целевая функция имеет одно и то же значение. Но это пока не позволяет утверждать, что в какой-либо вершине или на другом граничном многообразии нет точки x^1 , в которой значение целевой функции больше, чем в найденной нами точке (или точках) минимума x^0 , но в ее некоторой окрестности значение целевой функции еще больше, чем в x^1 .

Предположим, что \mathbf{x}^1 — точка локального минимума и рассмотрим отрезок $[\mathbf{x}^1, \mathbf{x}^0]$. В силу выпуклости ОДР он целиком принадлежит ОДР. При движении по лучу $\mathbf{x}^1, \mathbf{x}^0$ из точки \mathbf{x}^1 целевая функция в зависимости от λ получит приращение $\lambda(\mathbf{c}, \mathbf{p})$, где вектор $\mathbf{p} = \mathbf{x}^0 - \mathbf{x}^1$ (см. рис.6). При этом $(\mathbf{c}, \mathbf{p}) \geq 0$, так как в \mathbf{x}^1 локальный минимум. Величина (\mathbf{c}, \mathbf{p}) не меняется при движении по лучу и с увеличением λ приращение целевой функции остается неотрицательным и может только возрасти. Но при шаге, равном расстоянию от \mathbf{x}^1 до \mathbf{x}^0 (при $\lambda=1$), мы достигнем точки \mathbf{x}^0 , в которой значение целевой функции должно быть не больше, а меньше, чем в \mathbf{x}^0 . Полученное противоречие доказывает отсутствие локальных минимумов (аналогично и максимумов) в задаче линейного программирования.

6. Нет, так как это невыпуклое множество, а ОДР в задаче линейного программирования выпукла. Можно выбрать две точки сектора так, что соединяющий их отрезок содержит точки, не принадлежащие сектору, что и означает отсутствие выпуклости

7. Доказательство от противного. Предположим, что точка А как некоторая допустимая вершина ОДР не является крайней точкой, т.е. существует отрезок $[MN]$, целиком принадлежащий ОДР и содержащий точку А как внутреннюю. Берем любое ограничение, активное в точке А, т.е. выполненное как равенство.

Отрезок $[MN]$ не может принадлежать всем граням, пересекающимся в точке А, так как их пересечение имеет нулевую размерность (точка А — вершина). Берем ту из пересекающихся в точке А граней, которой он не принадлежит, и рассматриваем соответствующее ограничение. Этому ограничению точка А удовлетворяет как равенству, но, поскольку точка А есть внутренняя точка отрезка $[MN]$, то или в точке М, или в точке N ограничение не вы-

полнено. Пришли к противоречию. Следовательно, такого отрезка (при линейно независимых ограничениях) не существует.

Пусть теперь точка A крайняя, но не вершина, т.е. в ней активны меньше чем n ограничений. Следовательно, размерность граничного линейного многообразия не равна нулю и в нем можно построить допустимый отрезок с центром в точке A . Следовательно, точка A не является крайней. Пришли к противоречию, что доказывает эквивалентность понятий крайняя точка многогранника и *допустимая* вершина.

8. Да. Например, в двумерном случае каждая вершина — это точка пересечения двух прямых, соответствующих двум неравенствам. Не все такие точки удовлетворяют остальным неравенствам, т.е. являются допустимыми. Если ОДР треугольник, то недопустимых вершин может быть сколь угодно много за счет пересечения прямых, не влияющих на формирование ОДР. Эти «ложные» вершины можно видеть на экране при работе программы DANTZIG_1.exe

9. Берем две точки, принадлежащие *пересечению выпуклых* множеств, и соединяющий их отрезок. Концы отрезка принадлежат каждому из множеств по определению понятия «пересечение множеств». Значит, и любая внутренняя точка отрезка принадлежит *каждому из множеств* в силу его выпуклости. Следовательно, она принадлежит и пересечению, что и требовалось доказать.

10. Нет. Множество граничных точек ОДР в задаче линейного программирования не является выпуклым. Чтобы убедиться в этом, возьмем две точки ОДР, принадлежащие *разным* граням, и соединим их отрезком. Внутренние точки отрезка принадлежат ОДР, так как она выпукла, но они не обязательно являются граничными

(например, отрезок, соединяющий две точки на сторонах треугольника).

11. Нет, неверно. При неограниченной ОДР задача линейного программирования может иметь (а может и не иметь) решение. Все зависит от целевой функции. См. примеры, демонстрируемые программой DANTZIG_2.exe.

12. Представим $\mathbf{A}\mathbf{y}$ как линейную комбинацию столбцов \mathbf{a}^i матрицы \mathbf{A} с коэффициентами y_i , равными соответствующим компонентам вектора \mathbf{y} (формула 1.1), и используем свойства скалярного произведения.

$$\begin{aligned}(\mathbf{x}, \mathbf{A}\mathbf{y}) &= (\mathbf{x}, y_1\mathbf{a}^1 + y_2\mathbf{a}^2 + \dots + y_n\mathbf{a}^n) = \\ &= y_1(\mathbf{a}^1, \mathbf{x}) + y_2(\mathbf{a}^2, \mathbf{x}) + \dots + y_n(\mathbf{a}^n, \mathbf{x})\end{aligned}$$

Каждый столбец \mathbf{a}^i матрицы \mathbf{A} рассматриваем как i -ую строку транспонированной матрицы \mathbf{A}^T . Скалярное произведение $(\mathbf{a}^i, \mathbf{x})$ это i -ая компонента произведения $\mathbf{A}^T\mathbf{x}$, т.е. $(\mathbf{A}^T\mathbf{x})_i$. Следовательно, полученное выражение есть сумма произведений соответственных компонент двух векторов \mathbf{y} и $\mathbf{A}^T\mathbf{x}$, т.е.

$$(\mathbf{y}, \mathbf{A}^T\mathbf{x}) = (\mathbf{A}^T\mathbf{x}, \mathbf{y}).$$

Получили $(\mathbf{x}, \mathbf{A}\mathbf{y}) = (\mathbf{A}^T\mathbf{x}, \mathbf{y})$, что и требовалось доказать.

Ответы и решения к разделам 1.4–1.8

1, 2. **Рекомендация:** Использовать обучающую программу DANTZIG_2.exe.

3. Для первого механизма можно выбрать любую работу (n возможностей), для второго — любую, кроме выбранной для первого ($n-1$ возможность) и т.д. Для последнего механизма останется только одна возможность. Получаем $n!$.

4. Если в нижней строке симплексной таблицы стоит отрицательное число и в столбце над ним нет отрицательных чисел, то ОДР и целевая функция не ограничены и другие столбцы симплексной таблицы рассматривать не нужно.

5. Нуль в нижней строке симплексной таблицы означает, что увеличение соответствующей свободной переменной при неизменных остальных не меняет целевой функции и поэтому такой столбец не может быть разрешающим.

6. Неоднозначность при выборе разрешающего столбца принципиального значения не имеет, если при этом ни в одном из них не возникает неоднозначности выбора опорного элемента. Если же такая неоднозначность при выборе опорного элемента возникает, то это означает переход в вырожденную вершину, что нежелательно, и лучше выбрать другой разрешающий столбец.

7. Да. Как следует из правил преобразования симплексных таблиц, изменение целевой функции равно абсолютной величине отношения свободного члена к опорному элементу, умноженной на соответствующее число в нижней строке разрешающего столбца, так что важно не только это число в нижней строке, но и упомянутое отношение (шаг).

8. Вырожденная вершина появится, если минимум абсолютной величины отношения свободного члена к соответствующему отрицательному числу в разрешающем столбце достигается в нескольких строках. В этом случае появится нуль в столбце свободных членов, что и является признаком вырожденности вершины.

Даже в случае, если такая вершина одна, то возможна ситуация, при которой не удастся ее миновать. При выборе разрешающего столбца в нижней строке симплексной таблицы может быть только один отрицательный элемент и других столбцов, которые можно рассматривать как разрешающие, чтобы избежать попадания в вырожденную вершину, просто нет. Кстати, это не означает, что следующая вершина (пусть и вырожденная) будет точкой минимума, так как такие отрицательные числа могут появиться.

Поэтому часто рекомендуемый способ избежания заикливания путем выбора нужного разрешающего столбца не гарантирует успех. Однако возможны случаи, в которых нет выбора. Есть только одно отрицательное число в нижней строке и выбор соответствующего разрешающего столбца ведет в вырожденную вершину, которая не является решением задачи. Такой случай показан на рис. 46.

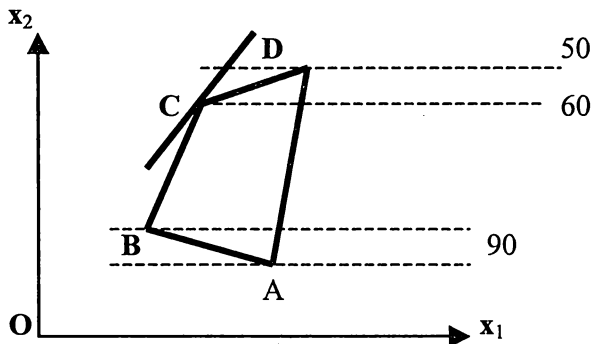


Рис. 46. Пример неизбежности попадания в вырожденную вершину

Допустимая область ABCD. Линии уровня показаны пунктиром. Минимум в точке D. Вершина C вырожденная. Если мы находимся в невырожденной вершине B, то из нее симплекс-метод может вести только в точку C, но это не минимум, и далее придется «выбираться» из вырожденной вершины C.

9. Симплекс-алгоритм «не заметит» попадание в вырожденную вершину, если в разрешающем столбце отрицательное число стоит в строке, которой соответствует ненулевой свободный член. В этом случае шаг не равен нулю. Если же нулевой свободный член и отрицательное число в разрешающем столбце стоят в одной строке и нет возможности выбрать другой разрешающий столбец, то закликивание неизбежно и нужно предпринимать дополнительные меры.

10. Нет, такое утверждение неверно, так как даже в двумерном случае возможно такое взаимное расположение ОДР и линий уровня целевой функции в задаче целочисленного линейного программирования, при котором отклонение по одной из координат точки минимума, полученной без учета требований целочисленности, от решения задачи, т.е. точки минимума, удовлетворяющей условию целочисленности, составляет 2 и более (рис. 47). Решением задачи является точка A с целочисленными координатами (0, 2), а решение, полученное без учета требований целочисленности, — это точка B(3.8, 3.7).

В многомерном случае также возможны варианты, в которых и сама точка минимума, полученная без учета требований целочисленности, и допустимая точка, полученная после округления значений ее координат, отстоят от решения задачи на несколько единиц по одной или нескольким координатам. Однако по величине целевой функции эти отклонения могут быть и невелики.

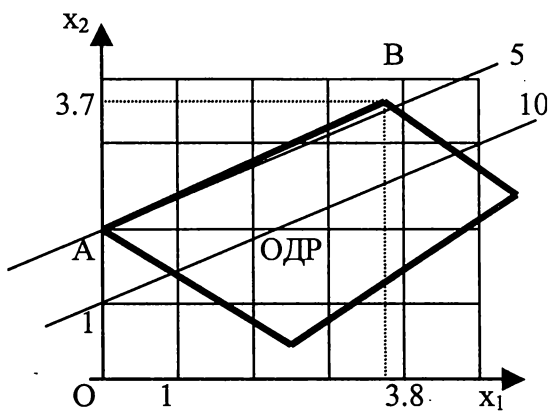


Рис. 47. Влияние требований целочисленности решений

Ответы и решения к разделам 2.1–2.3

2. Да. Линейная функция выпукла (не строго).

3, 4. Собственные числа $5/2$ и $-1/2$. Квадратичная форма знаконеопределенна. В точке $(0,0)$ не минимум и не максимум. Это седловая точка. $x_1^2 + 3x_1x_2 + x_2^2 = 5/4(x_1 + x_2)^2 - 1/4(x_1 - x_2)^2$. Линии уровня гиперболы с асимптотами $x_2 = (-3 + \sqrt{5})/2x_1$ и $x_2 = (-3 - \sqrt{5})/2x_1$.

5. Антиградиент в точке $(1, 1)$ равен $(-5, -5)^T$. Его скалярные произведения на первые два вектора положительны. Они задают направления спуска.

6. Это направление может быть ортогонально градиенту, и смена знака ничего не изменит.

7. Да, если по каждому направлению рассматривать шаги только в одну сторону, и нет, если по каждому направлению возможны шаги в обе стороны.

9, 10. Минимальное значение соответствует попаданию в точку минимума за один шаг. Такое направление всегда существует. В некоторых точках оно совпадает с антиградиентом. Действительно, градиент квадратичной формы:

$$\frac{1}{2}(\mathbf{Q}\mathbf{x}, \mathbf{x}) + (\mathbf{p}, \mathbf{x})$$

в произвольной точке \mathbf{x} вычисляется как $\mathbf{g} = \mathbf{Q}\mathbf{x} + \mathbf{p}$. Если \mathbf{g} есть собственный вектор матрицы \mathbf{Q} с собственным числом β , то $\mathbf{Q}\mathbf{g} = \beta\mathbf{g}$. На луче $\mathbf{x} - \lambda\mathbf{g}$ при произвольном λ вычислим новый градиент. Получим вектор:

$$\mathbf{Q}(\mathbf{x} - \lambda\mathbf{g}) + \mathbf{p} = \mathbf{Q}\mathbf{x} - \lambda\beta\mathbf{g} + \mathbf{p} = (1 - \lambda\beta)\mathbf{g}.$$

При $\lambda = 1/\beta$ получим точку, в которой градиент равен нулю, т.е. точку минимума.

Однако существует ли точка, в которой градиент есть собственный вектор матрицы \mathbf{Q} ? Найдем такие точки. Пусть \mathbf{f} — некоторый собственный вектор. Найдем точку \mathbf{x} , в которой градиент ему равен. $\mathbf{Q}\mathbf{x} + \mathbf{p} = \mathbf{f}$ и далее $\mathbf{x} = \mathbf{Q}^{-1}(\mathbf{f} - \mathbf{p})$. При произвольном λ в точке $\mathbf{x} = \mathbf{Q}^{-1}(\lambda\mathbf{f} - \mathbf{p})$ градиент тоже является собственным вектором. Кстати, при $\mathbf{p} = \mathbf{0}$ можно просто взять $\mathbf{x} = \mathbf{f}$.

12. Если вычисляются *только* значения функции на луче, то деление интервала неопределенности пополам не обязательно сокращает его длину вдвое, даже при наличии третьей точки внутри интервала. Интервал неопределенности сокращается вдвое при поиске нуля функции, в качестве которой может выступать скалярное произведение градиента на вектор спуска. Но при этом необходимо вычислять не значения функции, а градиент.

14. Да. Если \mathbf{p}_i и \mathbf{p}_j — два собственных вектора матрицы \mathbf{Q} , то они \mathbf{Q} -ортогональны, так как $(\mathbf{p}_i, \mathbf{Q}\mathbf{p}_j) = \lambda(\mathbf{p}_i, \mathbf{p}_j) = 0$, если λ является собственным числом для \mathbf{p}_j , но не для \mathbf{p}_i . Если же λ — кратное собственное число, то соответствующие ему собственные векторы \mathbf{p}_i и \mathbf{p}_j неортогональны.

Ответы и решения к разделам 2.4–2.5

1. Пусть $F(x)$ — строго выпуклая функция на некотором множестве M , x^* — точка ее глобального минимума на M . Покажем сначала, что во всех остальных точках x из M $F(x) > F(x^*)$. Предположим, что это не так и существует точка x^1 из M , в которой $F(x^1) = F(x^*)$. Предполагать $F(x^1) < F(x^*)$ бессмысленно, так как тогда x^* не была бы точкой глобального минимума. Далее, для любого $0 < \lambda \leq 1$ точка $x = \lambda x^1 + (1 - \lambda) x^*$ принадлежит множеству M , так как оно выпукло. В силу строгой выпуклости $F(x)$ имеем

$$\begin{aligned} F(x) &= F(\lambda x^1 + (1 - \lambda) x^*) \leq \lambda F(x^1) + (1 - \lambda) F(x^*) = \\ &= \lambda (F(x^1) - F(x^*)) + F(x^*). \end{aligned}$$

По предположению $F(x^1) - F(x^*) = 0$, и получается $F(x) < F(x^*)$, следовательно, x^* не является точкой глобального минимума. Полученное противоречие доказывает единственность решения задачи на минимум строго выпуклой функции. Однако отсюда пока не следует, что не может существовать точка локального минимума x^0 , т.е. такая точка, в которой $F(x^0) > F(x^*)$, но во всех точках x из некоторой окрестности x^0 $F(x) \geq F(x^0)$. Возьмем точку x^1 из такой окрестности x^0 , которая одновременно принадлежит отрезку $[x^0, x^*]$ (рис. 48). Займемся значением $F(x^1)$. С одной стороны, должно быть $F(x^1) \geq F(x^0)$, так как x^1 принадлежит окрестности точки локального минимума x^0 . С другой стороны, в силу выпуклости $F(x)$ (даже не строгой), это значение должно быть не больше, чем при линей-

ной интерполяции $F(x)$ на отрезке $[x^0, x^*]$, т.е. $F(x^1) \leq F_n(x^1)$. Но $F_n(x^1) < F(x^0)$, так как $F(x^*) < F(x^0)$. Значит, и $F(x^1) < F(x^0)$, и поэтому x^0 не может быть точкой локального минимума. Это противоречие доказывает, что у нестрого выпуклой функции не может быть локальных минимумов с различными значениями. Однако нестрого выпуклая функция может иметь много точек минимума с равными значениями в них. Пример тому — линейная функция на выпуклом многограннике.

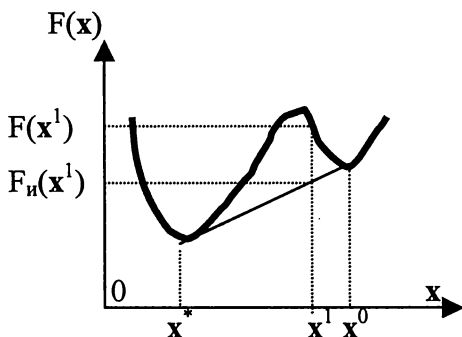


Рис. 48. Отсутствие локальных минимумов выпуклой функции

2. Нет, так как алгоритм может остановиться на границе допустимой области (рис. 49). Из точки M , изменяя *только* x_1 или *только* x_2 , нельзя уменьшить целевую функцию, так как соответствующие направления недопустимы. В точке M «тупик», а минимум в точке N .

3. Нет, так как при замене неравенств равенствами появляются дополнительные переменные, которые *должны быть неотрицательны*. Значит, неравенства остаются.

4. Да. Будет получено правильное решение, если нет локальных минимумов. В алгоритме метода проекции гради-

ента активный набор преобразуется, и в подобных случаях поочередно все ограничения будут выведены из активного набора.

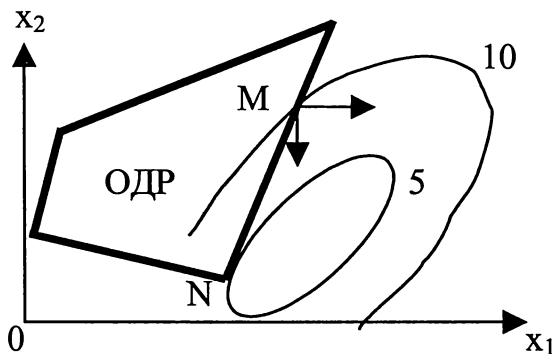


Рис. 49. «Тупик» при покоординатном спуске

5. Да. Антиградиент надо проектировать на одно и то же подпространство, определяемое ограничениями. Алгоритм проще, чем в случае неравенств, так как не надо преобразовывать активный набор. Ограничения-равенства всегда активны, а других ограничений нет.

6, 7. Да, можно. Ограничения-равенства присутствуют в активном наборе постоянно, а с неравенствами работать как обычно. Приводить систему к основной форме нет необходимости.

8. Да, можно. Вместо антиградиента надо использовать его проекцию на одно и то же подпространство, определяемое ограничениями.

9–12. Базисные векторы:

$$\mathbf{c}^1 = \begin{vmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{vmatrix} \text{ и } \mathbf{c}^2 = \begin{vmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{vmatrix}$$

Размерность нуль-пространства равна двум. Проекция любого вектора \mathbf{f} на это подпространство имеет три первых компоненты, равные некоторому числу a , и две оставшиеся, равные другому числу b . Нормаль \mathbf{n} имеет вид

$$\mathbf{n} = \begin{vmatrix} f_1 - a \\ f_2 - a \\ f_3 - a \\ f_4 - b \\ f_5 - b \end{vmatrix}$$

Условие ортогональности нормали базисным векторам: $(\mathbf{n}, \mathbf{c}^1) = 0$ и $(\mathbf{n}, \mathbf{c}^2) = 0$ дает $a = (f_1 + f_2 + f_3)/3$ и $b = (f_4 + f_5)/2$. Для заданного антиградиента получаем $a = -1$ и $b = 5$.

В итоге проекция \mathbf{p} имеет компоненты $(-1, -1, -1, 5, 5)$, а нормаль \mathbf{n} $(2, 3, -5, -1, 1)$. Матрица \mathbf{A} содержит три строки:

$$\begin{vmatrix} -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 \end{vmatrix}$$

Для поиска коэффициентов разложения \mathbf{u} нормали \mathbf{n} по нормальям к граничным гиперплоскостям составляем систему $\mathbf{A}^T \mathbf{u} = \mathbf{n}$.

$$\begin{cases} -u_1 = 2 \\ u_1 - u_2 = 3 \\ u_2 = -5 \\ -u_3 = -1 \\ u_3 = 1 \end{cases}$$

Из этой системы получаем $u_1 = -2$, $u_2 = -5$, $u_3 = 1$, что означает возможность исключения из активного набора первого или второго ограничения. Если мы исключим первое ограничение, то новая проекция будет $\mathbf{p}^1 = (1, -2, -2, 5, 5)^T$. Это ограничение было активно, т.е. выполнялось как равенство, следовательно, в исходной точке \mathbf{x} было $x_2 - x_1 = b_1$. При движении по проекции антиградиента \mathbf{p}^1 получим в левой части этого неравенства $x_2 + \lambda(-2) - (x_1 + \lambda) = x_2 - x_1 - 3\lambda$, что будет меньше b_1 при $\lambda > 0$. Если исключить второе ограничение, то получим проекцию $\mathbf{p}^2 = (3/2, 3/2, -6, 5, 5)^T$. При движении по этому направлению в левой части второго неравенства добавится $-15/2\lambda$, т.е. ограничение будет выполнено. Если исключить одновременно и первое и второе ограничение, то получим проекцию: $\mathbf{p}^3 = (1, 2, -6, 5, 5)^T$.

При движении в этом направлении вместо $x_2 - x_1 = b_1$ будет $x_2 + 2\lambda - (x_1 + \lambda) = x_2 - x_1 + \lambda$, что больше b_1 , т.е. это направление нарушает первое ограничение и потому недопустимо. Это еще раз подтверждает, что ограничения можно исключать только по одному. Приведенный антиградиент \mathbf{p}^4 получаем как $\mathbf{C}\mathbf{C}^T\mathbf{f}$, где столбцы матрицы \mathbf{C} это базисные векторы \mathbf{c}^1 и \mathbf{c}^2 . \mathbf{p}^4 имеет компоненты $(-3, -3, -3, 10, 10)$. Легко убедиться, что этот вектор принадлежит нуль-пространству матрицы \mathbf{A} . Скалярное произведение $(\mathbf{p}^4, \mathbf{f}) > 0$, значит, это направление спуска, так как \mathbf{f} — это антиградиент по условию задачи. Посмот-

рим, можно ли исключить первое ограничение из активного набора. Вектор \mathbf{b}^1 , который нарушает это и только это ограничение, имеет компоненты $(0, 1, 1, 0, 0)$. В этом легко убедиться, подставляя его компоненты в систему $\mathbf{A}\mathbf{x} \leq \mathbf{0}$. Его скалярное произведение на вектор первой нормали $\mathbf{a}^1 = (-1, 1, 0, 0, 0)^T$ и на градиент $-\mathbf{f}$ имеют одинаковые знаки. Действительно, $(\mathbf{b}^1, \mathbf{a}^1) = 1$ и $(\mathbf{b}^1, -\mathbf{f}) = 4$. Значит, первое ограничение можно исключить и добавить \mathbf{b}^1 в матрицу \mathbf{C} . Новый приведенный градиент: $\mathbf{p}^5 = (-3, -7, -7, 10, 10)^T$.

При движении по этому направлению второе и третье ограничение остаются активными, а первое выходит из активного набора. Поскольку $(\mathbf{p}^5, \mathbf{f}) > 0$, это направление является направлением спуска. Вектор $\mathbf{b}^2 = (0, 0, 1, 0, 0)^T$ нарушает второе и только второе ограничение.

$(\mathbf{b}^2, \mathbf{a}^2) = 1$ и $(\mathbf{b}^2, -\mathbf{f}) = 6$. Значит, второе ограничение тоже можно исключить из активного набора. Если исключить, только второе ограничение, то в матрицу \mathbf{C} третьим столбцом надо добавить \mathbf{b}^2 . Получим новый приведенный антиградиент $\mathbf{p}^6 = (-3, -3, -9, 10, 10)^T$. Исключим теперь одновременно и первое и второе ограничения. Это значит, что в матрицу \mathbf{C} добавим два столбца \mathbf{b}^1 и \mathbf{b}^2 . По той же формуле $\mathbf{p} = \mathbf{C}\mathbf{C}^T\mathbf{f}$ с новой матрицей \mathbf{C} получим новый вектор $\mathbf{p}^7 = (-3, -7, -13, 10, 10)^T$, который выводит из активного набора и первое и второе ограничение и сохраняет в нем третье ограничение. Этот вектор тоже является направлением спуска. Наконец, вектор, который нарушает третье и только третье ограничение $\mathbf{b}^3 = (0, 0, 0, 0, 1)^T$.

Далее, $\mathbf{a}^3 = (0, 0, 0, -1, 1)^T$ и $(\mathbf{b}^3, \mathbf{a}^3) = 1$, но $(\mathbf{b}^3, -\mathbf{f}) = -6 < 0$.

Следовательно, третье ограничение исключить нельзя.

Данный пример характерен тем, что мы нашли сразу несколько улучшающих направлений по заданному набору активных ограничений и антиградиенту без решения систем линейных уравнений (обращения матриц). Очевидно, что важное значение имеет не размер, а структура

матрицы активных ограничений. Одно из найденных направлений выводит сразу два ограничения из активного набора. Какое из этих трех улучшающих направлений лучше? Этого нельзя определить, не зная конкретной целевой функции. Заметим, что при поиске улучшающих направлений два первых и последнее ограничения можно было бы рассматривать независимо, так как они относятся к различным компонентам неизвестного вектора. Такая ситуация типична для систем ограничений с блочными матрицами.

13. Для системы ограничений

$$\begin{cases} x_3 - 2x_2 + x_1 = b_1, \\ x_5 - 2x_4 + x_3 = b_2. \end{cases}$$

с матрицей

$$A = \begin{pmatrix} 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 \end{pmatrix}$$

также легко найти базисные векторы в нуль-пространстве, если переписать ее в виде

$$\begin{cases} (x_3 - x_2) - (x_2 - x_1) = b_1, \\ (x_5 - x_4) - (x_4 - x_3) = b_2. \end{cases}$$

Вектор \mathbf{c}^1 , у которого все компоненты равны некоторому числу (например, единице), по-прежнему является базисным, так как $A\mathbf{c}^1 = \mathbf{0}$ для $\mathbf{c}^1 = (1, 1, 1, 1, 1)^T$. Другой базисный вектор \mathbf{c}^2 — это вектор, у которого равны между собой разности смежных компонент, например, вектор

$(1, 2, 3, 4, 5)^T$. Но вместо него мы для простоты вычислений возьмем $\mathbf{c}^2 = (0, 0, 0, 1, 2)^T$. Поскольку размерность нуль-пространства равна трем, нужен третий базисный вектор $\mathbf{c}^3 = (2, 1, 0, 0, 0)^T$. Искомую проекцию \mathbf{p} заданного вектора \mathbf{f} в этом базисе представим как:

$$\mathbf{p} = p_1 \mathbf{c}^1 + p_2 \mathbf{c}^2 + p_3 \mathbf{c}^3.$$

Вектор $\mathbf{f} - \mathbf{p}$ (нормаль) должен быть ортогонален каждому базисному вектору. Отсюда получаем систему

$$\begin{cases} (\mathbf{f} - p_1 \mathbf{c}^1 - p_2 \mathbf{c}^2 - p_3 \mathbf{c}^3, \mathbf{c}^1) = 0 \\ (\mathbf{f} - p_1 \mathbf{c}^1 - p_2 \mathbf{c}^2 - p_3 \mathbf{c}^3, \mathbf{c}^2) = 0 \\ (\mathbf{f} - p_1 \mathbf{c}^1 - p_2 \mathbf{c}^2 - p_3 \mathbf{c}^3, \mathbf{c}^3) = 0 \end{cases}$$

По условию задачи проектируемый вектор $\mathbf{f} = (1, 2, -6, 4, 6)^T$. Система сводится к

$$\begin{cases} 5p_1 + 3p_2 + 3p_3 = 7; \\ 3p_1 + 5p_2 = 20; \\ 3p_1 + 5p_3 = 4. \end{cases}$$

Отсюда находим координаты искомой проекции в базисе $\mathbf{c}^1, \mathbf{c}^2, \mathbf{c}^3$ $p_1 = -37/7$; $p_2 = 251$; $p_3 = 129/35$ и, наконец, в исходном базисе находим искомую проекцию:

$$\mathbf{p} = (73/35, -8/5, -37/5, 66/35, 317/35).$$

Размерность ортогонального дополнения равна двум. Вектор, нарушающий только первое ограничение — $\mathbf{b}^1 = (1, 0, 0, 0, 0)^T$, а вектор, нарушающий только второе ограничение — $\mathbf{b}^2 = (0, 0, 0, 0, 1)^T$.

Нормали $\mathbf{a}^1 = (1, -2, 1, 0, 0)^T$ и $\mathbf{a}^2 = (0, 0, 1, -2, 1)^T$. Градиент $-\mathbf{f} = (-1, -2, 6, -4, -6)^T$. $(\mathbf{b}^1, \mathbf{a}^1) = 1 > 0$. Но $(\mathbf{b}^1, -\mathbf{f}) = -1 < 0$. Значит, первое ограничение нельзя исключить из активного набора. Аналогично $(\mathbf{b}^2, \mathbf{a}^2) = 1 > 0$.

Но $(\mathbf{b}^2, -\mathbf{f}) = -6 < 0$. Значит, и второе ограничение нельзя исключить из активного набора. Характерно, что если вместо \mathbf{b}^1 взять $\mathbf{b}^3 = (0, 1, 0, 0, 0)^T$, то этот вектор тоже нарушает только первое ограничение и при этом $(\mathbf{b}^3, \mathbf{a}^1) = -2 < 0$ и $(\mathbf{b}^3, -\mathbf{f}) = -2 < 0$. Получается, что первое ограничение все-таки можно исключить из активного набора. Так можно или нельзя? Здесь нет никакого противоречия. Можно исключить, если в базисную матрицу к векторам $\mathbf{c}^1, \mathbf{c}^2, \mathbf{c}^3$ добавить вектор \mathbf{b}^3 , и нельзя, если добавлять \mathbf{b}^1 .

Напомним, что направление спуска (приведенный антиградиент) вычисляется как $\mathbf{p} = \mathbf{C}\mathbf{C}^T\mathbf{f}$ и, очевидно, зависит от того, какой вектор мы добавляем к базису, т.е. какой столбец мы добавляем в матрицу \mathbf{C} .

14. Свободные члены системы активных ограничений на построение проекции градиента и приведенного антиградиента не влияют, так как направление определяется в нуль-пространстве матрицы системы активных ограничений. Свободные члены учитываются при построении активного набора.

$$15. \mathbf{P} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \text{ для любого вектора } \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \mathbf{P}\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ 0 \end{bmatrix}.$$

Вектор $\mathbf{p} = \mathbf{P}\mathbf{x}$ принадлежит подпространству, на которое проектируем вектор \mathbf{x} , так как $\mathbf{p}_3 = 0$ и $\mathbf{x} - \mathbf{p} = (0 \ 0 \ x_3)^T$ (нормаль) ортогонален любому вектору в этом подпространстве. Следовательно, \mathbf{P} — матрица проектирования на подпространство $x_3 = 0$.

16. Если \mathbf{M} — подпространство, на которое проектирует матрица \mathbf{P} , а \mathbf{f} любой вектор из \mathbf{M} , то $\mathbf{Pf} = \mathbf{f}$. Следовательно, \mathbf{f} — собственный вектор с собственным числом 1. Если \mathbf{g} — любой вектор из ортогонального дополнения к \mathbf{M} , то $\mathbf{Pg} = \mathbf{0}$ или $\mathbf{Pg} = \mathbf{0g}$. Отсюда следует, что число 0 является собственным числом матрицы \mathbf{P} . Поскольку в \mathbf{E}_n всегда можно выбрать базис так, что каждый базисный вектор принадлежит \mathbf{M} или его ортогональному дополнению, то разложив произвольный вектор по такому базису, придем к выводу, что кратность собственного числа 1 равна размерности подпространства \mathbf{M} , а кратность собственного числа 0 равна размерности ортогонального дополнения к \mathbf{M} . Других собственных чисел у матрицы проектирования нет.

17. Нет. От противного. Берем вектор \mathbf{f} из подпространства, на которое проектирует матрица $2\mathbf{P}$. Тогда $2\mathbf{Pf} = \mathbf{f}$. Или $\mathbf{Pf} = 1/2\mathbf{f}$. Получается, что число $1/2$ является собственным для матрицы \mathbf{P} . Но матрица \mathbf{P} , как матрица проектирования, не может иметь такого собственного числа. Полученное противоречие доказывает, что $k\mathbf{P}$, где $k \neq 0$ не может быть матрицей проектирования, если \mathbf{P} — матрица проектирования.

Ответы и решения к разделу 3

1. Если в задаче поиска оптимального пути на двумерной сетке целевая функция представляет собой сумму шаговых затрат плюс количество изменений направления (поворотов), умноженное на заданное число, т.е. штраф за повороты, то нельзя сравнивать варианты, сходящиеся в узле сетки, и выбирать лучший из них по этой целевой функции. У этих вариантов множества возможных продолжений совпадают, но «предыстория» все-таки существенна, так как оценка продолжения каждого варианта зависит от того, делается поворот или нет. Поэтому продолжение отброшенного худшего варианта (без поворота) может оказаться предпочтительнее продолжения лучшего варианта (с поворотом). Для корректного применения динамического программирования следует сравнивать только те варианты, у которых последний шаг общий (сравнение не «в точку», а «в отрезок»).

2. Да, такая ситуация вырождения вариантов и прерывания процесса поиска оптимального решения возможна. Простой пример, иллюстрирующий влияние ограничения на максимальное число поворотов, приведен на рис. 50. В этом примере максимальное число поворотов равно единице. Затраты на переходы заданы и показаны на рис. 50. Оптимальный путь ACB. Допустимый (но не оптимальный) путь ADB. Все прочие пути *недопустимы*, так как содержат более чем один поворот.

При сравнении вариантов ACE и AME, сходящихся в точке E, отбрасывается вариант ACE, а оставшийся вариант AME не может быть продолжен из-за исчерпания «ресурса» поворотов. Аналогичная ситуация имеет место в точке N. Отбрасывается вариант ADN как худший, а оставшийся ва-

риант AKN не имеет допустимых продолжений. Выход состоит в переходе к другой формализации понятия «состояние системы» и сравнении «в отрезок, а не в точку». Варианты ACE и AME станут не сравнимыми и в итоге задача будет решена.

	C	10	E	1	B
1			1		2
K		1		1	N
1			1		10
A		1	M	1	D

Рис. 50. Влияние ограничения на число поворотов

3, 4, 5. Если дискретность вводится искусственно, то вопрос обоснования величины дискрета (шага сетки) достаточно сложен. Простое решение: разбить сетку с крупным шагом, а потом в полосе вокруг полученного решения шириной ± 1 или 2 шага разбить сетку с мелким шагом не гарантирует нахождение оптимального решения.

На шаг сетки и на результат могут повлиять ограничения. Например, если ограничена разность двух смежных ординат искомой кривой (рис. 42), то очевидно, что шаг сетки по вертикали должен быть меньше ограничивающей величины.

При наличии локальных экстремумов метод динамического программирования при крупном шаге сетки из-за до-

пускаемой при этом погрешности может дать решение в окрестности не глобального, а локального минимума и дальнейшее исследование этой окрестности может быть бесперспективно.

Это обстоятельство поясняет рис. 51, на котором изображены линии уровня целевой функции, имеющей две точки минимума. Сетка на рис. 51 условно отражает влияние дискретности поиска. Ее не следует путать с сеткой, используемой для поиска экстремальной кривой по методу динамического программирования (рис. 42). При этом поиске экстремальная кривая заменяется ломаной линией с фиксированными абсциссами точек перелома и ординатами, изменяющимися дискретно с заданным шагом сетки.

Каждый узел сетки на рис. 51 соответствует одной из искомым ломаных, которая может быть получена по методу динамического программирования. Координаты узла сетки на рис. 51 соответствуют ординатам двух узлов искомой ломаной.

Точки, соответствующие различным ломаным линиям, на рис. 51 обязаны располагаться в регулярном порядке. Таким образом возникает сетка, показанная на рис. 51, который иллюстрирует влияние дискретности появления этих точек.

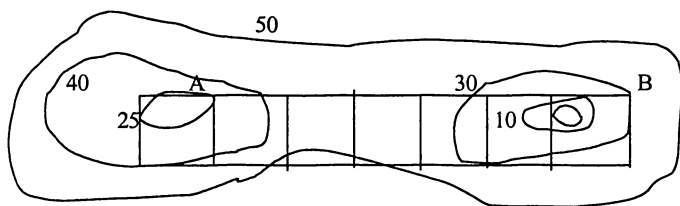


Рис. 51. Влияние дискретности поиска

Именно из-за искусственно введенной дискретности можно как в ситуации, изображенной на рис. 51, получить решение (точка А) в окрестности локального, а не глобального минимума. Исследовать далее окрестность точки А, разбивая сетку с мелким шагом, нет смысла, если минимум где-то в окрестности точки В.

В этой связи целесообразно исследовать не только оптимальное решение, полученное при некотором шаге сетки, но и субоптимальные решения, которые могут (геометрически) существенно отличаться от оптимального.

Что касается влияния ограничений, накладываемых на оптимальную кривую, которую мы пытаемся найти с помощью динамического программирования, то здесь можно усмотреть «аналогию наоборот» с ситуацией решения задачи целочисленного программирования обычными методами с последующим округлением до целых чисел. Естественно, что решение, полученное на дискретном множестве, не обязано быть близким (по геометрии) к решению, полученному на непрерывном множестве (см. рис. 47).

6. Если целевая функция представляет собой логарифм суммы шаговых затрат, то нельзя вычислить ее значение на отдельных шагах, так как логарифм суммы не равен сумме логарифмов. Казалось бы, из-за этого нельзя применить и метод динамического программирования.

Однако логарифм — это монотонная функция своего аргумента, и это дает возможность рассматривать в качестве целевой функции ее аргумент, т.е. сумму шаговых затрат, и решать задачу по методу динамического программирования. Если исходная целевая функция монотонно возрастающая (как, например, логарифм по основанию больше единицы), то ее минимум совпадает с минимумом аргумента (в нашем случае с минимумом суммы шаговых затрат). Если же исходная целевая функция монотонно убывающая

(как, например, логарифм по основанию меньше единицы), то ее минимум совпадает с максимумом аргумента. Таким образом, минимум любой монотонной функции от суммы шаговых затрат может быть найден по методу динамического программирования. Нужно найти минимум (или максимум) суммы шаговых затрат, полученное решение будет окончательным, но нужно еще по известной сумме затрат вычислить значение исходной целевой функции (например, логарифм).

Если же исходная целевая функция от суммы шаговых затрат не является монотонной, то метод динамического программирования применить не удастся.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Аоки М. Введение в методы оптимизации: Пер. с англ. — М.: Наука, 1977. — 343 с.
2. Беллман Р. Динамическое программирование. — М.: ИЛ, 1960.
3. Беллман Р., Дрейфус С. Прикладные задачи динамического программирования. — М.: Наука, 1964.
4. Вентцель Е.С. Исследование операций: задачи, принципы, методология. — 2-е изд. — М.: Наука, 1988. — 210 с.
5. Гилл Ф., Мюррей У., Райт М. Практическая оптимизация. Пер. с англ. — М.: Мир, 1985. — 509 с.
6. Зойтендейк Дж.Г. Методы возможных направлений. Пер. с англ. — М.: ИЛ, 1963. — 214 с.
7. Зуховицкий С.И., Авдеева Л.И. Линейное и выпуклое программирование, 2-е изд. — М.: Наука, 1967. — 348 с.
8. Карманов В.Г. Математическое программирование. — М.: Наука, 1980. — 256 с.
9. Кремер Н.Ш., Путка Б.А., Тришин И.М., Фридман М.Н. Исследование операций в экономике. Учебн. пособие для вузов. / Под ред. проф. Н.М. Кремера. — М.: Банки и Биржи, ЮНИТИ, 1999. — 408 с.
10. Михалевич В.С., Шор Н.З. Математические основы решения задачи выбора оптимального очертания продольного профиля // Тр. Всесоюзн. НИИ транспортного строительства. Вып. 51. — 1964
11. Муртаф Б. Современное линейное программирование. Пер. с англ. — М.: Мир, 1984. — 224 с.

12. Панченко В.М., Панов А.В. Теория принятия решений. Линейное программирование. Учебн. пособие. — М.: МИРЭА(ТУ), 2002. — 51 с.
13. Первозванский А.А. Поиск. — М.: Наука, 1970. — 146 с.
14. Струченков В.И. Компьютерная программа для изучения динамического программирования // Открытое образование. — 2002. — № 5. — С. 54–59.
15. Струченков В.И. Методы оптимизации в проектировании трасс линейных сооружений // Сб. научных трудов. Искусственный интеллект в технических системах. Вып. № 20. — М.: Гос. ИФТП, 1999. — С. 97–111.
16. Хедли Дж. Нелинейное и динамическое программирование. Пер. с англ. — М.: Мир, 1967. — 506 с.
17. Химмельбау Д. Прикладное нелинейное программирование. — М.: Мир, 1975.
18. Численные методы условной оптимизации. Редакторы Ф. Гилл и У. Мюррей. Пер. с англ. — М.: Мир, 1977. — 290 с.
19. Юдин Д.Б., Гольштейн Е.Г. Линейное программирование (теория и конечные методы). — М.: Физматгиз, 1963. — 455 с.

Учебное издание

Струченков Валерий Иванович

МЕТОДЫ ОПТИМИЗАЦИИ ОСНОВЫ ТЕОРИИ, ЗАДАЧИ, ОБУЧАЮЩИЕ КОМПЬЮТЕРНЫЕ ПРОГРАММЫ

Издательство «**ЭКЗАМЕН**»

ИД № 05518 от 01.08.01

Гигиенический сертификат

№ 77.99.02.953.Д.005320.08.04 от 12.08.2004 г.

Редактор *В.И. Осипов*

Корректор *С.Н. Липовицкая*

Дизайн обложки *Л.В. Демьянова*

Компьютерная верстка *Т.А. Турскова*

105066, Москва, ул. Александра Лукьянова, д. 4, стр. 1.

www.examen.biz

E-mail: по общим вопросам: info@examen.biz;

по вопросам реализации: sale@examen.biz

тел./факс 263-96-60

Общероссийский классификатор продукции

ОК 005-93, том 2; 953005 — книги, брошюры,

литература учебная

Текст отпечатан с диапозитивов

в ОАО «Владимирская книжная типография»

600000, г. Владимир, Октябрьский проспект, д. 7

Качество печати соответствует

качеству предоставленных диапозитивов

По вопросам реализации обращаться по тел.: 263-96-60

В.И. Струченков

МЕТОДЫ ОПТИМИЗАЦИИ

Цель настоящей работы – содействовать изучению и практическому применению современных методов решения задач оптимизации в различных областях практики. Изложены теоретические основы методов линейного, нелинейного и динамического программирования, приведены необходимые сведения о специально разработанных четырех обучающих компьютерных программах, а также конкретные примеры практического применения методов оптимизации.

Каждый раздел заканчивается контрольными вопросами и задачами; на большинство из них в приложениях даны ответы и решения. Детально разобраны примеры практических задач из различных областей, каждую из которых можно решать различными методами, и дан сопоставительный анализ этих вариантов.

Для студентов и аспирантов технических вузов, изучающих методы оптимизации, а также специалистов, сталкивающихся с проблемами выработки оптимальных решений в различных областях деятельности.



ЭКЗАМЕН

ИНН 7720026291 ГУП Эделькс-Ис маг. ОУ

Э.Методы оптимизации

Цена: 113р.40к.



9785472004657 02.12.04