Алгоритмы оптимизации

Algorithms of Optimization

MYKEL J. KOCHENDERFER
TIM A. WHEELER

The MIT Press
Cambridge, Massachusetts
London, England

Алгоритмы оптимизации

МАЙКЛ КОХЕНДЕРФЕР
ТИМ УИЛЕР

ДИАЛЕКТИКА

Москва ◆ Санкт-Петербург
2020

ББК 22.19
К75
УДК 519.626

ООО “Диалектика”

Зав. редакцией *С.Н. Тригуб*

Перевод с английского и редакция докт. физ.-мат. наук *Д.А. Клюшина*

По общим вопросам обращайтесь в издательство “Диалектика” по адресу:
info.dialektika@gmail.com, http://www.dialektika.com

Кохендерфер, Майкл Дж., **Уилер**, Тим А.

К75 Алгоритмы оптимизации. : Пер. с англ. — СПб. : ООО “Диалектика”, 2020. — 528 с. : ил. — Парал. тит. англ.

ISBN 978-5-907144-76-7 (рус.)

ББК 22.19

Научно-популярное издание

Майкл Дж. Кохендерфер, Тим А. Уилер

Алгоритмы оптимизации

ООО “Диалектика”, 195027, Санкт-Петербург, Магнитогорская ул., д. 30, лит. А, пом. 848

ISBN 978-5-907144-76-7 (рус.)

ISBN 978-0-262-03942-0 (англ.)

Оглавление

Содержание

Посвящается нашим семьям

Предисловие

Эта книга дает широкое введение в оптимизацию с упором на практические алгоритмы проектирования инженерных систем. Мы охватываем широкий спектр тем, связанных с оптимизацией, представляя базовые математические формулировки задач и алгоритмы их решения. Описать интуитивные предположения, лежащие в основе различных подходов, помогают рисунки, примеры и упражнения.

Материал книги предназначен для студентов старших курсов и аспирантов, а также специалистов. Его понимание требует некоторой математической зрелости и предполагает предварительное знакомство с анализом функций многих переменных, линейной алгеброй и теории вероятности. Некоторые обзорные материалы представлены в приложении. Дисциплины, в которых книга была бы особенно полезна, включают математику, статистику, информатику, аэрокосмическую отрасль, электротехнику и исследования операций.

Основой этого учебника являются алгоритмы, реализованные на языке программирования Julia. Мы считаем этот язык идеальным для определения алгоритмов в удобной для чтения форме. Читатели могут бесплатно использовать фрагменты кодов, приведенные в книге, при условии ссылки на источник кода. Мы предполагаем, что многие захотят перевести эти алгоритмы на другие языки программирования. По мере появления таких переводов мы будем ссылаться на них с веб-страницы книги.

Майкл Дж. Кохендерфер
Тим А. Уилер
Стэнфорд, Калифорния
12 августа 2018 г.
Вспомогательные материалы доступны на веб-странице книги:
`http://mitpress.mit.edu/algorithms-for-optimization`

Благодарности

Этот учебник возник в рамках курса по оптимизации инженерного проектирования, преподаваемого в Стэнфорде. Мы благодарны студентам и ассистентам, которые помогли сформировать курс за последние пять лет. Мы также в долгу перед преподавателями, которые вели варианты этого курса на нашем факультете: Хоакимом Мартинсом, Хуаном Алонсо, Иланом Кру, Девом Раджнараяном и Джейсоном Хикеном. Многие темы, обсуждаемые в этом учебнике, были вдохновлены их конспектами лекций.

Авторы хотели бы поблагодарить многих людей, которые предоставили ценные отзывы о нашей работе, в том числе Мохамеда Абделати, Атиша Агарвала, Пьерджорджо Алотто, Давида Ата, Риши Беди, Феликса Беркенкампа, Раунака Бхаттачарья, Ханса Борхерса, Максима Бутона, Абхишека Каулиги, Мо Чен, Винса Чиу, Джонатана Кокс, Кэтрин Дриггс-Кэмпбелл, Тай Дуонг, Хамзу Эль-Саави, Софиан Эннадир, Тамаса Гала, Кристофера Лазаря Гарсию, Майкла Гоббла, Роберта Гоедмана, Джаеша Гупта, Аарона Хейвенса, Богумила Каминski, Уокера Кихо, Петра Крысла, Джесси Лозон, Руилин Ли, Иблиса Лина, Эдварда Лонднера, Чарльза Лу, Майлза Любина, Маркуса Любке, Жаклин Мачески, Эша Магальеса, Зохейра Махбуби, Джереми Мортона, Роберта Моса, Сантьяго Падрона, Харша Пателя, Дерека Филипса, Сидда Рао, Андреаса Решка, Алекса Рейнелла, Пера Ратквиста, Орсона Сандовала, Джеффри Сарноффа, Сумит Сингха, Натана Стейси, Алекса Тоуэс, Памелу Томан, Рэйчел Томпа, Захария Тутена, Юрия Вишневского, Джули Уокер, Зиджана Вана, Патрика Вашингтона, Адама Виктора, Роберта Янга и Андреа Занетт. Кроме того, было очень приятно работать с Мари Луфкин Ли и Кристиной Бриджит Сэвидж из издательства MIT Press при подготовке этой рукописи для публикации.

Стиль этой книги был вдохновлен Эдвардом Туфте. Помимо других стилистических элементов, мы приняли его предложение использовать широкие поля и раскадровки. На самом деле набор текста этой книги во многом основан на пакете Tufte-LaTeX Кевина Годби, Била Клеба и Билла Вуда. Мы также были вдохновлены ясностью учебников Дональда Кнута и Стивена Бойда.

За последние несколько лет мы получили пользу от дискуссий с основными разработчиками языка Julia, включая Джеффа Безансона, Стефана Карпински и Вирала Шаха. Мы также изучили различные пакеты с открытым исходным кодом, от которых зависит этот учебник (см. приложение A.4). Набор текста в коде выполнен с помощью пакета `pythontex`, который поддерживает Джеффри Пур. Рисунки построены с помощью пакета `pgfplots`, который поддерживается Кристианом Фейерсангером. Цветовая схема книги была адаптирована из темы Monikai текстового редактора Sublime Text, созданного Джоном Скиннером. Для построения графиков мы использовали цветовую палитру viridis, созданную Стефаном ван дер Уолтом и Натаниэлем Смитом.

От издательства

Вы, читатель этой книги, и есть главный ее критик и комментатор. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересно услышать и любые другие замечания, которые вам хотелось бы высказать в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам электронное письмо либо просто посетить наш веб-сайт и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится или нет вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Посылая письмо или сообщение, не забудьте указать название книги и ее авторов, а также ваш обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию последующих книг.

Наши электронные адреса:

E-mail: `info@dialektika.com`

WWW: `http://www.dialektika.com`

Глава 1

Введение

Многие научные и инженерные дисциплины основаны на оптимизации. В физике системы стремятся к состоянию с наименьшей энергией в соответствии с законами природы. В бизнесе корпорации нацелены на максимальную стоимость акций. В биологии с большей вероятностью выживают более адаптивные организмы. Эта книга посвящена оптимизации с инженерной точки зрения, для которой целью является разработка систем, оптимизирующих набор показателей с учетом ограничений. Система может быть сложным физическим объектом, как самолет, или простым, как рама велосипеда. Система может даже не быть физической; например, нас может интересовать разработка системы управления беспилотным транспортным средством или системы компьютерного зрения, которая определяет, свидетельствует ли фотография биопсии о злокачественной опухоли. Мы хотим, чтобы эти системы работали как можно лучше. В зависимости от области применения соответствующие показатели могут включать эффективность, безопасность и точность. Проектные ограничения могут включать стоимость, вес и структурную прочность.

Эта книга об *алгоритмах* оптимизации. Имея некоторое представление о конструкции системы, например наборе чисел, описывающих геометрию аэродинамического профиля, с помощью этих алгоритмов мы можем проанализировать множество возможных проектных решений в поисках наилучшего. В зависимости от приложения этот поиск может включать проведение физических экспериментов, таких как испытания в аэродинамической трубе либо анализ аналитических выражений, или компьютерное моделирование. Мы обсудим вычислительные методы, в частности поиск в многомерных пространствах, решение задач, связанных с множеством конкурирующих целей, и учет неопределенности в параметрах.

1.1. История

Начнем обсуждение истории алгоритмов оптимизации[1] с древнегреческих философов. Пифагор Самосский (569–475 гг. до н.э.), автор теоремы Пифагора, утверждал: “Все вещи — суть числа” [3], популяризируя идею о том, что матема-

[1] Эта дискуссия не претендует на полноту. Более подробную историю можно найти в книге X.-S. Yang [163].

тика может моделировать мир. Платон (427–347 гг. до н.э.) и Аристотель (384–322 гг. до н.э.) использовали логические рассуждения с целью оптимизации общества [83]. Они размышляли о наилучшей организации жизни людей, которая предполагает оптимизацию как индивидуального образа жизни, так и функционирования государства. Аристотелевская логика была одним из первых формальных процессов — алгоритмом, с помощью которого можно делать выводы.

Оптимизация математических абстракций также датируется тысячелетиями. Евклид Александрийский (325–265 гг. до н.э.) решил первые задачи оптимизации в геометрии, в том числе показал, как найти самые короткие и длинные отрезки от точки до окружности. Он также доказал, что квадрат — это прямоугольник с максимальной площадью при фиксированном периметре [47]. Греческий математик Зенодор (200–140 гг. до н.э.) изучал задачу Дидоны, показанную на рис. 1.1.

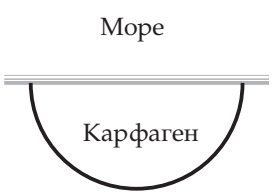


Рис. 1.1. Царица Дидона, основательница Карфагена, получила столько земли, сколько могла охватить ремешками из бычьей кожи. Она сделала полукруг, закрепив его концы на берегу Средиземного моря, таким образом, охватив максимально возможную область. Эта задача упоминается в “Энеиде” Вергилия (19 г. до н.э.)

Другие мыслители утверждали, что природа, по-видимому, оптимизируется самостоятельно. Герон Александрийский (10–75 г. н.э.) показал, что свет проходит между точками по кратчайшему пути. Папп Александрийский (290–350 гг. н.э.), помимо других многочисленных вкладов в оптимизацию, утверждал, что шестиугольник, повторяемый в сотах, является оптимальным регулярным многоугольником для хранения меда; его шестиугольная структура использует наименьшее количество материала для создания решетки из клеток на плоскости [65].

Центральное место в изучении оптимизации занимает использование *алгебры*, т.е. изучение правил манипулирования математическими символами. Открытие алгебры приписывается персидскому математику аль-Хорезми (790–850 гг. н.э.) в его трактате “Аль-Китаб алжабр валь-мукабала” (“Краткая книга восполнения и противопоставления”). Алгебра имела преимущество за счет использования индо-арабских цифр, включая применение нуля в позиционной системе счисления. Персидское слово *al'jabr* означает “восполнение” и является источником для западного слова *алгебра*. Термин *алгоритм* происходит от слова *algoritmi*, латинского перевода и произношения имени аль-Хорезми (al-Khwārismī).

Задачи оптимизации часто представляют собой поиск в пространстве, заданном набором координат. Использование координат происходит от Рене Декарта

(1596–1650), который применял два числа для описания точки на двумерной плоскости. Его проницательность связала алгебру и ее аналитические уравнения с описательными и визуальными средствами геометрии [38]. Он также открыл метод нахождения касательной к любой кривой, уравнение которой известно. Касательные полезны при определении минимумов и максимумов функций. Пьер де Ферма (1601–1665) начал решать задачи вычисления точек, в которых производная равна нулю, чтобы определить потенциально оптимальные точки.

Важную роль в нашем обсуждении оптимизации играет *математический анализ*, или наука о непрерывных изменениях. Современный математический анализ основан на результатах работы Готфрида Вильгельма Лейбница (1646–1716) и сэра Исаака Ньютона (1642–1727). Как дифференциальное, так и интегральное исчисление используют понятие сходимости бесконечных рядов к точно определенному пределу.

В середине двадцатого века стали популярными электронные компьютеры, что вызвало интерес к численным алгоритмам оптимизации. Простота вычислений позволила применить оптимизацию к более крупным задачам в различных областях. Одним из главных достижений стало изобретение линейного программирования, которое представляет собой оптимизацию с линейной целевой функцией и линейными ограничениями. Леонид Канторович (1912–1986) первым сформулировал задачу линейного программирования и алгоритм ее решения [78]. Этот алгоритм был применен к задачам оптимального распределения ресурсов во время Второй мировой войны. Джордж Данциг (1914–2005) разработал симплекс-метод, который представляет собой значительный прогресс в эффективном решении задач линейного программирования.[2] Ричард Беллман (1920–1984) разработал понятие динамического программирования, которое является широко используемым методом оптимального решения сложных задач с помощью их декомпозиции на более простые задачи [14]. Динамическое программирование широко применяется для оптимального управления. В этом учебнике описаны многие ключевые алгоритмы, разработанные для цифровых компьютеров, которые использовались для решения различных задач оптимизации инженерного проектирования.

Десятилетия достижений в области крупномасштабных вычислений привели к инновационным проектам физической инженерии, а также к разработке систем искусственного интеллекта. Интеллект этих систем был продемонстрирован в таких играх, как шахматы, Jeopardy! и Go. В 1996 г. программа IBM Deep Blue победила чемпиона мира по шахматам Гарри Каспарова, вычислив оптимальные ходы и оценив миллионы позиций. В 2011 г. компания Watson, разработанная компанией IBM, играла в Jeopardy! против бывших победителей Брэда Футтера

[2] Симплекс-метод рассматривается в главе 11.

и Кена Дженнингса. Программа Watson выиграла приз за первое место в размере одного миллиона долларов, оптимизировав свой ответ на основании вероятностных выводов о 200 миллионах страниц структурированных и неструктурированных данных. Впоследствии система развивалась, чтобы помочь в принятии медицинских решений и прогнозировании погоды. В 2017 г. программа AlphaGo компании Google победила Кэ Цзе, который имел первый рейтинг среди игроков в го. В этой системе использовались нейронные сети с миллионами параметров, которые были оптимизированы на основе самостоятельной игры и данных об играх между людьми. Оптимизация глубоких нейронных сетей способствует крупной революции в области искусственного интеллекта, которая, вероятно, продолжится [61].

1.2. Процесс оптимизации

Типичный процесс оптимизации инженерного проектирования показан на рис. 1.2.[3] Роль *конструктора* заключается в предоставлении *технического задания*, которое детализирует параметры, константы, цели и ограничения. Конструктор отвечает за постановку задачи и количественную оценку достоинств потенциальных проектов. Он также обычно предоставляет базовый проект или начальную точку проектирования для алгоритма оптимизации.

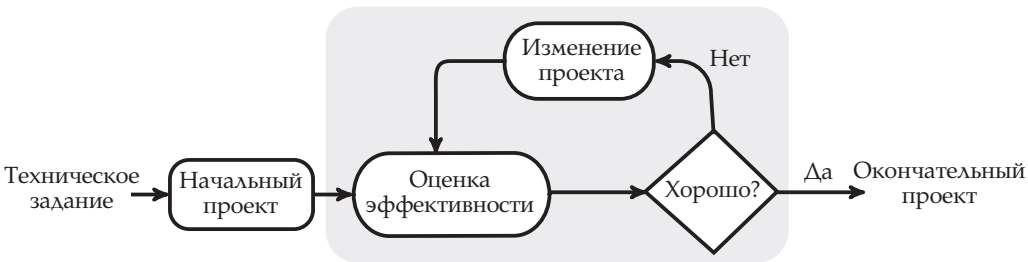


Рис. 1.2. Процесс оптимизации проекта. Мы стремимся автоматизировать процедуру оптимизации, выделенную синим цветом

Эта книга посвящена автоматизации процесса уточнения проекта для повышения его эффективности. Алгоритм оптимизации используется для постепенного улучшения проекта до тех пор, пока проект больше не может быть улучшен или пока не будет затрачено запланированное время либо превышена предельно допустимая стоимость. Конструктор несет ответственность за анализ результатов процесса оптимизации, чтобы обеспечить его пригодность для конечного применения. Неправильные спецификации в постановке задачи, плохой начальный

[3] Дальнейшее обсуждение процесса инженерного проектирования см. в [5].

проект и неправильно реализованные или неподходящие алгоритмы оптимизации могут привести к неоптимальным или опасным проектам.

Есть несколько преимуществ оптимизации подхода к проектированию. Прежде всего, процесс оптимизации обеспечивает систематическую, логичную процедуру проектирования. При ее правильном соблюдении алгоритмы оптимизации могут уменьшить вероятность ошибки человека при проектировании. Интуиция в инженерном проектировании может ввести в заблуждение; намного лучше оптимизировать данные. Оптимизация может ускорить процесс проектирования, особенно когда процедура может быть написана один раз, а затем повторно применена к другим задачам. Традиционные инженерные методы часто визуализируются и обосновываются людьми в двух или трех измерениях. В то же время современные методы оптимизации могут применяться к задачам с миллионами переменных и ограничений.

Есть также проблемы, связанные с использованием оптимизации для проектирования. Мы обычно ограничены в наших вычислительных ресурсах и времени, и поэтому наши алгоритмы должны быть избирательными в том, как они исследуют пространство проектных параметров. По сути, алгоритмы оптимизации ограничены способностью конструктора формулировать задачу. В некоторых случаях алгоритм оптимизации может использовать ошибки моделирования или найти решение, которое не позволяет адекватно решить поставленную задачу. Когда алгоритм приводит к оптимальному проекту, который противоречит интуиции, его может быть трудно интерпретировать. Другое ограничение заключается в том, что многие алгоритмы оптимизации не всегда гарантируют получение оптимальных проектов.

1.3. Основная задача оптимизации

Основная задача оптимизации формулируется следующим образом:

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) \\ &\text{при условии, что } \mathbf{x} \in \mathcal{X}. \end{aligned} \tag{1.1}$$

Здесь $\mathbf{x}$ — *расчетная точка* (design point). Расчетная точка может быть представлена как вектор значений, соответствующих различным *расчетным переменным* (design variables). Расчетная точка в n-мерном пространстве записывается следующим образом:[4]

$$[x_1, x_2, ..., x_n], \tag{1.2}$$

[4] В языке Julia квадратные скобки представляют векторы-столбцы. Расчетные точки — это векторы-столбцы.

где i-я расчетная переменная обозначена x_i. Элементы в этом векторе можно регулировать, чтобы минимизировать *целевую функцию* f. Любое значение x из всех точек в *допустимом множестве* $\mathcal{X}$, которое минимизирует целевую функцию, называется *решением* или *точкой минимума*. Конкретное решение записывается как $\mathbf{x}^*$. Пример задачи одномерной оптимизации показан на рис. 1.3.

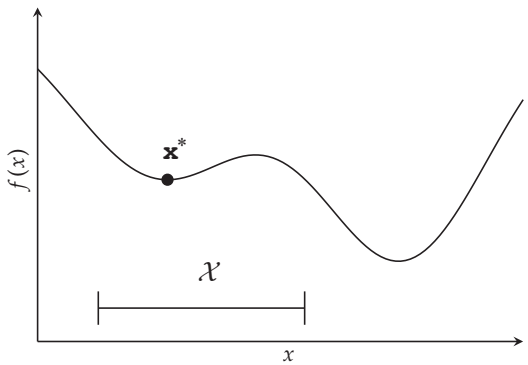


Рис. 1.3. Задача одномерной оптимизации. Обратите внимание на то, что минимум является просто лучшим вариантом в возможном наборе — вне допустимой области могут существовать точки с более низкими значениями

Эта формулировка является общей, т.е. любая задача оптимизации может быть переписана в соответствии с уравнением (1.1). В частности, задачу

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) \\ &\text{при условии, что } \mathbf{x} \in \mathcal{X}. \end{aligned} \tag{1.3}$$

можно переформулировать так:

$$\begin{aligned} &\max_{\mathbf{x}} \; -f(\mathbf{x}) \\ &\text{при условии, что } \mathbf{x} \in \mathcal{X}. \end{aligned} \tag{1.4}$$

Задача в новой формулировке имеет тот же самый набор решений.

Моделирование инженерных задач в рамках этой математической формулировки может быть сложной задачей. То, как мы формулируем задачу оптимизации, может сделать процесс решения простым или сложным [22]. Сосредоточимся на алгоритмических аспектах оптимизации, возникающих после того, как проблема была правильно сформулирована.[5]

Поскольку в этой книге обсуждается множество различных алгоритмов оптимизации, можно задаться вопросом, какой алгоритм лучше. Как утверждает *теорема об отсутствии бесплатного завтрака* Вольперта и Макриди, нет причин

[5] Многочисленные примеры преобразования практических задач проектирования в задачи оптимизации приведены, например, в [6], [13], [80], [118].

предпочитать один алгоритм другому, если только мы не сделаем предположений о распределении вероятностей по пространству возможных целевых функций. Если один алгоритм работает лучше, чем другой алгоритм для одного класса задач, то он будет работать хуже для другого класса задач.[6] Чтобы многие алгоритмы оптимизации работали эффективно, в целевой функции должна быть некоторая регулярность, например липшиц-непрерывность или выпуклость. Эти темы мы рассмотрим позже. Обсуждая различные алгоритмы, мы опишем их предположения, обоснование их работы, а также их преимущества и недостатки.

1.4. Ограничения

Многие задачи имеют ограничения. Каждое *ограничение* выделяет множество возможных решений, и в совокупности ограничения определяют допустимое множество $\mathcal{X}$. Допустимые расчетные точки не нарушают никаких ограничений. Например, рассмотрим следующую проблему оптимизации:

$$\begin{aligned} &\min_{x_1,x_2} f(x_1,x_2) \\ &\text{при условии, что} \quad x_1 \geq 0, \\ &\qquad\qquad\qquad\quad x_2 \geq 0, \\ &\qquad\qquad\qquad\quad x_1 + x_2 \leq 1. \end{aligned} \tag{1.5}$$

Допустимое множество изображено на рис. 1.4.

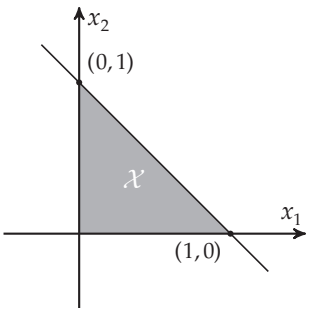


Рис. 1.4. Допустимое множество $\mathcal{X}$, заданное неравенствами (1.5)

Ограничения обычно записываются с помощью знаков $\leq$, $\geq$ или $=$. Если ограничения включают знаки $<$ или $>$ (т.е. строгие неравенства), то допустимое множество не включает границу ограничений. Потенциальная проблема, которая может возникнуть без учета границы иллюстрируется следующей задачей:

[6] Предпосылки и следствия из теоремы об отсутствии бесплатного завтрака изложены в работе [160].

$$\min_x x$$
$$\text{при условии, что } x > 1. \tag{1.6}$$

Допустимое множество показано на рис. 1.5. Точка $x = 1$ меньше любого x, превышающего единицу, но значение $x = 1$ недопустимо. Мы можем выбрать любой x, произвольно близкий к единице, но превышающей ее, и независимо от того, что мы выберем, всегда можно найти бесконечное количество значений, которые расположены еще ближе к единице. Мы вынуждены констатировать, что задача не имеет решения. Чтобы избежать таких проблем, часто лучше включать границу ограничений в допустимое множество.

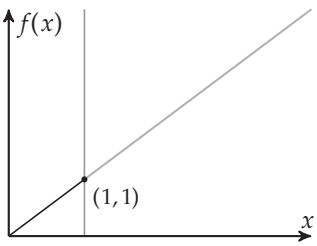


Рис. 1.5. Задача (1.6) не имеет решения, поскольку граница ограничения недопустима

1.5. Критические точки

На рис. 1.6 показана *одномерная функция*[7] $f(x)$ с несколькими помеченными *критическими точками*, в которых производная равна нулю и которые представляют интерес при обсуждении задач оптимизации. При минимизации функции f мы хотим найти *точку глобального минимума*, т.е. значение x, в котором значение $f(x)$ является минимальным. Функция может иметь не более одного глобального минимума, но может иметь несколько точек глобального минимума.

К сожалению, как правило, трудно доказать, что данная точка-кандидат является точкой глобального минимума. Часто лучшее, что мы можем сделать, это проверить, соответствует ли она локальному минимуму. Точка x^* является точкой *локального минимума*, если существует число $\delta > 0$ такое, что $f(x^*) \leq f(x)$ для всех x, удовлетворяющих условию $|x - x^*| < \delta$. В многомерном контексте это определение сводится к существованию числа $\delta > 0$ такого, что $f(\mathbf{x}^*) \leq f(\mathbf{x})$ для всех $\mathbf{x}$, удовлетворяющих условию $\|\mathbf{x} - \mathbf{x}^*\| < \delta$.

На рис. 1.6 показаны два типа локальных минимумов: *сильный* и *слабый*. *Точка сильного локального минимума*, которая также называется *точкой строгого локального минимума*, — это точка, которая однозначно минимизирует f в окрест-

[7] Одномерная функция — это функция одного скаляра.

ности. Иначе говоря, точка x^* является точкой строгого локального минимума, если существует число $\delta > 0$ такое, что $f(x^*) < f(x)$ всякий раз, когда $x^* \neq x$ и $|x - x^*| < \delta$. В многомерном контексте это определение сводится к существованию числа $\delta > 0$ такого, что $f(\mathbf{x}^*) < f(\mathbf{x})$ всякий раз, когда $\mathbf{x}^* \neq \mathbf{x}$ и $\|\mathbf{x} - \mathbf{x}^*\| < \delta$. Точка слабого локального минимума — это точка локального минимума, которая не является точкой сильного локального минимума.

Рис. 1.6. Примеры критических точек одномерной функции, представляющих интерес для алгоритмов оптимизации (в которых производная равна нулю)

Во всех точках локального и глобального минимума производная непрерывной неограниченной целевой функции равна нулю. Равенство производной нулю — *необходимое*[8], но не *достаточное* условие для локального минимума.

На рис. 1.6 показана *точка перегиба*, где производная равна нулю, но эта точка не является точкой локального минимума функции f. Точка перегиба — это место, где меняется знак второй производной функции f, что соответствует локальному минимуму или максимуму ее первой производной f'. Производная в точке перегиба не обязательно равна нулю.

1.6. Условия локального минимума

Многие методы численной оптимизации ищут локальные минимумы. Локальные минимумы локально оптимальны, но мы обычно не знаем, является ли локальный минимум глобальным минимумом. Условия, которые мы обсуждаем в этом разделе, предполагают, что целевая функция является дифференцируемой. Производные, градиенты и гессианы рассматриваются в следующей главе. В этом разделе мы также предполагаем, что задача не имеет ограничений. Условия оптимальности в задачах с ограничениями представлены в главе 10.

[8] Точки, в которых производная не равна нулю, не могут быть точками минимума.

1.6.1. Одномерный случай

Расчетная точка гарантированно будет точкой сильного локального минимума, если локальная производная равна нулю, а вторая производная положительна:

1. $f'(x^*) = 0$,
2. $f''(x^*) > 0$.

Нулевая производная гарантирует, что смещение точки на маленькие значения не влияет на значение функции. Положительная вторая производная гарантирует, что первая производная принимает нулевое значение на дне *чаши*.[9]

Точка также может быть точкой локального минимума, если функция в этой точке обращается в нуль, а вторая производная является просто неотрицательной.

1. $f'(x) = 0$, *необходимое условие первого порядка* (first-order necessary condition — FONC).[10]
2. $f''(x) \geq 0$, *необходимое условие второго порядка* (second-order necessary condition — SONC).

Эти условия называются необходимыми, поскольку все точки локального минимума подчиняются этим двум правилам. К сожалению, не все точки с нулевой производной и нулевой второй производной являются локальными минимумами, как показано на рис 1.7.

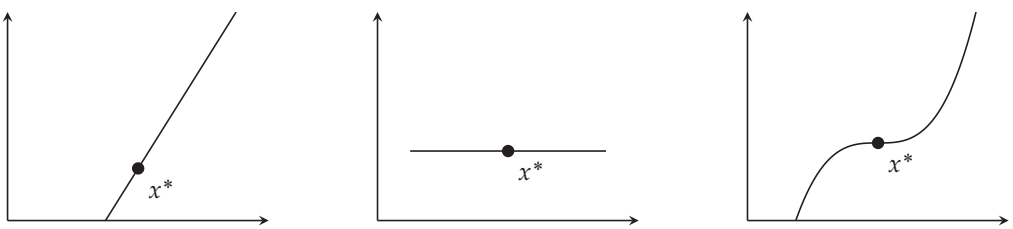


Рис. 1.7. Примеры необходимых, но недостаточных условий сильных локальных минимумов

Первое необходимое условие может быть получено с использованием разложения Тейлора[11] в окрестности точки-кандидата x^*.

$$f(x^* + h) = f(x^*) + hf'(x^*) + O(h^2), \tag{1.7}$$

$$f(x^* - h) = f(x^*) - hf'(x^*) + O(h^2), \tag{1.8}$$

---

[9] Если $f'(x) = 0$ и $f''(x) < 0$, то это — локальный максимум.

[10] Точка, которая удовлетворяет необходимому условию первого порядка, иногда называется *стационарной.*

[11] Разложение Тейлора описывается в приложении В.

$$f(x^* + h) \geq f(x^*) \Rightarrow hf'(x^*) \geq 0, \tag{1.9}$$

$$f(x^* - h) \geq f(x^*) \Rightarrow hf'(x^*) \leq 0, \tag{1.10}$$

$$\Rightarrow f'(x^*) = 0, \tag{1.11}$$

где асимптотическое обозначение $O(h^2)$ описывается в приложении B.

Необходимое условие второго порядка также можно получить из разложения Тейлора:

$$f\left(x^* + h\right) = f\left(x^*\right) + \underbrace{hf'\left(x^*\right)}_{=0} + \frac{h^2}{2} f''\left(x^*\right) + O\left(h^3\right). \tag{1.12}$$

Мы знаем, что должно быть выполнено необходимое условие первого порядка:

$$f\left(x^* + h\right) \geq f\left(x^*\right) \Rightarrow \frac{h^2}{2} f''\left(x^*\right) \geq 0, \tag{1.13}$$

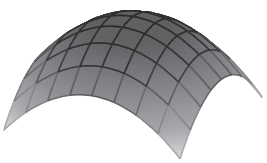
поскольку $h > 0$. Отсюда следует, что для того, чтобы точка x^* была точкой локального минимума, должно выполняться условие $f''(x^*) \geq 0$.

1.6.2. Многомерный случай

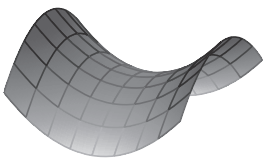
Для того чтобы точка $\mathbf{x}$ была локальным минимумом функции f, необходимы следующие условия.

1. $\nabla f(\mathbf{x}) = 0$, *необходимое условие первого порядка* (FONC).
2. $\nabla^2 f(\mathbf{x})$ — положительно полуопределенная матрица (обзор этого определения см. в приложении B.6), необходимое условие второго порядка (SONC).

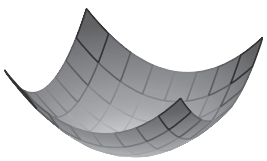
Условия FONC и SONC являются обобщениями одномерного случая. Условие FONC означает, что функция не изменяется в точке $\mathbf{x}$. На рис. 1.8 показаны примеры многомерных функций, для которых выполняется условие FONC. Условие SONC утверждает, что значение $\mathbf{x}$ лежит в чаше.



Локальный максимум. Градиент в центре равен нулю, но гессиан отрицательно определен



Седло. Градиент в центре равен нулю, но эта точка не является локальным минимумом



Чаша. Градиент в центре равен нулю и гессиан положительно определен. Это — локальный минимум

Рис. 1.8. Три локальные области, где градиент равен нулю

Условия FONC и SONC можно получить с помощью простого анализа. Для того чтобы точка $\mathbf{x}^*$ была локальным минимумом, функция в этой точке должна быть меньше, чем все значения вокруг нее:

$$f(\mathbf{x}^*) \leq f(\mathbf{x}^* + h\mathbf{y}) \Leftrightarrow f(\mathbf{x}^* + h\mathbf{y}) - f(\mathbf{x}^*) \geq 0. \tag{1.14}$$

Если мы напишем аппроксимацию второго порядка для $f(\mathbf{x}^*)$, то получим:

$$f\left(\mathbf{x}^* + h\mathbf{y}\right) = f\left(\mathbf{x}^*\right) + h\nabla f\left(\mathbf{x}^*\right)^{\mathrm{T}}\mathbf{y} + \frac{1}{2}h^2\mathbf{y}^{\mathrm{T}}\nabla^2 f\left(\mathbf{x}^*\right)\mathbf{y} + O\left(h^3\right). \tag{1.15}$$

Мы знаем, что в точке минимума первая производная должна быть равна нулю, и пренебрегаем членами более высокого порядка. Переставляя слагаемые, получаем:

$$\frac{1}{2}h^2\mathbf{y}^{\mathrm{T}}\nabla^2 f\left(\mathbf{x}^*\right)\mathbf{y} = f\left(\mathbf{x}^* + h\mathbf{y}\right) - f\left(\mathbf{x}^*\right) \geq 0. \tag{1.16}$$

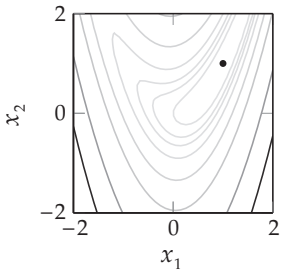
Это определение положительной полуопределенной матрицы, и мы получаем условие SONC. Пример 1.1 иллюстрирует, как эти условия можно применить к функции Розенброка (Rosenbrock function).

Пример 1.1. Проверка необходимых условий первого и второго порядка в точке для функции Розенброка. Точка минимума обозначена на рисунке черным кружочком

Рассмотрим функцию Розенброка

$$f\left(\mathbf{x}\right) = \left(1 - x_1\right)^2 + 5\left(x_2 - x_1^2\right)^2.$$

Удовлетворяет ли точка (1, 1) условиям FONC и SONC?



Градиент равен

$$\nabla f\left(\mathbf{x}\right) = \begin{bmatrix} \dfrac{\partial f}{\partial x_1} \\ \dfrac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2\left(10x_1^3 - 10x_1x_2 + x_1 - 1\right) \\ 10\left(x_2 - x_1^2\right) \end{bmatrix}.$$

Гессиан равен

$$\nabla^2 f(\mathbf{x}) = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1 \partial x_1} & \dfrac{\partial^2 f}{\partial x_1 \partial x_2} \\ \dfrac{\partial^2 f}{\partial x_2 \partial x_1} & \dfrac{\partial^2 f}{\partial x_2 \partial x_2} \end{bmatrix} = \begin{bmatrix} -20\left(x_2 - x_1^2\right) + 40x_1^2 + 2 & -20x_1 \\ -20x_1 & 10 \end{bmatrix}.$$

Поскольку $\nabla f([1, 1]) = 0$, условие FONC выполняется. Гессиан в точке [1, 1] равен

$$\begin{bmatrix} 42 & -20 \\ -20 & 10 \end{bmatrix}.$$

Эта матрица является положительно определенной, поэтому условие SONC выполняется.

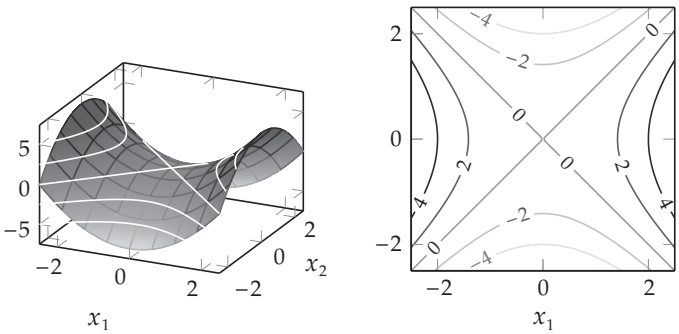
Условия FONC и SONC являются необходимыми, но не достаточными условиями оптимальности. Для оптимизации без ограничений условие, что функция является дважды дифференцируемой, гарантирует, что функция имеет сильный локальный минимум, если условие FONC удовлетворяется, а матрица $\nabla^2 f(\mathbf{x})$ положительно определена. Эти условия, взятые в совокупности, называются *достаточными условиями второго порядка* (second-order sufficient condition — SOSC).

1.7. Изолинии

Эта книга содержит задачи разной размерности, и нам нужен инструмент для отображения информации в одном, двух или трех измерениях. Функции вида $f(x_1, x_2) = y$ могут быть представлены в трехмерном пространстве, но не все ориентации обеспечивают полный обзор области. *Диаграмма изолиний* (contour plot) — это визуальное представление трехмерной поверхности, полученное путем построения областей с постоянными значениями y, которые называются *изолиниями*, на двухмерной диаграмме с осями, индексированными по x_1 и x_2. Интерпретация диаграммы изолиний показана в примере 1.2.

Пример 1.2. Пример диаграммы изолиний и связанная с ней трехмерная визуализация

Функция $f(x_1, x_2) = x_1^2 - x_2^2$. Эта функция может быть представлена в трехмерном пространстве с помощью двух переменных и одного значения. Ее также можно представить с помощью диаграммы изолиний, на которой показаны линии с постоянными значениями y. Трехмерная визуализация и диаграмма изолиний показаны ниже.



1.8. Обзор

В этом разделе представлен краткий обзор глав книги. Концептуальные зависимости между главами приведены на рис. 1.9.

Рис. 1.9. Зависимости между главами книги.
Более светлые стрелки означают более слабые связи

Глава 2 начинается с обсуждения *производных* и их обобщения на многомерный случай. Производные используются во многих алгоритмах для выбора направления поиска оптимальной точки. Часто производные невозможно вычислить аналитически, поэтому мы обсуждаем способы их вычислений и методы автоматического дифференцирования.

Глава 3 посвящена *методу интервалов*, который предназначен для определения интервала, в котором лежит точка локального минимума одномерной функции. Различные алгоритмы метода интервалов используют разные схемы последовательного сокращения интервалов на основе оценки функций. Один из подходов, которые мы обсуждаем, применяет знание константы Липшица для

управления методом интервалов. Эти алгоритмы метода интервалов часто используются в качестве подпрограмм в алгоритмах оптимизации, обсуждаемых далее в тексте.

В глава 4 описывается *метод локального спуска* как общий подход к оптимизации многомерных функций. Локальный спуск включает в себя итеративный выбор направления спуска, а затем выполнение шага в этом направлении и повторение этого процесса до тех пор, пока не будут выполнены критерий сходимости или какое-либо условие завершения алгоритма. Существуют разные схемы выбора длины шага. Мы также обсудим методы, которые адаптивно ограничивают размер шага областью, в которой локальная модель является достоверной.

Глава 5 основана на предыдущей главе, в которой объясняется, как использовать информацию *первого порядка*, полученную посредством оценки градиента, в качестве локальной модели для выбора направлении снижения. Простой шаг в направлении наискорейшего спуска — часто не лучшая стратегия для нахождения минимума. В этой главе обсуждается широкий спектр различных методов использования прошлой последовательности оценок градиента, чтобы повысить эффективность поиска.

В главе 6 показано, как использовать локальные модели на основе аппроксимаций *второго порядка* в методе локального спуска. Эти модели основаны на оценках гессиана целевой функции. Преимущество аппроксимаций второго порядка состоит в том, что они могут определять как направление, так и размер шага.

В главе 7 представлен набор *прямых методов* поиска точек минимума, которые избегают использования информации о градиенте для выбора направления поиска. Мы начнем с обсуждения методов, которые итеративно выполняют одномерный поиск среди набора направлений. Затем мы обсудим методы поиска по шаблону, которые не выполняют одномерный поиск, а вычисляют оценки на некотором расстоянии от текущей точки по набору направлений. Размер шага постепенно изменяется по мере продолжения поиска. Другой метод использует симплекс, который адаптивно расширяется и сжимается при прохождении пространства проектных параметров в очевидном направлении улучшения. Наконец, мы обсудим метод, основанный на свойстве липшиц-непрерывности для увеличения точности метода в областях, которые могут содержать глобальный минимум.

В главе 8 представлены *стохастические методы*, в которых процесс оптимизации содержит случайность. Мы покажем, как стохастичность может улучшить такие алгоритмы, рассмотренные в предыдущих главах, как метод скорейшего спуска и поиск по шаблону. Некоторые методы содержат постепенный обход пространства проектных параметров, а другие включают изучение распределения вероятностей в областях проектных параметров, присваивая больший вес областям, которые с большей вероятностью содержат оптимальную точку.

В главе 9 обсуждаются *популяционные* методы, в которых для исследования пространства проектных параметров используется набор точек. Распределение большого количества точек в пространстве позволяет снизить риск застрять в точке локального минимума. Популяционные методы, как правило, полагаются на стохастичность для поощрения разнообразия в популяции, и их можно сочетать с методами локального спуска.

Глава 10 вводит понятие *ограничений* в задачах оптимизации. Начнем с обсуждения математических условий оптимальности с ограничениями. Затем введем методы включения ограничений в алгоритмы оптимизации, которые обсуждались ранее, с использованием штрафных функций. Кроме того, обсудим методы, гарантирующие, что, отправляясь из допустимой точки, мы никогда не выйдем за пределы допустимого множества.

В главе 11 делается предположение, что и целевая функция, и ограничения являются *линейными*. Хотя линейность может показаться сильным предположением, многие инженерные проблемы могут быть сформулированы как задачи оптимизации с линейными ограничениями. Для использования этой линейной структуры было разработано несколько методов. Эта глава посвящена симплексному алгоритму, который гарантированно приведет к глобальному минимуму.

В главе 12 показано, как решить проблему *многокритериальной* оптимизации, имея несколько целей, которые мы пытаемся оптимизировать одновременно. В инженерном деле часто приходится искать компромисс между несколькими целями, испытывая трудности с расставкой приоритетов. Мы обсуждаем, как преобразовать многокритериальные задачи в скалярные целевые функции, чтобы использовать алгоритмы, рассмотренные в предыдущих главах. Кроме того, описываем алгоритмы для нахождения набора расчетных точек, которые представляют наилучший компромисс между целями.

В главе 13 показано, как создавать *планы выборки*, состоящей из точек, распределенных в пространстве проектных параметров. Случайная выборка из пространства проектных параметров часто не обеспечивает адекватного охвата. Мы обсуждаем методы обеспечения равномерного покрытия по каждому измерению, а также методы измерения и оптимизации охвата пространства. Кроме того, мы обсуждаем квазислучайные последовательности, которые можно использовать для создания планов выборки.

Глава 14 объясняет, как построить *суррогатные модели* целевой функции. Суррогатные модели часто используются для задач, когда оценка целевой функции очень дорога. Затем алгоритм оптимизации может использовать оценки суррогатной модели вместо фактической целевой функции для улучшения проекта. Оценки могут вычисляться на основе исторических данных, возможно, полученных с использованием плана выборки, представленного в предыдущей главе. Мы

обсуждаем различные типы суррогатных моделей, объясняем, как адаптировать их к данным и как определить подходящую суррогатную модель.

Глава 15 знакомит читателя с *вероятностными суррогатными моделями*, которые позволяют количественно оценить уверенность в предсказаниях моделей. Эта глава посвящена конкретному типу суррогатной модели, называемой *гауссовским процессом*. Мы покажем, как использовать гауссовские процессы для прогнозирования, учесть измерения градиента и шум, а также оценить некоторые параметры, регулирующие гауссовский процесс, на основе данных.

Глава 16 показывает, как использовать вероятностные модели из предыдущей главы для управления *суррогатной оптимизацией*. В этой главе изложены методы выбора расчетной точки, которая должна оцениваться следующей. Мы также обсуждаем, как суррогатные модели могут быть использованы для безопасной оптимизации целевой функции.

Глава 17 объясняет, как выполнять *оптимизацию в условиях неопределенности*, ослабляя допущение, сделанное в предыдущих главах, что целевая функция является детерминированной функцией проектных параметров. Мы обсуждаем различные подходы к представлению неопределенности, в том числе основанные на множествах и вероятности, и объясняем, как преобразовать задачу, чтобы обеспечить устойчивость к неопределенности.

Глава 18 описывает подходы к *распространению неопределенности*, где известные входные распределения используются для оценки статистических величин, связанных с выходным распределением. Понимание выходного распределения целевой функции важно для оптимизации в условиях неопределенности. Мы обсуждаем различные подходы, некоторые из которых основаны на математических понятиях, таких как метод Монте-Карло, аппроксимации с помощью рядов Тейлора, ортогональные многочлены и гауссовские процессы. Они отличаются по своим предположениям и качеству оценок.

Глава 19 показывает, как подходить к задачам, в которых переменные проекта являются *дискретными*. Общий подход состоит в том, чтобы ослабить предположение, что переменные являются дискретными, но это может привести к недопустимым схемам. Другой подход предполагает постепенное добавление линейных ограничений до тех пор, пока оптимальная точка остается дискретной. Мы также обсуждаем метод ветвей и границ наряду с подходами динамического программирования, которые гарантируют оптимальность. В этой же главе упоминается популяционный метод, который часто масштабируется до больших пространств проектных параметров, но не дает гарантий.

В главе 20 обсуждаются методы поиска в пространствах проектных параметров, состоящих из *выражений*, определенных грамматикой. Для многих задач количество переменных неизвестно, например при оптимизации графовых структур или компьютерных программ. Мы наметили несколько алгоритмов, которые

учитывают грамматическую структуру пространства проектных параметров, чтобы сделать поиск более эффективным.

Глава 21 объясняет подходы к *междисциплинарной оптимизации проектов*. Многие инженерные проблемы связаны со сложным взаимодействием между несколькими дисциплинами, и индивидуальная оптимизация по отдельным дисциплинам может не привести к оптимальному решению. В этой главе рассматриваются различные методы использования структуры междисциплинарных задач для сокращения усилий, необходимых для поиска хороших проектов.

Приложения содержат дополнительный материал. Приложение А начинается с краткого введения в язык программирования Julia и содержит понятия, используемые для определения алгоритмов, перечисленных в этой книге. Приложение Б включает определения тестовых функций, используемых для оценки производительности различных алгоритмов. Приложение В охватывает математические понятия, используемые при выводе и анализе методов оптимизации, обсуждаемых в книге.

1.9. Резюме

- Оптимизация в инженерном деле — это процесс поиска наилучшей конструкции системы с учетом ряда ограничений.
- Оптимизация связана с поиском глобальных минимумов функции.
- Минимумы возникают, когда градиент равен нулю, но нулевой градиент не означает оптимальность.

1.10. Упражнения

Упражнение 1.1. Приведите пример функции с локальным минимумом, который не является глобальным минимумом.

Упражнение 1.2. Укажите точку минимума функции $f(x) = x^3 - x$.

Упражнение 1.3. Выполняется ли условие первого порядка $f'(x) = 0$, если x является оптимальным решением задачи с ограничениями?

Упражнение 1.4. Сколько минимумов имеет функция $f(x, y) = x^2 + y$ при условии, что $x > y \geq 1$?

Упражнение 1.5. Сколько точек перегиба имеет функция $f(x) = x^3 - 10$?

Глава 2

Производные и градиенты

Оптимизация связана с поиском расчетной точки, которая минимизирует (или максимизирует) целевую функцию. Знание того, как значение функции изменяется при изменении входных данных, полезно, потому что оно говорит нам, в каком направлении следует двигаться, чтобы улучшить результат по сравнению с предыдущими точками. Изменение значения функции измеряется производной в одном измерении и градиентом в нескольких измерениях. В этой главе кратко рассматриваются некоторые важные факты математического анализа [26].

2.1. Производные

Производная $f'(x)$ функции f одной переменной x — это скорость, с которой значение f изменяется в точке x. Данное определение часто иллюстрируют рисунком, похожим на рис. 2.1, с использованием касательной к графику функции в точке x. Значение производной равно тангенсу угла наклона касательной.

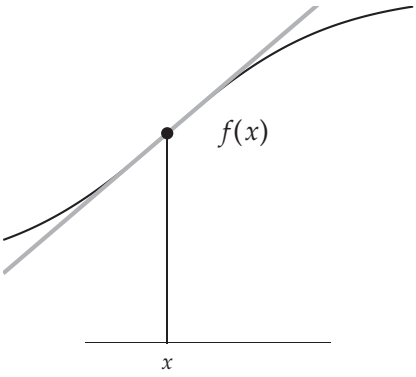


Рис. 2.1. График функции f показан в виде черной кривой, а касательная к $f(x)$ — синей прямой. Производная от f в точке x равна тангенсу угла наклона касательной

Мы можем использовать производную, чтобы построить линейное приближение функции в окрестности точки x:

$$f(x + \Delta x) \approx f(x) + f'(x)\Delta x. \tag{2.1}$$

Производная — это предел отношения между приращением функции f и приращением переменной в точке x, когда приращение переменной стремится к нулю (рис. 2.2):

$$f'(x) = \lim_{\Delta x \to 0} \frac{\Delta f}{\Delta x}. \tag{2.2}$$

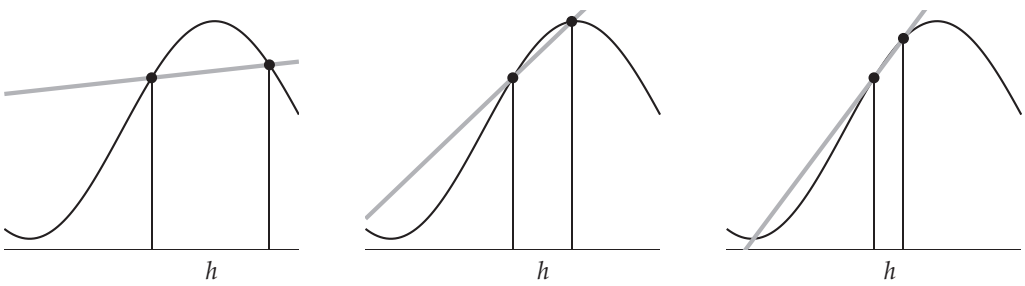


Рис. 2.2. Касательная линия получается путем соединения точек с достаточно малым шагом по x

Авторство обозначения $f'(x)$ приписывают Лагранжу. Мы также используем обозначение, изобретенное Лейбницем,

$$f'(x) \equiv \frac{df(x)}{dx}, \tag{2.3}$$

которое подчеркивает тот факт, что производная — это предел отношения изменения f к изменению переменной в точке x.

Предельную формулу, определяющую производную, можно представить тремя различными способами: с помощью *правой разности*, *центральной разности* и *левой разности*. Каждый из этих способов использует бесконечно малый размер шага h:

$$\begin{aligned} f'(x) &= \underbrace{\lim_{h \to 0} \frac{f(x+h) - f(x)}{h}}_{\text{Правая разность}} = \\ &= \underbrace{\lim_{h \to 0} \frac{f(x+h/2) - f(x-h/2)}{h}}_{\text{Центральная разность}} = \\ &= \underbrace{\lim_{h \to 0} \frac{f(x) - f(x-h)}{h}}_{\text{Левая разность}}. \end{aligned} \tag{2.4}$$

Если функцию f можно представить символически, то *символьное дифференцирование* часто может обеспечить точное аналитическое выражение для f' на основе правил дифференцирования из математического анализа. Затем аналити-

ческое выражение можно вычислить в любой точке x. Процесс иллюстрирует пример 2.1.

Пример 2.1. Символьное дифференцирование позволяет найти производную в аналитическом виде

Детали реализации символьного дифференцирования выходят за рамки данной книги. Для этого существуют разнообразные программные пакеты, такие как **`SymEngine.jl`** на языке Julia и **`SymPy`** на языке Python. Ниже мы используем пакет **`SymEngine.jl`**, чтобы вычислить производную функции $x^2 + x/2 - \sin x/x$.

```
julia> using SymEngine
julia> @vars x; # определение x как символьной переменной
julia> f = x ^ 2 + x / 2 - sin(x) / x;
julia> diff(f, x)
1 / 2 + 2 * x + sin(x) / x ^ 2 - cos(x) / x
```

2.2. Производные нескольких переменных

Градиент является обобщением производной для многомерных функций. Он несет информацию о локальном наклоне функции, что позволяет нам предсказать эффект от небольшого шага от точки в любом направлении. Напомним, что производная является тангенсом угла наклона касательной. Градиент указывает в направлении наискорейшего роста касательной гиперплоскости, как показано на рис. 2.3. Гиперплоскость в n-мерном пространстве — это множество точек, удовлетворяющих уравнению

$$w_1 x_1 + ... + w_n x_n = b \tag{2.5}$$

для некоторого вектора $\mathbf{w}$ и скаляра b. Гиперплоскость имеет $n - 1$ измерений.

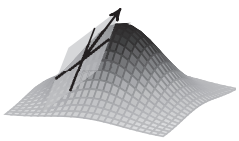


Рис. 2.3. Каждый компонент градиента определяет локальную касательную линию. Эти касательные линии определяют локальную касательную гиперплоскость. Вектор градиента указывает в направлении наибольшего увеличения функции

Градиент f в точке x записывается как $\nabla f(\mathbf{x})$ и является вектором. Каждый компонент этого вектора является частной производной[1] от f по этому компоненту:

$$\nabla f(\mathbf{x}) = \left[\frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, ..., \frac{\partial f(\mathbf{x})}{\partial x_n} \right]. \tag{2.6}$$

[1] *Частной производной* функции по переменной является производная, предполагающая, что все остальные входные переменные остаются постоянными. Обозначается как $\partial f/\partial x$.

Мы используем соглашение, по которому векторы, написанные через запятую, являются векторами-столбцами. Например, $[a, b, c] = [a\ b\ c]^{\mathrm{T}}$. В примере 2.2 показано, как вычислить градиент функции в конкретной точке.

Пример 2.2. Вычисление градиента в определенной точке

Вычислим градиент функции $f(\mathbf{x}) = x_1 \sin x_2 + 1$ в точке $\mathbf{c} = [2, 0]$.

$$\nabla f(\mathbf{x}) = \left[\frac{\partial f}{\partial x_1},\ \frac{\partial f}{\partial x_2}\right] = \left[\sin x_2,\ x_1 \cos x_2\right],$$
$$\nabla f(\mathbf{c}) = \left[0, 2\right].$$

Гессиан многомерной функции — это матрица, содержащая все вторые производные по входным переменным.[2] Вторые производные содержат информацию о локальной кривизне функции.

$$\nabla^2 f(\mathbf{x}) = \begin{pmatrix} \dfrac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_1} & \dfrac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_2} & \cdots & \dfrac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \dfrac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_1} & \dfrac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_2} & \cdots & \dfrac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_n} \end{pmatrix}. \tag{2.7}$$

Производная по направлению $\nabla_{\mathbf{s}} f(\mathbf{x})$ многомерной функции f — это мгновенная скорость изменения $f(\mathbf{x})$ при перемещении точки $\mathbf{x}$ со скоростью $\mathbf{s}$. Это определение тесно связано с определением производной одномерной функции:[3]

$$\begin{aligned} \nabla_{\mathbf{s}} f(\mathbf{x}) &= \underbrace{\lim_{h \to 0} \frac{f(\mathbf{x} + h\mathbf{s}) - f(\mathbf{x})}{h}}_{\text{Правая разность}} = \\ &= \underbrace{\lim_{h \to 0} \frac{f(\mathbf{x} + h\mathbf{s}/2) - f(\mathbf{x} - h\mathbf{s}/2)}{h}}_{\text{Центральная разность}} = \\ &= \underbrace{\lim_{h \to 0} \frac{f(\mathbf{x}) - f(\mathbf{x} - h\mathbf{s})}{h}}_{\text{Левая разность}}. \end{aligned} \tag{2.8}$$

Производную по направлению можно вычислить с помощью градиента функции:

[2] Гессиан симметричен, только если все вторые производные функции f непрерывны в окрестности заданной точки: $\dfrac{\partial^2 f}{\partial x_1 \partial x_2} = \dfrac{\partial^2 f}{\partial x_2 \partial x_1}$.

[3] В некоторых учебниках вектор $\mathbf{s}$ считается единичным (см., например, [154]).

$$\nabla_{\mathbf{s}} f(\mathbf{x}) = \nabla f(\mathbf{x})^{\mathrm{T}} \mathbf{s}. \tag{2.9}$$

Другой способ вычисления производной по направлению $\nabla_{\mathbf{s}} f(\mathbf{x})$ — определить функцию $g(\alpha) = f(\mathbf{x} + \alpha\mathbf{s})$, а затем вычислить $g'(0)$, как показано в примере 2.3.

Пример 2.3. Вычисление производной по направлению

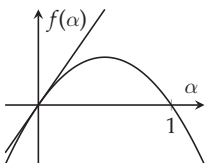
Мы хотим вычислить производную по направлению от функции $f(\mathbf{x}) = x_1 x_2$ в точке $\mathbf{x} = [1, 0]$ в направлении $\mathbf{s} = [-1, -1]$.

$$\nabla f(\mathbf{x}) = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}\right] = [x_2, x_1],$$

$$\nabla_{\mathbf{s}} f(\mathbf{x}) = \nabla f(\mathbf{x})^{\mathrm{T}} \mathbf{s} = \begin{bmatrix}0 & 1\end{bmatrix}\begin{bmatrix}-1\\-1\end{bmatrix} = -1.$$

Мы также можем вычислить производную по направлению следующим образом:

$$g(\alpha) = f(\mathbf{x} + \alpha\mathbf{s}) = (1 - \alpha)(-\alpha) = \alpha^2 - \alpha,$$
$$g'(\alpha) = 2\alpha - 1,$$
$$g'(0) = -1.$$



Производная по направлению принимает самое большое значение в направлении градиента и самая маленькое — в направлении, противоположном градиенту. Эта зависимость от направления возникает из скалярного произведения в определении производной по направлению и того факта, что градиент определяет локальную касательную гиперплоскость.

2.3. Численное дифференцирование

Численным дифференцированием называется процесс вычисления значения производной в точке. Эти значения могут быть по-разному получены, исходя из значений функции.

В этом разделе обсуждаются методы конечных разностей и комплексного шага.[4]

[4] Более полное рассмотрение тем, обсуждаемых в оставшейся части этой главы, см. в книге [62].

2.3.1. Методы конечных разностей

Как следует из названия, методы конечных разностей вычисляют разности между двумя значениями, вычисленными в точках, которые отличаются на величину конечного шага. Они аппроксимируют производные в определении (2.4) при небольшой разности:

$$f'(x) \approx \underbrace{\frac{f(x+h)-f(x)}{h}}_{\text{Правая разность}} \approx \\ \approx \underbrace{\frac{f(x+h/2)-f(x-h/2)}{h}}_{\text{Центральная разность}} \approx \underbrace{\frac{f(x)-f(x-h)}{h}}_{\text{Левая разность}}. \tag{2.10}$$

Теоретически, чем меньше размер шага h, тем точнее значение производной. Однако на практике слишком малые значения h могут привести к потере значимости. Этот эффект показан ниже на рис. 2.4. Алгоритм 2.1 обеспечивает реализации для этих методов.

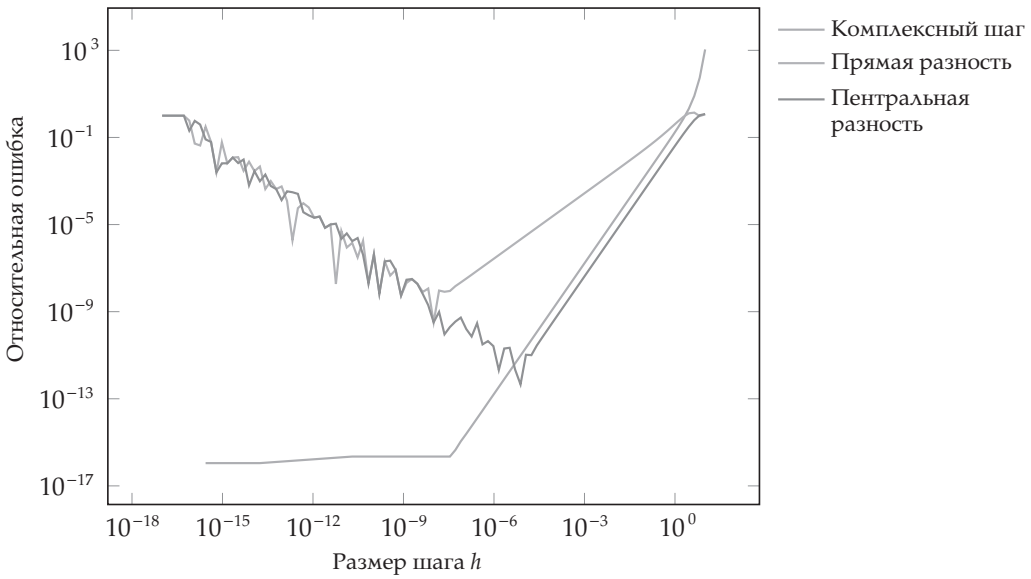


Рис. 2.4. Сравнение ошибок при вычислении производной функции $\sin x$ в точке $x = 1/2$ при изменении размера шага. Линейную погрешность метода правой разности и квадратичную погрешность метода центральной разности и комплексного шага можно оценить по постоянным наклонам кривых на правой стороне рисунка. Метод комплексного шага позволяет избежать ошибки вычитания, возникающей при вычитании двух значений функции, которые находятся близко друг к другу

Алгоритм 2.1. Конечно-разностные методы оценки производной функции `f` в точке `x` с конечной разностью `h`. Размеры шагов по умолчанию — квадратный или кубический корень из машинной точности для значений с плавающей запятой. Эти размеры шага уравновешивают машинную ошибку округления с ошибкой, связанной с размером шага. Функция `eps` обеспечивает размер шага между 1,0 и следующим бо́льшим представимым значением с плавающей точкой

```
diff_forward(f, x; h = sqrt(eps(Float64))) = (f(x + h) - f(x)) / h
diff_central(f, x; h = sqrt(eps(Float64))) = (f(x + h / 2) - f(x - h / 2)) / h
diff_backward(f, x; h = sqrt(eps(Float64))) = (f(x) - f(x - h)) / h
```

Методы конечных разностей могут быть получены с использованием разложения Тейлора. Мы получим правую разностную производную, начиная с разложения Тейлора f относительно x:

$$f(x+h) = f(x) + \frac{f'(x)}{1!}h + \frac{f''(x)}{2!}h^2 + \frac{f'''(x)}{3!}h^3 + \ldots \tag{2.11}$$

Мы можем выполнить перестановку слагаемых и выразить первую производную:

$$f'(x)h = f(x+h) - f(x) - \frac{f''(x)}{2!}h^2 - \frac{f'''(x)}{3!}h^3 - \ldots \tag{2.12}$$

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{f''(x)}{2!}h - \frac{f'''(x)}{3!}h^2 - \ldots \tag{2.13}$$

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}. \tag{2.14}$$

Прямая разность аппроксимирует истинную производную при малых h с ошибкой, зависящей от $\frac{f''(x)}{2!}h + \frac{f'''(x)}{3!}h^2 + \ldots$. Эта величина имеет порядок $O(h)$, т.е. погрешность правой разностной производной является линейной при приближении h к нулю.[5]

Метод центральной разности имеет погрешность $O(h^2)$ [103]. Мы можем вывести эту погрешность, используя разложение Тейлора в окрестности точки x для $f(x + h/2)$ и $f(x - h/2)$:

$$f(x+h/2) = f(x) + f'(x)\frac{h}{2} + \frac{f''(x)}{2!}\left(\frac{h}{2}\right)^2 + \frac{f'''(x)}{3!}\left(\frac{h}{2}\right)^3 + \ldots \tag{2.15}$$

$$f(x-h/2) = f(x) - f'(x)\frac{h}{2} + \frac{f''(x)}{2!}\left(\frac{h}{2}\right)^2 - \frac{f'''(x)}{3!}\left(\frac{h}{2}\right)^3 + \ldots \tag{2.16}$$

[5] Асимптотические обозначения объясняются в приложении В.

Вычитание этих разложений дает:

$$f\left(x+h/2\right)-f\left(x-h/2\right)\approx 2f'\left(x\right)\frac{h}{2}+\frac{2}{3!}f'''\left(x\right)\left(\frac{h}{2}\right)^3. \tag{2.17}$$

Выражая $f'(x)$, получаем формулу

$$f'\left(x\right)\approx\frac{f\left(x+h/2\right)-f\left(x-h/2\right)}{h}-\frac{f'''\left(x\right)h^2}{24}, \tag{2.18}$$

которая означает, что это приближение имеет квадратичную ошибку.

2.3.2. Метод комплексного шага

Мы часто сталкиваемся с проблемой необходимости выбора размера шага h, достаточно малого, чтобы обеспечить хорошее приближение, но не слишком маленького, чтобы привести к потери значащих цифр при вычитании близких чисел. *Метод комплексного шага* позволяет избежать потери значащих цифр при вычитании близких чисел за счет однократного вычисления функции. Мы вычисляем функцию только один раз, делая шаг в мнимом направлении [102].[6]

Разложение Тейлора с мнимым шагом имеет следующий вид:

$$f\left(x+ih\right)=f\left(x\right)+ihf'\left(x\right)-h^2\frac{f''\left(x\right)}{2!}-ih^3\frac{f'''\left(x\right)}{3!}+\ldots. \tag{2.19}$$

Сравнивая мнимые компоненты левой и правой части, мы получаем приближение производной:

$$\mathrm{Im}\left(f\left(x+ih\right)\right)=hf'\left(x\right)-h^3\frac{f'''\left(x\right)}{3!}+\ldots\Rightarrow \tag{2.20}$$

$$\Rightarrow f'\left(x\right)=\frac{\mathrm{Im}\left(f\left(x+ih\right)\right)}{h}+h^2\frac{f'''\left(x\right)}{3!}-\ldots= \tag{2.21}$$

$$=\frac{\mathrm{Im}\,f\left(x+ih\right)}{h}+O\left(h^2\right)\ \text{при}\ h\to 0. \tag{2.22}$$

Реализация обеспечивается алгоритмом 2.2. Действительная часть $f(x)$ приближается с точностью $O(h^2)$ при $h \to 0$:

$$\mathrm{Re}\left(f\left(x+ih\right)\right)=f\left(x\right)-h^2\frac{f''\left(x\right)}{2!}+\ldots\Rightarrow \tag{2.23}$$

$$\Rightarrow f\left(x\right)=\mathrm{Re}\left(f\left(x+ih\right)\right)+h^2\frac{f''\left(x\right)}{2!}-\ldots. \tag{2.24}$$

---

[6] При этом следует обратить особое внимание на то, чтобы реализация функция f поддерживала работу с комплексными числами.

Таким образом, мы можем вычислить как $f(x)$, так и $f'(x)$, используя одно вычисление функции f с комплексными аргументами. В примере 2.4 показаны вычисления, используемые для оценки производной функции в определенной точке. Метод комплексного шага реализует алгоритм 2.2. На рис. 2.4 показаны результаты сравнения численной ошибки метода комплексного шага с методами правой и центральной разностной производной при изменении размера шага.

Алгоритм 2.2. Метод комплексного шага для вычисления производной функции f в точке x с конечной разностью h

```
diff_complex(f, x; h = 1e-20) = imag(f(x + h*im))/h
```

Пример 2.4. Метод комплексного шага для вычисления производных

Рассмотрим $f(x) = \sin x^2$. Значение функции при $x = \pi/2$ составляет приблизительно 0,624266, а производная равна $\pi\cos(\pi^2/4) \approx -2{,}44525$. Мы можем получить этот результат, используя метод комплексного шага:

```
julia> f = x -> sin(x ^ 2);
julia> v = f(π / 2 + 0.001im);
julia> real(v) # f(x)
0,6242698144866649
julia> imag(v) / 0.001 # f'(x)
-2.4542516170381785
```

2.4. Автоматическое дифференцирование

В этом разделе представлены алгоритмы для численного вычисления производных функций, заданных компьютерной программой. Ключом к этим методам автоматического дифференцирования является применение правила дифференцирования сложной функции:

$$\frac{d}{dx} f\left(g\left(x\right)\right) = \frac{d}{dx}\left(f \circ g\right)\left(x\right) = \frac{df}{dg}\frac{dg}{dx}. \tag{2.25}$$

Программа состоит из элементарных операций, таких как сложение, вычитание, умножение и деление.

Рассмотрим функцию $f(a, b) = \ln(ab + \max(a, 2))$. Если мы хотим вычислить частную производную по a в некоторой точке, то должны несколько раз применить правило дифференцирования сложной функции:[7]

[7] Мы принимаем соглашение, что значение логических выражений, таких как $(2 < a)$, равны единице, если они являются истинными, и нулю, если ложными.

$$\frac{\partial f}{\partial a} = \frac{\partial}{\partial a}\ln\left(ab + \max\left(a,\, 2\right)\right) = \tag{2.26}$$

$$= \frac{1}{ab + \max\left(a,\, 2\right)} \frac{\partial}{\partial a}\left(ab + \max\left(a,\, 2\right)\right) = \tag{2.27}$$

$$= \frac{1}{ab + \max\left(a,\, 2\right)} \left[\frac{\partial\left(ab\right)}{\partial a} + \frac{\partial \max\left(a,\, 2\right)}{\partial a}\right] = \tag{2.28}$$

$$= \frac{1}{ab + \max\left(a,\, 2\right)} \left[\left(b\frac{\partial a}{\partial a} + a\frac{\partial b}{\partial a}\right) + \left(\left(2 > a\right)\frac{\partial 2}{\partial a} + \left(2 < a\right)\frac{\partial a}{\partial a}\right)\right] = \tag{2.29}$$

$$= \frac{1}{ab + \max\left(a,\, 2\right)} \left[b + \left(2 < a\right)\right]. \tag{2.30}$$

Этот процесс можно автоматизировать с помощью *вычислительного графа*. Вычислительный граф представляет собой функцию, в которой узлы являются операциями, а ребра — отношениями ввода-вывода. Листовые узлы вычислительного графа соответствуют входным переменным или константам, а терминальные узлы — значениям функции. Пример вычислительного графа показан на рис. 2.5.

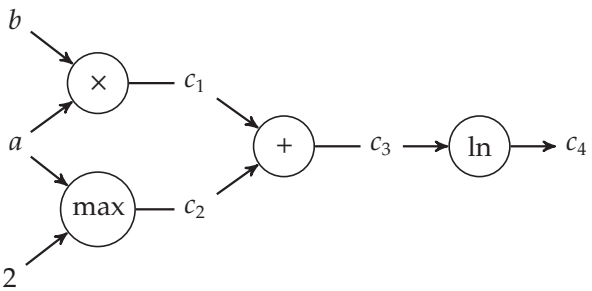


Рис. 2.5. Вычислительный граф для $\ln(ab + \max(a, 2))$

Существует два метода автоматического дифференцирования функции f с использованием ее вычислительного графа. *Метод прямого аккумулирования*, используемый для работы с дуальными числами, перемещается по дереву от входов к выходам, тогда как *метод обратного аккумулирования* выполняет обратный проход через граф.

2.4.1. Метод прямого аккумулирования

Метод прямого аккумулирования (forward accumulation) автоматически дифференцирует функцию с помощью одного прямого прохода по ее вычислительному графу. Этот метод эквивалентен итеративному расширению правила дифференцирования сложных функций относительно внутренней операции:

$$\frac{df}{dx} = \frac{df}{dc_4}\frac{dc_4}{dx} = \frac{df}{dc_4}\left(\frac{dc_4}{dc_3}\frac{dc_3}{dx}\right) = \frac{df}{dc_4}\left(\frac{dc_4}{dc_3}\left(\frac{dc_3}{dc_2}\frac{dc_2}{dx} + \frac{dc_3}{dc_1}\frac{dc_1}{dx}\right)\right). \qquad (2.31)$$

Чтобы проиллюстрировать метод прямого аккумулирования, мы применим его к функции $f(a, b) = \ln(ab + \max(a, 2))$ для вычисления частной производной по a при $a = 3$, $b = 2$.

1. Процедура начинается с исходных узлов графа, состоящих из входных данных функции и любых постоянных значений. Для каждого из этих узлов отмечаем как значение, так и частную производную по целевой переменной, как показано на рисунке 2.6.[8]
2. Затем переходим вниз по дереву, по одному узлу за раз, выбирая в качестве следующего узла тот, в котором аргументы уже были вычислены. Мы можем вычислить значение, пройдя через значения предшествующих ему узлов, и вычислить частную производную по a, используя как значения предыдущих узлов, так и значения их частных производных. Вычисления продемонстрированы на рис. 2.7.

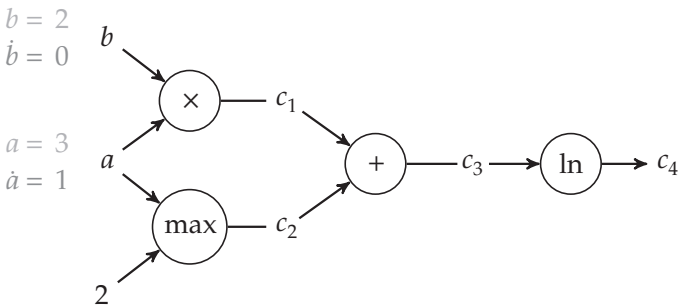


Рис. 2.6. Вычислительный граф для функции $\ln(ab + \max(a, 2))$ и его начальное состояние при вычислении $\partial f/\partial a$ при $a = 3$ и $b = 2$ методом прямого аккумулирования

В итоге мы получим правильный результат: $f(3, 2) = \ln 9$ и $\partial f/\partial a = 1/3$. Мы получили его, используя один проход через вычислительный граф.

Этот процесс может быть удобно автоматизирован с помощью компьютера, использующего язык программирования, который переопределяет каждую операцию для получения как значения, так и его производной. Такие пары называются *дуальными числами*.

---

[8] Для компактности на этом рисунке мы используем *точечное*, или *ньютоновское,* обозначение производной. Например, если ясно, что мы берем производную по a, можем записать $\partial b/\partial a$ как $\dot{b}$.

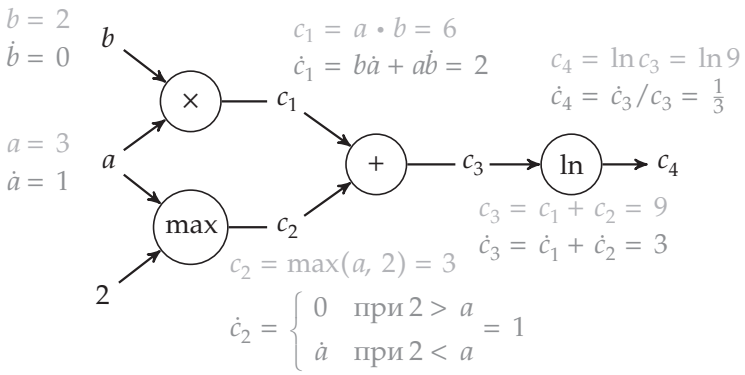


Рис. 2.7. Вычислительный граф для функции $\ln(ab + \max(a, 2))$ и результаты вычисления $\partial f/\partial a$ при $a = 3$ и $b = 2$ методом прямого аккумулирования

Дуальные числа могут быть выражены математически с помощью абстрактной величины ε, где ε^2 по определению равно нулю. Как и комплексное число, дуальное число записывается как $a + b\varepsilon$. где a и b — действительные числа. В таком случае выполняются следующие тождества.

$$(a + b\varepsilon) + (c + d\varepsilon) = (a + c) + (b + d)\varepsilon. \tag{2.32}$$

$$(a + b\varepsilon) \cdot (c + d\varepsilon) = (ac) + (ad + bc)\varepsilon. \tag{2.33}$$

Фактически, передавая дуальное число в любую гладкую функцию f, мы получаем значение этой функции и ее производной. Можно показать это, используя ряд Тейлора:

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k, \tag{2.34}$$

$$f(a+b\varepsilon) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(a+b\varepsilon-a)^k = \tag{2.35}$$

$$= \sum_{k=0}^{\infty} \frac{f^{(k)}(a)b^k\varepsilon^k}{k!} = \tag{2.36}$$

$$= f(a) + bf'(a)\varepsilon + \varepsilon^2 \sum_{k=2}^{\infty} \frac{f^{(k)}(a)b^k\varepsilon^{k-2}}{k!} = \tag{2.37}$$

$$= f(a) + bf'(a)\varepsilon. \tag{2.38}$$

Реализация этих вычислений показана в примере 2.5.

Пример 2.5. Реализация дуальных чисел с автоматическим прямым аккумулированием

Дуальные числа могут быть реализованы путем определения структуры `Dual`, которая содержит дуальные числа, значение **v** и производную ∂.

```
struct Dual
    v
    ∂
end
```

Затем мы должны реализовать методы для каждой базовой операции. Эти методы получают дуальные числа и создают новые дуальные числа, используя логику правила дифференцирования сложной функции.

```
Base. :+ (a :: Dual, b :: Dual) = Dual(a.v + b.v, a.∂ + b.∂)
Base. :* (a :: Dual, b :: Dual) = Dual(a.v * b.v, a.v * b.∂ + b.v * a.∂)
Base.log(a :: Dual) = Dual(log(a.v), a.∂ / a.v)
function Base.max(a :: Dual, b :: Dual)
     v = max(a.v, b.v)
     ∂ = a.v > b.v ? a.∂ : a.v < b.v ? b.∂ : NaN
     return Dual(v, ∂)
end
function Base.max(a :: Dual, b :: Int)
      v = max(a.v, b)
      ∂ = a.v > b ? a.∂ : a.v < b ? 0: NaN
      return Dual(v, ∂)
end
```

Пакет **`ForwardDiff.jl`** поддерживает обширный набор математических операций и дополнительно обеспечивает вычисление градиентов и гессиана.

```
julia> using ForwardDiff
julia> a = ForwardDiff.Dual(3, 1);
julia> b = ForwardDiff.Dual(2, 0);
julia> log(a * b + max(a, 2))
Dual{Void}(2.1972245773362196, 0.3333333333333333)
```

2.4.2. Метод обратного аккумулирования

Метод прямого аккумулирования требует n проходов для вычисления n-мерного градиента. *Обратное акукумулирование* [98] вычисляет полный градиент за один раз, но требует двух проходов по графу: *прямого прохода*, во время которого вычисляются необходимые промежуточные значения, и *обратного прохода*, во

время которого вычисляется градиент. При вычислении градиентов обратное аккумулирование часто предпочтительнее прямого, хотя необходимо соблюдать осторожность в системах с ограниченным объемом памяти, когда вычислительный граф становится очень большим.[9]

Как и метод прямого аккумулирования, метод обратного аккумулирования вычисляет частную производную по выбранной переменной, но при этом итеративно подставляет внешнюю функцию:

$$\frac{df}{dx}=\frac{df}{dc_4}\frac{dc_4}{dx}=\left(\frac{df}{dc_3}\frac{dc_3}{dc_4}\right)\frac{dc_4}{dx}=\left(\left(\frac{df}{dc_2}\frac{dc_2}{dc_3}+\frac{df}{dc_1}\frac{dc_1}{dc_3}\right)\frac{dc_3}{dc_4}\right)\frac{dc_4}{dx}. \qquad (2.39)$$

Процесс выполняется во время обратного обхода графа, и для вычислений ему требуются промежуточные значения, полученные во время прямого прохода.

Метод обратного аккумулирования может быть реализован посредством перегрузки операций[10] аналогично тому, как дуальные числа используются для реализации метод прямого аккумулирования. Для каждой основной операции должны быть реализованы две функции: прямая операция, которая перегружает операцию по сохранению локальной информации о градиенте во время прямого прохода, и обратная операция, которая использует эту информацию для распространения градиента в обратном направлении. Такие пакеты, как **`TensorFlow`**[11] или **`AutoDiffSource.jl`**, могут автоматически создавать вычислительные графы и связанные с ними прямые и обратные операции из элементарных операций. Пример 2.6 демонстрирует, как можно использовать пакет **`AutoDiffSource.jl`**.

Пример 2.6. Пакет **`AutoDiffSource.jl`** обеспечивает автоматическое дифференцирование с помощью метода обратного аккумулирования. Макрос **`@δ`** создает как обычную функцию **`f`**, так и ее автоматически сгенерированную производную **`δf`**.

```
julia> using AutoDiffSource
julia> @δ f(a, b) = log(a * b + max(a, 2));
julia> y, ∇ = δf(3, 2);
julia> y
2.1972245773362196
```

[9] Метод обратного аккумулирования является центральным в алгоритме обратного распространения ошибки, используемом для обучения нейронных сетей [134].

[10] Перегрузка операций означает переопределение реализаций общих операций, таких как + или = для пользовательских типов переменных. Перегрузка обсуждается в приложении А.2.5.

[11] **`TensorFlow`** — это библиотека программ с открытым исходным кодом для вычислений с использованием графов потоков данных, которая часто используется для приложений, связанных с глубоким обучением. Ее можно получить на сайте **`tensorflow.org`**.

```
julia> ∂a, ∂b = ∇()
(0.3333333333333333, 0.3333333333333333)
```

2.5. Резюме

- Производные полезны при оптимизации, поскольку они предоставляют информацию о том, как изменить заданную точку, чтобы улучшить целевую функцию.
- Для многомерных функций различные концепции на основе производных полезны для выбора направления поиска оптимума, включая градиент, гессиан и производную по направлению.
- Один из подходов к численному дифференцированию включает конечно-разностные аппроксимации.
- Метод комплексных шагов может устранить влияние ошибки вычитания при выполнении небольших шагов, что приводит к получению значений градиента высокой точности.
- Методы аналитического дифференцирования включают прямое и обратное аккумулирование на вычислительных графах.

2.6. Упражнения

Упражнение 2.1. Примените метод правой разности для аппроксимации гессиана $f(\mathbf{x})$, используя его градиент $\nabla f(\mathbf{x})$.

Упражнение 2.2. В чем недостаток метода центральных разностей по сравнению с другими методами конечных разностей, если мы уже знаем $f(x)$?

Упражнение 2.3. Вычислите градиент функции $f(x) = \ln x + e^x + 1/x$ в точке x, близкой к нулю. Какое слагаемое доминирует в выражении?

Упражнение 2.4. Предположим, что $f(x)$ является действительной функцией, которая также определена для комплексных входных данных. Если $f(3 + ih) = 2 + 4ih$, чему равно значение $f'(3)$?

Упражнение 2.5. Нарисуйте вычислительный граф для функции $f(x, y) = \sin(x + y^2)$. Используйте вычислительный граф с прямым аккумулированием для вычисления $\partial f/\partial y$ в точке $(x, y) = (1, 1)$. Разметьте промежуточные значения и частные производные по мере их распространения по графу.

Упражнение 2.6. Объедините методы левой и правой разности, чтобы получить разностный метод для оценки производной второго порядка функции f в точке x, используя три вычисления значения функции.

Глава 3

Метод интервалов

В этой главе представлены различные варианты *метода интервалов* (bracketing) для функций одной переменной. Метод интервалов — это процесс определения интервала, в котором лежит точка локального минимума, и его последовательного сжатия. Для многих функций при выборе направления поиска оптимального значения может оказаться полезной информация о ее производной, но для некоторых функций эта информация может быть недоступной. В этой главе описывается широкий спектр подходов, в которых используются различные допущения. Последующие главы, в которых рассматривается многомерная оптимизация, будут основываться на представленных здесь концепциях.

3.1. Одномодальность

Некоторые из алгоритмов, представленных в этой главе, предполагают *одномодальность* целевой функции. *Одномодальная функция* f — это функция, у которой существует единственное значение x^*, такое что f монотонно убывает при $x \leq x^*$ и монотонно возрастает при $x \geq x^*$. Из этого определения следует, что точка x^* является точкой единственного глобального минимума, а других локальных минимумов нет.[1]

Для одномодальной функции можно найти *интервал* (a, c), содержащий глобальный минимум, если мы можем найти три такие точки $a < b < c$, что $f(a) > f(b) < f(c)$. Пример показан на рис. 3.1.

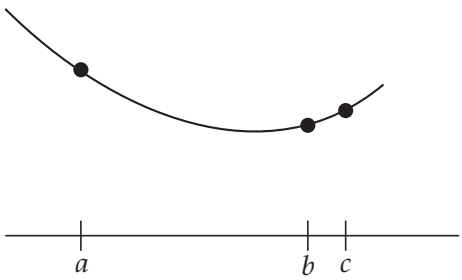


Рис. 3.1. Три точки, задающие интервалы, содержащие минимум

[1] Возможно, обычно одномодальные функции определяют иначе, как функции, у которых существует уникальный глобальный максимум, а не минимум. Тем не менее в этой книге мы пытаемся минимизировать функции и поэтому используем именно такое определение.

3.2. Нахождение начального интервала

При оптимизации функции мы часто сначала ищем интервал, содержащий локальный минимум. Затем последовательно уменьшаем размер этого интервала, чтобы приблизиться к точке локального минимума. Для поиска начального интервала можно использовать простую процедуру (алгоритм 3.1). Начиная с заданной точки, мы делаем шаг в положительном направлении. Выбранное расстояние является *гиперпараметром* этого алгоритма[2], но описанный алгоритм имеет значение по умолчанию $1 \cdot 10^{-2}$. Затем мы перемещаемся в направлении спуска, чтобы найти новую точку, в которой значение функции больше минимума. С каждым шагом мы увеличиваем размер шага на некоторый коэффициент, который является еще одним гиперпараметром этого алгоритма и часто равен двум. Пример показан на рис. 3.2. Функции, не имеющие локальных минимумов, такие как $\exp(x)$, не допускают применения метода интервалов, и алгоритм **`brac_minimum`** для них использовать нельзя.

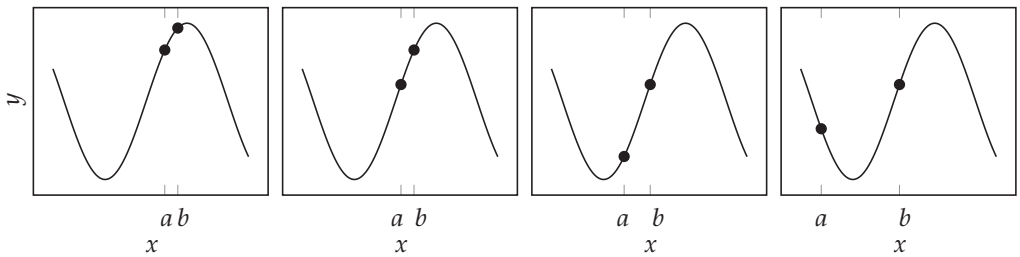


Рис. 3.2. Пример применения алгоритма **`brac_minimum`** к функции. Метод меняет направление поиска между первой и второй итерациями, а затем выполняет расширение до тех пор, пока на четвертой итерации не будет найден интервал, содержащий минимум

Алгоритм 3.1. Метод поиска интервала, в котором должна лежать точка локального минимума. Он принимает в качестве входных данных одномерную функцию **`f`** и начальную позицию **`x`**, которая по умолчанию равна **`0`**. Можно указать размер **`s`** начального шага и коэффициент расширения **`k`**. Возвращает кортеж, содержащий новый интервал $[a, b]$

```
function bracket_minimum(f, x = 0; s = 1e-2, k = 2.0)
    a, ya = x, f(x)
```

[2] Гиперпараметр — это параметр, который управляет работой алгоритма. Он может быть задан экспертом или настроен с использованием алгоритма оптимизации. Многие из алгоритмов в этом тексте имеют гиперпараметры. Мы часто предоставляем значения по умолчанию, предложенные в литературе. Успех алгоритма может сильно зависеть от выбора гиперпараметра.

```
    b, yb = a + s, f(a + s)
    if yb > ya
       a, b = b, a
       ya, yb = yb, ya
       s = -s
    end
    while true
        c, yc = b + s, f(b + s)
        if yc > yb
           return a < c ? (a, c) : (c, a)
        end
        a, ya, b, yb = b, yb, c, yc
        s *= k
    end
end
```

3.3. Алгоритм поиска Фибоначчи

Предположим, что у нас есть одномодальная функция f, заданная на интервале $[a, b]$. *Алгоритм поиска Фибоначчи* (алгоритм 3.2) гарантированно максимально сжимает интервал при ограниченном количестве обращений к целевой функции.

Предположим, что мы можем вычислить функцию f только дважды. Вычислив f для точек, соответствующих одной трети и двум третям интервала, мы гарантированно удалим одну треть нашего интервала, какой бы ни была функция f, как показано на рис. 3.3.

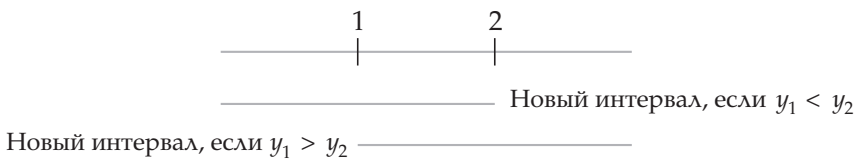


Рис. 3.3. Первоначальное предположение о двух обращениях удаляет одну треть начального интервала

Мы можем гарантировать более узкий интервал, смещая наши приближения к центру. В пределе при $\varepsilon \to 0$ мы гарантированно сожмем первоначальный интервал с коэффициентом, равным двум, как показано на рис. 3.4.

С помощью трех обращений мы можем сократить интервал в три раза. Сначала вычисляем функцию f в точках одной и двух третей на интервале, отбрасываем одну треть интервала, а затем выбираем точку, расположенную справа от той, которая ближе к минимуму, как показано на рис. 3.5.

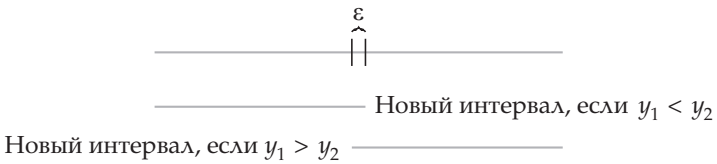


Рис. 3.4. Максимум, что можно гарантировать, — это сократить интервал чуть менее чем в два раза

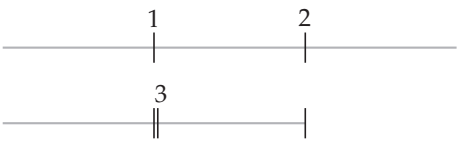


Рис. 3.5. С помощью трех запросов можно уменьшить область происка в три раза. Третий запрос сделан на основе результатов первых двух запросов

При n обращениях к функции длины интервалов образуют последовательность Фибоначчи: 1, 1, 2, 3, 5, 8 и т.д. Первые два члена этой последовательности равны единице, а следующие члены всегда являются суммой двух предыдущих:

$$F_n = \begin{cases} 1, & \text{если } n \le 2, \\ F_{n-1} + F_{n-2} & \text{в противном случае.} \end{cases} \tag{3.1}$$

Отношения между интервалами показаны на рис. 3.6. Применение этого алгоритма к функции одной переменной иллюстрирует пример 3.1.

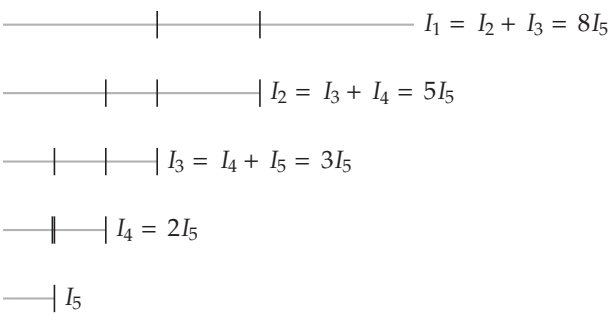


Рис. 3.6. После n обращений к функции мы гарантированно сократим наш интервал в F_{n+1} раз. Длина каждого интервала, построенного алгоритмом поиска Фибоначчи, может быть выражена через длину последнего интервала, умноженную на число Фибоначчи. Если последний, наименьший интервал имеет длину I_n, то второй наименьший интервал имеет длину $I_{n-1} = F_3 I_n$, третий наименьший интервал имеет длину $I_{n-2} = F_4 I_n$ и т.д.

Последовательность Фибоначчи можно определить аналитически по *формуле Бине* (Binet):

$$F_n = \frac{\varphi^n - (1-\varphi)^n}{\sqrt{5}}, \tag{3.2}$$

где $\varphi = \left(1+\sqrt{5}\right)\Big/2 \approx 1,61803$ — это *золотое сечение*.

Соотношение между соседними значениями в последовательности Фибоначчи задается формулой:

$$\frac{F_n}{F_{n-1}} = \varphi\frac{1-s^{n+1}}{1-s^n}, \tag{3.3}$$

где $s = \left(1-\sqrt{5}\right)\Big/\left(1+\sqrt{5}\right) \approx -0,382.$

Алгоритм 3.2. Алгоритм поиска Фибоначчи для одномерной функции **f**, заданной на интервале [*a*, *b*] для *n* > 1 обращений к функции. Возвращает новый интервал (**a**, **b**). Дополнительный параметр **ϵ** контролирует длину интервала самого низкого уровня

```
function fibonacci_search(f, a, b, n; ϵ = 0.01)
   s = (1 - √5) / ( 1 + √5)
   ρ =1 / (φ * (1 - s ^ (n + 1)) / (1 - s ^ n))
   d = ρ * b + (1 - ρ) * a
   yd = f(d)
   for i in 1 : n - 1
        if i == n - 1
           c = ϵ * a + (1 - ϵ) * d
        else
           c = ρ * a + (1 - ρ) * b
        end
        yc = f(c)
        if yc < yd
            b, d, yd = d, c, yc
        else
            a, b = b, c
        end
        ρ = 1 / (φ * (1 - s ^ (n - i + 1)) / (1 - s ^ (n - i)))
   end
   return a < b ? (a, b) : (b, a)
end
```

3.4. Метод золотого сечения

Если мы возьмем предел для больших значений n, то увидим, что отношение между соседними значениями последовательности Фибоначчи стремится к золотому сечению:

$$\lim_{n\to\infty}\frac{F_n}{F_{n-1}}=\varphi. \tag{3.4}$$

Метод золотого сечения (алгоритм 3.3) использует золотое сечение для аппроксимации метода поиска Фибоначчи. На рис. 3.7 показана взаимосвязь между интервалами. На рис. 3.8 и 3.9 показаны результаты сравнения метода Фибоначчи с методом золотого сечения для одномодальных и неодномодальных функций соответственно.

Пример 3.1. Использование метода поиска Фибоначчи с пятью обращениями к функции для оптимизации одномерной функции

Рассмотрим метод поиска Фибоначчи с пятью обращениями к функции на примере минимизации функции $f(x) = \exp(x - 2) - x$ на интервале $[a, b] = [-2, 6]$. Первые два вычисления функции выполняются в точках $\frac{F_5}{F_6}$ и $1-\frac{F_5}{F_6}$ начального отрезка:

$$f\left(x^{(1)}\right)=f\left(a+(b-a)\left(1-\frac{F_5}{F_6}\right)\right)=f(1)=-0,632,$$

$$f\left(x^{(2)}\right)=f\left(a+(b-a)\frac{F_5}{F_6}\right)\qquad=f(3)=-0,282.$$

Значение функции в точке $x^{(1)}$ меньше, чем в точке $x^{(2)}$, следовательно, новый интервал $[a, b]$ равен $[-2, 3]$. Для следующего разделения интервала необходимы два значения:

$$x_{\text{left}}=a+(b-a)\left(1-\frac{F_4}{F_5}\right)=0,$$

$$x_{\text{right}}=a+(b-a)\frac{F_4}{F_5}=1.$$

Таким образом, третье обращение к функции выполняется в точке x_{left}, поскольку значение x_{right} уже было вычислено.

$$f(x^{(3)}) = f(0) = 0,135.$$

Значение функции в точке $x^{(1)}$ меньше, что дает новый интервал $[a, b] = [0, 3]$. Для следующего разделения интервала необходимы два значения:

$$x_{\text{left}} = a + (b-a)\left(1 - \frac{F_3}{F_4}\right) = 1,$$

$$x_{\text{right}} = a + (b-a)\frac{F_3}{F_4} = 2.$$

Таким образом, четвертое вычисление функции выполняется в правильном направлении, поскольку значение в точке x_{left} уже было вычислено:

$$f\left(x^{(4)}\right) = f(2) = -1.$$

Новый интервал $[a, b] = [1, 3]$. Последнее вычисление выполняется в непосредственной близости от центра интервала в точке $2 + \varepsilon$, и оказалось, что она имеет немного большее значение, чем $f^{(2)}$. Последний интервал равен интервалу $[1, 2 + \varepsilon]$.

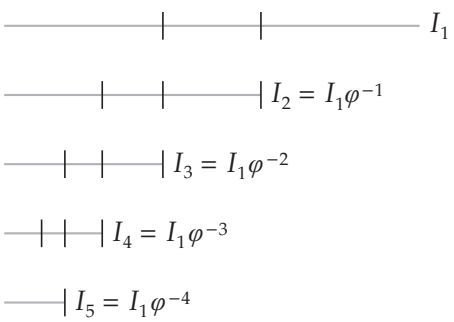


Рис. 3.7. При n вычислениях одномерной функции мы гарантируем сжатие интервала с коэффициентом φ^{n-1}

Алгоритм 3.3. Метод золотого сечения для функции `f` одной переменной на интервале $[a, b]$ при $n > 1$ вычислениях функции. Возвращает новый интервал `(a, b)`. В языке Julia золотое сечение `φ` уже определено по умолчанию. Гарантия сходимости с точностью до `ε` требует $n = (b - a)/(\varepsilon \ln \varphi)$ итераций

```
function golden_section_search(f, a, b, n)
    ρ = φ - 1
    d = ρ * b + (1 - ρ) * a
    yd = f(d)
    for i = 1 : n - 1
        c = ρ * a + (1 - ρ) * b
```

```
        yc = f(c)
        if yc < yd
            b, d, yd = d, c, yc
        else
            a, b = b, c
        end
    end
    return a < b ? (a, b) : (b, a)
end
```

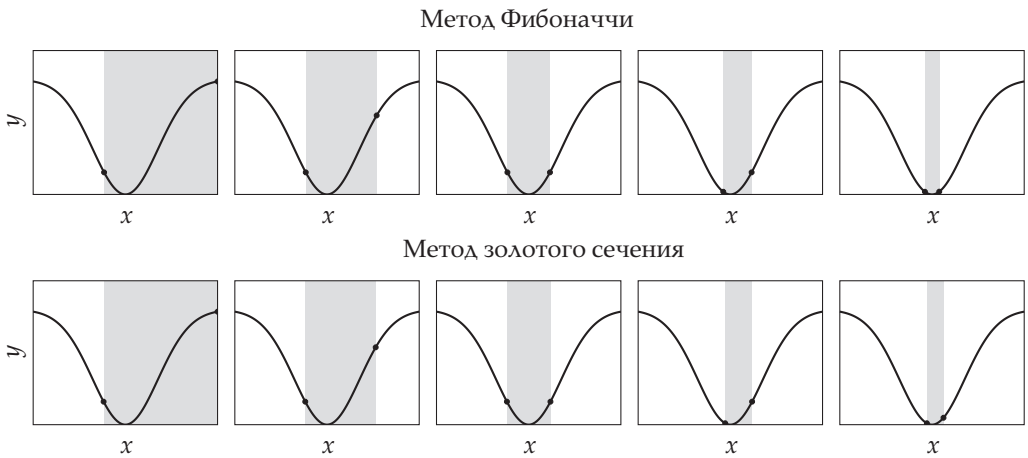


Рис. 3.8. Методы Фибоначчи и золотого сечения для одномодальной функции

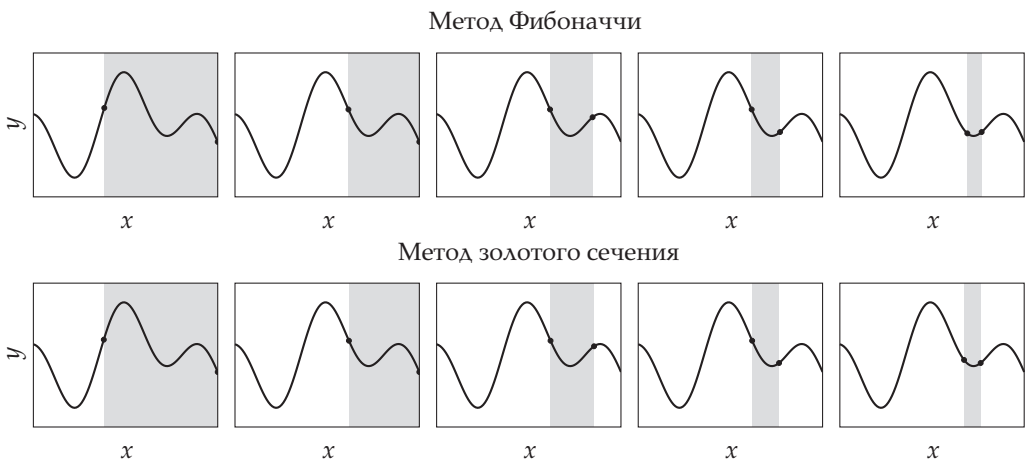


Рис. 3.9. Методы Фибоначчи и золотого сечения для неодномодальной функции

3.5. Метод квадратичной аппроксимации

Метод квадратичной аппроксимации использует возможность аналитически вычислять точку минимума квадратичной функции. Многие локальные минимумы выглядят квадратично, если мы приближаемся достаточно близко. Метод квадратичной аппроксимации итеративно приближает квадратичную функцию по трем точкам методом интервалов, находит точку минимума, выбирает новый набор точек для метода интервалов и повторяет итерацию, как показано на рис. 3.10.

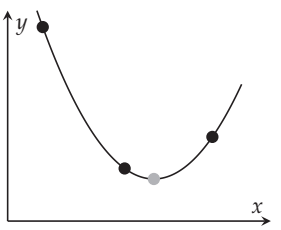


Рис. 3.10. Метод квадратичной аппроксимации приближает квадратичную функцию по трем точкам методом интервалов (черные точки) и использует аналитическое выражения для точки минимума (синяя точка) для определения следующего набора точек для метода интервалов

Для заданных точек метода интервалов $a < b < c$ мы хотим найти коэффициенты p_1, p_2 и p_3 для квадратичной функции q, которая проходит через точки (a, y_a), (b, y_b) и (c, y_c):

$$q(x) = p_1 + p_2 x + p_3 x^2, \tag{3.5}$$

$$y_a = p_1 + p_2 a + p_3 a^2, \tag{3.6}$$

$$y_b = p_1 + p_2 b + p_3 b^2, \tag{3.7}$$

$$y_c = p_1 + p_2 c + p_3 c^2. \tag{3.8}$$

В матричной форме имеем

$$\begin{bmatrix} y_a \\ y_b \\ y_c \end{bmatrix} = \begin{bmatrix} 1 & a & a^2 \\ 1 & b & b^2 \\ 1 & c & c^2 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix}. \tag{3.9}$$

Можем найти коэффициенты, обращая матрицу:

$$\begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} 1 & a & a^2 \\ 1 & b & b^2 \\ 1 & c & c^2 \end{bmatrix}^{-1} \begin{bmatrix} y_a \\ y_b \\ y_c \end{bmatrix}. \tag{3.10}$$

Тогда квадратичная функция примет вид

$$q(x) = y_a \frac{(x-b)(x-c)}{(a-b)(a-c)} + y_b \frac{(x-a)(x-c)}{(b-a)(b-c)} + y_c \frac{(x-a)(x-b)}{(c-a)(c-b)}. \quad (3.11)$$

Можем найти точку единственного минимума, найдя точку, в которой производная равна нулю:

$$x^* = \frac{1}{2} \frac{y_a\left(b^2-c^2\right) + y_b\left(c^2-a^2\right) + y_c\left(a^2-b^2\right)}{y_a(b-c) + y_b(c-a) + y_c(a-b)}. \quad (3.12)$$

Метод квадратичной аппроксимации обычно работает быстрее, чем метод золотого сечения. Может потребоваться особая предосторожность в случаях, когда следующая точка очень близка к другим точкам. Основная реализация представлена в алгоритме 3.4. На рис. 3.11 показано несколько итераций алгоритма.

Алгоритм 3.4. Метод квадратичной аппроксимации для одномерной функции **f**, определенной на интервале $[a, c]$, где $a < b < c$. Метод использует n вычислений функции. Он возвращает новые концы интервала в виде кортежа (**a**, **b**, **c**)

```
function quadratic_fit_search(f, a, b, c, n)
    ya, yb, yc = f(a), f(b), f(c)
    for i in 1 : n - 3
        x = 0.5 * (ya * (b ^ 2 - c ^ 2) + yb * (c ^ 2 - a ^ 2)
                + yc * (a ^ 2 - b ^ 2)) / (ya * (b - c) + yb * (c - a)
                + yc * (a - b))
        yx = f(x)
        if x > b
            if yx > yb
                c, yc = x, yx
            else
                a, ya, b, yb = b, yb, x, yx
            end
        elseif x < b
            if yx > yb
                a, ya = x, yx
            else
                c, yc, b, yb = b, yb, x, yx
            end
        end
    end
    return (a, b, c)
end
```

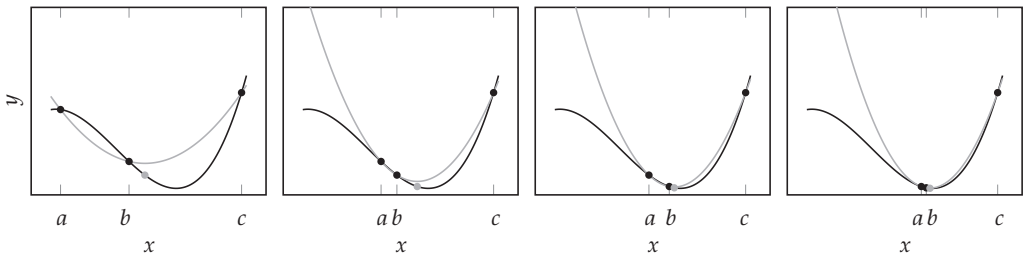


Рис. 3.11. Четыре итерации метода квадратичной аппроксимации

3.6. Метод Шуберта–Пиявского

В отличие от предыдущих методов в этой главе, *метод Шуберта–Пиявского* [121, 142] является *методом глобальной оптимизации* в области $[a, b]$, что означает, что он гарантированно сходится к точке глобального минимума функции независимо от количества локальных минимумов или одномодальности функции. Базовая реализация описывается алгоритмом 3.5.

Метод Шуберта–Пиявского требует, чтобы функция была *липшиц-непрерывной*, а это означает, что она непрерывна и существует верхняя граница величины ее производной. Функция f является липшиц-непрерывной на $[a, b]$, если существует $l > 0$ такое, что:[3]

$$|f(x) - f(y)| \leq l|x - y| \text{ для всех } x, y \in [a, b]. \tag{3.13}$$

Интуитивно понятно, что l равно величине наибольшей мгновенной скорости изменения без знака, которую функция достигает на $[a, b]$. При заданной точке $(x_0, f(x_0))$ линии $f(x_0) - l(x - x_0)$ для $x > x_0$ и $f(x_0) + l(x - x_0)$ для $x < x_0$ образуют нижнюю границу функции f.

Метод Шуберта–Пиявского итеративно строит все более точную нижнюю границу функции. При заданной действительной константе Липшица l алгоритм начинается с выбора средней точки, $x^{(1)} = (a + b)/2$. Нижняя граница пилообразной формы строится с использованием линий наклона $\pm l$ от этой точки. Эти линии всегда будут лежать ниже f, если l — действительная константа Липшица, как показано на рис. 3.12.

Верхние вершины пилообразной ломаной соответствуют выбранным точкам. Нижние вершины соответствуют пересечениям между липшицевыми линиями, исходящими из каждой выбранной точки. На дальнейших итерациях выполняется поиск минимальной точки пилообразной ломаной, вычисляется функция в этом значении x, а затем этот результат используется для обновления пилообразной ломаной. Этот процесс иллюстрирует рис. 3.13.

[3] Мы можем расширить определение липшиц-непрерывности на многомерные функции, где **x** и **y** — векторы, а абсолютное значение заменяется любой векторной нормой.

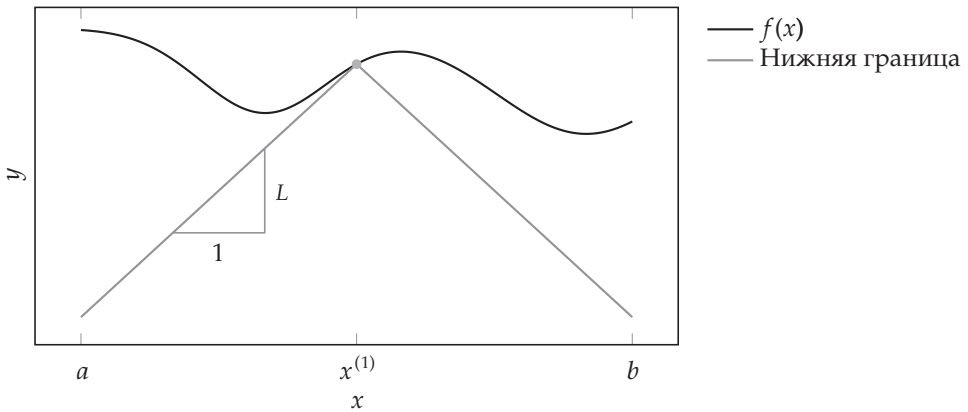


Рис. 3.12. Первая итерация метода Шуберта–Пиявского

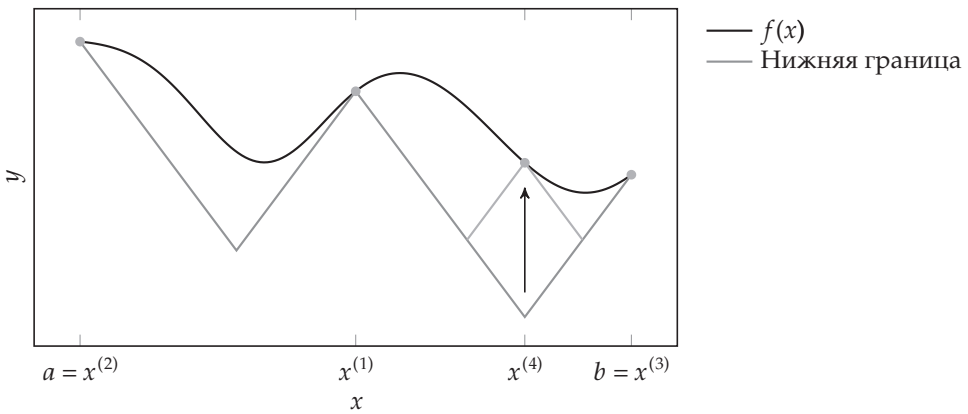


Рис. 3.13. Уточнение нижней границы включает выбор новой точки и пересечение новых линий с существующей пилообразной ломаной

Алгоритм обычно останавливается, когда разница в высоте между минимальным значением пилообразной ломаной и значением функции в этой точке становится меньше заданного допуска ε. Таким образом, для минимального пика $\left(x^{(n)}, y^{(n)}\right)$ и значения функции $f\left(x^{(n)}\right)$ мы завершаем работу, если $y^{(n)} - f\left(x^{(n)}\right) < \varepsilon$.

Области, в которых может находиться точка минимума, вычисляются с использованием этой информации об уточнении. Для каждого пика область неопределенности может быть определена по формуле

$$\left[x^{(i)} - \frac{1}{l}\left(y_{\min} - y^{(i)}\right),\ x^{(i)} + \frac{1}{l}\left(y^{(i)} - y_{\min}\right)\right] \tag{3.14}$$

для каждой вершины нижней пилообразной ломаной $\left(x^{(i)}, y^{(i)}\right)$ и минимальной вершины верхней пилообразной ломаной $\left(x_{\min}, y_{\min}\right)$. Точка $x^{(i)}$ влияет на область

неопределенности, только если $y^{(i)} < y_{\min}$. Точка минимума находится в одной из этих областей неопределенности.

Основной недостаток метода Шуберта–Пиявского заключается в том, что он требует знания действительной константы Липшица. Большие константы Липшица приведут к плохим нижним оценкам. На рис. 3.14 показано несколько итераций метода Шуберта–Пиявского.

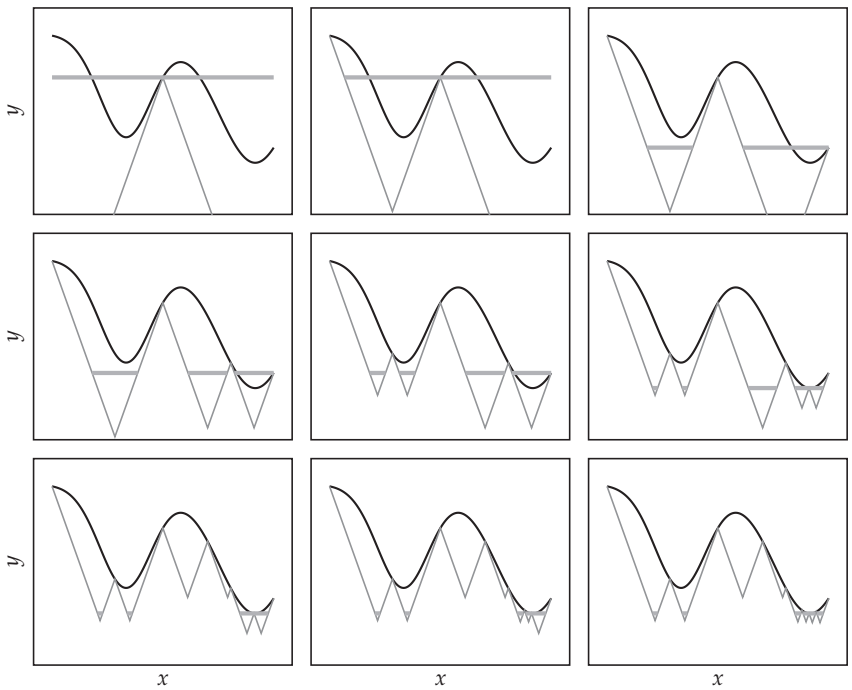


Рис. 3.14. Девять итераций метода Шуберта–Пиявского, идущих слева направо и сверху вниз. Синие линии — это области неопределенности, в которых может находиться точка глобального минимума

Алгоритм 3.5. Метод Шуберта–Пиявского для одномерной функции `f` с интервалом $a < b$ и постоянной Липшица `l`. Алгоритм работает до тех пор, пока уточнение не станет меньше допуска. Возвращаются как лучшая точка, так и набор интервалов неопределенности. Интервалы неопределенности возвращаются в виде массива кортежей `(a, b)`. Параметр `δ` — это допуск, используемый для объединения интервалов неопределенности

```
struct Pt
    x
    y
end
```

```
function _get_sp_intersection(A, B, l)
    t = ((A.y - B.y) - l * (A.x - B.x)) / 2l
    return Pt(A.x + t, A.y - t * l)
end
function shubert_piyavskii(f, a, b, l, ε, δ = 0.01)
    m = (a + b) / 2
    A, M, B = Pt(a, f(a)), Pt(m, f(m)), Pt(b, f(b))
    pts = [A, _get_sp_intersection(A, M, l),
           M, _get_sp_intersection(M, B, l), B]
    Δ = Inf
    while Δ > ε
        i = argmin([P.y for P in pts])
        P = Pt(pts[i].x, f(pts[i].x))
        Δ = P.y - pts[i].y

        P_prev = _get_sp_intersection(pts[i - 1], P, l)
        P_next = _get_sp_intersection(P, pts[i + 1], l)
        deleteat!(pts, i)
        insert!(pts, i, P_next)
        insert!(pts, i, P)
        insert!(pts, i, P_prev)
    end

    intervals = []
    i = 2 * (argmin([P.y for P in pts[1 : 2 : end]])) - 1
    for j in 2 : 2 : length(pts)
        if pts[j].y < pts[i].y
            dy = pts[i].y - pts[j].y
            x_lo = max(a, pts[j].x - dy/l)
            x_hi = min(b, pts[j].x + dy/l)
            if !isempty(intervals) && intervals[end][2] + δ ≥ x_lo
                intervals[end] = (intervals[end][1], x_hi)
            else
                push!(intervals, (x_lo, x_hi))
            end
        end
    end
    return (pts[i], intervals)
end
```

3.7. Метод бисекции

Метод бисекции (алгоритм 3.6) можно использовать для вычисления *корней* функции или точек, где функция равна нулю. Такие *методы нахождения корней* можно использовать для оптимизации, применяя их к производной целевой функции, определяя местоположение, где $f'(x) = 0$. В общем, мы должны убедиться, что результирующие точки действительно являются точками локального минимума.

Метод бисекции определяет интервал $[a, b]$, в котором существует хотя бы один корень. Если функция f непрерывна на $[a, b]$ и существует некоторое значение $y \in [f(a), f(b)]$, то *теорема о промежуточном значении* утверждает, что существует хотя бы одна точка $x \in (a, b)$, такая что $f(x) = y$, как показано на рис. 3.15. Отсюда следует, что интервал (a, b) гарантированно содержит нуль, если $f(a)$ и $f(b)$ имеют противоположные знаки.

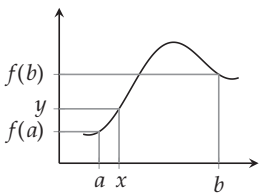


Рис. 3.15. Горизонтальная линия, проведенная из любого значения $y \in [f(a), f(b)]$, должна пересекать график хотя бы один раз

Метод бисекции на каждой итерации разрезает область неопределенности пополам. Для этого вычисляется средняя точка $(a + b)/2$ и формируется новый интервал, концами которого является средняя точка и такая точка, что интервал содержит нуль. Если средняя точка равна нулю, процесс прекращается немедленно. В противном случае он может завершиться после фиксированного количества итераций. На рис. 3.16 показаны пять итераций метода бисекции. Этот метод гарантированно сходится в пределах ε от x^* в течение итераций $\lg\frac{|b-a|}{\varepsilon}$, где lg обозначает логарифм с основанием 2.

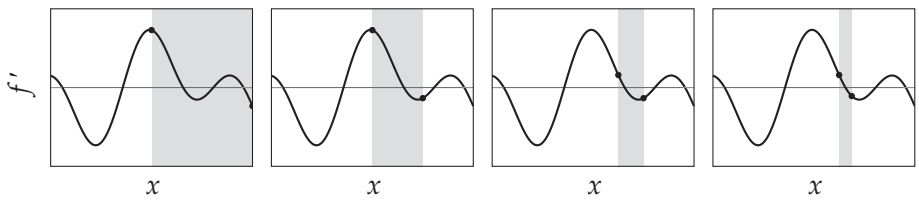


Рис. 3.16. Четыре итерации метода бисекции. Горизонтальная линия соответствует $f'(x) = 0$. Обратите внимание на то, что в начальном интервале существует несколько корней

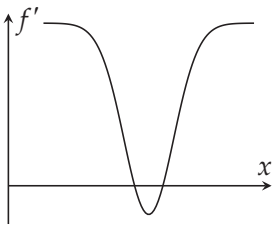


Рис. 3.17. Метод интервалов, инициализированный таким образом, что он охватывает два корня на этом рисунке, будет расширяться вечно, никогда не найдя изменения знака. Кроме того, если начальный интервал находится между двумя корнями, удвоение интервала может привести к тому, что оба конца интервала одновременно пройдут мимо двух корней

Алгоритмы поиска корней, такие как метод бисекции, требуют, чтобы концы начальных интервалов (a, b) были расположены по разные стороны от нуля, т.е. $\text{sign}(f'(a)) \neq \text{sign}(f'(b))$ или, что то же самое, $f'(a)f'(b) \leq 0$. Метод для автоматического определения такого интервала описан в алгоритме 3.7. Он начинается со случайного интервала $[a, b]$. Пока интервал не содержит нуля, его ширина увеличивается на постоянный коэффициент. Как правило, его размер удваивается. Этот метод не всегда будет успешным, как показано на рис. 3.17. Если функция имеет два близких корня, они могут быть пропущены, что приводит к бесконечному увеличению интервала без завершения.

Обобщением метода бисекции является *метод Брента–Деккера* (Brent–Dekker method). Это алгоритм поиска корня, который сочетает в себе элементы метода секущих (раздел 6.2) и обратной квадратичной интерполяции. Он обладает надежными и быстрыми свойствами сходимости и является предпочтительным алгоритмом одномерной оптимизации во многих популярных пакетах численной оптимизации.[4]

Алгоритм 3.6. Алгоритм бисекции, где `f'` — производная одномерной функции, которую мы стремимся оптимизировать. Если `a` < `b`, то интервал содержит нуль производной `f'`. Допуск ширины интервала равен `ε`. Вызов функции `bisection` возвращает новый интервал `(a, b)` как кортеж. Символ `'` — не апостроф. Таким образом, `f'` — это имя переменной, а не транспонированный вектор `f`. Этот символ можно создать, набрав команду `\prime` и нажав клавишу <Tab>

```
function bisection(f', a, b, ε)
    if a > b; a,b = b,a; end # гарантия условия a < b
```

[4] Подробности этого алгоритма можно найти в книге [25]. Этот алгоритм является продолжением работы [37].

```
    ya, yb = f′(a), f′(b)
    if ya == 0; b = a; end
    if yb == 0; a = b; end

    while b - a > ε
       x = (a + b) / 2
       y = f′(x)
       if y == 0
            a, b = x, x
       elseif sign(y) == sign(ya)
            a = x
       else
            b = x
       end
    end

    return (a,b)
end
```

Алгоритм 3.7. Алгоритм нахождения интервала, в котором происходит смена знака. Входными данными являются действительная функция **f′**, определенная на действительных числах, и начальный интервал [**a**, **b**]. Он возвращает новый интервал в виде кортежа, увеличивая ширину интервала, пока не произойдет смена знака между функцией, вычисленной в границах интервала. Коэффициент расширения **k** по умолчанию равен 2

```
function bracket_sign_change(f′, a, b; k = 2)
    if a > b; a,b = b,a; end # гарантия условия a < b

    center, half_width = (b + a) / 2, (b - a) / 2
    while f′(a) * f′(b) > 0
        half_width *= k
        a = center - half_width
        b = center + half_width
    end

    return (a,b)
end
```

3.8. Резюме

- Многие методы оптимизации сжимают интервалы, содержащие точку минимума, — метод Фибоначчи, метод золотого сечения и метод квадратичной аппроксимации.
- Метод Шуберта–Пиявского вычисляет набор интервалов, содержащих глобальные минимумы с учетом константы Липшица.
- Для определения того, где производная функции равна нулю, можно использовать методы поиска корней, такие как метод бисекции.

3.9. Упражнения

Упражнение 3.1. Приведите пример задачи, для которой метод Фибоначчи предпочтительнее метода бисекции.

Упражнение 3.2. В чем заключается недостаток метода Шуберта–Пиявского?

Упражнение 3.3. Приведите пример нетривиальной функции, для которой поиск с помощью квадратичной аппроксимации будет правильно определять минимум, если доступны значения функции в трех различных точках.

Упражнение 3.4. Предположим, что задана функция $f(x) = x^2/2 - x$. Примените метод бисекции для поиска интервала, содержащего точку минимума функции f, начиная с отрезка [0, 1000]. Выполните три шага алгоритма.

Упражнение 3.5. Предположим, что задана функция $f(x) = (x + 2)^2$ на отрезке [0, 1]. Является ли число 2 действительной константой Липшица для функции f на этом отрезке?

Упражнение 3.6. Предположим, что задана одномодальная функция, определенная на отрезке [1, 32]. Сможем ли мы сузить оптимум до интервала, длина которого не превышает 10, после трех вычислений функций? Объясните ответ.

Глава 4

Метод локального спуска

До этого момента мы рассматривали оптимизацию по единственному проектному параметру. В этой главе представлен общий подход к оптимизации с использованием функций с несколькими переменными. В этой главе основное внимание уделяется тому, как применять локальные модели для постепенного улучшения расчетной точки, пока не будет выполнено некоторое условие критерия сходимости. Мы начнем с обсуждения методов, которые на каждой итерации выбирают направление спуска на основе *локальной модели*, а затем выбирают размер шага. Затем мы обсудим методы, которые ограничивают шаг в пределах области, где локальная модель считается корректной. Эта глава заканчивается обсуждением условий сходимости. В следующих двух главах мы обсудим, как использовать модели первого и второго порядка, построенные на основе информации о градиенте или гессиане.

4.1. Метод спуска

Обычный подход к оптимизации заключается в постепенном улучшении проектного параметра x путем выполнения шага, который минимизирует значение целевой функции на основе локальной модели. Локальная модель может быть получена, например, из аппроксимации Тейлора первого или второго порядка. Алгоритмы оптимизации, которые следуют этому общему подходу, называются *методами спуска*. Они начинаются с расчетной точки $\mathbf{x}^{(1)}$, а затем генерируют последовательность точек, иногда называемых *итерационными приближениями*, сходящуюся к локальному минимуму.[1]

[1] Выбор точки $\mathbf{x}^{(1)}$ может повлиять на успех алгоритма в поиске точки минимума. Для выбора разумного значения часто используется знание предметной области. Если это невозможно, мы можем выполнить поиск в области проектных параметров, используя методы, которые будут рассмотрены в главе 13.

Итерационная процедура спуска включает в себя следующие этапы.

1. Проверяем, удовлетворяет ли точка $\mathbf{x}^{(k)}$ условиям завершения. Если да, прекращаем алгоритм; в противном случае переходим к следующему этапу.
2. Определяем *направление спуска* $\mathbf{d}^{(k)}$, используя локальную информацию, такую как градиент или гессиан. Некоторые алгоритмы предполагают, что $\|\mathbf{d}^{(k)}\| = 1$, а другие — нет.
3. Определяем шаговый множитель, или скорость обучения $\alpha^{(k)}$. Некоторые алгоритмы пытаются оптимизировать размер шага так, чтобы он максимально уменьшал функцию f.[2]
4. Вычисляем следующую расчетную точку по формуле:

$$\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{d}^{(k)} \tag{4.1}$$

Существует множество различных методов оптимизации, каждый из которых имеет свои способы определения параметра α и вектора $\mathbf{d}$.

4.2. Линейный поиск

На данный момент предположим, что мы выбрали направление спуска $\mathbf{d}$, возможно, используя один из методов, обсуждаемых в одной из последующих глав. Нам нужно выбрать шаговый множитель α, чтобы получить следующую расчетную точку. Один из подходов заключается в использовании *линейного поиска*, который выбирает скорость обучения, минимизирующую одномерную функцию:

$$\min_{\alpha} f\left(\mathbf{x} + \alpha\mathbf{d}\right). \tag{4.2}$$

Линейный поиск — это один из методов одномерной оптимизации, которая была рассмотрена в главе 3. Можно применить любой метод одномерной оптимизации.[3] Для направленного поиска используем производную от целевой функции линейного поиска, которая является просто производной по направлению вдоль вектора $\mathbf{x} + \alpha\mathbf{d}$. Линейный поиск продемонстрирован в примере 4.1 и реализован в алгоритме 4.1.

---

[2] Термин *шаговый множитель* используется для обозначения величины общего шага. Получение новой итерации с использованием уравнения (4.1) с шаговым множителем $\alpha^{(k)}$ подразумевает, что вектор направления спуска $\mathbf{d}^{(k)}$ имеет единичную длину. Мы используем термин *скорость обучения* для обозначения скалярного коэффициента, на который умножается вектор направления спуска, не обязательно имеющий единичную длину.

[3] Широко используемым методом одномерной оптимизации является метод Брента–Деккера, упомянутый в предыдущей главе. Он сочетает в себе надежность метода бисекции со скоростью метода секущих.

Алгоритм 4.1. Способ проведения линейного поиска, который находит оптимальный шаговый множитель α вдоль направления спуска $\mathbf{d}$ от расчетной точки $\mathbf{x}$, чтобы минимизировать функцию $\mathbf{f}$. Минимизация функции может быть реализована с помощью одномерного алгоритма оптимизации, такого как метод Брента–Деккера

```
function line_search(f, x, d)
    objective = α -> f(x + α * d)
    a, b = bracket_minimum(objective)
    α = minimize(objective, a, b)
    return x + α * d
end
```

Одним из недостатков проведения линейного поиска на каждом этапе является вычислительная стоимость оптимизации с высокой степенью точности. Вместо этого обычно лучше быстро найти разумное значение и затем двигаться дальше, выбирая сначала $\mathbf{x}^{(k+1)}$, а затем — новое направление $\mathbf{d}^{(k+1)}$.

Некоторые алгоритмы используют фиксированный шаговый множитель. Большие шаги, как правило, приводят к более быстрой сходимости, но возникает риск промахнуться мимо точки минимума. Меньшие шаги обычно приводят к более устойчивому процессу, но замедляют сходимость. Фиксированный шаговый множитель иногда называется *скоростью обучения*.

Другой метод заключается в использовании *затухающего шагового множителя*:

$$\alpha^{(k)} = \alpha^{(1)}\gamma^{k-1}, \text{ где } \gamma \in (0, 1]. \tag{4.3}$$

Затухающие шаговые множители особенно популярны при минимизации шумовых целевых функций[4] и широко используются в приложениях машинного обучения.

Пример 4.1. Линейный поиск используется для минимизации функции вдоль направления спуска

Рассмотрим возможность линейного поиска по $f(x_1, x_2, x_3) = \sin(x_1 x_2) + \exp(x_2 + x_3) - x_3$ из точки $\mathbf{x} = [1, 2, 3]$ в направлении $\mathbf{d} = [0, -1, -1]$. Соответствующая задача оптимизации заключается в следующем:

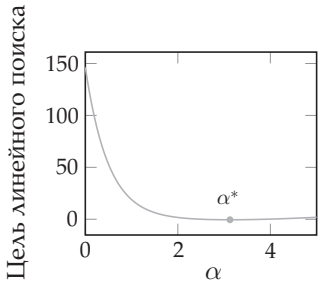
$$\min_{\alpha} \sin((1 + 0\alpha)(2 - \alpha)) + \exp((2 - \alpha) + (3 - \alpha)) - (3 - \alpha)$$

Она упрощается до задачи

$$\min_{\alpha} \sin(2 - \alpha) + \exp(5 - 2\alpha) + \alpha - 3$$

[4] Мы обсудим оптимизацию при наличии шума и других форм неопределенности в главе 17.

Точка минимума: $\mathbf{x} \approx [1, -1{,}126, -0{,}126]$ при $\alpha \approx 3{,}127$.



4.3. Приближенный линейный поиск

Часто в вычислительном отношении эффективнее выполнять больше итераций метода спуска, чем проводить точный линейный поиск на каждой итерации, особенно если вычисления функций и производных являются дорогостоящими. Многие из обсуждаемых методов могут извлечь пользу из использования *приближенного линейного поиска* для определения подходящего шагового множителя с небольшим количеством вычислений. Поскольку методы спуска должны стремиться к точке минимума, то размер шага можно считать приемлемым, если он приводит к уменьшению значения целевой функции. Тем не менее могут использоваться и другие условия для ускорения сходимости.

Условие *достаточного убывания*[5] (condition for sufficient decrease) требует, чтобы размер шага приводил к достаточному уменьшению значения целевой функции:

$$f\left(\mathbf{x}^{(k+1)}\right) \leq f\left(\mathbf{x}^{(k)}\right) + \beta \nabla_{\mathbf{d}^{(k)}} f\left(\mathbf{x}^{(k)}\right), \tag{4.4}$$

где $\beta \in [0, 1]$ часто устанавливается равным $\beta = 1 \cdot 10^{-4}$. Это условие проиллюстрировано на рис. 4.1. Если $\beta = 0$, то любое уменьшение является допустимым. Если $\beta = 1$, то уменьшение не должно превышать значение, которое было бы предсказано приближением первого порядка.

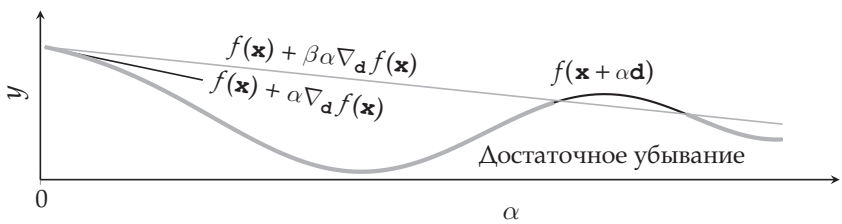


Рис. 4.1. Условие достаточного убывания, или первое условие Вольфе, всегда может быть выполнено с помощью выбора достаточно малого размера шага в направлении спуска

[5] Это условие иногда называется *условием Армихо* (Armijo condition).

Если направление спуска является недопустимым, то должен существовать достаточно маленький размер шага, который удовлетворяет условию достаточного уменьшения. Таким образом, мы можем начать с большого размера шага и уменьшать его с помощью постоянного коэффициента уменьшения, пока не будет выполнено условие достаточного уменьшения. Этот алгоритм называется *обратным линейным поиском*[6] из-за того, что он выполняет обратное отслеживание вдоль направления спуска. Обратный линейный поиск показан на рис. 4.2 и реализован в алгоритме 4.2. Пройдемся по процедуре в примере 4.2.

Алгоритм 4.2. Алгоритм обратного линейного поиска, который принимает целевую функцию `f`, ее градиент ∇`f`, текущую расчетную точку `x`, направление спуска `d` и максимальный размер шага `α`. Мы можем произвольно задавать коэффициент уменьшения параметра `α` и первый параметр условия Вольфе `β`

```
function backtracking_line_search(f, ∇f, x, d, α; p = 0.5, β = 1e-4)
    y, g = f(x), ∇f(x)
    while f(x + α * d) > y + β * α * (g ⋅ d)
        α *= p
    end
    α
end
```

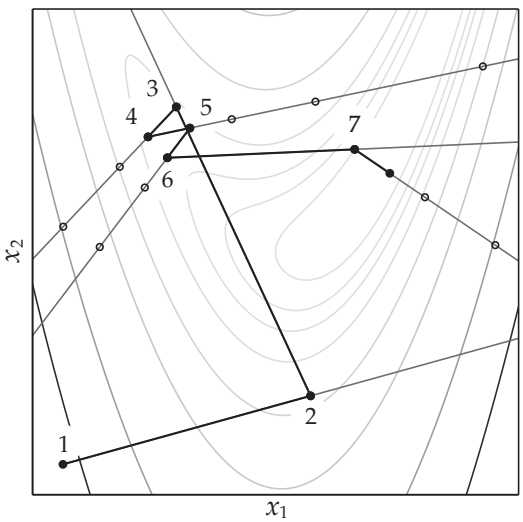


Рис. 4.2. Обратный линейный поиск, примененный к функции Розенброка (приложение Б.6). Черные линии показывают семь итераций, выполненных методом спуска, а красные линии показывают точки, рассматриваемые при каждом шаге линейного поиска

[6] Этот метод также называется *методом линейного поиска Армихо* [4].

Первого условия недостаточно, чтобы гарантировать сходимость к точке локального минимума. Очень маленькие размеры шага будут удовлетворять первому условию, но могут преждевременно сходиться. Обратный линейный поиск позволяет избежать преждевременной сходимости, вычисляя наибольший удовлетворительный размер шага, полученный последовательным уменьшением масштаба, и гарантированно сходится к точке локального минимума.

Пример 4.2. Пример приближенного линейного поиска

Рассмотрим приближенный линейный поиск минимума функции $f(x_1, x_2) = x_1^2 + x_1x_2 + x_2^2$ из точки $\mathbf{x} = [1, 2]$ в направлении $\mathbf{d} = [-1, -1]$, используя максимальный размер шага, равный 10, коэффициент затухания 0,5, параметр первого условия Вольфе $\beta = 1 \cdot 10^{-4}$ и параметр второго условия Вольфе $\sigma = 0{,}9$.

Проверим, удовлетворяет ли максимальный размер шага первому условию Вольфе, с учетом того, что градиент в точке $\mathbf{x}$ равен $\mathbf{g} = [4, 5]$:

$$f(\mathbf{x} + \mathbf{d}) \le f(\mathbf{x}) + (\mathbf{g}^{\mathrm{T}}\mathbf{d}),$$

$$f([1, 2] + 10 \cdot [-1, -1]) \le 7 + 1 \cdot 10^{-4} \cdot 10 \cdot [4, 5]^{\mathrm{T}}[-1, -1],$$

$$217 \le 6{,}991.$$

Условие не выполняется.

Умножим размер шага на 0,5, получим 5 и снова проверим первое условие Вольфе:

$$f([1, 2] + 5 \cdot [-1, -1]) \le 7 + 1 \cdot 10^{-4} \cdot 5 \cdot [4, 5]^{\mathrm{T}}[-1, -1],$$

$$37 \le 6{,}996.$$

Условие не выполняется.

Умножим размер шага на 0,5, получим 2,5 и снова проверим первое условие Вольфе:

$$f([1, 2] + 2{,}5 \cdot [-1, -1]) \le 7 + 1 \cdot 10^{-4} \cdot 2{,}5 \cdot [4, 5]^{\mathrm{T}}[-1, -1],$$

$$3{,}25 \le 6{,}998.$$

Первое условие Вольфе выполняется.

Проверим, удовлетворяет ли точка-кандидат $\mathbf{x}' = \mathbf{x} + \alpha\mathbf{d} = [-1{,}5; -0{,}5]$ второму условию Вольфе:

$$\nabla_{\mathbf{d}} f(\mathbf{x}') \ge \sigma \nabla_{\mathbf{d}} f(\mathbf{x});$$

$$[-3{,}5; -2{,}5]^{\mathrm{T}}[-1, -1] \ge \sigma [4, 5]^{\mathrm{T}}[-1, -1];$$

$$6 \ge -8{,}1.$$

Приближенный линейный поиск прекращается в точке $\mathbf{x} = [-1{,}5; -0{,}5]$.

Другое условие, называемое *условием кривизны*, требует, чтобы производная по направлению на следующей итерации была меньше текущей:

$$\nabla_{\mathbf{d}^{(k)}} f\left(\mathbf{x}^{(k+1)}\right) \geq \sigma \nabla_{\mathbf{d}^{(k)}} f\left(\mathbf{x}^{(k)}\right), \tag{4.5}$$

где параметр σ контролирует, насколько маленькой должна быть следующая производная по направлению. Это условие иллюстрируют рис. 4.3 и 4.4. Как правило, $\beta < \sigma < 1$ и $\sigma = 0{,}1$ при использовании приближенного линейного поиска с использованием метода сопряженных градиентов и 0,9 при использовании метода Ньютона.[7]

Альтернативой условию кривизны выступает *условие сильной кривизны*, которое является более строгим в том смысле, что тангенс угла наклона не должен быть большим положительным числом:

$$\left|\nabla_{\mathbf{d}^{(k)}} f\left(\mathbf{x}^{(k+1)}\right)\right| \leq -\sigma \nabla_{\mathbf{d}^{(k)}} f\left(\mathbf{x}^{(k)}\right). \tag{4.6}$$

Это условие иллюстрирует рис. 4.5.

Условие достаточного уменьшения и первое условие кривизны называются *условиями Вольфе*. Условие достаточного убывания часто называют *первым условием Вольфе*, а условие кривизны называют *вторым условием Вольфе*. Условие достаточного убывания в сочетании со вторым условием кривизны называют *сильными условиями Вольфе*.

Удовлетворение строгим условиям Вульфа требует более сложного алгоритма — *сильного обратного линейного поиска* (алгоритм 4.3) [113]. Этот метод работает в два этапа. Первый этап, называемый *этапом поиска интервала*, проверяет последовательно возрастающие размеры шагов, чтобы найти интервал $[\alpha^{(k-1)}, \alpha^{(k)}]$, гарантированно содержащий длины шагов, удовлетворяющие условиям Вольфе. Такой интервал обнаруживается при выполнении одного из следующих условий

$$f\left(\mathbf{x} + \alpha^{(k)}\mathbf{d}\right) \geq f\left(\mathbf{x}\right), \tag{4.7}$$

$$f\left(\mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{d}^{(k)}\right) > f\left(\mathbf{x}^{(k)}\right) + \beta\alpha^{(k)}\nabla_{\mathbf{d}^{(k)}} f\left(\mathbf{x}^{(k)}\right), \tag{4.8}$$

$$\nabla f\left(\mathbf{x} + \alpha^{(k)}\mathbf{d}\right) \geq 0. \tag{4.9}$$

Удовлетворение неравенства (4.8) эквивалентно нарушению первого условия Вольфе, гарантирующего, что уменьшение шагового множителя обеспечит удовлетворительную длину шага. Аналогично неравенства (4.7) и (4.9) гарантируют, что шаг снижения превысит локальный минимум, и поэтому область между ними должна содержать удовлетворительную длину шага.

---

[7] Метод сопряженных градиентов представлен в разделе 5.2, а метод Ньютона — в разделе 6.1.

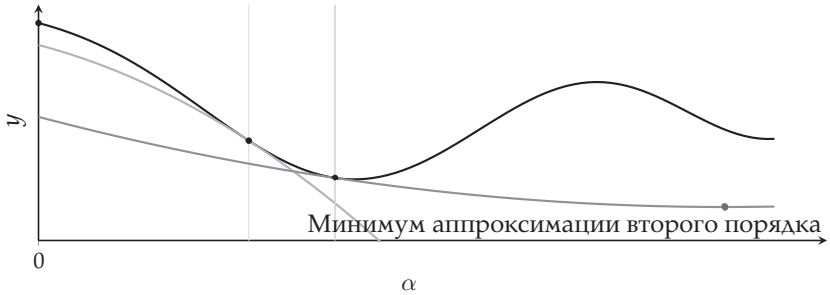


Рис. 4.3. Условие кривизны, или второе условие Вольфа, необходимо для обеспечения того, чтобы приближения функции второго порядка имели положительную кривизну, тем самым обеспечивая единственный глобальный минимум

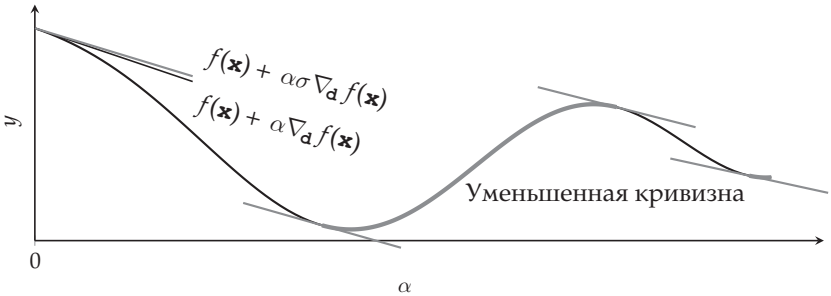


Рис. 4.4. Области, в которых выполняется условие кривизны

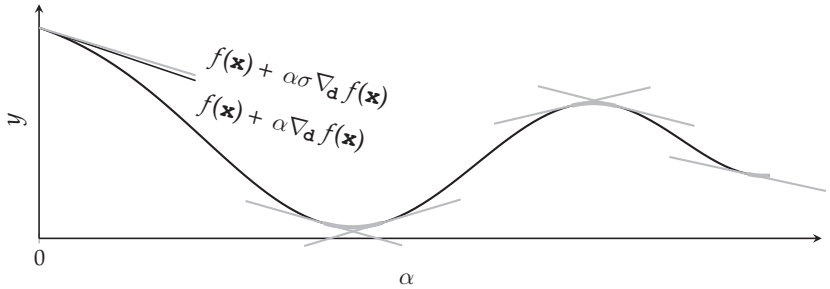


Рис. 4.5. Области, в которых выполняется условие сильной кривизны

На рис. 4.6 показано, где выполняется каждое из условий (4.7)–(4.9), на примере линейного поиска. На рисунке показаны интервалы $[0, \alpha]$, в то время как метод обратного линейного поиска последовательно увеличивает длину шага для получения интервала $[\alpha^{(k-1)}, \alpha^{(k)}]$.

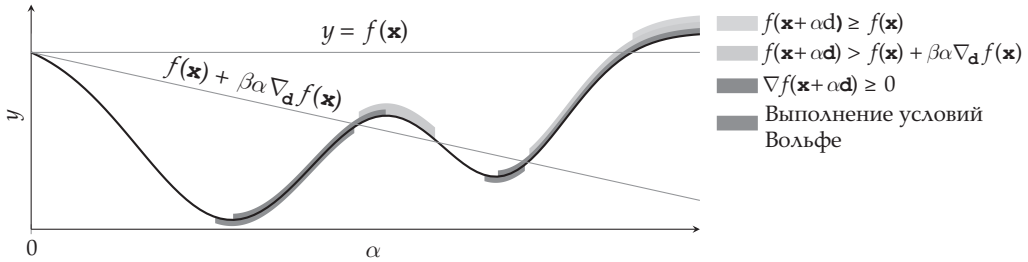


Рис. 4.6. Если выполняется хотя бы одно из трех условий, интервал [0, α] гарантированно охватывает интервал, содержащий шаговый множитель, удовлетворяющий сильным условиям Вульфа

На *этапе масштабирования* мы сжимаем интервал, чтобы найти размер шага, удовлетворяющий сильным условиям Вольфе. Это сжатие может быть выполнено с помощью метода бисекции (раздел 3.7), уточняя границы интервала в соответствии с теми же условиями. Этот процесс показан на рис. 4.7.

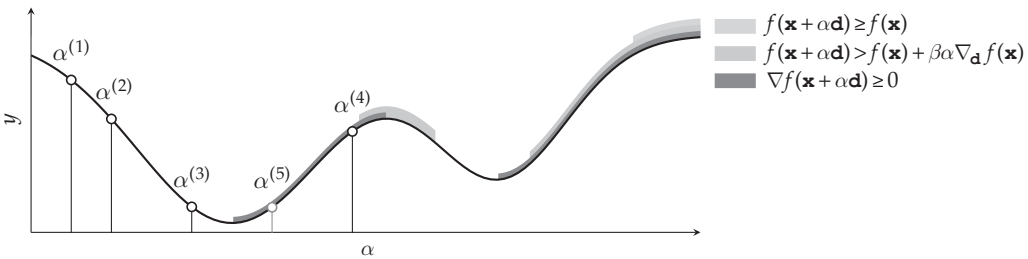


Рис. 4.7. Первый шаг метода сильного обратного линейного поиска, обозначенный черными кружками, используется для ограничения интервала. В этом случае вступает в действие условие $\nabla f\left(\mathbf{x} + \alpha^{(4)}\mathbf{d}\right) \geq 0$, в результате чего искомый интервал принимает вид [$\alpha^{(3)}$, $\alpha^{(4)}$]. Второй этап, обозначенный красным кружком, сжимает ограниченный интервал, пока не будет найден подходящий шаговый множитель

4.4. Методы доверительных областей

Методы спуска могут слишком сильно зависеть от условий первого или второго порядка, что может привести к чрезмерно большим шагам или преждевременной сходимости. *Доверительная область* (trust region) [97] — это локальная область пространства проектных параметров, в которой локальная модель считается надежной. Метод доверительной области, или *метод ограниченного шага*, поддерживает локальную модель доверительной области, которая ограничивает шаг, выполняемый традиционным методом линейного поиска, и прогнозирует улучшение, связанное с выполнением шага. Если улучшение хорошо совпадает с прогнозируемым значением, то доверительная область расширяется. Если улуч-

шение отклоняется от прогнозируемого значения, то доверительная область сокращается.[8] На рис. 4.8 показана расчетная точка с центром в круговой доверительной области.

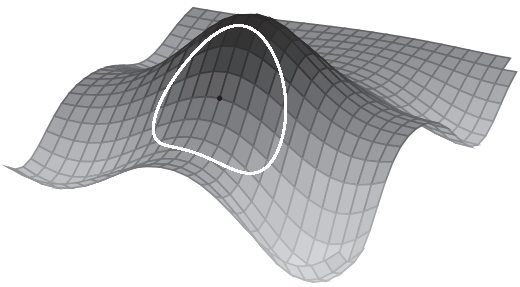


Рис. 4.8. Методы доверительной области ограничивают следующий шаг пределами локальной области. Доверительная область расширяется и сжимается на основе прогнозирующих характеристик моделей целевой функции

Методы доверительной области сначала выбирают максимальный размер шага, а затем направление. Этим они отличаются от методов линейного поиска, которые сначала выбирают направление, а затем оптимизируют размер шага. Метод доверительных областей находит следующий шаг, минимизируя модель целевой функции $\hat{f}$ для доверительной области с центром в расчетной текущей x. Примером модели $\hat{f}$ является аппроксимация Тейлора второго порядка (см. приложение В.2). Радиус доверительной области δ расширяется и сжимается в зависимости от того, насколько хорошо модель прогнозирует оценки функций. Следующая точка проектирования $\mathbf{x}'$ получается путем решения задачи:

$$\begin{aligned} &\min_{\mathbf{x}'} \hat{f}(\mathbf{x}') \\ &\text{при условии, что } \|\mathbf{x}-\mathbf{x}'\| \le \delta, \end{aligned} \tag{4.10}$$

где доверительная область определяется положительным радиусом δ и векторной нормой.[9] Вышеупомянутое уравнение представляет собой задачу оптимизации с ограничениями, которые рассматриваются в главе 10.

Радиус доверительной области δ расширяется или сокращается в зависимости от прогнозируемой эффективности локальной модели. Методы доверительной области сравнивают предсказанное улучшение $\Delta y_{\text{pred}} = f(\mathbf{x}) - \hat{f}(\mathbf{x}')$ с фактическим улучшением $\Delta y_{\text{act}} = f(\mathbf{x}) - f(\mathbf{x}')$:

---

[8] Свежий обзор методов доверительных областей приведен в [166].

[9] Существует множество эффективных методов решения уравнения (4.10). Обзор методов доверительных областей, примененных к квадратичным моделям, см. в [146].

$$\eta = \frac{\text{Фактическое улучшение}}{\text{Предсказанное улучшение}} = \frac{f(\mathbf{x}) - f(\mathbf{x}')}{f(\mathbf{x}) - \hat{f}(\mathbf{x}')}. \tag{4.11}$$

Отношение η близко к единице, когда прогнозируемый размер шага совпадает с фактическим размером шага. Если отношение слишком мало, например ниже порога η_1, то считается, что улучшение меньше, чем ожидалось, а радиус доверительной области уменьшается с коэффициентом $\gamma_1 < 1$. Если отношение достаточно велико, например, выше порога η_2, то наш прогноз считается точным, а радиус доверительной области доверия увеличивается в два раза. Реализация этого метода описана в алгоритме 4.4 , а визуализация процедуры оптимизации — на рис. 4.9. Пример 4.3 показывает, как построить некруглые доверительные области.

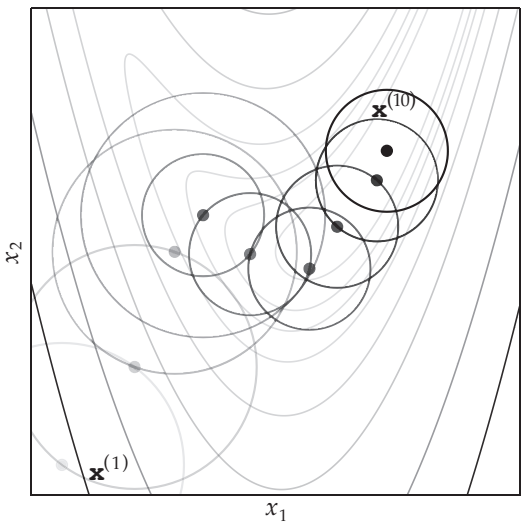


Рис. 4.9. Оптимизация доверительной области для функции Розенброка (приложение Б.6)

Алгоритм 4.3. Сильный обратный приближенный линейный поиск для удовлетворения сильных условий Вольфе. Он принимает в качестве входных данных целевую функцию `f`, градиент функции ∇, расчетную точку `x` и направление `d`, в котором проводится линейный поиск, а также начальный размер шага и параметры условий Вольфе `β` и `ε`. На этапе охвата интервала алгоритм сначала определяет интервал, содержащий размер шага, который удовлетворяет строгим условиям Вольфе. Затем он уменьшает этот интервал на этапе масштабирования, пока не будет найден подходящий размер шага. Мы выполняем интерполяцию с помощью бисекции, но можно использовать и другие схемы

```
function strong_backtracking(f, ∇, x, d; α = 1, β = 1e-4, σ = 0.1)
    y0, g0, y_prev, α_prev = f(x), ∇(x) ⋅ d, NaN, 0
```

```
    αlo, αhi = NaN, NaN

    # Этап поиска интервала
    while true
        y = f(x + α * d)
        if y > y0 + β * α * g0 || (!isnan(y_prev) && y ≥ y_prev)
            αlo, αhi = α_prev, α
            break
        end
        g = ∇(x + α * d) ⋅ d
        if abs(g) ≤ -σ * g0
            return α
        elseif g ≥ 0
            αlo, αhi = α, α_prev
            break
        end
        y_prev, α_prev, α = y, α, 2α
    end

    # этап масштабирования
    ylo = f(x + αlo * d)
    while true
        α = (αlo + αhi) / 2
        y = f(x + α * d)
        if y > y0 + β * α * g0 || y ≥ ylo
            αhi = α
        else
            g = ∇(x + α * d) ⋅ d
            if abs(g) ≤ -σ * g0
                return α
            elseif g * (αhi - αlo) ≥ 0
                αhi = αlo
            end
            αlo = α
        end
    end
end
```

Алгоритм 4.4. Метод доверительной области, где `f` — целевая функция, ∇`f` — производная, `H` — гессиан, `x` — начальная расчетная точка, а `k_max` — число итераций. Необязательные параметры η_1 и η_2 определяют, когда радиус доверительной области δ увеличивается или уменьшается, а параметры γ_1 и γ_2 контролируют величину изменения. Должна быть предусмотрена реализация функции `solve_trust_region_subproblem`, которая решает уравнение (4.10). Мы предоставили пример реализации, которая использует аппроксимацию Тейлора второго порядка в окрестности точки `x0` с круговой областью доверия

```
function trust_region_descent(f, ∇f, H, x, k_max;
    η1 = 0.25, η2 = 0.5, γ1 = 0.5, γ2 = 2.0, δ = 1.0)
    y = f(x)
    for k in 1 : k_max
        x′, y′ = solve_trust_region_subproblem(∇f, H, x, δ)
        r = (y - f(x′)) / (y - y′)
        if r < η1
            δ *= γ1
        else
            x, y = x′, y′
            if r > η2
                δ *= γ2
             end
        end
    end
    return x
end

using Convex
function solve_trust_region_subproblem(∇f, H, x0, δ)
    x = Variable(length(x0))
    p = minimize(∇f(x0) ⋅ (x - x0) + quadform(x - x0, H(x0)) / 2)
    p.constraints += norm(x - x0) <= δ
    solve!(p)
    return (x.value, p.optval)
end
```

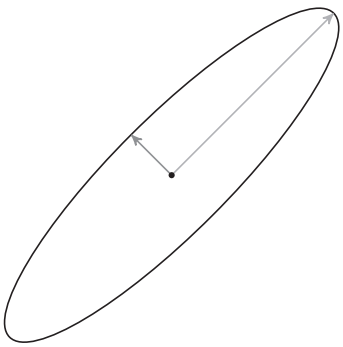
Пример 4.3. Методы доверительной области не обязаны использовать круглые области. Дополнительные подробности см. в работе [113]

Доверительные области не обязаны быть круглыми. В некоторых случаях определенные направления могут иметь более высокое доверие, чем другие.

Норму можно построить для создания эллиптических областей:

$$\|\mathbf{x} - \mathbf{x}_0\|_{\mathbf{E}} = (\mathbf{x} - \mathbf{x}_0)^{\mathrm{T}} \mathbf{E} (\mathbf{x} - \mathbf{x}_0),$$

где $\|\mathbf{x} - \mathbf{x}_0\|_{\mathbf{E}} \leq 1$, а $\mathbf{E}$ — симметричная матрица, которая определяет эллипс.



Матрица эллипса $\mathbf{E}$ может обновляться на каждой итерации спуска, что может потребовать более сложных корректировок, чем масштабирование доверительной области.

4.5. Условия завершения итераций

Существуют четыре общих условия завершения итераций для методов спуска.

- *Максимальное количество итераций*. Можно завершить алгоритм, когда число итераций k превысит некоторый порог $k_{\max}$, или как только будет превышено максимальное количество прошедшего времени.

$$k > k_{\max}. \tag{4.12}$$

- *Абсолютное улучшение*. Это условие завершения определяет изменение значения функции на последующих этапах. Если изменение меньше заданного порогового значения, оно прекращается:

$$f\left(\mathbf{x}^{(k)}\right) - f\left(\mathbf{x}^{(k+1)}\right) < \varepsilon_a. \tag{4.13}$$

- *Относительное улучшение*. Это условие завершения также учитывает изменение значения функции, но сравнивает шаговый множитель с текущим значением функции:

$$f\left(\mathbf{x}^{(k)}\right) - f\left(\mathbf{x}^{(k+1)}\right) < \varepsilon_r \left| f\left(\mathbf{x}^{(k)}\right) \right|. \tag{4.14}$$

- *Величина градиента*. Мы также можем завершить алгоритм, когда градиент становится очень маленьким:

$$\left\| \nabla f\left(\mathbf{x}^{(k+1)}\right) \right\| < \varepsilon_g. \tag{4.15}$$

При наличии нескольких локальных минимумов может быть полезно выполнять случайные перезапуски, после того как будут выполнены наши условия завершения, из случайно выбранных начальных точек.

4.6. Резюме

- Методы спуска постепенно снижаются до точки локального оптимума.
- Во время линейного поиска может быть применена одномерная оптимизация.
- Для определения подходящих размеров шага в методе спуска может использоваться приближенный линейный поиск.
- Методы доверительной области ограничивают шаг, лежащий в локальной области, которая расширяется или сжимается в зависимости от точности прогнозирования.
- Условия завершения для методов спуска могут основываться на таких критериях, как изменение значения целевой функции или величины градиента.

4.7. Упражнения

Упражнение 4.1. Почему важно иметь более одного условия завершения?

Упражнение 4.2. Первое условие Вольфе требует, чтобы выполнялось неравенство

$$f\left(\mathbf{x}^{(k)} + \alpha \mathbf{d}^{(k)}\right) \leq f\left(\mathbf{x}^{(k)}\right) + \beta \alpha \nabla_{\mathbf{d}^{(k)}} f\left(\mathbf{x}^{(k)}\right). \tag{4.16}$$

Какова максимальная длина шага, который удовлетворяет этому условию, учитывая, что $f\left(\mathbf{x}\right) = 5 + x_1^2 + x_2^2$, $\mathbf{x}^{(k)} = \left[-1,\ -1\right]$, $\mathbf{d} = \left[1,\ 0\right]$ и $\beta = 10^{-4}$?

Глава 5

Методы первого порядка

В предыдущей главе была представлена общая концепция методов спуска. В этой главе рассматриваются различные алгоритмы, которые используют *методы первого порядка* для выбора подходящего направления спуска. Методы первого порядка опираются на информацию о градиенте, чтобы помочь найти минимум, который можно получить с помощью методов, описанных в главе 2.

5.1. Градиентный спуск

Интуитивно понятный выбор направления спуска $\mathbf{d}$ — это направление наискорейшего спуска. Следование направлению наискорейшего спуска гарантированно приведет к улучшению при условии, что целевая функция является гладкой, размер шага достаточно мал, и мы еще не достигли точки, где градиент равен нулю.[1] Направлением наискорейшего спуска является направление, противоположное градиенту ∇f, отсюда и название *метода градиентного спуска*. Для удобства в этой главе мы определим градиент как вектор

$$\mathbf{g}^{(k)} = \nabla f(\mathbf{x}^{(k)}), \tag{5.1}$$

где $\mathbf{x}^{(k)}$ — расчетная точка на k-й итерации метода спуска.

При градиентном спуске мы обычно нормализуем направление самого крутого спуска (см. пример 5.1):

$$\mathbf{d}^{(k)} = \frac{\mathbf{g}^{(k)}}{\left\|\mathbf{g}^{(k)}\right\|}. \tag{5.2}$$

Если мы выберем размер шага, который приведет к максимальному уменьшению функции f, получаются зигзагообразные пути поиска. Фактически следующее направление всегда будет ортогонально текущему направлению. Мы можем показать это следующим образом.

[1] Точка, в которой градиент равен нулю, называется *стационарной*.

Пример 5.1. Расчет направления градиентного спуска

Предположим, что $f(\mathbf{x}) = x_1 x_2^2$. Градиент равен $\nabla f = \left[x_2^2,\ 2x_1 x_2\right]$. Для $\mathbf{x}^{(k)} = [1, 2]$ мы получаем ненормированное направление наискорейшего спуска, задаваемое вектором $\mathbf{d} = [-4, -4]$, который после нормализации превращается в вектор $\mathbf{d} = \left[-\frac{1}{\sqrt{2}},\ -\frac{1}{\sqrt{2}}\right]$.

Если мы оптимизируем шаговый множитель на каждом шаге, то получим

$$\alpha^{(k)} = \arg\min_{\alpha} f\left(\mathbf{x}^{(k)} + \alpha \mathbf{d}^{(k)}\right). \tag{5.3}$$

Приведенная выше оптимизация подразумевает, что производная по направлениям равна нулю. Используя уравнение (2.9), получим

$$\nabla f(\mathbf{x}^{(k)} + \alpha \mathbf{d}^{(k)})^{\mathrm{T}} \mathbf{d}^{(k)} = 0. \tag{5.4}$$

Мы знаем, что

$$\mathbf{d}^{(k+1)} = -\frac{\nabla f\left(\mathbf{x}^{(k)} + \alpha \mathbf{d}^{(k)}\right)}{\left\|\nabla f\left(\mathbf{x}^{(k)} + \alpha \mathbf{d}^{(k)}\right)\right\|}. \tag{5.5}$$

Следовательно,

$$\mathbf{d}^{(k+1)\mathrm{T}} \mathbf{d}^{(k)} = 0, \tag{5.6}$$

это означает, что векторы $\mathbf{d}^{(k+1)}$ и $\mathbf{d}^{(k)}$ ортогональны.

Узкие долины, ориентированные по направлению спуска, не являются проблемой. Но если направления спуска пересекают долину, необходимо предпринять много шагов, чтобы продвинуться по дну долины, как показано на рис. 5.1. Реализация градиентного спуска описана алгоритмом 5.1.

Алгоритм 5.1. Метод градиентного спуска, который следует направлению градиентного спуска с фиксированной скоростью обучения. Функция `step!` выполняет переход на следующую итерацию, тогда как функция `init` ничего не делает

```
abstract type DescentMethod end
struct GradientDescent <: DescentMethod
    α
end
init!(M :: GradientDescent, f, ∇f, x) = M
function step!(M :: GradientDescent, f, ∇f, x)
```

```
    α, g = M.α, ∇f(x)
    return x - α * g
end
```

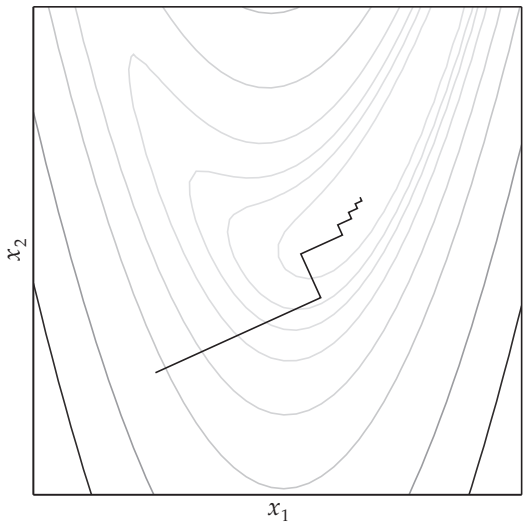


Рис. 5.1. Градиентный спуск может привести к зигзагообразным узким каньонам. Здесь мы видим путь спуска для функции Розенброка (приложение Б.6)

5.2. Метод сопряженных градиентов

Метод градиентного спуска может плохо работать в узких долинах. *Метод сопряженных градиентов* преодолевает эту проблему, используя идеи методов оптимизации квадратичных функций:

$$\min_{\mathbf{x}} f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} + \mathbf{b}^{\mathrm{T}}\mathbf{x} + c, \tag{5.7}$$

где $\mathbf{A}$ — симметрично и положительно определенная матрица, и, следовательно, функция f имеет единственный локальный минимум (раздел 1.6.2).

Метод сопряженных градиентов может оптимизировать n-мерные квадратичные функции за n шагов, как показано на рис. 5.2. Его направления взаимно сопряжены относительно матрицы $\mathbf{A}$:

$$\mathbf{d}^{(i)\mathrm{T}}\mathbf{A}\mathbf{d}^{(j)} = 0 \text{ для всех } i \neq j. \tag{5.8}$$

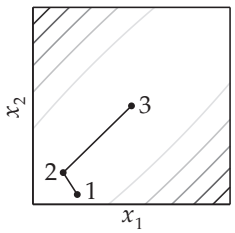


Рис. 5.2. Метод сопряженных градиентов, примененный к n-мерной квадратичной функции, сходится за n шагов

Взаимно сопряженные векторы являются базисными векторами матрицы **A**. Они, как правило, не ортогональны друг другу.

Последовательные сопряженные направления вычисляются с использованием информации о градиенте и предыдущем направлении спуска. Алгоритм начинается с определения направления наискорейшего спуска:

$$\mathbf{d}^{(1)} = -\mathbf{g}^{(1)}. \tag{5.9}$$

Затем используется линейный поиск, чтобы найти следующую расчетную точку. Для квадратичных функций шаговый множитель α можно вычислить точно (пример 5.2). Тогда итерационное приближение вычисляется так:

$$\mathbf{x}^{(2)} = \mathbf{x}^{(1)} + \alpha^{(1)}\mathbf{d}^{(1)}. \tag{5.10}$$

Пример 5.2. Оптимальный шаговый множитель для линейного поиска точки минимума квадратичной функции

Предположим, мы хотим вычислить оптимальный шаговый множитель для линейного поиска точки минимума квадратичной функции:

$$\min_{\alpha} f\left(\mathbf{x} + \alpha\mathbf{d}\right).$$

Мы можем вычислить производную по α:

$$\begin{aligned}\frac{\partial f\left(\mathbf{x}+\alpha\mathbf{d}\right)}{\partial\alpha} &= \frac{\partial}{\partial\alpha}\left[\frac{1}{2}\left(\mathbf{x}+\alpha\mathbf{d}\right)^{\mathrm{T}}\mathbf{A}\left(\mathbf{x}+\alpha\mathbf{d}\right)+\mathbf{b}^{\mathrm{T}}\left(\mathbf{x}+\alpha\mathbf{d}\right)+c\right] = \\ &= \mathbf{d}^{\mathrm{T}}\mathbf{A}\left(\mathbf{x}+\alpha\mathbf{d}\right)+\mathbf{d}^{\mathrm{T}}\mathbf{b} = \\ &= \mathbf{d}^{\mathrm{T}}\left(\mathbf{A}\mathbf{x}+\mathbf{d}\right)+\alpha\mathbf{d}^{\mathrm{T}}\mathbf{A}\mathbf{d}.\end{aligned}$$

Полагая $\dfrac{\partial f\left(\mathbf{x}+\alpha\mathbf{d}\right)}{\partial\alpha} = 0$, приходим к результату:

$$\alpha = -\frac{\mathbf{d}^{\mathrm{T}}\left(\mathbf{A}\mathbf{x}+\mathbf{b}\right)}{\mathbf{d}^{\mathrm{T}}\mathbf{A}\mathbf{d}}.$$

Последующие итерации выбирают $\mathbf{d}^{(k+1)}$ на основе следующего градиента и вклада текущего направления спуска

$$\mathbf{d}^{(k+1)} = -\mathbf{g}^{(k+1)} + \beta^{(k)}\mathbf{d}^{(k)}, \tag{5.11}$$

где β — скалярный параметр. Большие значения β указывают на то, что предыдущее направление снижения влияет сильнее.

Можно получить оптимальное значение β для заданной матрицы $\mathbf{A}$, используя тот факт, что вектор $\mathbf{d}^{(k+1)}$ сопряжен с вектором $\mathbf{d}^{(k)}$:

$$\mathbf{d}^{(k+1)}\mathbf{A}\mathbf{d}^{(k)} = 0, \tag{5.12}$$

$$\Rightarrow \left(-\mathbf{g}^{(k+1)} + \beta^{(k)}\mathbf{d}^{(k)}\right)^{\mathrm{T}}\mathbf{A}\mathbf{d}^{(k)} = 0, \tag{5.13}$$

$$\Rightarrow -\mathbf{g}^{(k+1)\mathrm{T}}\mathbf{A}\mathbf{d}^{(k)} + \beta^{(k)}\mathbf{d}^{(k)\mathrm{T}}\mathbf{A}\mathbf{d}^{(k)} = 0, \tag{5.14}$$

$$\Rightarrow \beta^{(k)} = \frac{\mathbf{g}^{(k+1)\mathrm{T}}\mathbf{A}\mathbf{d}^{(k)}}{\mathbf{d}^{(k)\mathrm{T}}\mathbf{A}\mathbf{d}^{(k)}}. \tag{5.15}$$

Метод сопряженных градиентов может применяться и к неквадратичным функциям. В окрестности локального минимума гладкие, непрерывные функции ведут себя как квадратичные функции, и метод сопряженных градиентов будет очень быстро сходиться в таких областях.

К сожалению, мы не знаем матрицу $\mathbf{A}$, которая наилучшим образом приближает функцию f в окрестности точки $\mathbf{x}^{(k)}$. Однако существуют несколько вариантов выбора $\beta^{(k)}$, которые хорошо зарекомендовали себя на практике:

Метод Флетчера–Ривса (Fletcher–Reeves) [48]:

$$\beta^{(k)} = \frac{\mathbf{g}^{(k)\mathrm{T}}\mathbf{g}^{(k)}}{\mathbf{g}^{(k-1)\mathrm{T}}\mathbf{g}^{(k-1)}}. \tag{5.16}$$

Метод Полака–Рибьера (Polak–Ribière) [122]:

$$\beta^{(k)} = \frac{\mathbf{g}^{(k)\mathrm{T}}\left(\mathbf{g}^{(k)} - \mathbf{g}^{(k-1)}\right)}{\mathbf{g}^{(k-1)\mathrm{T}}\mathbf{g}^{(k-1)}}. \tag{5.17}$$

Сходимость метода Полака–Рибьера (алгоритм 5.2) может быть гарантирована, если мы допустим автоматическую подстановку:

$$\beta \leftarrow \max(\beta, 0) \tag{5.18}$$

Пример поиска с использованием этого метода показан на рис. 5.3.

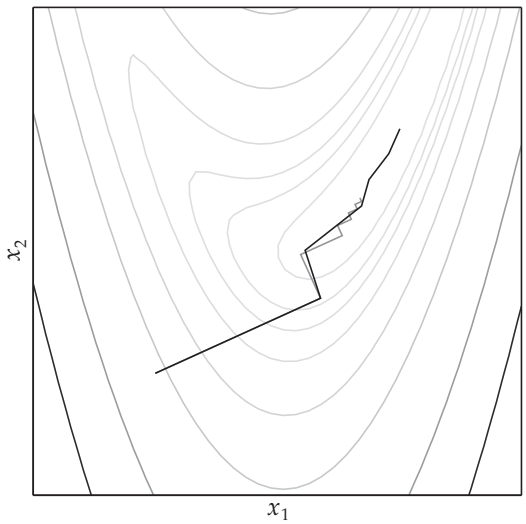


Рис. 5.3. Метод сопряженных градиентов с модификацией Полака–Рибьера

Алгоритм 5.2. Метод сопряженного градиента с модификацией Полака–Рибьера, где `d` — предыдущее направление поиска, а `g` — предыдущий градиент

```
mutable struct ConjugateGradientDescent <: DescentMethod
    d
    g
end
function init(M :: ConjugateGradientDescent, f, ∇f, x)
   M.g = ∇f(x)
   M.d = -M.g
   return M
end
function step! (M ::ConjugateGradientDescent, f, ∇f, x)
    d, g = M.d, M.g
    g′ = ∇f(x)
    β = max(0, dot(g′, g′ - g) / (g ⋅ g))
    g′ = -g′ + β * d
    x′ = line_search(f, x, d′)
    M.d, M.g = d′, g′
    return x′
end
```

5.3. Метод моментов

Градиентный спуск будет очень долго пересекать почти плоскую поверхность, как показано на рис. 5.4. Одним из способов ускорить процесс является возможность накопления момента. Мы можем изменить метод градиентного спуска, чтобы включить в него момент.

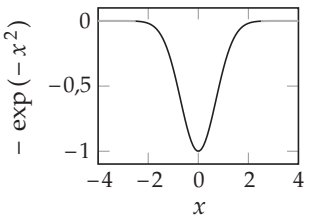


Рис. 5.4. Области, которые являются почти плоскими, имеют очень маленькие градиенты, поэтому методу градиентного спуска потребуется много итераций

Формулы *метода моментов* имеют следующий вид:

$$\mathbf{v}^{(k+1)} = \beta\mathbf{v}^{(k)} - \alpha\mathbf{g}^{(k)}, \tag{5.19}$$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k+1)} + \mathbf{v}^{(k+1)}. \tag{5.20}$$

При $\beta = 0$ мы получим формулы градиентного спуска. Метод моментов можно интерпретировать как описание движения шара, катящегося по почти горизонтальному наклону. Шар естественным образом набирает обороты, поскольку гравитация заставляет его ускоряться, так же, как градиент приводит к накоплению момента в этом варианте методе спуска. Реализация представлена этого метода представлена в алгоритме 5.3. Результаты сравнения метода моментов с градиентным спуском приведены на рис. 5.5.

Алгоритм 5.3. Метод моментов. Первая строка в функции `step!` делает копии скаляров `α` и `β`, но создает ссылку на вектор `v`. Таким образом, следующая строка `v[:] = β * v - α * g` изменяет исходный вектор моментов в структуре `M`

```
mutable struct Momentum <: DescentMethod
    α # скорость обучения
    β # затухание момента
    v # момент
end
function init!(M :: Momentum, f, ∇f, x)
    M.v = zeros(length(x))
    return M
end
```

```
function step!(M :: Momentum, f, ∇f, x)
    α, β, v, g = M.α, M.β, M.v, ∇f(x)
    v[:] = β * v - α * g
    return x + v
end
```

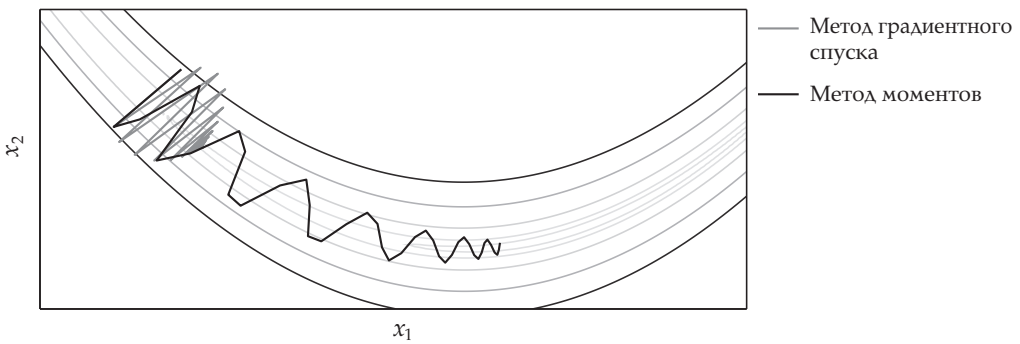


Рис. 5.5. Результаты сравнения градиентного спуска и метода моментов на примере функции Розенброка с $b = 100$; см. приложение Б.6

5.4. Метод Нестерова

Одна из проблем заключается в том, что шаги на дне долины замедляются недостаточно быстро и имеют тенденцию перескакивать через него. *Метод Нестерова* [112] модифицирует метод моментов:

$$\mathbf{v}^{(k+1)} = \beta \mathbf{v}^{(k)} - \alpha \nabla f\left(\mathbf{x}^{(k)} + \beta \mathbf{v}^{(k)}\right), \tag{5.21}$$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{v}^{(k+1)}. \tag{5.22}$$

Реализация метода Нестерова описана в алгоритме 5.4. Результаты сравнения метода Нестерова и метода моментов показаны на рис. 5.6.

Алгоритм 5.4. Метод Нестерова

```
mutable struct NesterovMomentum <: DescentMethod
    α # скорость обучения
    β # затухание момента
    v # момент
end
function init!(M :: NesterovMomentum, f, ∇f, x)
    M.v = zeros(length(x))
return M
end
```

```
function step!(M :: NesterovMomentum, f, ∇f, x)
    α, β, v = M.α, M.β, M.v
    v[:] = β * v - α * ∇f(x + β * v)
return x + v
end
```

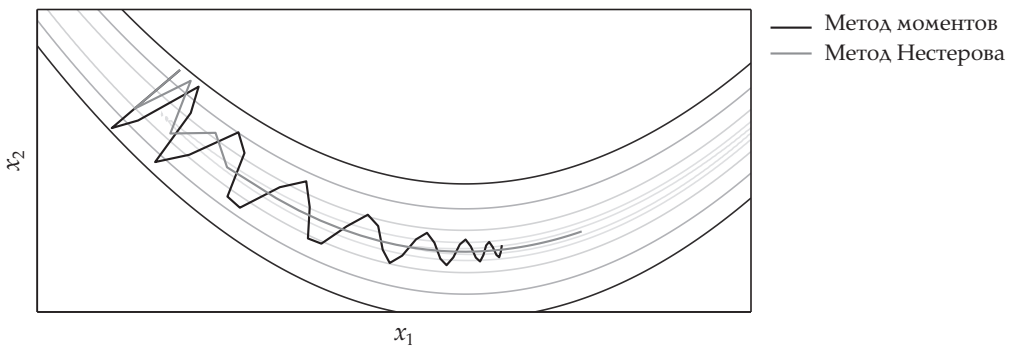


Рис. 5.6. Сравнение метода моментов и метода Нестерова на примере функции Розенброка с $b = 100$; см. приложение Б.6

5.5. Метод Adagrad

Метод моментов и метод Нестерова обновляют все компоненты вектора $\mathbf{x}$ с одинаковой скоростью обучения. *Адаптивный субградиентный метод* (*Adagrad* [42]) адаптирует скорость обучения для каждого компонента вектора $\mathbf{x}$. Метод Adagrad смягчает влияние параметров с постоянно высокими градиентами, тем самым увеличивая влияние параметров с редкими модификациями.[2]

Формулы метода Adagrad имеют вид:

$$x_i^{(k+1)} = x_i^{(k)} - \frac{\alpha}{\varepsilon + \sqrt{s_i^{(k)}}} g_i^{(k)}, \tag{5.23}$$

где $\mathbf{s}^{(k)}$ — вектор, i-й элемент которого является суммой квадратов частичных производных по x_i до временного шага k:

$$s_i^{(k)} = \sum_{j=1}^{k} \left(g_i^{(j)} \right)^2, \tag{5.24}$$

[2] Метод Adagrad превосходно работает, когда градиент является разреженным вектором. Источником вдохновения для исходной статьи был метод стохастического градиентного спуска, выбирающий случайную серию обучающих данных для каждой итерации, по которым можно вычислить градиент. Разреженные градиенты порождаются многими наборами данных для глубокого обучения при решении реальных задач, в которых некоторые признаки встречаются гораздо реже остальных.

а ε — это небольшое значение, порядка $1 \cdot 10^{-8}$, чтобы предотвратить деление на нуль.

Метод Adagrad гораздо менее чувствителен к параметру скорости обучения α. Параметр скорости обучения обычно имеет значение по умолчанию, равное 0,01. Главная слабость метода Adagrad заключается в том, что компоненты вектора **s** строго не убывают. Накопленная сумма приводит к снижению эффективной скорости обучения, часто становясь бесконечно малой еще до выполнения критерия сходимости. Реализация описана в алгоритме 5.5.

Алгоритм 5.5. Метод ускоренного спуска Adagrad

```
mutable struct RMSProp <: DescentMethod
    α # скорость обучения
    ε # малое значение
    s # сумма квадратов градиентов
end
function init!(M :: Adagrad, f, ∇f, x)
    M.s = zeros(length(x))
    return M
end
function step!(M :: Adagrad, f, ∇f, x)
    α, ε, s, g = M.α, M.ε, M.s, ∇f(x)
    s[:] += g .* g
    return x - α * g ./ (sqrt.(s) .+ ε)
end
```

5.6. Метод RMSProp

Метод RMSProp[3] обобщает метод Adagrad, чтобы избежать эффекта монотонно убывающей скорости обучения. Он поддерживает затухание среднего квадратов градиентов. Это среднее значение обновляется в соответствии с формулой:[4]

$$\hat{\mathbf{s}}^{(k+1)} = \gamma \hat{\mathbf{s}}^{(k)} + (1-\gamma)\left(\mathbf{g}^{(k)} \odot \mathbf{g}^{(k)}\right), \tag{5.25}$$

где коэффициент затухания $\gamma \in [0, 1]$ обычно близко к 0,9.

[3] Метод RMSProp не опубликован и взят из курса лекций Джеффа Хинтона Coursera Lecture 6e.

[4] Операция $\mathbf{a} \odot \mathbf{b}$ — поэлементное произведение векторов **a** и **b**.

Затухающее среднее квадратов градиентов на прошлых итерациях можно подставить в уравнение обновления RMSProp:[5]

$$x_i^{(k+1)} = x_i^{(k)} - \frac{\alpha}{\varepsilon + \sqrt{\hat{s}_i^{(k+1)}}} g_i^{(k)} = \tag{5.26}$$

$$= x_i^{(k)} - \frac{\alpha}{\varepsilon + \mathrm{RMS}\left(g_i\right)} g_i^{(k)}. \tag{5.27}$$

Реализация описывается алгоритмом 5.6.

Алгоритм 5.6. Метод ускоренного спуска RMSProp

```
mutable struct RMSProp <: DescentMethod
    α # скорость обучения
    γ # коэффициент затухания
    ε # малое значение
    s # сумма квадратов градиентов
end
function init!(M :: RMSProp, f, ∇f, x)
    M.s = zeros(length(x))
    return M
end
function step!(M :: RMSProp, f, ∇f, x)
    α, γ, ε, s, g = M.α, M.γ, M.ε, M.s, ∇f(x)
    s[:] = γ * s + (1 - γ) * (g .* g)
    return x - α * g ./ (sqrt.(s) .+ ε)
end
```

5.7. Метод Adadelta

Метод Adadelta [168] — это еще один метод преодоления монотонно убывающей скорости обучения в методе Adagrad. Независимо разработав метод RMSProp, авторы заметили, что единицы в формулах градиентного спуска, моментов и Adagrad не совпадают. Чтобы исправить это, они применили экспоненциально затухающее среднее квадратов градиентов:

$$x_i^{(k+1)} = x_i^{(k)} - \frac{\mathrm{RMS}\left(\Delta x_i\right)}{\varepsilon + \mathrm{RMS}\left(\Delta g_i\right)} g_i^{(k)}, \tag{5.28}$$

[5] Знаменатель похож на среднеквадратичное значение (root mean square — RMS) компонентов градиента. В этой главе мы используем выражение $\mathrm{RMS}(x)$ для обозначения затухающего среднеквадратичного значения временного ряда x.

которое полностью исключает параметр скорости обучения. Реализация описывается алгоритмом 5.7.

Алгоритм 5.7. Метод ускоренного спуска Adadelta. Маленькая константа `ε` добавляется в числитель, чтобы предотвратить полное замедление прогресса до нуля и начать первую итерацию, где $\Delta x = 0$

```
mutable struct Adadelta <: DescentMethod
    γs # затухание градиента
    γx # обновление коэффициента затухания
    ε  # малое значение
    s  # сумма квадратов градиентов
    u  # обновленная сумма квадратов градиентов
end
function init!(M :: Adadelta, f, ∇f, x)
    M.s = zeros(length(x))
    M.u = zeros(length(x))
    return M
end
function step!(M :: Adadelta, f, ∇f, x)
    γs, γx, ε, s, u, g = M.γs, M.γx, M.ε, M.s, M.u, ∇f(x)
    s[:] = γs * s + (1 - γs) * g .* g
    Δx = -(sqrt.(u) .+ ε) ./ (sqrt.(s) .+ ε) .* g
    u[:] = γx * u + (1 - γx) * Δx .* Δx
    return x + Δx
end
```

5.8. Метод Adam

Метод адаптивной оценки момента, или *Adam* [82], также адаптирует скорости обучения к каждому параметру (алгоритм 5.8). Он сохраняет экспоненциально затухающий квадрат градиента, как методы RMSProp и Adadelta, и сохраняет экспоненциально затухающий градиент, аналогично моменту.

Инициализация градиента и квадрата градиента нулем вносит смещение. Этап исправления смещения помогает облегчить проблему.[6] Формулы каждой итерации метода Adam имеют вид:

[6] Согласно материалу оригинальной статьи, хорошими настройками по умолчанию являются $a = 0{,}001$; $\gamma_v = 0{,}9$; $\gamma_s = 0{,}999$ и $\varepsilon = 1 \cdot 10^{-8}$.

смещенный затухающий момент: $$\mathbf{v}^{(k+1)} = \gamma_v \mathbf{v}^{(k)} + (1-\gamma_v)\mathbf{g}^{(k)}, \tag{5.29}$$

смещенный затухающий квадрат градиента: $$\mathbf{s}^{(k+1)} = \gamma_s \mathbf{s}^{(k)} + (1-\gamma_s)(\mathbf{g}^{(k)} \odot \mathbf{g}^{(k)}), \tag{5.30}$$

скорректированный затухающий момент: $$\hat{\mathbf{v}}^{(k+1)} = \mathbf{v}^{(k+1)} \Big/ \left(1-\gamma_v^k\right), \tag{5.31}$$

исправленный затухающий квадрат градиента: $$\hat{\mathbf{s}}^{(k+1)} = \mathbf{s}^{(k+1)} \Big/ \left(1-\gamma_s^k\right), \tag{5.32}$$

следующая итерация: $$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha \hat{\mathbf{v}}^{(k+1)} \Big/ \left(c + \sqrt{\hat{\mathbf{s}}^{(k+1)}}\right). \tag{5.33}$$

Алгоритм 5.8. Метод ускоренного спуска Adam

```
mutable struct Adam <: DescentMethod
    α  # скорость обучения
    γv # коэффициент затухания
    γs # коэффициент затухания
    ε  # малое значение
    k  # счетчик шагов
    v  # оценка первого момента
    s  # оценка второго момента
end
function init!(M :: Adam, f, ∇f, x)
    M.k = 0
    M.v = zeros(length(x))
    M.s = zeros(length(x))
    return M
end
function step!(M::Adam, f, ∇f, x)
    α, γv, γs, ε, k = M.α, M.γv, M.γs, M.ε, M.k
    s, v, g = M.s, M.v, ∇f(x)
    v[:] = γv * v + (1 - γv) * g
    s[:] = γs * s + (1 - γs) * g .* g
    M.k = k += 1
    v_hat = v ./ (1 - γv ^ k)
    s_hat = s ./ (1 - γs ^ k)
    return x - α * v_hat ./ (sqrt.(s_hat) .+ ε)
end
```

5.9. Гиперградиентный спуск

Методы ускоренного спуска либо чрезвычайно чувствительны к скорости обучения, либо переходят на крупные шаги, чтобы адаптировать скорость обучения во время выполнения. Скорость обучения определяет, насколько чувствителен метод к величине градиента. Слишком высокая или слишком низкая скорость зачастую сильно влияет на производительность.

Гиперградиентный спуск [12] был разработан с пониманием того, что производная скорости обучения должна быть полезна для повышения производительности оптимизатора. *Гиперградиент* — это производная по гиперпараметру. Гиперградиентные алгоритмы снижают чувствительность к гиперпараметру, позволяя ему быстрее адаптироваться.

Метод гиперградиентного спуска применяет метод градиентного спуска к скорости обучения основного метода спуска. Метод требует вычисления частной производной целевой функции по отношению к скорости обучения. В методе градиентного спуска эта частная производная имеет вид:

$$\frac{\partial f\left(\mathbf{x}^{(k)}\right)}{\partial \alpha} = \left(\mathbf{g}^{(k)}\right)^{\mathrm{T}} \frac{\partial}{\partial \alpha}\left(\mathbf{x}^{(k-1)} - \alpha \mathbf{g}^{(k-1)}\right) = \tag{5.34}$$

$$= \left(\mathbf{g}^{(k)}\right)^{\mathrm{T}} \left(-\mathbf{g}^{(k-1)}\right). \tag{5.35}$$

Таким образом, вычисление гиперградиента требует отслеживания последнего градиента. Полученное правило модификации имеет вид:

$$\alpha^{(k+1)} = \alpha^{(k)} - \mu \frac{\partial f\left(\mathbf{x}^{(k)}\right)}{\partial \alpha} = \tag{5.36}$$

$$= \alpha^{(k)} + \mu \left(\mathbf{g}^{(k)}\right)^{\mathrm{T}} \mathbf{g}^{(k-1)}, \tag{5.37}$$

где μ — скорость гиперградиентного обучения.

Этот вывод может быть применен к любому методу градиентного спуска, который следует уравнению (4.1). Ниже приведены реализации гиперградиентной версии градиентного спуска (алгоритм 5.9) и метода Нестерова (алгоритм 5.10). Результаты сравнения этих методов представлены на рис. 5.7.

Алгоритм 5.9. Гиперградиентная форма градиентного спуска

```
mutable struct HyperGradientDescent <: DescentMethod
    α0 # начальная скорость обучения
    μ  # скорость обучения скорости обучения
    α  # текущая скорость обучения
```

```
    g_prev # предыдущий градиент
end
function init!(M :: HyperGradientDescent, f, ∇f, x)
    M.α = M.α0
    M.g_prev = zeros(length(x))
    return M
end
function step!(M :: HyperGradientDescent, f, ∇f, x)
    α, μ, g, g_prev = M.α, M.μ, ∇f(x), M.g_prev
    α = α + μ * (g ⋅ g_prev)
    M.g_prev, M.α = g, α
    return x - α * g
end
```

Алгоритм 5.10. Гиперградиентная форма метода Нестерова

```
mutable struct HyperNesterovMomentum <: DescentMethod
    α0 # начальная скорость обучения
    μ  # скорость обучения скорости обучения
    β  # коэффициент затухания момента
    v  # момент
    α  # текущая скорость обучения
    g_prev # предыдущий градиент
end
function init!(M :: HyperNesterovMomentum, f, ∇f, x)
    M.α = M.α0
    M.v = zeros(length(x))
    M.g_prev = zeros(length(x))
    return M
end
function step!(M :: HyperNesterovMomentum, f, ∇f, x)
    α, β, μ = M.α, M.β, M.μ
    v, g, g_prev = M.v, ∇f(x), M.g_prev
    α = α - μ *(g ⋅ (-g_prev - β * v))
    v[:] = β * v + g
    M.g_prev, M.α = g, α
    return x - α * (g + β * v)
end
```

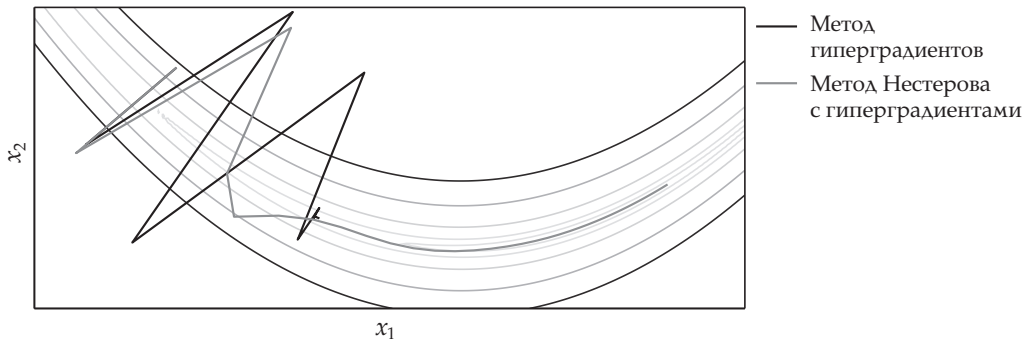


Рис. 5.7. Сравнение метода гиперградиентов и метода Нестерова на примере функции Розенброка с $b = 100$; см. приложение Б.6

5.10. Резюме

- Градиентный спуск следует в направлении наискорейшего спуска.
- Метод сопряженного градиента может автоматически адаптироваться к локальным долинам.
- Методы спуска с использованием момента наращивают прогресс в благоприятных направлениях.
- Широкий спектр методов ускоренного спуска использует специальные методы для ускорения спуска.
- Метод гиперградиентного спуска применяет метод градиентного спуска к скорости обучения основного метода спуска.

5.11. Упражнения

Упражнение 5.1. Вычислите градиент $\mathbf{x}^\mathsf{T}\mathbf{A}\mathbf{x} + \mathbf{b}^\mathsf{T}\mathbf{x}$, когда матрица $\mathbf{A}$ симметрична.

Упражнение 5.2. Примените метод градиентного спуска с единичным шаговым множителем к функции $f(x) = x^4$, начиная с точки, выбранной по вашему усмотрению. Вычислите две итерации.

Упражнение 5.3. Примените один шаг градиентного спуска к функции $f(x) = e^x + e^{-x}$ из точки $x^{(1)} = 10$ как с единичным шаговым множителем, так и с точным линейным поиском.

Упражнение 5.4. Метод сопряженных градиентов также можно использовать для определения направления поиска $\mathbf{d}$, когда в текущей точке доступна локальная

квадратичная модель функции. Пусть $\mathbf{d}$ — направление поиска. Тогда модель имеет вид

$$q(\mathbf{d}) = \mathbf{d}^{\mathrm{T}}\mathbf{H}\mathbf{d} + \mathbf{b}^{\mathrm{T}}\mathbf{d} + \mathbf{c}$$

для симметричной матрицы $\mathbf{H}$. Как выглядит гессиан в этом случае? Каков градиент q при $\mathbf{d} = 0$? Что может пойти не так, если метод квадратичного градиента применяется к квадратичной модели, чтобы определить направление поиска $\mathbf{d}$?

Упражнение 5.5. Какие преимущества метод Нестерова имеет над методом моментов?

Упражнение 5.6. Почему метод сопряженных градиентов лучше, чем метода наискорейшего спуска?

Упражнение 5.7. Каково нормализованное направление спуска на первой итерации метода сопряженных градиентов для функции $f(x, y) = x^2 + xy + y^2 + 5$ при начальной точке $(x, y) = (1, 1)$? Какова конечная точка после двух шагов метода сопряженных градиентов?

Упражнение 5.8. Допустим, у нас есть полиномиальная функция f такая, что $f(\mathbf{x}) > 2$ для всех $\mathbf{x}$ в трехмерном евклидовом пространстве. Предположим, что мы используем метод наискорейшего спуска с длиной шага, оптимизированной на каждом шаге, и мы хотим найти точку локального минимума функции f. Если на шаге k наше ненормированное направление спуска задается вектором [1, 2, 3], возможно ли, чтобы ненормированное направление спуска на шаге $k + 1$ задавалось вектором [0, 0, −3]? Обоснуйте свой ответ.

Глава 6

Методы второго порядка

В предыдущей главе основное внимание уделялось методам оптимизации, включающим аппроксимации первого порядка целевой функции с использованием градиента. Эта глава посвящена аппроксимациям *второго порядка*, которые используют для выбора направления поиска вторую производную в одномерной оптимизации или гессиан в многомерной оптимизации. Эта дополнительная информация может помочь улучшить локальную модель, используемую для информирования о выборе направлений и длин шагов в алгоритмах спуска.

6.1. Метод Ньютона

Знание значения функции и градиента в расчетной точке может помочь определить направление движения, но эта информация первого порядка напрямую не помогает определить, как далеко нужно продвинуться, чтобы достичь локального минимума. Информация второго порядка, с другой стороны, позволяет выполнить квадратичную аппроксимацию целевой функции и приблизить правильный размер шага для достижения локального минимума, как показано на рис. 6.1. Как было показано при построении квадратичной аппроксимации в главе 3, можно

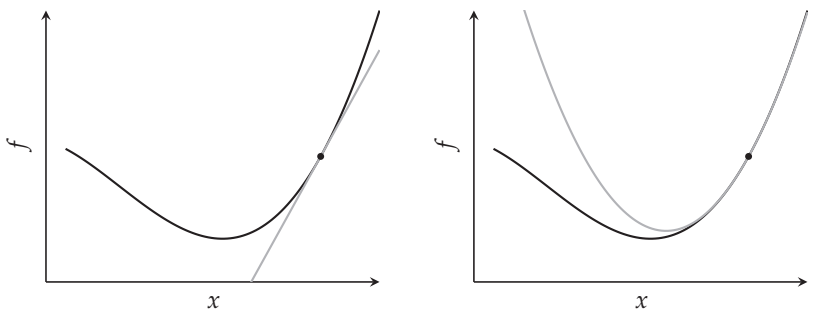


Рис. 6.1. Сравнение аппроксимаций первого и второго порядка. Чашеобразные квадратичные приближения имеют единственную точку, в которой производная равна нулю

аналитически вычислить точку, в которой квадратичная аппроксимация имеет нулевой градиент. Затем мы можем использовать эту точку в качестве следующей итерации, чтобы приблизиться к локальному минимуму.

При одномерной оптимизации квадратичная аппроксимация в окрестности точки $x^{(k)}$ следует из разложения Тейлора второго порядка:

$$q(x) = f\left(x^{(k)}\right) + \left(x - x^{(k)}\right) f'\left(x^{(k)}\right) + \frac{\left(x - x^{(k)}\right)^2}{2} f''\left(x^{(k)}\right). \tag{6.1}$$

Приравнивая производную к нулю и решая возникшее уравнение, получаем итерационные формулы для *метода Ньютона*:

$$\frac{\partial}{\partial x} q(x) = f'\left(x^{(k)}\right) + \left(x - x^{(k)}\right) f''\left(x^{(k)}\right) = 0, \tag{6.2}$$

$$x^{(k+1)} = x^{(k)} - \frac{f'\left(x^{(k)}\right)}{f''\left(x^{(k)}\right)}. \tag{6.3}$$

Эта итерация показана на рис. 6.2.

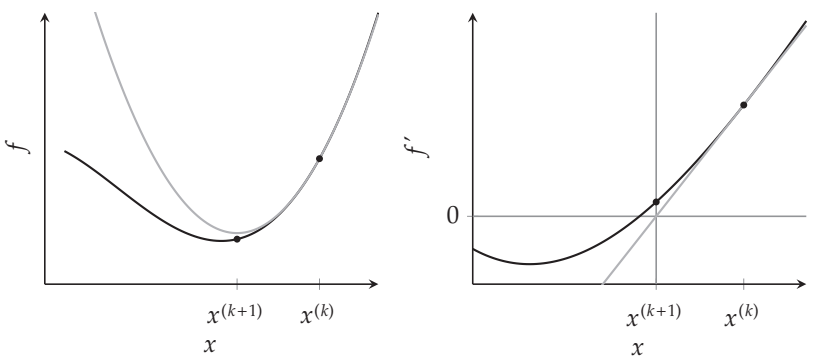


Рис. 6.2. Метод Ньютона можно интерпретировать как метод поиска корня, применяемый к функции f', который итеративно улучшает расчетную точку, строя касательную линию в точке $(x, f'(x))$, находя ее пересечение с осью x и используя это значение x в качестве следующей расчетной точки

Итерационная формула метода Ньютона включает деление на вторую производную. Итерация не определена, если вторая производная равна нулю, т.е. когда квадратичное приближение является горизонтальной линией. Кроме того, если вторая производная очень близка к нулю, возникает неустойчивость. В этом случае следующее итерационное приближение будет находиться очень далеко от текущей расчетной точки, и локальное квадратичное приближение станет некорректным. Плохие локальные приближения могут привести к низкой точности метода Ньютона. На рис. 6.3 показаны три вида проблем.

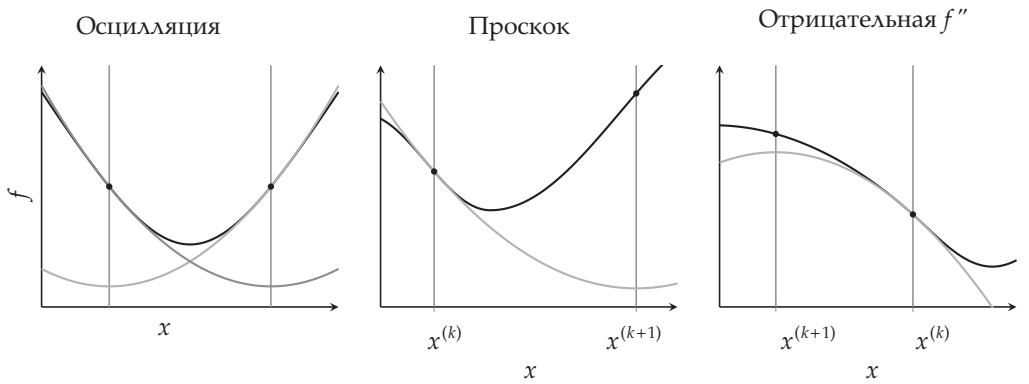


Рис. 6.3. Примеры проблем, связанных с методом Ньютона

Метод Ньютона, как правило, быстро сходится, если область имеет форму чаши и расчетная точка находится достаточно близко к локальному минимуму. Он имеет *квадратичную сходимость*, т.е. разность между минимумом и итерационным приближением на каждой итерации возводится в квадрат. Эта скорость сходимости имеет место для метода Ньютона, начиная с $x^{(1)}$ на расстоянии не больше чем δ от корня x*, если[1]

- $f''(x) \neq 0$ для всех точек в интервале I,
- $f'''(x)$ непрерывно в интервале I,
- $\frac{1}{2}\left|\frac{f'''\left(x^{(1)}\right)}{f''\left(x^{(1)}\right)}\right| < c\left|\frac{f'''\left(x^*\right)}{f''\left(x^*\right)}\right|$ для некоторой константы $c < \infty$,

где $I = [x^* - \delta, x^* + \delta]$. Конечное условие защищает метод от проскакивания мимо решения.

Метод Ньютона можно обобщить до многомерной оптимизации (алгоритм 6.1). Многомерное разложение Тейлора второго порядка в окрестности точки $\mathbf{x}^{(k)}$ имеет вид:

$$f(\mathbf{x}) \approx q(\mathbf{x}) = f\left(\mathbf{x}^{(k)}\right) + \left(\mathbf{g}^{(k)}\right)^{\mathrm{T}}\left(\mathbf{x} - \mathbf{x}^{(k)}\right) + \frac{1}{2}\left(\mathbf{x} - \mathbf{x}^{(k)}\right)^{\mathrm{T}} \mathbf{H}^{(k)}\left(\mathbf{x} - \mathbf{x}^{(k)}\right), \quad (6.4)$$

где $\mathbf{g}^{(k)}$ и $\mathbf{H}^{(k)}$ — градиент и гессиан в точке $\mathbf{x}^{(k)}$ соответственно.

Вычислим градиент и приравняем его к нулю:

$$\nabla q\left(\mathbf{x}^{(k)}\right) = \mathbf{g}^{(k)} + \mathbf{H}^{(k)}\left(\mathbf{x} - \mathbf{x}^{(k)}\right) = 0. \quad (6.5)$$

Решая это уравнение, мы получаем метод Ньютона в многомерном варианте:

---

[1] Последнее условие обеспечивает *достаточную близость*, гарантируя, что функция достаточно точно аппроксимируется разложением Тейлора [149].

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \left(\mathbf{H}^{(k)}\right)^{-1}\mathbf{g}^{(k)}. \tag{6.6}$$

Если функция f квадратичная, а ее гессиан положительно определен, то итерационный процесс сходится к глобальному минимуму за один шаг. Для общих функций метод Ньютона обычно завершается, как только $\mathbf{x}$ перестает меняться более чем на заданную величину.[2]

$$\mathbf{d}^{(k)} = -\left(\mathbf{H}^{(k)}\right)^{-1}\mathbf{g}^{(k)}. \tag{6.7}$$

Пример 6.1. Метод Ньютона может быть использован для минимизации функции Бута (см. приложение Б.2)

Пусть $\mathbf{x}^{(1)} = [9, 8]$. Применим метод Ньютона для минимизации функции Бута (Booth):

$$f(\mathbf{x}) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2.$$

Градиент функции Бута равен

$$\nabla f(\mathbf{x}) = [10x_1 + 8x_2 - 34,\ 8x_1 + 10x_2 - 38].$$

Гессиан функции Бута равен

$$\mathbf{H}(\mathbf{x}) = \begin{bmatrix} 10 & 8 \\ 8 & 10 \end{bmatrix}.$$

Первая итерация метода Ньютона дает следующий результат:

$$\mathbf{x}^{(2)} = \mathbf{x}^{(1)} - \left(\mathbf{H}^{(1)}\right)^{-1}\mathbf{g}^{(1)} = \begin{bmatrix} 9 \\ 8 \end{bmatrix} - \begin{bmatrix} 10 & 8 \\ 8 & 10 \end{bmatrix}^{-1}\begin{bmatrix} 10 \cdot 9 + 8 \cdot 8 - 34 \\ 8 \cdot 9 + 10 \cdot 8 - 38 \end{bmatrix} =$$

$$= \begin{bmatrix} 9 \\ 8 \end{bmatrix} - \begin{bmatrix} 10 & 8 \\ 8 & 10 \end{bmatrix}^{-1}\begin{bmatrix} 120 \\ 114 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}.$$

Градиент в точке $\mathbf{x}^{(2)}$ равен нулю, поэтому метод сходится за одну итерацию. Гессиан положительно определен во всех точках, поэтому точка $\mathbf{x}^{(2)}$ является точкой глобального минимума.

Метод Ньютона также может использоваться для выбора направления спуска в линейном поиске либо может быть изменен с учетом шагового множителя.[3] Меньшие шаги по направлению к минимуму или линейный поиск в направле-

[2] Условия прекращения методов спуска приведены в главе 5.

[3] См. главу 5.

нии спуска могут повысить надежность метода. Направление спуска задается вектором[4].

Алгоритм 6.1. Метод Ньютона, который принимает градиент функции ∇`f`, гессиан целевой функции `H`, начальную точку `x`, допуск на размер шага ε и максимальное количество итераций `k_max`

```
function newtons_method(∇f, H, x, ε, k_max)
    k, Δ = 1, Inf
    while norm(Δ) > ε && k ≤ k_max
        Δ = H(x) \ ∇f(x)
        x -= Δ
        k += 1
    end
    return x
end
```

6.2. Метод секущих

Метод Ньютона для минимизации одномерной функции требует знания первой и второй производных f' и f''. Во многих случаях значение первой производной известно, а второй — нет. Метод секущих (алгоритм 6.2) применяет метод Ньютона с использованием оценок второй производной и, следовательно, требует знания только f'. Это свойство делает метод секущих более удобным для использования на практике.

Метод секущих использует последние две итерации для аппроксимации второй производной:

$$f''\left(x^{(k)}\right) \approx \frac{f'\left(x^{(k)}\right) - f'\left(x^{(k-1)}\right)}{x^{(k)} - x^{(k-1)}}. \tag{6.8}$$

Эта оценка подставляется в метод Ньютона:

$$x^{(k+1)} \leftarrow x^{(k)} - \frac{x^{(k)} - x^{(k-1)}}{f'\left(x^{(k)}\right) - f'\left(x^{(k-1)}\right)} f'\left(x^{(k)}\right). \tag{6.9}$$

Метод секущих требует наличия дополнительной начальной расчетной точки. Он имеет те же недостатки, что и метод Ньютона, и может требовать больше итераций для сходимости из-за аппроксимации второй производной.

[4] Направление спуска, заданное методом Ньютона, аналогично *естественному градиенту* или *ковариантному градиенту* [2].

6.3. Квазиньютоновские методы

Так же, как метод секущих аппроксимирует f'' в одномерном случае, *квазиньютоновские методы* аппроксимируют обратный гессиан. Итерации квазиньютоновского метода имеют следующий вид:

$$\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} - \alpha^{(k)}\mathbf{Q}^{(k)}\mathbf{g}^{(k)}, \tag{6.10}$$

где $\alpha^{(k)}$ — скалярный шаговый множитель, а $\mathbf{Q}^{(k)}$ аппроксимирует обратный гессиан в точке $\mathbf{x}^{(k)}$.

Алгоритм 6.2. Метод секущих для минимизации одномерных функций. Входные данные состоят из первой производной `f'` целевой функции, двух начальных точек `x0` и `x1` и допустимой погрешности `ε`. Возвращается конечная координата `x`

```
function secant_method(f', x0, x1, ε)
    g0 = f'(x0)
    Δ = Inf
    while abs(Δ) > ε
        g1 = f'(x1)
        Δ = (x1 - x0) / (g1 - g0) * g1
        x0, x1, g0 = x1, x1 - Δ, g1
    end
    return x1
end
```

В этих методах $\mathbf{Q}^{(1)}$ обычно задают как единичную матрицу и затем применяют итерационные формулы, чтобы учесть информацию, полученную на каждой итерации. Чтобы упростить формулы для различных квазиньютоновских методов, определим следующие величины:

$$\boldsymbol{\gamma}^{(k+1)} = \mathbf{g}^{(k+1)} - \mathbf{g}^{(k)}, \tag{6.11}$$

$$\boldsymbol{\delta}^{(k+1)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}. \tag{6.12}$$

Метод Дэвидона–Флетчера–Пауэлла (Davidon–Fletcher–Powell — DFP) (алгоритм 6.3) использует следующую формулу:[5]

$$\mathbf{Q} \leftarrow \mathbf{Q} - \frac{\mathbf{Q}\boldsymbol{\gamma}\boldsymbol{\gamma}^{\mathrm{T}}\mathbf{Q}}{\boldsymbol{\gamma}^{\mathrm{T}}\mathbf{Q}\boldsymbol{\gamma}} + \frac{\boldsymbol{\delta}\boldsymbol{\delta}^{\mathrm{T}}}{\boldsymbol{\delta}^{\mathrm{T}}\boldsymbol{\gamma}}, \tag{6.13}$$

где все члены в правой части вычисляются на итерации k.

[5] Первоначальная концепция была изложена в техническом отчете [33]. Позднее она была опубликована в статье [34]. Впоследствии этот метод был модифицирован Флетчером и Пауэллом [48].

Итерационное приближение матрицы $\mathbf{Q}$ в методе DFP имеет три свойства.

1. Она остается симметричной и положительно определенной.
2. Если $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} + \mathbf{b}^{\mathrm{T}}\mathbf{x} + c$, то $\mathbf{Q} = \mathbf{A}^{-1}$. Таким образом, метод DFP имеет те же свойства сходимости, что и метод сопряженных градиентов.
3. Для задач большой размерности сохранение и уточнение матрицы $\mathbf{Q}$ может потребовать значительных ресурсов по сравнению с другими методами, такими как метод сопряженных градиентов.

Алгоритм 6.3. Метод спуска Дэвидона–Флетчера–Пауэлла

```
mutable struct DFP <: DescentMethod
    Q
end
function init!(M :: DFP, f, ∇f, x)
    m = length(x)
    M.Q = Matrix(1.0I, m, m)
    return M
end
function step!(M :: DFP, f, ∇f, x)
     Q, g = M.Q, ∇f(x)
     x' = line_search(f, x, -Q * g)
     g' = ∇f(x')
     δ = x' - x
     γ = g' - g
     Q[:] = Q - Q * γ * γ' * Q / (γ' * Q * γ) + δ * δ' / (δ' * γ)
     return x'
end
```

Альтернатива методу DFP, которая называется *методом Бройдена–Флетчера–Голдфарба–Шанно* (Broyden–Fletcher–Goldfarb–Shanno — BFGS) (алгоритм 6.4), использует следующую формулу [50]:

$$\mathbf{Q} \leftarrow \mathbf{Q} - \frac{\boldsymbol{\delta}\boldsymbol{\gamma}^{\mathrm{T}}\mathbf{Q} + \mathbf{Q}\boldsymbol{\gamma}\boldsymbol{\delta}^{\mathrm{T}}}{\boldsymbol{\delta}^{\mathrm{T}}\boldsymbol{\gamma}} + \left(1 + \frac{\boldsymbol{\gamma}^{\mathrm{T}}\mathbf{Q}\boldsymbol{\gamma}}{\boldsymbol{\delta}^{\mathrm{T}}\boldsymbol{\gamma}}\right)\frac{\boldsymbol{\delta}\boldsymbol{\delta}^{\mathrm{T}}}{\boldsymbol{\delta}^{\mathrm{T}}\boldsymbol{\gamma}}, \tag{6.14}$$

Алгоритм 6.4. Метод спуска Бройдена–Флетчера–Гольдфарба–Шанно

```
mutable struct BFGS <: DescentMethod
    Q
end
```

```
function init!(M :: BFGS, f, ∇f, x)
    m = length(x)
    M.Q = Matrix(1.0I, m, m)
    return M
end
function step!(M :: BFGS, f, ∇f, x)
    Q, g = M.Q, ∇f(x)
    x' = line_search(f, x, -Q * g)
    g' = ∇f(x')
    δ = x' - x
    γ = g' - g
    Q[:] = Q - (δ * γ' * Q + Q * γ * δ) / (δ' * γ) +
               (1 + (γ' * Q * γ) / (δ' * γ))[1] * (δ * δ') / (δ' * γ)
    return x'
end
```

Метод BFGS работает лучше, чем DFP с приближенным линейным поиском, но все еще использует плотную матрицу $n \times n$. Для очень больших задач, в которых объем памяти является проблемой, в качестве замены BFGS может использоваться *метод BFGS с ограниченной памятью* (алгоритм 6.5), или *L-BFGS* [113]. Метод L-BFGS хранит последние m значений векторов δ и γ, а не полный обратный гессиан, где $i = 1$ индексирует самое старое значение, а $i = m$ индексирует самое последнее.

Алгоритм 6.5. Метод спуска BFGS с ограниченной памятью, позволяющий избежать хранения приближенного обратного гессиана. Параметр `m` определяет глубину истории. Тип `LimitedMemoryBFGS` также хранит разности шагов `δs`, изменения градиента `γs` и векторы хранения `qs`

```
mutable struct LimitedMemoryBFGS <: DescentMethod
    m
    δs
    γs
    qs
end
function init!(M :: LimitedMemoryBFGS, f, ∇f, x)
    M.δs = []
    M.γs = []
    M.qs = []
    return M
end
```

```
function step!(M :: LimitedMemoryBFGS, f, ∇f, x)
    δs, γs, qs, g = M.δs, M.γs, M.qs, ∇f(x)
    m = length(δs)
    if m > 0
        q = g
        for i in m : -1 : 1
            qs[i] = copy(q)
            q -= (δs[i] ⋅ q) / (γs[i] ⋅ δs[i]) * γs[i]
        end
        z = (γs[m] .* δs[m] .* q) / (γs[m] ⋅ γs[m])
        for i in 1 : m
            z += δs[i] * (δs[i] ⋅ qs[i] - γs[i].z) / (γs[i] ⋅ δs[i])
        end
        x′ = line_search(f, x, -z)
    else
        x′ = line_search(f, x, -g)
    end
    g′ = ∇f(x′)
    push!(δs, x′ - x); push!(γs, g′ - g)
    push!(qs, zeros(length(x)))
    while length(δs) > M.m
       popfirst!(δs); popfirst!(γs); popfirst!(qs)
    end
    return x′
end
```

Процесс вычисления направления спуска $\mathbf{d}$ из точки $\mathbf{x}$ начинается с вычисления вектора $\mathbf{q}^{(m)} = \nabla f(\mathbf{x})$. Остальные векторы $\mathbf{q}^{(i)}$ для i от $m - 1$ до 1 вычисляются по формуле

$$\mathbf{q}^{(i)} = \mathbf{q}^{(i+1)} - \frac{\left(\boldsymbol{\delta}^{(i+1)}\right)^{\mathrm{T}} \mathbf{q}^{(i+1)}}{\left(\boldsymbol{\gamma}^{(i+1)}\right)^{\mathrm{T}} \boldsymbol{\delta}^{(i+1)}} \boldsymbol{\gamma}^{(i+1)}. \tag{6.15}$$

Эти векторы используются для вычисления других $m + 1$ векторов, начиная с вектора

$$\mathbf{z}^{(0)} = \frac{\boldsymbol{\gamma}^{(m)} \odot \boldsymbol{\delta}^{(m)} \odot \mathbf{q}^{(m)}}{\left(\boldsymbol{\gamma}^{(m)}\right)^{\mathrm{T}} \boldsymbol{\gamma}^{(m)}} \tag{6.16}$$

и продолжая векторами $\mathbf{z}^{(i)}$ для i от 1 до m согласно формуле

$$\mathbf{z}^{(i)} = \mathbf{z}^{(i-1)} + \boldsymbol{\delta}^{(i-1)} \left(\frac{\left(\boldsymbol{\delta}^{(i-1)}\right)^{\mathrm{T}} \mathbf{q}^{(i-1)}}{\left(\boldsymbol{\gamma}^{(i-1)}\right)^{\mathrm{T}} \boldsymbol{\delta}^{(i-1)}} - \frac{\left(\boldsymbol{\gamma}^{(i-1)}\right)^{\mathrm{T}} \mathbf{z}^{(i-1)}}{\left(\boldsymbol{\gamma}^{(i-1)}\right)^{\mathrm{T}} \boldsymbol{\delta}^{(i-1)}} \right). \tag{6.17}$$

Направление спуска определяется вектором $\mathbf{d} = -\mathbf{z}^{(m)}$.

Для минимизации обратный гессиан $\mathbf{Q}$ должен оставаться положительно определенным. Начальный гессиан часто выбирается в виде диагональной матрицы

$$\mathbf{Q}^{(1)} = \frac{\boldsymbol{\gamma}^{(1)} \left(\boldsymbol{\delta}^{(1)}\right)^{\mathrm{T}}}{\left(\boldsymbol{\gamma}^{(1)}\right)^{\mathrm{T}} \boldsymbol{\gamma}^{(1)}}. \tag{6.18}$$

Вычисление диагонали для вышеприведенного выражения и подстановка результата в $\mathbf{z}^{(1)} = \mathbf{Q}^{(1)}\mathbf{q}^{(1)}$ приводят к формуле для $\mathbf{z}^{(1)}$.

Квазиньютоновские методы, обсуждаемые в этом разделе, сравниваются на рис. 6.4. Их точность часто совпадает.

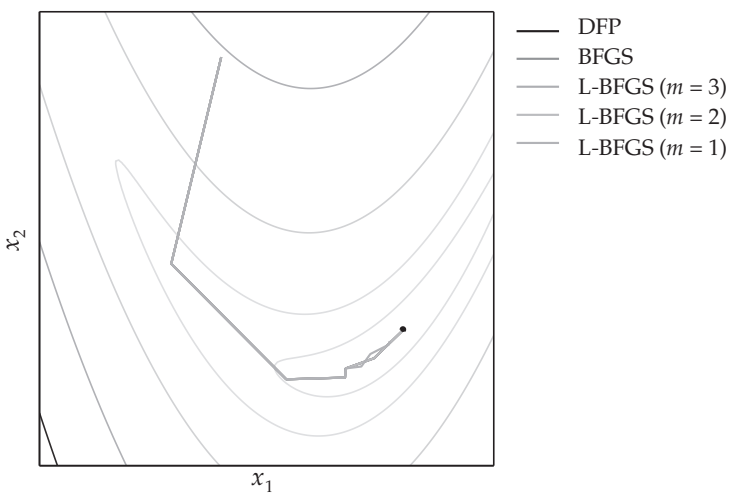


Рис. 6.4. Результаты сравнения нескольких квазиньютоновских методов на примере функции Розенброка (см. приложение Б.6). Все методы имеют почти идентичные итерационные приближения, причем метод L-BFGS заметно отклоняется только тогда, когда глубина его истории равна единице ($m = 1$)

6.4. Резюме

- Включение информации второго порядка в методы спуска часто ускоряет сходимость.

- Метод Ньютона — это метод поиска корня, который использует информацию второго порядка для быстрого спуска к локальному минимуму.
- Метод секущих и квазиньютоновские методы аппроксимируют метод Ньютона, когда информация второго порядка недоступна напрямую.

6.5. Упражнения

Упражнение 6.1. Какое преимущество дает информация второго порядка о сходимости, которой не хватает информации первого порядка?

Упражнение 6.2. Допустим, что мы ищем корень одномерной функции. Как использовать метод Ньютона вместо метода бисекции?

Упражнение 6.3. Примените метод Ньютона к функции $f(x) = x^2$ из начальной точки по вашему выбору. Сколько шагов нужно, чтобы метод сошелся?

Упражнение 6.4. Примените метод Ньютона к функции $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{H}\mathbf{x}$, начиная с точки $\mathbf{x}^{(1)} = [1, 1]$. Что вы заметили? Выберите матрицу $\mathbf{H}$ следующим образом:

$$H = \begin{bmatrix} 1 & 0 \\ 0 & 1000 \end{bmatrix}. \tag{6.19}$$

Затем примените метод градиентного спуска к той же задаче оптимизации, используя ненормализованный градиент. Выполните два шага алгоритма. Что вы заметили? Наконец, примените метод сопряженных градиентов. Сколько шагов нужно, чтобы метод сошелся?

Упражнение 6.5. Сравните метод Ньютона и метод секущих метод для функции $f(x) = x^2 + x^4$ с $x^{(1)} = -3$ и $x^{(0)} = -4$. Выполните по 10 итераций каждого метода. Постройте два графика.

1. Графики функции f и итерационного приближения для каждого метода.
2. График производной f' и решения x. Продемонстрируйте прогресс каждого метода, рисуя линии от $(x^{(i)}, f'(x^{(i)}))$ до $(x^{(i+1)}, 0)$ до $(x^{(i+1)}, f'(x^{(i+1)}))$ на каждом шаге. Какой вывод можно сделать из этого сравнения?

Упражнение 6.6. Приведите пример последовательности точек $x^{(1)}$, $x^{(2)}$, ... и функции f такой, что $f(x^{(1)}) > f(x^{(2)}) > \ldots$, но последовательность все же не сходится к локальному минимуму. Предположим, что функция f ограничена снизу.

Упражнение 6.7. В чем преимущество квазиньютоновского метода над методом Ньютона?

Упражнение 6.8. Приведите пример, где итерационное приближение BFGS не существует. Что бы вы сделали в этом случае?

Упражнение 6.9. Предположим, что задана функция $f(x) = (x_1 + 1)^2 + (x_2 + 3)^2 + 4$. Какова будет конечная точка после одного шага метода Ньютона, если мы начнем с начала координат?

Упражнение 6.10. Сформулируем задачу оптимизации, из которой был получен итерационный метод Дэвидона–Флетчера–Пауэлла. Начнем с квадратичного приближения в точке $\mathbf{x}^{(k)}$:

$$f^{(k)}\left(\mathbf{x}\right) = y^{(k)} + \left(\mathbf{g}^{(k)}\right)^{\mathrm{T}}\left(\mathbf{x} - \mathbf{x}^{(k)}\right) + \frac{1}{2}\left(\mathbf{x} - \mathbf{x}^{(k)}\right)^{\mathrm{T}}\mathbf{H}^{(k)}\left(\mathbf{x} - \mathbf{x}^{(k)}\right),$$

где $y^{(k)}$, $\mathbf{g}^{(k)}$ и $\mathbf{H}^{(k)}$ — это значение целевой функции, истинный градиент и положительно определенная аппроксимация гессиана в точке $\mathbf{x}^{(k)}$.

Следующая итерация выбирается с помощью линейного поиска:

$$\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} - \alpha^{(k)}\left(\mathbf{H}^{(k)}\right)^{-1}\mathbf{g}^{(k)}.$$

Мы можем построить новое квадратичное приближение $f^{(k+1)}$ в точке $\mathbf{x}^{(k+1)}$. Это приближение должно обеспечить правильность вычисления локальной функции

$$f^{(k+1)}\left(\mathbf{x}^{(k+1)}\right) = y^{(k+1)},$$

локального градиента

$$\nabla f^{(k+1)}\left(\mathbf{x}^{(k+1)}\right) = \mathbf{g}^{(k+1)}$$

и предыдущего градиента

$$\nabla f^{(k+1)}\left(\mathbf{x}^{(k)}\right) = \mathbf{g}^{(k)}.$$

Покажите, что итерационное приближение матрицы Гессе $\mathbf{H}^{(k+1)}$ требует выполнения следующего условия:[6]

$$\mathbf{H}^{(k+1)}\boldsymbol{\delta}^{(k+1)} = \boldsymbol{\gamma}^{(k+1)}.$$

Затем покажите, что для того, чтобы матрицы $\mathbf{H}^{(k+1)}$ была положительно определенной, нам требуется условие[7]

$$\left(\boldsymbol{\delta}^{(k+1)}\right)^{\mathrm{T}}\boldsymbol{\gamma}^{(k+1)} > 0.$$

---

[6] Это условие называется *уравнением секущей*. Векторы $\boldsymbol{\delta}$ и $\boldsymbol{\gamma}$ определены в формуле (6.11).

[7] Это условие называется *условием кривизны.* Оно следует из условий Вольфе на этапе линейного поиска.

Наконец, предполагая, что условие кривизны выполняется, объясните, почему этот итерационный процесс решает следующую задачу оптимизации, чтобы получить $\mathbf{H}^{(k+1)}$:[8]

$$\min_{\mathbf{H}} \left\| \mathbf{H} - \mathbf{H}^{(k)} \right\|$$

при условии, что $\mathbf{H} = \mathbf{H}^{\mathrm{T}}$,

$$\mathbf{H}\boldsymbol{\delta}^{(k+1)} = \boldsymbol{\gamma}^{(k+1)},$$

где $\left\| \mathbf{H} - \mathbf{H}^{(k)} \right\|$ — *матричная норма*, определяющая расстояние между матрицами $\mathbf{H}$ и $\mathbf{H}^{(k)}$.

[8] Итерационное приближение Дэвидона–Флетчера–Пауэлла получается путем решения такой задачи оптимизации для получения аналитического решения для аппроксимации обратного гессиана.

Глава 7

Прямые методы

Прямые методы опираются исключительно на целевую функцию *f*. Эти методы также называются *методами нулевого порядка*, *черного ящика*, *поиска по шаблону* или *методами без производных*. Прямые методы не полагаются на информацию о производной при поиске направления к локальному минимуму или определении момента, когда они достигли локального минимума. Они используют другие критерии, чтобы выбрать следующее направление поиска и оценить выполнение критерия сходимости.

7.1. Циклический поиск координат

Циклический поиск координат, также известный как *покоординатный спуск* и *поиск такси*, просто чередует направления вдоль координатных осей в ходе линейного поиска. Поиск начинается с начального $\mathbf{x}^{(1)}$ и оптимизирует первое приближение:

$$\mathbf{x}^{(2)} = \arg\min_{x_1} f\left(x_1, x_2^{(1)}, x_3^{(1)}, \ \ldots, x_n^{(1)}\right). \tag{7.1}$$

Решив эту задачу, он оптимизирует следующую координату:

$$\mathbf{x}^{(3)} = \arg\min_{x_2} f\left(x_1^{(2)}, x_2, x_3^{(2)}, \ \ldots, x_n^{(2)}\right). \tag{7.2}$$

Этот процесс эквивалентен выполнению последовательности линейного поиска по набору из *n* базисных векторов, где все элементы *i*-го базисного вектора равны нулю, за исключением *i*-го компонента, который равен единице (алгоритм 7.1). Например, третья базисная функция, обозначаемая $\mathbf{e}^{(3)}$, в четырехмерном пространстве имеет вид:

$$\mathbf{e}^{(3)} = [0, 0, 1, 0] \tag{7.3}$$

Алгоритм 7.1. Функция для построения `i`-го базисного вектора длины `n`

```
basis(i, n) = [k == i ? 1.0 : 0.0 for k in 1 : n]
```

На рис. 7.1 показан пример поиска в двумерном пространстве.

Подобно методу наискорейшего спуска, циклический координатный поиск координат гарантированно либо уменьшает, либо оставляет неизменной значение

функции на каждой итерации. Отсутствие значительного улучшения после полного цикла по всем координатам указывает на то, что метод сходится. Реализация описана алгоритмом 7.2. Как показано на рис. 7.2, циклический поиск координат может не найти даже локальный минимум.

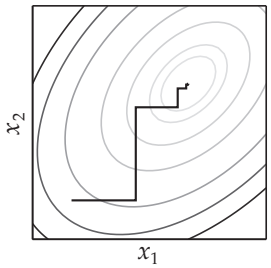


Рис. 7.1. Циклический координатный спуск чередует направления вдоль осей координат

Алгоритм 7.2. Метод циклического покоординатного спуска принимает в качестве входных данных целевую функцию `f` и начальную точку `x` и работает до тех пор, пока размер шага за полный цикл не станет меньше заданного допуска

```
function cyclic_coordinate_descent(f, x, ε)
    Δ, n = Inf, length(x)
    while abs(Δ) > ε
        x′ = copy(x)
        for i in 1:n
            d = basis(i, n)
            x = line_search(f, x, d)
        end
        Δ = norm(x - x′)
    end
    return x
end
```

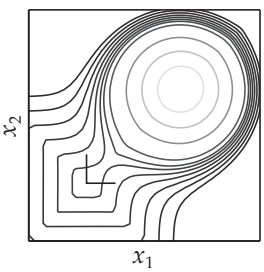


Рис. 7.2. Выше приведен пример того, как циклический поиск координат может застрять. Перемещение в любом из направлений координат приведет только к увеличению функции f, но перемещение по диагонали, невозможное при циклическом поиске координат, может привести к снижению f

Метод может быть дополнен шагом ускорения, чтобы помочь ему пересечь диагональные долины. Для каждого полного цикла, начиная с оптимизации приближения $\mathbf{x}^{(1)}$ вдоль вектора $\mathbf{e}^{(1)}$ и заканчивая приближением $\mathbf{x}^{(n+1)}$ после оптимизации вдоль вектора $\mathbf{e}^{(n)}$ в направлении $\mathbf{x}^{(n+1)} - \mathbf{x}^{(1)}$ проводится дополнительный линейный поиск. Реализация метода представлена в алгоритме 7.3, и пример траектории поиска показан на рис. 7.3.

7.2. Метод Пауэлла

Метод Пауэлла[1] может выполнять поиск в направлениях, которые не ортогональны друг другу. Метод может автоматически настраиваться на длинные узкие долины, которые в противном случае могут потребовать большого количества итераций для циклического покоординатного спуска или других методов, которые осуществляют поиск в направлениях, ориентированных по оси.

Рис. 7.3. Добавление шага ускорения к циклическому покоординатному спуску помогает пересечь долины. Показаны шесть шагов как для оригинальной, так и для ускоренной версии

Алгоритм 7.3. Метод циклического покоординатного спуска с шагом ускорения принимает в качестве входных данных целевую функцию `f` и начальную точку `x` и работает до тех пор, пока размер шага за полный цикл не станет меньше заданного допуска

```
function cyclic_coordinate_descent_with_acceleration_step(f, x, ε)
    Δ, n = Inf, length(x)
    while abs(Δ) > ε
        x′ = copy(x)
        for i in 1 : n
            d = basis(i, n)
            x = line_search(f, x, d)
        end
        x = line_search(f, x, x - x′) # шаг ускорения
```

[1] Метод Пауэлла был впервые представлен в статье [123]. Обзор представлен в работе [124].

```
        Δ = norm(x - x′)
    end
    return x
end
```

Алгоритм ведет список направлений поиска $\mathbf{u}^{(1)}$, ..., $\mathbf{u}^{(n)}$, которые изначально являются векторами базисных координат, $\mathbf{u}^{(i)} = \mathbf{e}^{(i)}$ для всех i. Начиная с $\mathbf{x}^{(1)}$, метод Пауэлла последовательно выполняет линейный поиск в каждом направлении, каждый раз уточняя расчетную точку:

$$\mathbf{x}^{(i+1)} \leftarrow \texttt{line\_search}\left(f, \mathbf{x}^{(i)}, \mathbf{u}^{(i)}\right) \text{ для всех } i \text{ в } \{1, ..., n\} \tag{7.4}$$

Затем все направления поиска смещаются вниз на один индекс, отбрасывая самое старое направление поиска, $\mathbf{u}^{(1)}$:

$$\mathbf{u}^{(i)} \leftarrow \mathbf{u}^{(i+1)} \text{ для всех } i \text{ в } \{1, ..., n-1\} \tag{7.5}$$

Последнее направление поиска заменяется направлением от $\mathbf{x}^{(1)}$ до $\mathbf{x}^{(n+1)}$, которое является общим направлением прогресса за последний цикл:

$$\mathbf{u}^{(n)} \leftarrow \mathbf{x}^{(n+1)} - \mathbf{x}^{(1)}, \tag{7.6}$$

а другой линейный поиск проводится вдоль нового направления, чтобы получить новый $\mathbf{x}^{(1)}$. Этот процесс повторяется до выполнения критерия сходимости. Реализация описана в алгоритме 7.4 Пример поисковой траектории показан на рис. 7.4 .

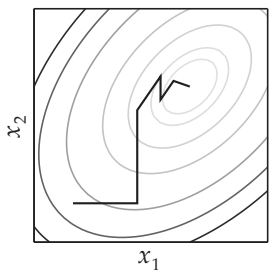


Рис. 7.4. Метод Пауэлла начинается так же, как циклический покоординатный спуск, но итеративно исследует сопряженные направления

Пауэлл показал, что для квадратичных функций после k полных итераций последние k направлений будут взаимно сопряжены. Напомним, что n полных итераций линейного поиска по взаимно сопряженным направлениям оптимизирует квадратичную функцию. Таким образом, n полных итераций метода Пауэлла, насчитывающих $n(n + 1)$ итераций линейного поиска, минимизируют квадратичную функцию.

Алгоритм 7.4. Метод Пауэлла, который принимает целевую функцию `f`, начальную точку `x` и допуск

```
function powell(f, x, ε)
    n = length(x)
    U = [basis(i, n) for i in 1 : n]
    Δ = Inf
    while Δ > ε
        x′ = x
        for i in 1 : n
            d = U[i]
            x′ = line_search(f, x′, d)
        end
        for i in 1 : n - 1
            U[i] = U[i + 1]
        end
        U[n] = d = x′ - x
        x′ = line_search(f, x, d)
        Δ = norm(x′ - x)
        x = x'
    end
    return x
end
```

Процедура отбрасывания самого старого направления поиска в пользу общего направления прогресса может привести к тому, что направления поиска станут линейно зависимыми. Без векторов поиска, которые являются линейно независимыми, направления поиска могут больше не охватывать все проектное пространство, и метод может не найти минимум. Эта слабость может быть уменьшена путем периодического сброса поиска направления к базисным векторам. Одна из рекомендаций — сбрасывать каждые n или $n + 1$ итераций.

7.3. Метод Хука–Дживса

Метод Хука–Дживса (алгоритм 7.5) пересекает пространство поиска на основе оценок с небольшими шагами в каждом направлении координат [71]. На каждой итерации метод Хука–Дживса вычисляет значения $f(\mathbf{x})$ и $f\left(\mathbf{x} \pm \alpha \mathbf{e}^{(i)}\right)$ для заданного размера шага α в каждом направлении координатных осей от *точки привязки* $\mathbf{x}$. Он принимает любые улучшения, которые может найти. Если улучшений не обнаружено, размер шага уменьшится. Процесс повторяется до тех пор, пока

размер шага не станет достаточно маленьким. На рис. 7.5 показано несколько итераций алгоритма.

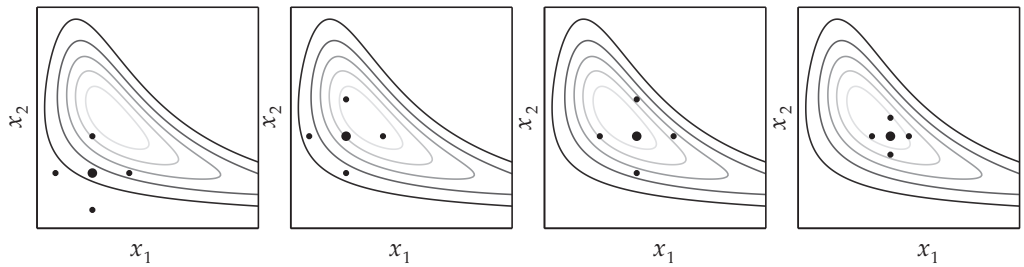


Рис. 7.5. Метод Хука–Дживса, действующий слева направо. Он начинается с большого размера шага, но затем уменьшает его, если его невозможно улучшить, делая шаг в любом направлении координат

Один шаг метода Хука–Дживса требует $2n$ вычислений функций для n-мерной задачи, что может сделать его слишком дорогостоящим для задач со многими измерениями. Метод Хука–Дживса чувствителен к локальным минимумам. Доказано, что метод сходится на определенных классах функций [39].

7.4. Обобщенный поиск по шаблону

В отличие от метода Хука–Дживса, который осуществляет поиск в направлениях координатных осей, *обобщенный поиск по шаблону* может осуществлять поиск в произвольных направлениях. *Шаблон* $\mathcal{P}$ может быть построен из набора направлений $\mathcal{D}$ вокруг точки привязки $\mathbf{x}$ с размером шага α в соответствии с правилом:

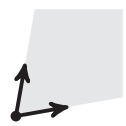
$$\mathcal{P} = \{ \mathbf{x} + \alpha \mathbf{d} \text{ для каждого } \mathbf{d} \text{ в } \mathcal{D} \} \tag{7.7}$$

Метод Хука–Дживса использует $2n$ направлений для задач в n измерениях, но обобщенный поиск по шаблону может использовать всего лишь $n + 1$ направлений.

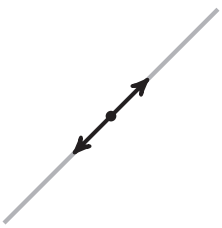
Чтобы обобщенный поиск по шаблону сходился к локальному минимуму, должны быть выполнены определенные условия. Набор направлений должен быть положительным базисом. Это означает, что мы можем построить любую точку в $\mathbb{R}^n$, используя неотрицательную линейную комбинацию направлений в $\mathcal{D}$. Положительный базис гарантирует, что по крайней мере одно из направлений является направлением спуска из точки с ненулевым градиентом.[2]

[2] Гарантии сходимости обобщенного поиска по шаблону требуют, чтобы все точки выборки находились на масштабированной решетке. Таким образом, каждое направление должно быть произведением $\mathbf{d}^{(j)} = \mathbf{G}\mathbf{z}^{(j)}$ для фиксированной невырожденной матрицы $n \times n$ и целочисленного вектора $\mathbf{z}$ [155].

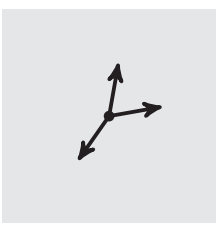
Мы можем определить, является ли данный набор направлений $\mathcal{D} = \{\mathbf{d}^{(1)}, \mathbf{d}^{(2)}, ..., \mathbf{d}^{(m)}\}$ в $\mathbb{R}^n$ положительным базисом. Сначала построим матрицу $\mathbf{D}$, столбцы которой являются направлениями в $\mathcal{D}$ (см. рис. 7.6). Множество направлений $\mathcal{D}$ является положительным базисом, если матрица $\mathbf{D}$ имеет полный строковый ранг и если $\mathbf{Dx} = -\mathbf{D1}$ с $\mathbf{x} \geq 0$ имеет решение [129]. Эта проблема оптимизации идентична фазе инициализации линейной программы, которая рассматривается в главе 11.



Положительный базис конуса



Положительный базис одномерного пространства



Положительный базис $\mathbb{R}^2$

Рис. 7.6. Корректный шаблон для поиска по обобщенному шаблону требует наличия положительного базиса

Алгоритм 7.5. Метод Хука–Дживса [79], который принимает целевую функцию `f`, начальную точку `x`, начальный размер шага `α`, допуск `ε` и затухание шага `γ`. Метод выполняется до тех пор, пока размер шага не станет меньше, чем `ε` и точки, выбранные вдоль координатных направлений, не будут обеспечивать улучшения

```
function hooke_jeeves(f, x, α, ε, γ = 0.5)
    y, n = f(x), length(x)
    while α > ε
        improved = false
        x_best, y_best = x, y
        for i in 1 : n
            for sgn in (-1, 1)
                x′ = x + sgn * α * basis(i, n)
                y′ = f(x′)
                if y′ < y_best
                    x_best, y_best, improved = x′, y′, true
                end
            end
        end
        x, y = x_best, y_best
```

```
            if !improved
                α *= γ
            end
        end
        return x
    end
```

Реализация поиска по обобщенному шаблону в алгоритме 7.6 содержит дополнительные усовершенствования по сравнению с исходным методом Хука–Дживса [8]. Во-первых, реализация является *оппортунистической* — как только оценка улучшает текущую расчетную точку, она принимается в качестве точки привязки для следующей итерации. Во-вторых, реализация использует *динамическое упорядочение* для ускорения сходимости — направление, которое приводит к улучшению, переводится в начало списка направлений. На рис. 7.7 показаны несколько итераций алгоритма.

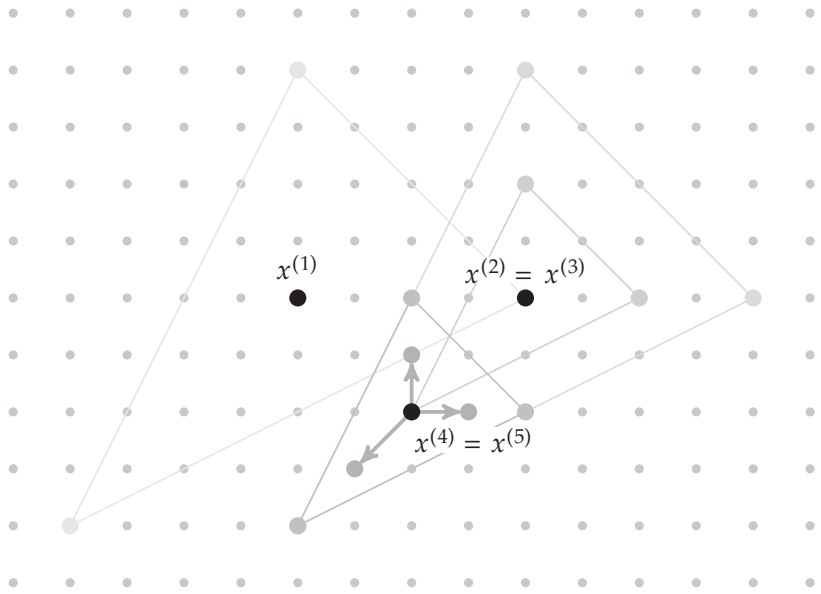


Рис. 7.7. Все предыдущие точки в обобщенном поиске по шаблону лежат на масштабированной решетке или сетке. Решетка строится явно и не обязана быть выровненной по осям

7.5. Симплекс-метод Нелдера–Мида

Симплексный метод Нелдера–Мида [92, 111] использует симплекс для обхода пространства в поисках минимума. *Симплекс* — это обобщение тетраэдра на n-мерное пространство. Симплекс в одном измерении — это отрезок, а в двух измере-

ниях — треугольник (рис. 7.8). Симплекс получил свое название от того факта, что он является самым простым многогранником в любом данном пространстве.

Алгоритм 7.6. Обобщенный поиск по шаблону, который принимает целевую функцию `f`, начальную точку `x`, начальный размер шага `α`, набор направлений поиска `D`, допуск `ε` и затухание шага `γ`. Метод выполняется до тех пор, пока размер шага не станет меньше, чем `ε`, и точки, выбранные вдоль координатных направлений, не перестанут обеспечивать улучшения

```
function generalized_pattern_search(f, x, α, D, ε, γ = 0.5)
    y, n = f(x), length(x)
    while α > ε
        improved = false
        for(i, d)in enumerate(D)
            x′ = x + α * d
            y′ = f(x′)
            if y′ < y
                x, y, improved = x′, y′, true
                D = pushfirst!(deleteat!(D, i), d)
                break
            end
        end
        if !improved
            α *= γ
        end
    end
    return x
end
```

Метод Нелдера–Мида использует ряд правил, которые определяют, как симплекс обновляется на основе значений целевой функции в его вершинах. Процедура представлена в виде блок-схемы на рис. 7.9 и алгоритма 7.7. Как и метод Хука–Дживса, симплекс может перемещаться, приблизительно сохраняя свой размер, и уменьшаться по мере приближения к оптимальному.

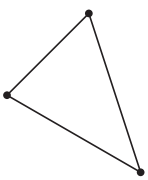


Рис. 7.8. Симплекс в двух измерениях — это треугольник. Чтобы симплекс был корректным, он должен иметь непустую внутренность

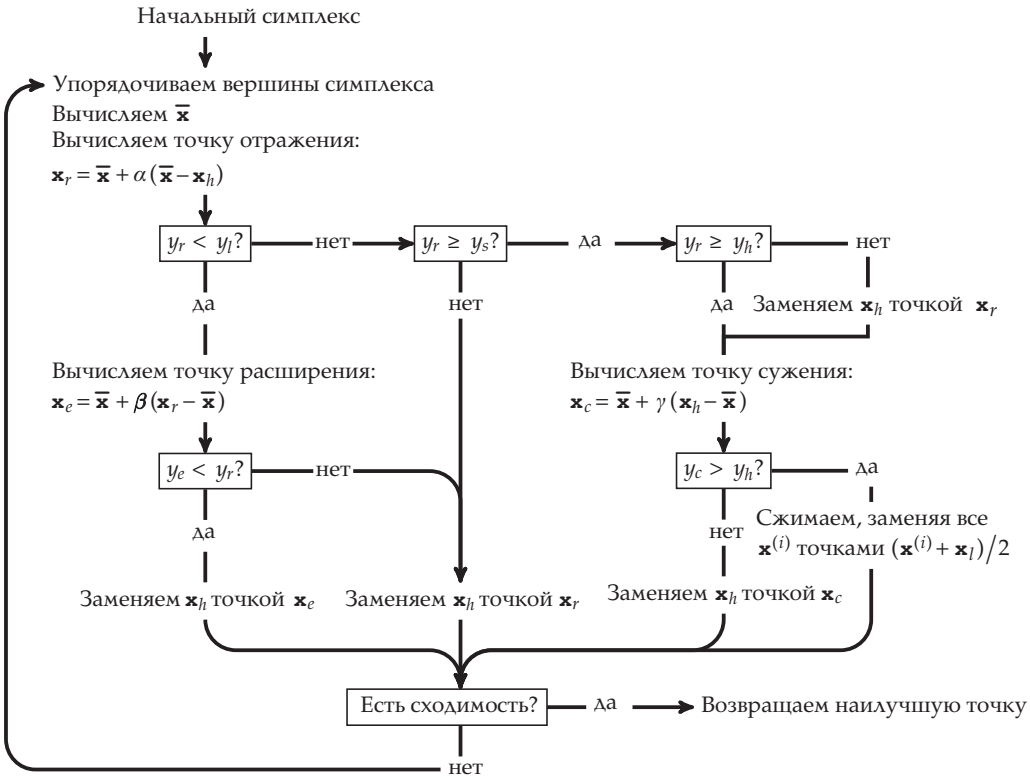


Рис. 7.9. Блок-схема алгоритма Нелдера–Мида

Алгоритм 7.7. Симплексный метод Нелдера–Мида, который принимает целевую функцию `f`, стартовый симплекс `S`, состоящий из списка векторов и допуска `ε`. Параметры метода Нелдера–Мида также могут быть указаны и по умолчанию рекомендованы к использованию

```
function nelder_mead(f, S, ε; α = 1.0, β = 2.0, γ = 0.5)
    Δ, y_arr = Inf, f.(S)
    while Δ > ε
        p = sortperm(y_arr)                    # упорядочивание по возрастанию
        S, y_arr = S[p], y_arr[p]
        xl, yl = S[1], y_arr[1]                # наименьшее значение
        xh, yh = S[end], y_arr[end]            # наибольшее значение
        xs, ys = S[end - 1], y_arr[end - 1] # второе наибольшее
        xm = mean(S[1 : end - 1])              # центроид
        xr = xm + α * (xm - xh)                # точка отражения
        yr = f(xr)
```

```
        if yr < yl
            xe = xm + β * (xr - xm) # точка расширения
            ye = f(xe)
            S[end], y_arr[end] = ye < yr ? (xe, ye) : (xr, yr)
        elseif yr ?> ys
            if yr > yh
               xh, yh, S[end], y_arr[end] = xr, yr, xr, yr
            end
            xc = xm + γ * (xh - xm) # точка сужения
            yc = f(xc)
            if yc > yh
               for i in 2 : length(y_arr)
                   S[i] = (S[i] + xl) / 2
                   y_arr[i] = f(S[i])
               end
            else
               S[end], y_arr[end] = xc, yc
            end
        else
            S[end], y_arr[end] = xr, yr
        end

        Δ = std(y_arr, corrected = false)
    end
    return S[argmin(y_arr)]
end
```

Симплекс состоит из точек $\mathbf{x}^{(1)}$, ..., $\mathbf{x}^{(n+1)}$. Пусть $\mathbf{x}_h$ — это вершина с наибольшим значением функции, $\mathbf{x}_s$ — вершина со вторым наибольшим значением функции, а $\mathbf{x}_l$ — вершина с наименьшим значением функции. Пусть $\bar{x}$ означает среднее значение всех вершин, кроме наивысшей точки $\mathbf{x}_h$. Наконец, для любой расчетной точки $\mathbf{x}_\theta$ пусть $y_\theta = f(\mathbf{x}_\theta)$ будет значением его целевой функции. На одной итерации выполняются четыре симплексных операции.

Отражение. $\mathbf{x}_r = \bar{\mathbf{x}} + \alpha(\bar{\mathbf{x}} - \mathbf{x}_h)$ отражает самую высокую точку над центроидом. Это обычно приводит к перемещению симплекса от высоких областей к более низким. В данном случае $\alpha > 0$ и обычно устанавливается равным единице.

Расширение. $\mathbf{x}_l = \bar{\mathbf{x}} + \beta(\mathbf{x}_r - \bar{\mathbf{x}})$ подобно отражению, но отраженная точка отправляется еще дальше. Это делается, когда значение целевой функции в отраженной точке меньше, чем во всех точках симплекса. В данном случае $\beta > \max(1, \alpha)$ и обычно устанавливается равным 2.

Сужение. $\mathbf{x}_c = \bar{\mathbf{x}} + \gamma\left(\mathbf{x}_h - \bar{\mathbf{x}}\right)$, симплекс сокращается, удаляясь от наихудшей точки. Этот шаг параметризован параметром $\gamma \in (0, 1)$, который обычно равен 0,5.

Сжатие. Все точки перемещаются к лучшей точке, как правило, уменьшая вдвое разделяющее их расстояние.

На рис. 7.10 показаны четыре симплексные операции. На рис. 7.11 показано несколько итераций алгоритма.

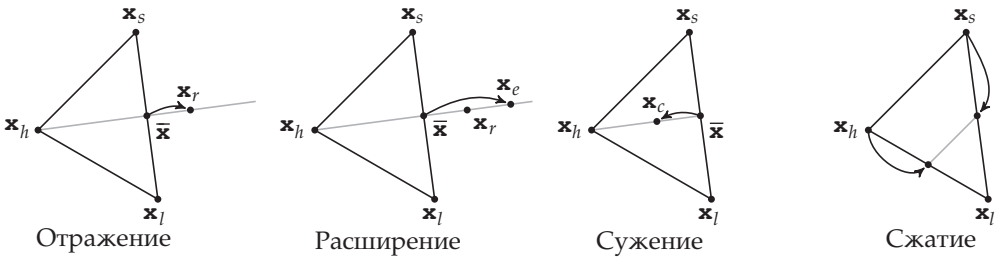


Рис. 7.10. Симплексные операции Нелдера–Мида, визуализированные в двух измерениях

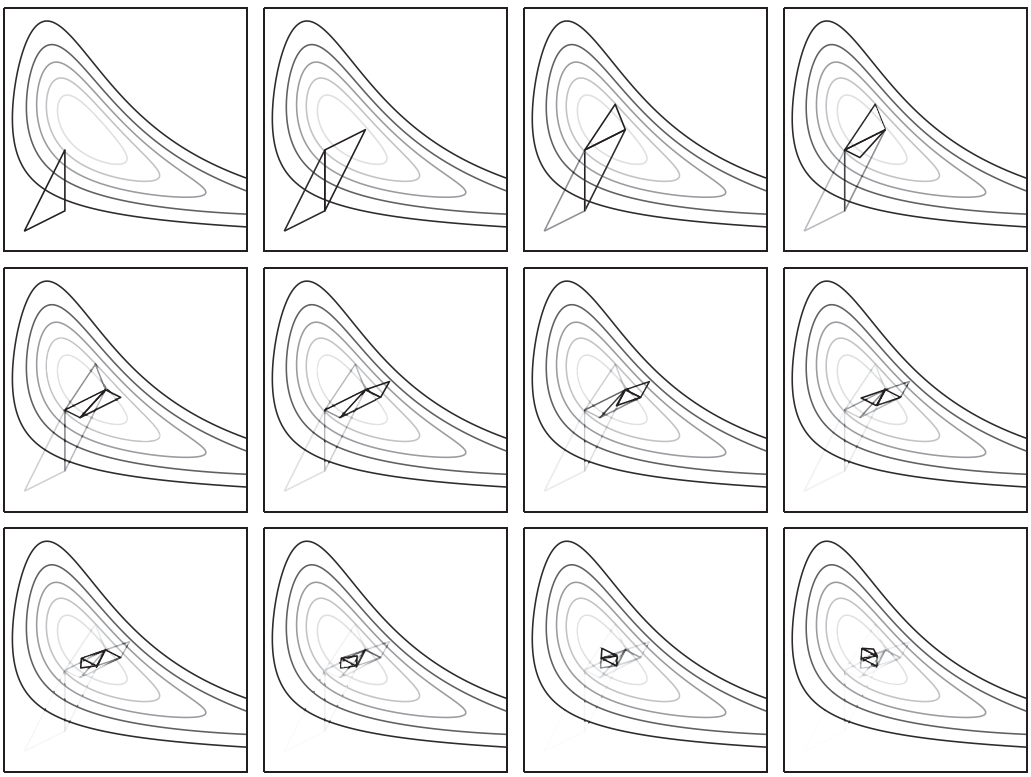


Рис. 7.11. Метод Нелдера–Мида, действующий слева направо и сверху вниз

Критерий сходимости для симплекс-метода Нелдера–Мида отличается от метода Пауэлла тем, что он учитывает изменение значений функции, а не изменения точек в области проектирования. Он сравнивает стандартное отклонение выборки[3] $y^{(1)}$, ..., $y^{(n+1)}$ с допуском ε. Это значение является высоким для симплекса в сильно изогнутой области и низким для симплекса в плоской области. Сильно изогнутая область указывает, что возможна дальнейшая оптимизация.

7.6. Разделенные прямоугольники

Алгоритм разделенных прямоугольников, или *DIRECT* (DIvided RECTangles), представляет собой липшицевый оптимизационный подход, в некотором роде сходный с методом Шуберта–Пиявского, описанным в разделе 3.6 [74]. Однако он устраняет необходимость указания константы Липшица и может быть более эффективно обобщен на несколько измерений.

Понятие липшиц-непрерывности можно распространить на несколько измерений. Если функция f является липшиц-непрерывной в области $\mathcal{X}$ с константой Липшица $l > 0$, то для заданной расчетной точки $\mathbf{x}^{(1)}$ и $y = f(\mathbf{x}^{(1)})$ круговой конус

$$f(\mathbf{x}^{(1)}) - l\|\mathbf{x} - \mathbf{x}^{(1)}\|_2 \tag{7.8}$$

образует нижнюю границу функции f. Имея m значений функций в расчетных точках $\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(m)}\}$, мы можем построить суперпозицию этих нижних границ, взяв их максимум:

$$\max_i f\left(\mathbf{x}^{(i)}\right) - l\left\|x - x^{(i)}\right\|_2. \tag{7.9}$$

Метод Шуберта–Пиявского осуществляет выборку в точке, где граница, полученная из известной константы Липшица, является наименьшей. К сожалению, нижняя граница Липшица имеет сложную геометрию, сложность которой увеличивается с размерностью проектного пространства. График левого контура на рис. 7.12 демонстрирует такую нижнюю границу с использованием пяти значений функций. Правый контурный график демонстрирует аппроксимацию, построенную с помощью метода DIRECT, которая делит область на гиперпрямоугольники с центром в каждой расчетной точке. Принятие этого предположения позволяет быстро вычислить минимум нижней границы.

---

[3] *Стандартное выборочное отклонение* также называется *нескорректированным выборочным стандартным отклонением*. В нашем случае оно равно $\sqrt{\frac{1}{n+1}\sum_{i=1}^{n+1}\left(y^{(i)} - \bar{y}\right)^2}$, где $\bar{y}$ — среднее значение выборки $y^{(1)}$, ..., $y^{(n+1)}$.

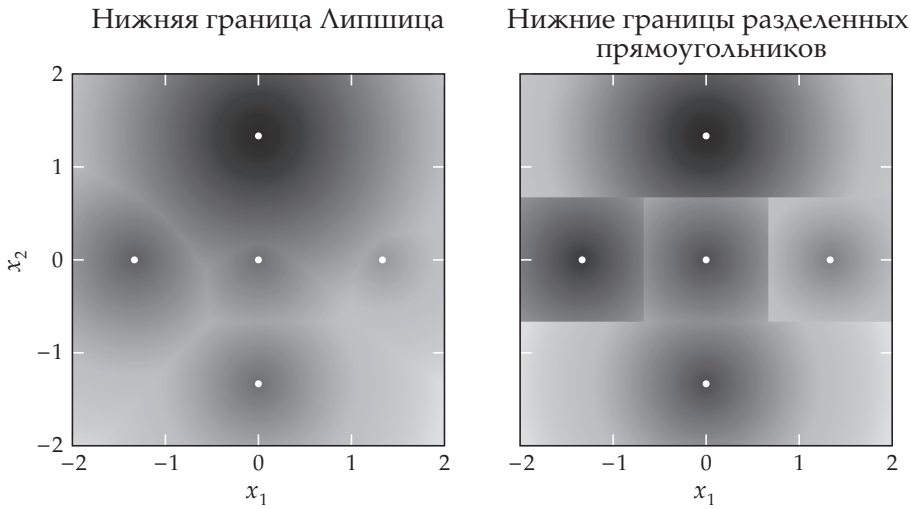


Рис. 7.12. Нижняя граница Липшица — это пересечение конусов, которое образует сложные поверхности в многомерном пространстве. Нижняя граница разделенного прямоугольника изолирует каждый нижний конус от своей гиперпрямоугольной области, что делает тривиальным вычисление минимального значения в каждой области с учетом константы Липшица

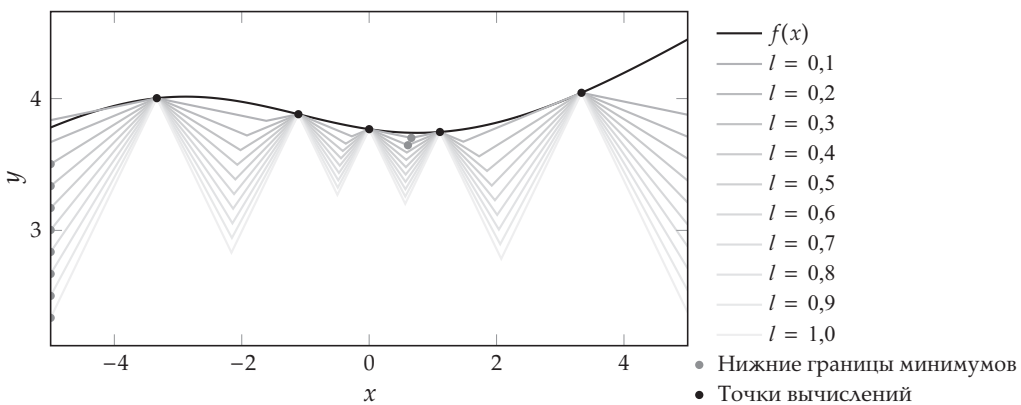


Рис. 7.13. Нижняя оценка Липшица для разных постоянных Липшина l. При изменении константы Липшица l локально изменяется не только вычисленный минимум, но и область, в которой лежит минимум

7.6.1. Односторонний метод DIRECT

В одном измерении DIRECT рекурсивно делит интервалы на трети, а затем выбирает целевую функцию в центре интервалов, как показано на рис. 7.15. Эта схема отличается от метода Шуберта–Пиявского, где выборка происходит в точке, где граница, полученная по известной константе Липшица, является самой низкой.

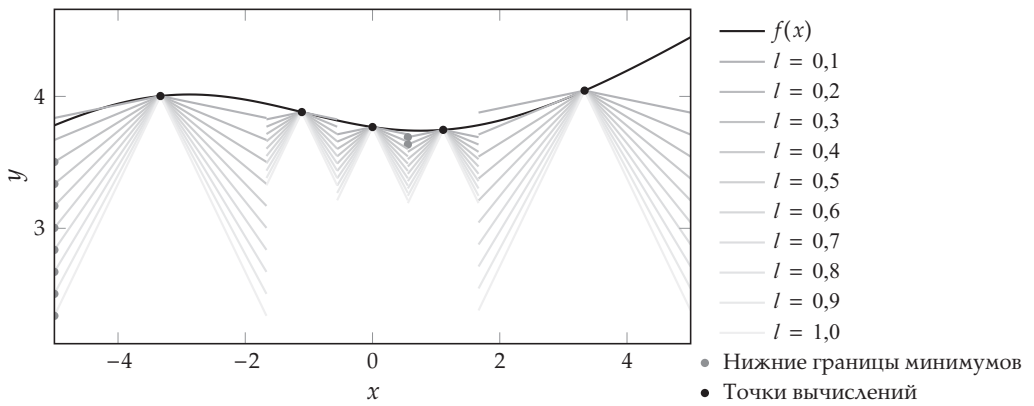


Рис. 7.14. Нижняя оценка метода DIRECT для разных липшицевых постоянных l. Нижняя граница не является непрерывной. При изменении константы Липшица минимум не изменяется локально, но может изменяться регионально

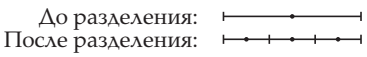


Рис. 7.15. Выборка центральной точки, использующая схему DIRECT, делит интервалы на три части

Для интервала $[a, b]$ с центром $c = (a + b)/2$ нижняя граница, основанная на $f(c)$, равна:

$$f(x) \geq f(c) - l|x - c|, \tag{7.10}$$

где l — константа Липшица, которая нам неизвестна. Наименьшее значение, которое граница достигает на этом интервале, равна $f(c) - l(b - a)/2$, и это происходит на краях интервала.

Даже если мы не знаем l, мы можем сделать вывод, что нижняя граница некоторых интервалов ниже других. Например, если у нас есть два интервала одинаковой длины, и первый имеет более низкое значение в центральной точке, чем второй, то нижняя граница первого интервала ниже, чем у второго. Это не означает, что первый интервал содержит точку минимума, но указывает на то, что мы можем сосредоточить поиск в этом интервале.

Во время поиска возникает много интервалов $[a_1, b_1]$, ..., $[a_n, b_n]$ различной ширины. Мы можем построить наши интервалы в соответствии с их центральным значением и шириной интервала, как показано на рис. 7.16. Нижняя граница для каждого интервала является вертикальным пересечением линии наклона, проходящей через его центральную точку. Центр интервала с самой низкой нижней границей будет первой точкой, пересекаемой при смещении линии с наклоном снизу вверх.

Метод DIRECT разбивает все интервалы, для которых существует константа Липшица так, что они имеют наименьшую нижнюю границу, как показано на рис. 7.17. Мы называем эти выбранные интервалы потенциально оптимальными. Теоретически любой интервал может содержать точку минимума, хотя выбранные точки эвристически имеют наилучшие шансы.

Одна итерация одномерного алгоритма DIRECT состоит из определения набора потенциально оптимальных интервалов и последующего деления каждого интервала на трети. Одномерный метод DIRECT демонстрирует пример 7.1.

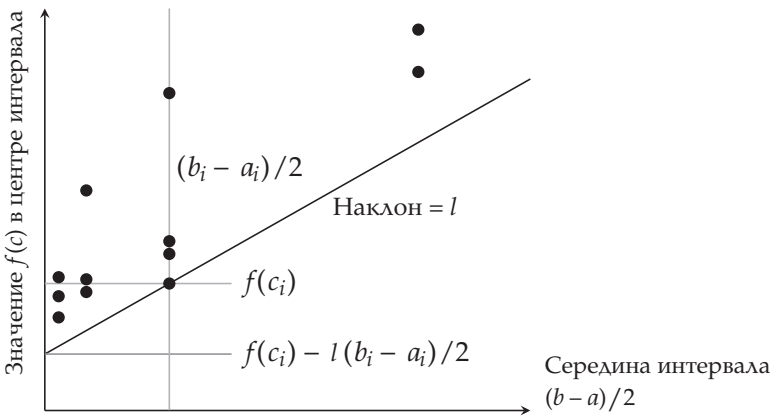


Рис. 7.16. Интервал выбора для конкретной константы Липшица *l*. Черные точки представляют интервалы метода DIRECT и соответствующие значения функций в их центрах. Черная линия с наклоном *l* проводится через точку, принадлежащую выбранному интервалу. Все остальные точки должны находиться на этой линии или выше нее

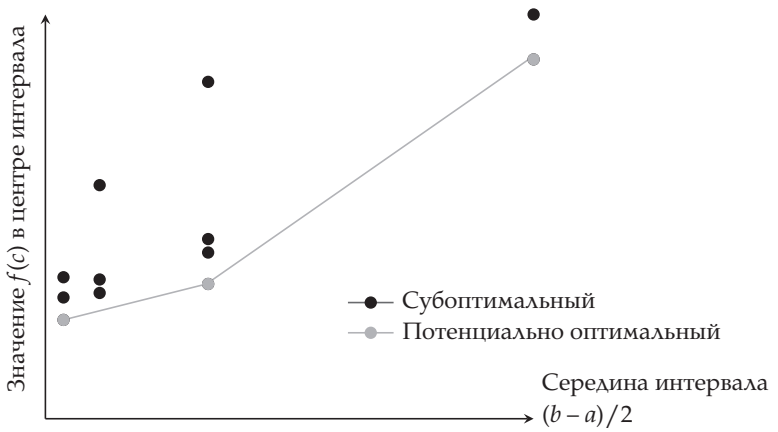
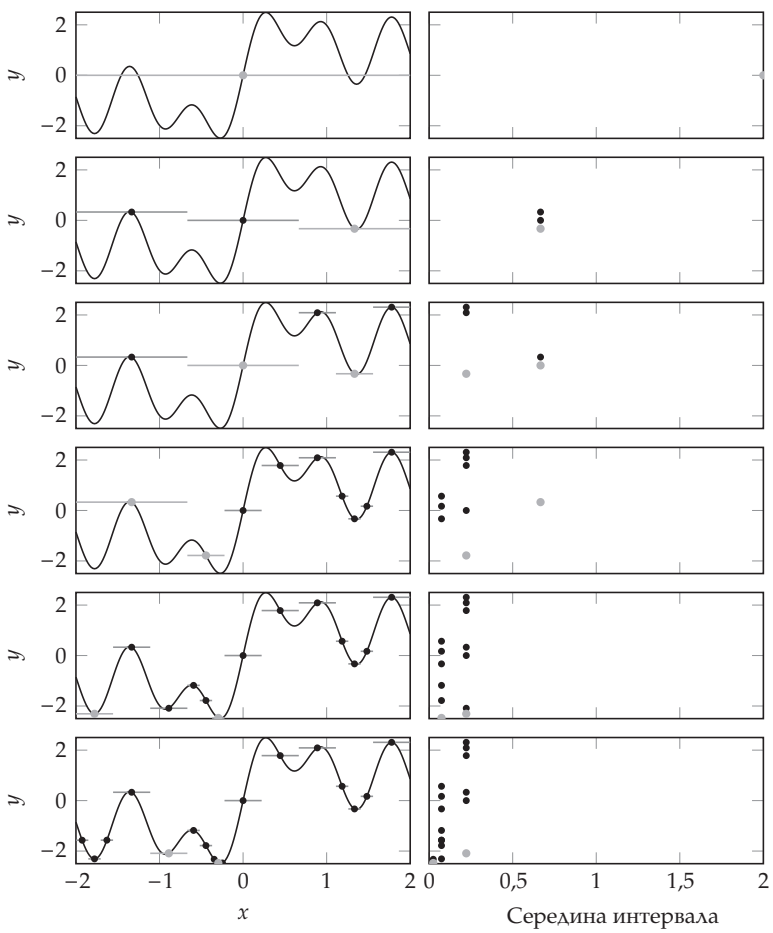


Рис. 7.17. Потенциально оптимальные интервалы для метода DIRECT образуют кусочную границу, охватывающую все интервалы вдоль нижнего правого угла. Каждая точка соответствует отдельному интервалу

Пример 7.1. Применение метода DIRECT к одномерной функции

Рассмотрим функцию $f(x) = \sin x + \sin 2x + \sin 4x + \sin 8x$ на интервале $[-2, 2]$ с точкой глобального минимума в окрестности точки $-0{,}272$. Оптимизация затруднена из-за множества локальных минимумов.

На рисунке ниже показана последовательность выполнения одномерного метода DIRECT с интервалами, выбранными для разделения, окрашенными в синий цвет. Левая сторона показывает интервалы, наложенные на целевую функцию. Правая сторона показывает интервалы в виде точек разброса на расстоянии полуширины интервала в зависимости от значения в центре.



7.6.2. Многомерный метод DIRECT

В пространстве нескольких измерений мы делим прямоугольники (или гиперпрямоугольники в пространстве более чем двух измерений) вместо интервалов. Как и в одномерном случае, мы делим прямоугольники на трети по направлениям осей. Перед началом деления прямоугольников DIRECT нормализует пространство поиска, чтобы оно стало единичным гиперкубом.

Как показано на рис. 7.18, выбор порядка направлений при разбиении единичного гиперкуба имеет значение. Метод DIRECT отдает приоритет назначению больших прямоугольников для точек с более низкими значениями функций. Большие прямоугольники являются приоритетными для дополнительного разделения.

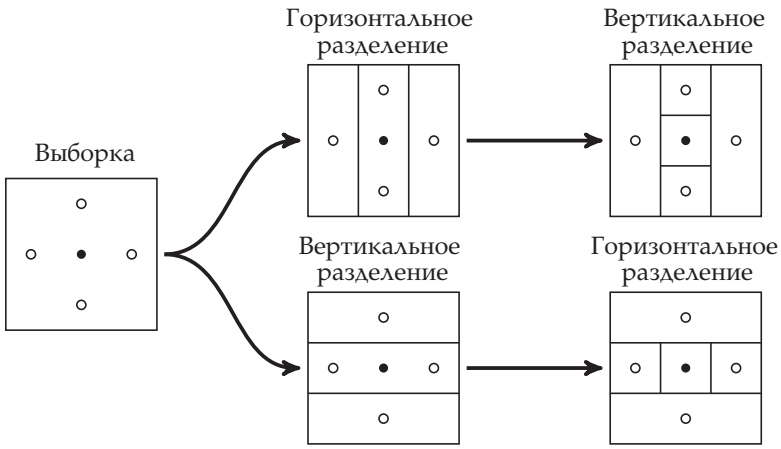


Рис. 7.18. Разделение интервалов в нескольких измерениях (используя метод DIRECT) требует выбора порядка разделения

При разделении области без равных длин сторон разделяются только самые длинные размеры (рис. 7.19). Затем в этих измерениях происходит расщепление таким же образом, как и в случае гиперкуба.

Множество потенциально оптимальных интервалов получается таким же, как в одномерном пространстве. Нижняя граница для каждого гиперпрямоугольника может быть вычислена на основе длины самой длинной стороны и значения центра. Мы можем построить диаграмму, похожую на рис. 7.16, чтобы определить потенциально оптимальные прямоугольники.[4]

[4] В качестве дополнительного требования для выбора метод DIRECT также требует, чтобы нижняя граница интервала улучшала текущее наилучшее значение на ненулевую величину.

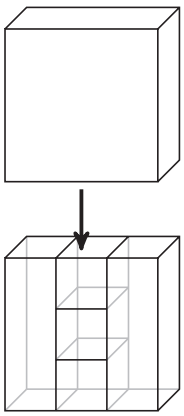


Рис. 7.19. Метод DIRECT будет разбивать только самые длинные стороны гиперпрямоугольников

7.6.3. Реализация

Метод DIRECT (алгоритм 7.8) лучше всего понимается, когда он разбит на подпрограммы. Мы представляем эти подпрограммы ниже.

Нормализация на единичный гиперкуб выполняется по алгоритму 7.9.

Алгоритм 7.8. Метод `DIRECT`, который принимает многомерную целевую функцию `f`, вектор нижних границ `a`, вектор верхних границ `b`, параметр допуска `ε` и число итераций `k_max`. Возвращает лучшую координату

```
function direct(f, a ,b, ε, k_max)
    g = reparametrize_to_unit_hupercuve(f, a, b)
    intervals = Intervals()
    n = length(a)
    c = fill(0.5, n)
    interval = Interval(c, g(c), fill(0, n))
    add_interval!(intervals,interval)
    c_best, y_best = copy(interval.c), interval.y

    for k in 1:k_max
        S = get_opt_intervals(intervals, ε, y_best)
        to_add = Interval[]
        for interval in S
            append!(to_add, divide(g, interval))
            dequeue!(intervals[min_depth(interval)])
        end
        for interval in to_add
            add_interval!(intervals, interval)
            if interval.y < y_best
```

```
                c_best, y_best = copy(interval.c), interval.y
            end
        end
    end

    return rev_unit_hypercube_parameterization(c_best, a, b)
end
```

Алгоритм 7.9. Функция, создающая функцию, определенную для единичного гиперкуба, которая является перепараметризованной версией функции `f`, определенной для гиперкуба с нижними и верхними границами `a` и `b`

```
rev_unit_hypercube_parameterization(x, a, b) = x. * (b - a) + a
function reparameterize_to_unit_hypercube(f, a, b)
    Δ = b - a
    return x -> f(x .* Δ + a)
end
```

Вычисление набора потенциально оптимальных прямоугольников может быть выполнено эффективно с помощью алгоритма 7.10. Мы используем тот факт, что ширина интервала может быть равной только одной трети, и, следовательно, многие из точек будут иметь одинаковую координату x. Для любой данной координаты x потенциально оптимальным интервалом может быть только тот, который содержит наименьшее значение y. Мы сохраняем прямоугольные интервалы в соответствии с их глубиной, а затем в связи с приоритетами согласно значению их центральной точки.

Алгоритм 7.10. Структура данных, используемая в методе `DIRECT`. Здесь у структуры `Interval` есть три поля: центр интервала `c`, значение в центральной точке `y = f(c)` и количество делений по каждому измерению `depths`. Функция `add_interval!` вставляет новый интервал в структуру данных

```
using DataStructures
struct Interval
    c
    y
    depths
end
min_depth(interval) = minimum(interval.depths)
const Intervals = Dict{Int, PriorityQueue{Interval, Float64}}
function add_interval!(intervals, interval)
```

```
    d = min_depth(interval)
    if !haskey(intervals, d)
        intervals[d] = PriorityQueue{Interval, Float64}()
    end
    return enqueue!(intervals[d], interval, interval.y)
end
```

Мы можем использовать эту структуру данных для получения всех потенциально оптимальных интервалов (алгоритм 7.11). Алгоритм переходит от минимальной ширины интервала к максимальной. Для каждой точки мы сначала определяем, находится она выше или ниже линии, соединяющей две предыдущие точки. Если ниже, мы ее пропускаем. Такая же процедура применяется к следующей точке.

Алгоритм 7.11. Процедура получения потенциально оптимальных интервалов, где объект **intervals** имеет тип **Intervals**, **ε** является параметром допуска, а **y_best** является наилучшим значением функции

```
function get_opt_intervals(intervals, ε, y_best)
    max_depth = maximum(keys(intervals))
    stack = [DataStructures.peek(intervals[max_depth])[1]]
    d = max_depth - 1
    while d ≥ 0
        if haskey(intervals, d) && !isempty(intervals[d])
            interval = DataStructures.peek(intervals[d])[1]
            x, y = 0.5 * 3.0 ^ (-min_depth(interval)), interval.y

            while !isempty(stack)
                interval1 = stack[end]
                x1 = 0.5 * 3.0 ^ (-min_depth(interval1))
                y1 = interval1.y
                l1 = (y - y1) / (x - x1)
                if y1 - l1 * x1 > y_best - ε || y < y1
                    pop!(stack)
                elseif length(stack) > 1
                    interval2 = stack[end - 1]
                    x2 = 0.5 * 3.0 ^ (-min_depth(interval2))
                    y2 = interval2.y
                    l2 = (y1 - y2) / (x1 - x2)
                    if l2 > l1
                        pop!(stack)
```

```
                else
                    break
                end
            else
                break
            end
        end

        push!(stack, interval) #добавляем новую точку
      end
      d -= 1
    end
    return stack
end
```

Наконец, нам нужен метод для разделения интервалов. Он реализуется методом деления (алгоритм 7.12).

Алгоритм 7.12. Процедура для разделения интервала, где **f** — целевая функция, а **intervals** — интервал, который нужно разделить. Возвращает список полученных меньших интервалов

```
function divide(f, interval)
    c, d, n = interval.c, min_depth(interval), length(interval.c)
    dirs = findall(interval.depths .== d)
    cs = [(c + 3.0 ^ (-d -1) * basis(i, n),
           c - 3.0 ^ (-d -1) * basis(i, n)) for i in dirs]
     vs = [(f(C[1]), f(C[2])) for C in cs]
     minvals = [min(V[1], V[2]) for V in vs]

     intervals = Interval[]
     depths = copy(interval.depths)
     for j in sortperm(minvals)
         depths[dirs[j]] += 1
         C, V = cs[j], vs[j]
         push!(intervals, Interval(C[1], V[1], copy(depths)))
         push!(intervals, Interval(C[2], V[2], copy(depths)))
     end
     push!(intervals, Interval(c, interval.y, copy(depths)))
     return intervals
end
```

Интервалы, полученные при запуске метода DIRECT в двух измерениях, показаны на рис. 7.20. Две итерации DIRECT в двух измерениях проиллюстрированы в примере 7.2.

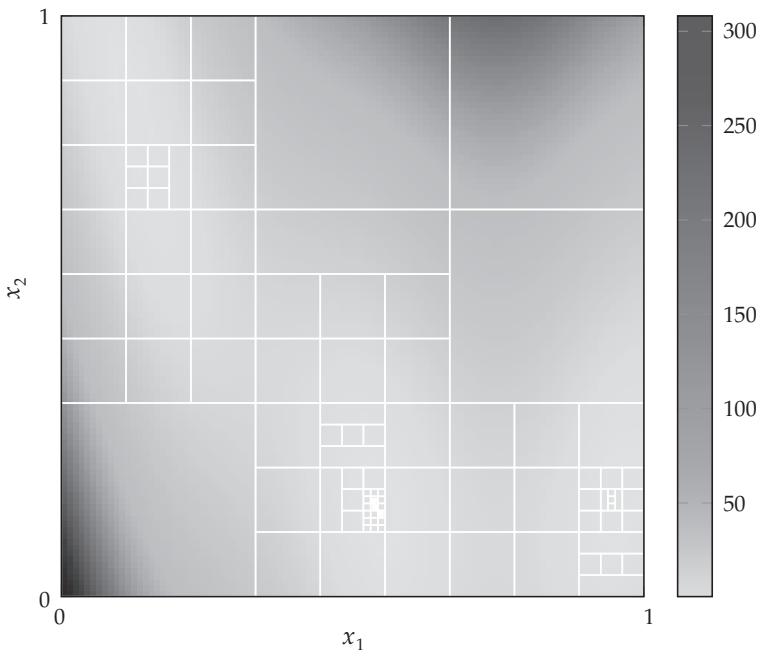


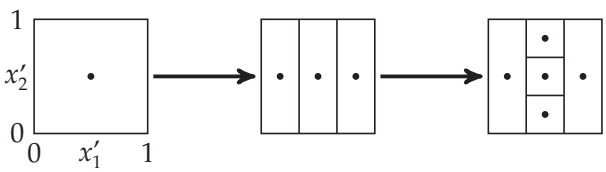
Рис. 7.20. Метод DIRECT после 16 итераций на примере функции Бранина (см. приложение Б.3)

Пример 7.2. Первые две итерации метода DIRECT

Рассмотрим использование метода DIRECT для оптимизации функции цветка (приложение Б.4) по $x_1 \in [-1, 3]$, $x_2 \in [-2, 1]$. Функция сначала нормализуется на единичный гиперкуб, так что мы оптимизируем $x_1', x_2' \in [0, 1]$:

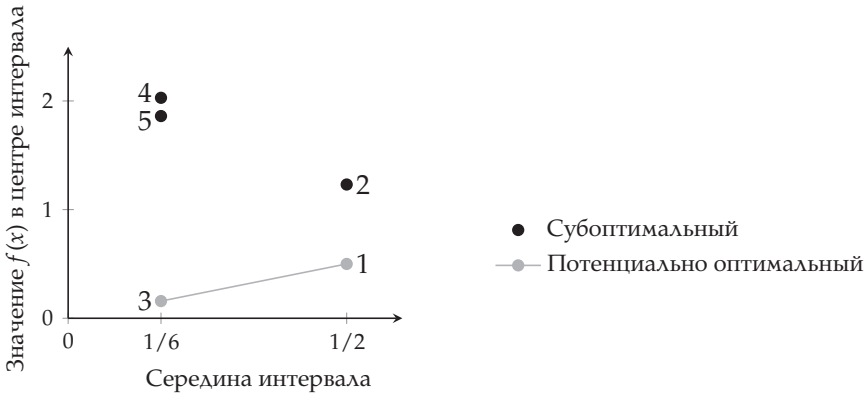
$$f(x_1', x_2') = \text{flower}(4x_1' - 1, 3x_2' - 2).$$

Значение целевой функции в точке [0,5; 0,5] равно 0,158. У нас есть один интервал с центром [0,5; 0,5] и длинами сторон [1, 1]. Интервал делится дважды: сначала на трети по x_1', а затем центральный интервал делится на трети по x_2'.



Теперь у нас есть пять интервалов.

Интервал	Центр	Длины сторон	Половина ширины	Значение в центре
1	[0,25; 0,50]	[1/3, 1]	1/2	0,500
2	[0,75; 0,50]	[1/3, 1]	1/2	1,231
3	[0,50; 0,50]	[1/3, 1/3]	1/6	0,158
4	[0,50; 0,25]	[1/3, 1/3]	1/6	2,029
5	[0,50; 0,75]	[1/3, 1/3]	1/6	1,886



Далее мы разбиваем два интервала с центрами в [0,25; 0,5] и [0,5; 0,5].

7.7. Резюме

- Прямые методы опираются исключительно на целевую функцию и не используют информацию о производной.
- Циклический покоординатный спуск оптимизирует одно направление координат за один раз.
- Метод Пауэлла адаптирует набор направлений поиска в зависимости от направления прогресса.
- Метод Хука–Дживса выполняет поиск в каждом направлении вдоль осей координат от текущей точки, используя размер шага, который адаптируется с течением времени.
- Обобщенный поиск по шаблону аналогичен методу Хука–Дживса, но использует меньше направлений поиска, которые образуют положительный базис пространства параметров проектирования.

- Симплекс-метод Нелдера–Мида использует симплекс для поиска в пространстве параметров проектирования, адаптивно расширяя и сокращая размер симплекса в ответ на значения целевой функции.
- Алгоритм разделенных прямоугольников обобщает подход Шуберта–Пиявского на нескольким измерениям и не требует указания действительной константы Липшица.

7.8. Упражнения

Упражнение 7.1. В предыдущих главах рассматривались методы, которые используют производную для снижения до минимума. Прямые методы могут использовать только информацию нулевого порядка — значения функции f. Сколько значений необходимо для аппроксимации производной и гессиана n-мерной целевой функции с использованием методов конечных разностей? Как вы думаете, почему важно иметь методы нулевого порядка?

Упражнение 7.2. Придумайте целевую функцию и начальную точку x_0 так, чтобы метод Хука–Дживса не смог добиться снижения целевой функции. Вам нужно выбрать точку x_0 так, чтобы она не была точкой локального минимума.

Упражнение 7.3. Всегда ли расчетная точка, полученная с использованием метода Хука–Дживса, лежит в ε-окрестности локального минимума?

Упражнение 7.4. Приведите пример конкретной инженерной задачи, в которой вы не сможете вычислить аналитические производные.

Упражнение 7.5. Укажите разницу между алгоритмом разделенных прямоугольников в одномерном пространстве и методом Шуберта–Пиявского.

Упражнение 7.6. Предположим, что наш алгоритм поиска выполняет переход от $\mathbf{x}^{(k)} = [1, 2, 3, 4]$ к $\mathbf{x}^{(k+1)} = [2, 2, 2, 4]$. Может ли этот алгоритм быть *а*) циклическим координатным поиском, *б*) методом Пауэлла, *в*) и *а*, и *б* или *г*) ни *а*, ни *б*? Почему?

Глава 8

Стохастические методы

В этой главе представлены различные *стохастические методы*, которые стратегически используют рандомизацию, чтобы помочь оптимально исследовать пространство параметров проекта. Случайность может помочь избежать локальных оптимумов и увеличить шансы на поиск глобального оптимума. Стохастические методы обычно используют генераторы *псевдослучайных* чисел для обеспечения повторяемости.[1] Большая случайность, как правило, неэффективна, поскольку мешает эффективно использовать при поиске предыдущие расчетные точки. В этой главе рассматриваются различные способы контроля степени случайности в поиске минимума.

8.1. Шумный спуск

Добавление стохастичности к градиентному спуску может быть полезным в больших задачах нелинейной оптимизации. Седловые точки, где градиент очень близок к нулю, могут привести к тому, что методы спуска выберут размеры шагов, которые слишком малы, чтобы быть полезными. Один из подходов заключается в добавлении гауссовского шума на каждом шаге спуска [70]:

$$\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} + \alpha \mathbf{g}^{(k)} + \boldsymbol{\varepsilon}^{(k)}, \tag{8.1}$$

где $\boldsymbol{\varepsilon}^{(k)}$ — это гауссовский шум с нулевым математическим ожиданием и стандартным отклонением σ. Величина шума обычно уменьшается со временем. Стандартное отклонение шума представляет собой убывающую последовательность $\sigma^{(k)}$, например $1/k$.[2] Реализация этого метода представлена в алгоритме 8.1.

---

[1] Хотя генераторы псевдослучайных чисел производят числа, которые кажутся случайными, на самом деле они являются результатом детерминированного процесса. Псевдослучайные числа могут быть получены с помощью вызова функции `rand`. Процесс может быть сброшен в исходное состояние с помощью функции `srand`.

[2] В [70] использовалось фиксированное стандартное отклонение для первых 3500 итераций, после чего стандартное отклонение устанавливалось равным нулю.

На рис. 8.1 приведены результаты сравнения метода наискорейшего спуска с шумом и без шума на седловой функции.

Алгоритм 8.1. Метод шумного спуска, который дополняет другой метод спуска аддитивным гауссовским шумом. Метод принимает другой подметод `DescentMethod`, шумовую последовательность и сохраняет счетчик итераций `k`

```
mutable struct NoisyDescent <: DescentMethod
    submethod
    σ
    k
end
function init!(M :: NoisyDescent, f, ∇f, x)
    init!(M.submethod, f, ∇f, x)
    M.k = 1
    return M
end
function step!(M :: NoisyDescent, f, ∇f, x)
    x = step!(M.submethod, f, ∇f, x)
    σ = M.σ(M.k)
    x += σ. * randn(length(x))
    M.k += 1
    return x
end
```

Распространенным подходом к обучению нейронных сетей является метод *стохастического градиентного спуска*, который использует аппроксимацию шумового градиента. Помимо помощи в обходе седловых точек, вычисление шумовых градиентов с использованием случайно выбранных подмножеств обучающих данных[3] значительно дешевле в вычислительном отношении, чем вычисление истинного градиента на каждой итерации.

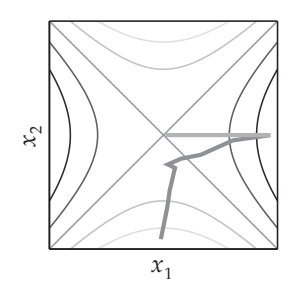


Рис. 8.1. Добавление стохастичности в метод спуска помогает обходить седловые точки, которые есть, например, у функции $f(\mathbf{x}) = x_1^2 - x_2^2$, показанной на рисунке. Благодаря инициализации метод наискорейшего спуска сходится к седловой точке, где градиент равен нулю

[3] Эти подмножества называются *пакетами*.

Гарантии сходимости для стохастического градиентного спуска требуют выбирать положительные шаговые множители так, чтобы:

$$\sum_{k=1}^{\infty}\alpha^{(k)}=\infty,\quad \sum_{k=1}^{\infty}\left(\alpha^{(k)}\right)^2<\infty. \tag{8.2}$$

Эти условия гарантируют, что размеры шагов уменьшаются и позволяют методу сходиться, но не так быстро, чтобы застревать вдали от локального минимума.

8.2. Метод адаптивного прямого поиска на сетке

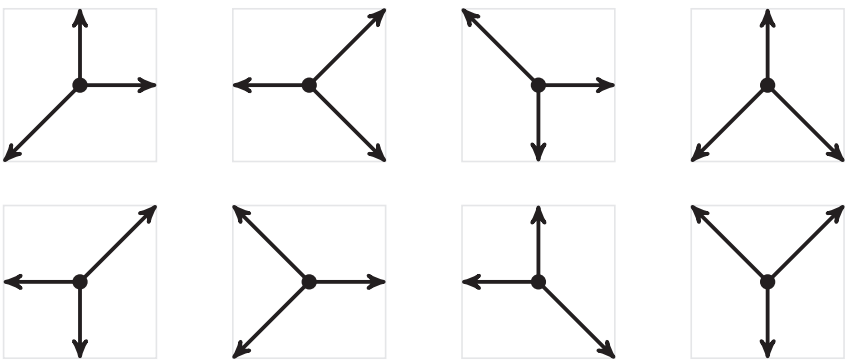
Обобщенные методы поиска по шаблону, описанные в разделе 7.4, ограничивают локальное исследование фиксированным набором направлений. Напротив, в методе прямого поиска на адаптивной сетке используются случайные направления положительного базиса.[4]

Процедура, используемая для выборки положительных базисов (см. пример 8.1), начинается с построения начального линейного базиса в виде нижнетреугольной матрицы **L**. Диагональные члены матрицы **L** выбираются из множества $\pm 1/\sqrt{\alpha^{(k)}}$, где $\alpha^{(k)}$ — размер шага на итерации k. Нижние компоненты матрицы **L** выбираются из множества

$$\left\{-1/\sqrt{\alpha^{(k)}}+1,-1/\sqrt{\alpha^{(k)}}+2,\ldots,-1/\sqrt{\alpha^{(k)}}-1\right\}. \tag{8.3}$$

Пример 8.1. Положительные базисы с направлениями, имеющими единичную l_1-длину в пространстве $\mathbb{R}^2$

Рассмотрим множество положительных базисов, построенных из ненулевых направлений $d_1, d_2 \in \{-1, 0, 1\}$. Существует восемь положительных базисов с тремя элементами, которые можно построить из этих направлений:



[4] В этом разделе описывается метод прямого поиска на нижнетреугольной адаптивной сетке [8].

Затем строки и столбцы матрицы **L** случайным образом переставляются, образуя матрицу **D**, столбцы которой соответствуют n направлениям, образующим линейный базис пространства $\mathbb{R}^n$. Максимальная величина среди этих направлений составляет $1/\sqrt{\alpha^{(k)}}$.

Два распространенных метода получения положительного базиса из линейного базиса заключаются в добавлении одного дополнительного направления $\mathbf{d}^{(n+1)} = -\sum_{i=1}^{n} \mathbf{d}^{(i)}$ и n дополнительных направлений $\mathbf{d}^{(n+j)} = -\mathbf{d}^{(j)}$ для j из множества $\{1, ..., n\}$. В алгоритме 8.2 мы используем первый метод.

Алгоритм 8.2. Произвольная выборка положительного базиса из $n + 1$ направлений в соответствии с методом адаптивного прямого поиска на сетке с шаговым множителем **α** и количеством измерений **n**

```
function rand_positive_spanning_set(α, n)
    δ = round(Int, 1 / sqrt(α))
    L = Matrix(Diagonal(δ * rand([1, -1], n)))
    for i in 1 : n - 1
        for j in i + 1 : n
          L[i, j]=rand(-δ + 1 : δ - 1)
        end
    end
    D = L[randperm(n), :]
    D = L[:, randperm(n)]
    D = hcat(D, -sum(D, dims = 2))
    return [D[:, i] for i in 1 : n + 1]
end
```

Шаговый множитель имеет начальное значение, равное единице, всегда является степенью четверки и никогда не превышает единицы. Использование степени четверки приводит к тому, что максимально возможный размер шага на каждой итерации умножается на два, так что максимальный размер шага $\alpha/\sqrt{\alpha}$ имеет длину $4^m/\sqrt{4^m} = 2^m$ для целого числа $m < 1$. Шаговый множитель вычисляется по формулам

$$\alpha^{(k+1)} \leftarrow \begin{cases} \alpha^{(k)}/4, \text{ если итерация не привела к улучшению,} \\ \min\left(1, 4\alpha^{(k)}\right) \text{ в противном случае.} \end{cases} \tag{8.4}$$

Метод адаптивного прямого поиска на сетке является оппортунистическим, но не может поддерживать динамическое упорядочение[5], поскольку после успешной

[5] См. раздел 7.4.

итерации размер шага увеличивается, и другой шаг в успешном направлении будет находиться за пределами сетки. Алгоритм запрашивает новую расчетную точку вдоль принятого направления спуска. Если $f(x^{(k)} = x^{(k-1)} + \alpha\mathbf{d}) < f(x^{(k-1)})$, то запрашиваемая точка имеет вид $x^{(k-1)} + 4\alpha\mathbf{d} = x^{(k)} + 3\alpha\mathbf{d}$. Процедура изложена в алгоритме 8.3. На рис. 8.2 показано, как этот алгоритм исследует пространство поиска.

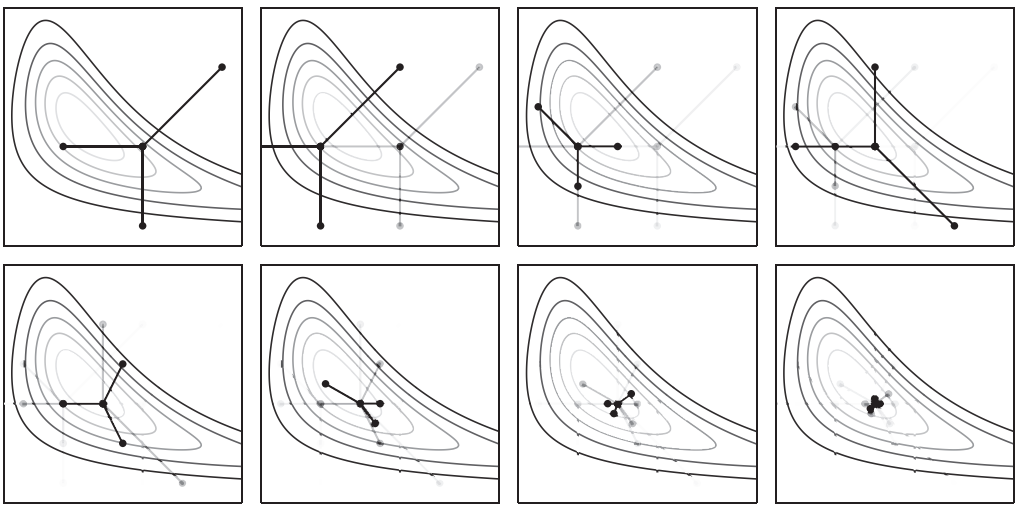


Рис. 8.2. Адаптивный прямой поиск на сетке осуществляется слева направо и сверху вниз

Алгоритм 8.3. Адаптивный прямой поиск на сетке с целевой функцией `f`, исходной точкой `x` и допуском `ε`

```
function mesh_adaptive_direct_search(f, x, ε)
    α, y, n = 1, f(x), length(x)
    while α > ε
        improved = false
        for (i, d) in enumerate(rand_positive_spanning_set(α, n))
            x′ = x + α * d
            y′ = f(x′)
            if y′ < y
                x, y, improved = x′ y′, true
                x′ = x + 3α * d
                y′ = f(x′)
                if y′ < y
                    x, y = x′, y′
                end
```

```
            break
          end
        end
        α = improved ? min(4α, 1) : α / 4
      end
      return x
end
```

8.3. Имитация отжига

Идея *имитации отжига* [84] возникла по ассоциации с металлургическим процессом.[6] Для контроля степени стохастичности при рандомизированном поиске используется *температура*. Процесс начинается с высокой температуры, что позволяет свободно перемещаться по пространству поиска, в надежде, что на этом этапе будет обнаружена хорошая область с лучшим локальным минимумом. Затем температура медленно понижается, уменьшая стохастичность и заставляя поиск сходиться к минимуму. Имитация отжига часто используется в функциях со многими локальными минимумами благодаря своей способности избегать таких точек.

На каждой итерации переход кандидата от $\mathbf{x}$ к $\mathbf{x}'$ выбирается на основе распределения переходов T с вероятностью

$$\begin{cases} 1, & \text{если } \Delta y \le 0, \\ \min\left(e^{-\Delta y/t},\ 1\right), & \text{если } \Delta y > 0, \end{cases} \tag{8.5}$$

где $\Delta y = f(\mathbf{x}') - f(\mathbf{x})$ — разность между значениями целевой функции, а t — температура. Эта вероятность перехода, известная как *критерий Метрополиса*, позволяет алгоритму выходить из локальных минимумов при высокой температуре.

Температурный параметр t контролирует вероятность перехода. Для медленного понижения температуры в процессе работы алгоритма используется график отжига, как показано на рис. 8.3. Чтобы обеспечить сходимость, температура должна быть снижена. Если она снижается слишком быстро, то метод поиска может не охватить часть пространства поиска, содержащую глобальный минимум.

---

[6] Отжиг — это процесс, при котором материал нагревается, а затем охлаждается, что делает его более пластичным. В горячем состоянии атомы в материале перемещаются более свободно и благодаря случайному движению стремятся занять более выгодные позиции. Медленное охлаждение приводит материал к упорядоченному кристаллическому состоянию. Быстрая, резкая закалка вызывает дефекты, поскольку материал вынужден затвердевать в своем текущем состоянии.

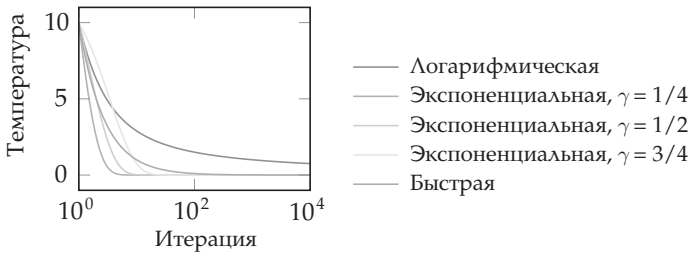


Рис. 8.3. Несколько графиков отжига, обычно используемых для имитации отжига. Графики имеют начальную температуру, равную 10

Можно показать, что *график логарифмического отжига* $t^{(k)} = t^{(1)} \ln 2 / \ln(k+1)$ для k-й итерации гарантированно асимптотически достигает глобального оптимума при определенных условиях [64], но может быть очень медленным на практике. *График экспоненциального отжига*, который является более распространенным, использует простой коэффициент затухания:

$$t^{(k+1)} = \gamma t^{(k)} \tag{8.6}$$

для некоторых $\gamma \in (0, 1)$. Другой распространенный график отжига — *быстрый отжиг* [153] — использует температуру

$$t^{(k)} = \frac{t^{(1)}}{k}. \tag{8.7}$$

Базовая реализация имитации отжига описана в алгоритме 8.4. В примере 8.2 показано влияние различных распределений переходов и графиков отжига на процесс оптимизации.

Алгоритм 8.4. Имитация отжига, которая принимает в качестве входных данных целевую функцию **f**, начальную точку **x**, распределение **T** перехода, график отжига **t** и количество итераций **k_max**

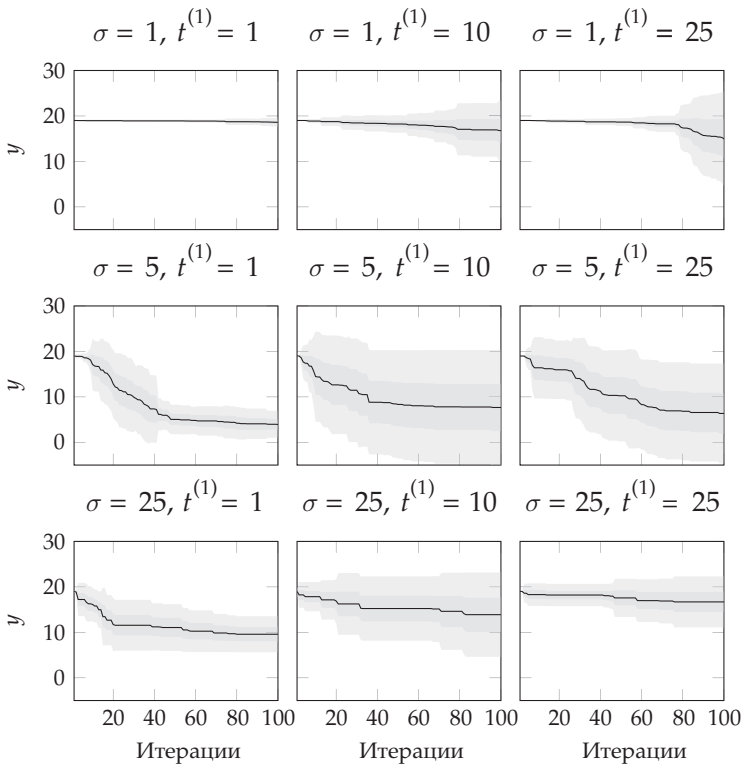
```
function simulated_annealing(f, x, T, t, k_max)
    y= f(x)
    x_best, y_best = x,  y
    for k in 1 : k_max
        x' = x + rand(T)
        y' = f(x')
        Δy = y' - y
        if Δy ≤ 0 || rand() < exp(-Δy / t(k))
            x, y = x', y'
        end
        if y' < y_best
```

```
            x_best, y_best = x', y'
        end
    end
    return x_best
end
```

Пример 8.2. Изучение влияния дисперсии распределения и температуры на производительность имитации отжига. Синие области указывают эмпирические квантили Гаусса от 5 до 95% и от 25 до 75% значения целевой функции.

Мы можем использовать имитацию отжига для оптимизации функции Экли (см. приложение Б.1). Функция Экли имеет множество локальных минимумов, в которых могут застрять градиентные методы.

Предположим, мы начинаем с точки $\mathbf{x}^{(1)} = [15, 15]$ и выполняем 100 итераций. Ниже мы показываем распределение по итерациям для нескольких вариантов с различными комбинациями трех распределений гауссовских переходов с нулевым математическим ожиданием, диагональной ковариацией ($\sigma\mathbf{I}$) и тремя различными температурными графиками $t^{(k)} = t^{(1)}/k$.



В этом случае распространение переходного распределения оказывает наибольшее влияние на производительность.

В 1987 г. в работе [30] был представлен более сложный алгоритм, позволяющий изменять шаговый множитель во время поиска. Вместо использования фиксированного распределения переходов этот адаптивный метод имитации отжига отслеживает отдельный шаговый множитель **v** для каждого направления координат. Для данной точки **x** цикл случайных перемещений выполняется в каждом направлении координат i в соответствии с формулой:

$$\mathbf{x}' = \mathbf{x} + r v_i \mathbf{e}_i, \tag{8.8}$$

где число r выбирается случайным образом из диапазона [–1, 1], а v_i — максимальный размер шага в i-м направлении координат. Каждая новая точка принимается в соответствии с критерием Метрополиса. Количество принятых точек в каждом направлении координат сохраняется в векторе **a**.

После n_s циклов шаговые множители корректируются с целью поддержания примерно равного числа принятых и отклоненных переходов со средним уровнем принятия около половины. Отклонение слишком большого количества переходов означает пустую трату вычислительных усилий, в то время как принятие слишком большого количества переходов указывает на то, что конфигурация развивается слишком медленно, поскольку точки-кандидаты слишком близки к текущей точке. Формула итерационного приближения, предложенная в работе [30], имеет вид

$$v_i = \begin{cases} v_i\left(1 + c_i \dfrac{\alpha_i / n_s - 0{,}6}{0{,}4}\right), & \text{если } a_i > 0{,}6 n_s; \\ v_i\left(1 + c_i \dfrac{0{,}4 - \alpha_i / n_s}{0{,}4}\right), & \text{если } a_i < 0{,}4 n_s; \\ v_i, & \text{в противном случае.} \end{cases} \tag{8.9}$$

Параметр c_i управляет изменением шага вдоль каждого направления и обычно устанавливается равным 2, как показано на рис. 8.4. Реализация итерационного приближения показана в алгоритме 8.5.

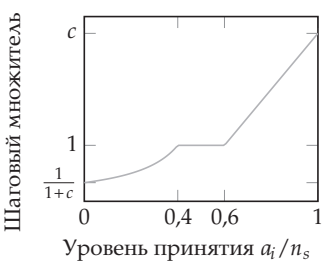


Рис. 8.4. Влияние шагового множителя на уровень принятия переходов при $c = 2$

Алгоритм 8.5. Формула итерационного приближения, используемая в работе [30] в адаптивной имитации отжига, где **v** — вектор шаговых множителей по координатам, **a** — вектор числа принятых шагов в каждом направлении координат, **c** — вектор шаговых множителей для каждого направления координат, а **ns** — количество циклов до регулировки шагового множителя

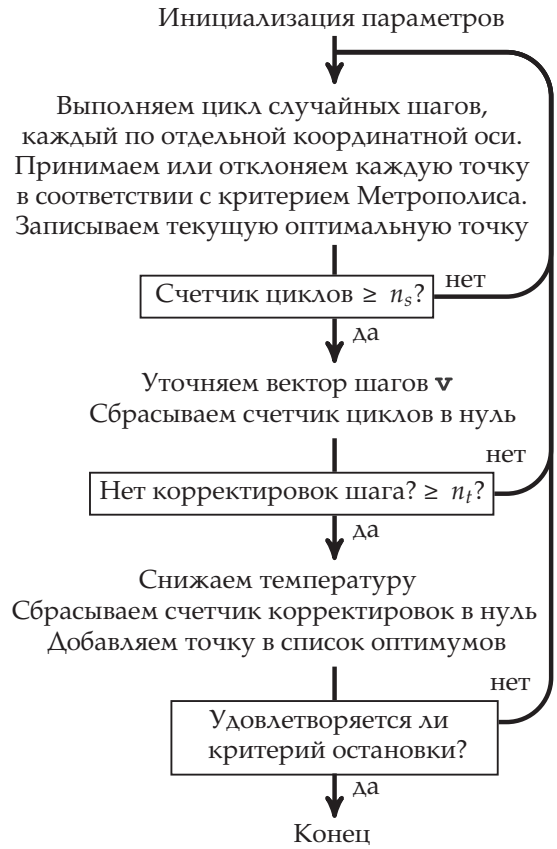
```
function corana_update!(v, a, c, ns)
   for i in 1 : length(v)
       ai, ci = a[i], c[i]
       if ai > 0.6ns
          v[i] *= (1 + ci * (ai / ns - 0.6) / 0.4)
       elseif ai < 0.4ns
          v[i] /= (1 + ci * (0.4 - ai / ns) / 0.4)
       end
   end
   return v
end
```

Снижение температуры происходит при каждом шаге регулировки, т.е. через каждые $n_s \cdot n_t$ циклов. В исходной реализации температура просто умножалась на коэффициент уменьшения.

Процесс прекращается, когда температура падает достаточно низко, так что улучшения уже нельзя было ожидать. Завершение происходит, когда самое последнее значение функции не отличается от предыдущих n_ε итераций больше чем на ε, и текущего лучшего значения, полученного в ходе выполнения. Реализация метода описана в алгоритме 8.6 и проиллюстрирована на рис. 8.5.

Алгоритм 8.6. Алгоритм адаптивной имитации отжига, где **f** — многомерная целевая функция, **x** — начальная точка, **v** — начальный вектор шаговых множителей, **t** — начальная температура и **ε** — параметр критерия завершения. Необязательными параметрами являются количество циклов перед выполнением настройки размера шага **ns**, количество циклов перед снижением температуры **nt**, количество последовательных снижений температуры для проверки на завершение **nε**, коэффициент снижения температуры **γ** и критерий перемены направления **c**. Ниже приведена блок-схема алгоритма адаптивной имитации отжига, представленного в оригинальной статье

```
function adaptive_simulated_annealing(f, x, v, t, ε;
     ns = 20, nε = 4, nt = max(100, 5length(x)),
     γ = 0.85, c = fill(2, length(x)))
```




```
y = f(x)
x_best, y_best = x, y
y_arr, n, U = [], length(x), Uniform(-1.0, 1.0)
a,counts_cycles,counts_resets = zeros(n), 0, 0

while true
    for i in 1:n
        x' = x + basis(i,n) * rand(U) * v[i]
        y' = f(x')
        Δy = y' - y
        if Δy < 0 || rand() < exp(-Δy / t)
           x, y = x', y'
           a[i] += 1
           if y' < y_best; x_best, y_best = x', y'; end
        end
    end
```

```
            counts_cycles += 1
            counts_cycles ≥ ns || continue

            counts_cycles = 0
            corana_update!(v, a, c, ns)
            fill!(a, 0)
            counts_resets += 1
            counts_resets ≥ nt || continue

            t *= γ
            counts_resets = 0
            push!(y_arr, y)

            if !(length(y_arr) > nε && y_arr[end] - y_best ≤ ε &&
                all(abs(y_arr[end] - y_arr[end - u]) ≤ ε for u in 1 : nε))
                x, y = x_best, y_best
            else
                break
            end
        end
        return x_best
    end
```

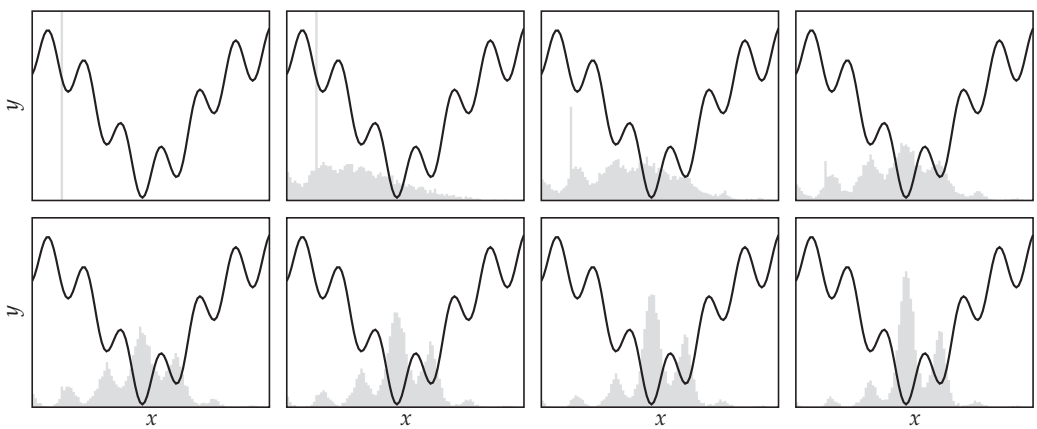


Рис. 8.5. Имитация отжига с экспоненциально затухающей температурой, где гистограммы показывают вероятность того, что имитированный отжиг будет находиться в конкретной позиции на этой итерации

8.4. Метод перекрестной энтропии

Метод перекрестной энтропии [133], в отличие от методов, которые мы обсуждали до сих пор в этой главе, поддерживает явное распределение вероятностей в пространстве параметров проектов.[7] Это распределение вероятностей, часто называемое *распределением предложений*, используется для предложения новых выборок для следующей итерации. На каждой итерации мы извлекаем выборку из распределения предложений, а затем обновляем его, чтобы оно обучалось на коллекции лучших выборок. Цель сходимости заключается в том, чтобы при распределении предложений основное внимание уделялось глобальной оптимальности. Реализация метода описана в алгоритме 8.7.

Алгоритм 8.7. Метод перекрестной энтропии, который получает целевую функцию `f`, распределение предложений `P`, число итераций `k_max`, размер выборки `m` и количество выборок, которые следует использовать при перекомпоновке распределения `m_elite`

```
using Distributions
function cross_entropy_method(f, P, k_max, m = 100, m_elite = 10)
    for k in 1 : k_max
        samples = rand(P, m)
        order = sortperm([f(samples[:, i]) for i in 1 : m])
        P = fit(typeof(P), samples[:, order[1 : m_elite]])
    end
    return P
end
```

Метод перекрестной энтропии требует выбора семейства распределений, параметризованных параметром θ. Одним из распространенных вариантов является семейство многомерных нормальных распределений, параметризованных вектором математических ожиданий и ковариационной матрицей. Алгоритм также требует, чтобы мы указали количество *элитных выборок* m_{elite} для использования при подборе параметров для следующей итерации.

В зависимости от выбора семейства распределения процесс аппроксимации распределения на элитных выборках может быть выполнен аналитически. В случае многомерного нормального распределения параметры обновляются в соответствии с оценкой максимального правдоподобия:

[7] Название этого метода происходит от того факта, что процесс аппроксимации распределения включает в себя минимизацию перекрестной энтропии, которая также называется *дивергенцией Кульбака–Лейблера*. При определенных условиях минимизация перекрестной энтропии соответствует нахождению оценки максимального правдоподобия параметров распределения.

$$\boldsymbol{\mu}^{(k+1)} = \frac{1}{m_{\text{elite}}} \sum_{i=1}^{m_{\text{elite}}} \mathbf{x}^{(i)}, \tag{8.10}$$

$$\boldsymbol{\Sigma}^{(k+1)} = \frac{1}{m_{\text{elite}}} \sum_{i=1}^{m_{\text{elite}}} \left(\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(k+1)}\right)\left(\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(k+1)}\right)^{\mathrm{T}}. \tag{8.11}$$

В примере 8.3 метод перекрестной энтропии применяется к простой функции. На рис. 8.6 показано несколько итераций для более сложной функции. Пример 8.4 демонстрирует потенциальное ограничение использования многомерного нормального распределения для подбора элитных выборок.

Пример 8.3. Пример использования метода перекрестной энтропии

Для представления, извлечения выборок и аппроксимации распределений предложений можно использовать пакет `Distributions.jl`. Вектор параметров θ заменяется распределением `P`. Вызов функции `rand(P, m)` создает матрицу $n \times m$, соответствующую m выборкам, извлеченным из множества n-мерных выборок, имеющих распределение `P`, а вызов функции `fit` аппроксимирует новое распределение по заданным входным данным.

```
srand(0) # начальное значение датчика случайных чисел
f = x -> norm(x)
μ = [0.5, 1.5]
Σ = [1.0 0.2; 0.2 2.0]
P = MvNormal(μ, Σ)
k_max = 10
P = cross_entropy_method(f, P, k_max)
@show P.μ

P.μ = [-6.13623e-7, -1.37216e-6]
```

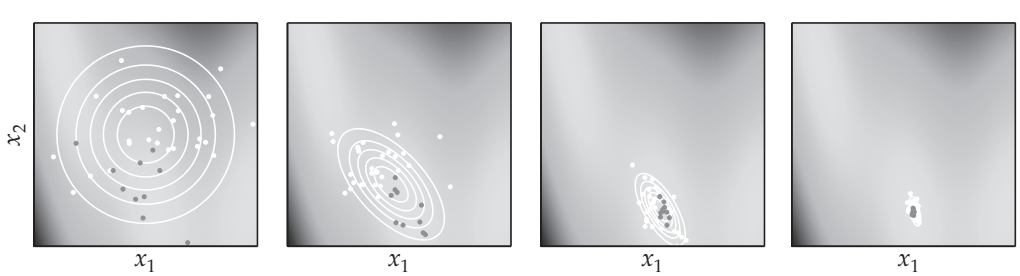
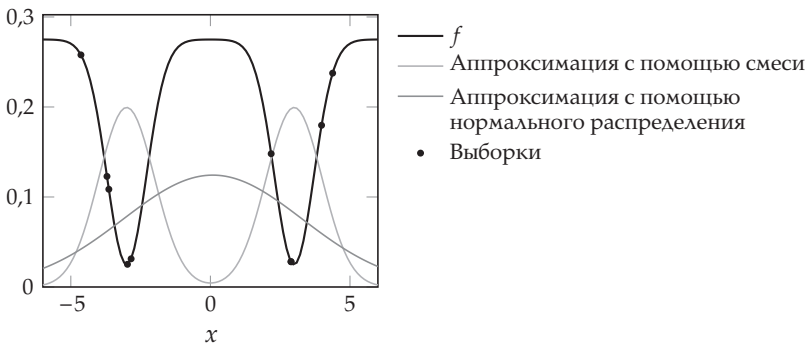


Рис. 8.6. Метод перекрестной энтропии с $m = 40$ применяется к функции Бранина (приложение Б.3) с использованием многомерного нормального распределения предложений. Десять элитных выборок на каждой итерации выделены красным

Пример 8.4. Нормальное распределение не в состоянии охватить несколько локальных минимумов, в отличие от моделей смеси, которые могут поддерживать несколько минимумов

Семейство распределения должно быть достаточно гибким, чтобы охватить соответствующие функции целевой функции. Здесь мы показываем ограничения использования нормального распределения для мультимодальной целевой функции, которая назначает бо́льшую плотность между двумя минимумами. Смешанная модель способна центрироваться над каждым минимумом.



8.5. Стратегии естественной эволюции

Как и метод перекрестной энтропии, стратегии естественной эволюции [128] оптимизируют распределение предложений, параметризованное параметром θ. Мы должны указать семейство распределений предложений и количество выборок. Цель состоит в том, чтобы минимизировать $\mathbb{E}_{\mathbf{x}\sim p(\cdot|\boldsymbol{\theta})}[f(\mathbf{x})]$. Вместо подбора элитных выборок в стратегиях эволюции применяется градиентный спуск. Градиент вычисляется по выборкам:[8]

$$\nabla_{\boldsymbol{\theta}}\mathbb{E}_{\mathbf{x}\sim p(\cdot|\boldsymbol{\theta})}\left[f(\mathbf{x})\right] = \int \nabla_{\boldsymbol{\theta}}\, p(\mathbf{x}|\boldsymbol{\theta})\, f(\mathbf{x})\, d\mathbf{x} = \tag{8.12}$$

$$= \int \frac{p(\mathbf{x}|\boldsymbol{\theta})}{p(\mathbf{x}|\boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} p(\mathbf{x}|\boldsymbol{\theta})\, f(\mathbf{x})\, d\mathbf{x} = \tag{8.13}$$

$$= \int p(\mathbf{x}|\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}|\boldsymbol{\theta})\, f(\mathbf{x}) d\mathbf{x} = \tag{8.14}$$

$$= \mathbb{E}_{\mathbf{x}\sim p(\cdot|\boldsymbol{\theta})}\left[f(\mathbf{x}) \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}|\boldsymbol{\theta})\right] \approx \tag{8.15}$$

[8] Эта процедура вычисления градиента была недавно успешно применена к распределению предложений, представленному глубокими нейронными сетями [136].

$$\approx \frac{1}{m}\sum_{i=1}^{m} f\left(\mathbf{x}^{(i)}\right)\nabla_{\boldsymbol{\theta}} \log p\left(\mathbf{x}^{(i)}\middle|\boldsymbol{\theta}\right). \tag{8.16}$$

Нам не нужен градиент целевой функции, в отличие от градиента логарифмического правдоподобия $\log p(\mathbf{x}|\boldsymbol{\theta})$. В примере 8.5 показано, как вычислить градиент логарифмического правдоподобия для многомерного нормального распределения. Предполагаемый градиент может быть использован вместе с любым из методов спуска, обсуждавшихся в предыдущих главах для улучшения параметра $\boldsymbol{\theta}$. Алгоритм 8.8 использует градиентный спуск с фиксированным размером шага. На рис. 8.7 показаны несколько итераций алгоритма.

Пример 8.5. Вывод формул для градиента логарифмического правдоподобия для многомерного нормального распределения. Исходный вывод и несколько более сложных решений для работы с положительно определенной ковариационной матрицей см. в [158]

Многомерное нормальное распределение $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ с математическим ожиданием $\boldsymbol{\mu}$ и ковариационной матрицей $\boldsymbol{\Sigma}$ является популярным семейством распределений благодаря наличию аналитических решений. Функция правдоподобия в d измерениях имеет вид

$$p\left(\mathbf{x}\middle|\boldsymbol{\mu},\boldsymbol{\Sigma}\right) = \left(2\pi\right)^{-d/2}\left|\boldsymbol{\Sigma}\right|^{-1/2}\exp\left(-\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}\right)^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}\right)\right),$$

где $|\boldsymbol{\Sigma}|$ — определитель матрицы $\boldsymbol{\Sigma}$. Функция логарифмического правдоподобия имеет вид:

$$\log p\left(\mathbf{x}\middle|\boldsymbol{\mu},\boldsymbol{\Sigma}\right) = -\frac{d}{2}\log 2\pi - \frac{1}{2}\log\left|\boldsymbol{\Sigma}\right| - \frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}\right)^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}\right).$$

Параметры могут вычисляться с использованием градиентов их логарифмического правдоподобия:

$$\nabla_{(\boldsymbol{\mu})}\log p\left(\mathbf{x}\middle|\boldsymbol{\mu},\boldsymbol{\Sigma}\right) = \boldsymbol{\Sigma}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}\right),$$

$$\nabla_{(\boldsymbol{\Sigma})}\log p\left(\mathbf{x}\middle|\boldsymbol{\mu},\boldsymbol{\Sigma}\right) = \frac{1}{2}\boldsymbol{\Sigma}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}\right)\left(\mathbf{x}-\boldsymbol{\mu}\right)^{\mathrm{T}}\boldsymbol{\Sigma}^{-1} - \frac{1}{2}\boldsymbol{\Sigma}^{-1}.$$

Член $\nabla_{(\boldsymbol{\Sigma})}$ содержит частную производную каждого элемента матрицы $\boldsymbol{\Sigma}$ по логарифмическому правдоподобию.

Непосредственное вычисление матрицы $\boldsymbol{\Sigma}$ может не привести к положительно определенной матрице, как это требуется для ковариационных матриц. Одним из решений является представление матрицы $\boldsymbol{\Sigma}$ как произведения $\mathbf{A}^{\mathrm{T}}\mathbf{A}$, которое гарантирует, что матрица $\boldsymbol{\Sigma}$ остается положительно полуопределенной при обновлении матрицы $\mathbf{A}$. Изменение $\boldsymbol{\Sigma}$ на $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ и вычисление градиента по $\mathbf{A}$ дает:

$$\nabla_{(\mathbf{A})} \log p\left(\mathbf{x}|\boldsymbol{\mu},\mathbf{A}\right) = \mathbf{A}\left[\nabla_{(\boldsymbol{\Sigma})} \log p\left(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}\right) + \nabla_{(\boldsymbol{\Sigma})} \log p\left(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}\right)^{\mathrm{T}}\right].$$

Алгоритм 8.8. Метод стратегий естественной эволюции, который получает целевую функцию `f`, вектор начального параметра распределения `θ`, счетчик итераций `k_max`, размер выборки `m` и шаговый множитель `α`. Возвращается оптимизированный вектор параметров `θ`. Метод `rand(θ)` должен производить выборку из распределения, параметризованного параметром `θ`, а функция `∇logp(x, θ)` — возвращать градиент логарифмического правдоподобия

```
using Distributions
function natural_evolution_strategies(f, θ,k_max; m = 100, α = 0.01)
    for k in 1 : k_max
        samples = [rand(θ) for i in 1 : m]
        θ -= α * sum(f(x) * ∇logp(x, θ) for x in samples) / m
    end
    return θ
end
```

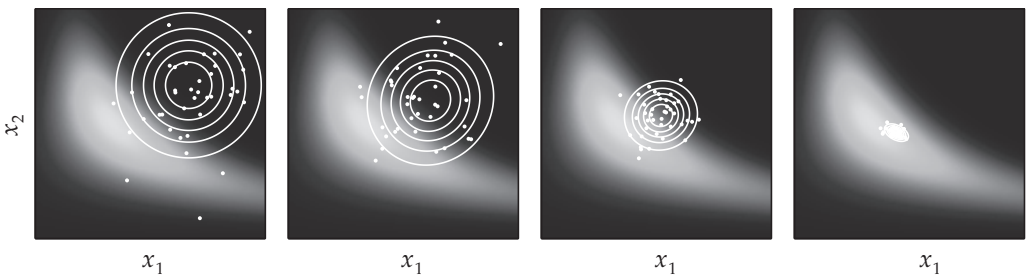


Рис. 8.7. Стратегии естественной эволюции с использованием многомерных нормальных распределений, применяемые к функции Уилера, приложение Б.7

8.6. Адаптация ковариационной матрицы

Другим популярным методом является *адаптация ковариационной матрицы*[9] (covariance matrix adaptation evolutionary strategy — *CMA-ES*). Она имеет сходство со стратегиями естественной эволюции из раздела 8.5, но их не следует путать. Этот метод поддерживает ковариационную матрицу и является надежным и эффективным способов выбора. Как и метод перекрестной энтропии и стратегии

[9] Обычно под эволюционной стратегией подразумевают именно адаптацию ковариационной матрицы.

естественной эволюции, распределение со временем улучшается на основе выборок. При адаптации ковариационной матрицы используются многомерные нормальные распределения [67].

Адаптация ковариационной матрицы поддерживает вектор математических ожиданий $\boldsymbol{\mu}$, ковариационную матрицу $\boldsymbol{\Sigma}$ и дополнительный скаляр — шаговый множитель σ. Ковариационная матрица с каждой итерацией увеличивается или уменьшается только в одном направлении, тогда как шаговый множитель адаптируется для управления общим разбросом распределения. На каждой итерации из многомерного нормального распределения извлекаются m выборок:[10]

$$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2\boldsymbol{\Sigma}). \tag{8.17}$$

Затем точки сортируются в соответствии со значениями целевой функции, так что $f(\mathbf{x}^{(1)}) \leq f(\mathbf{x}^{(2)}) \leq \ldots \leq f(\mathbf{x}^{(m)})$. Новый вектор математических ожиданий $\boldsymbol{\mu}^{(k+1)}$ формируется с использованием взвешенного среднего значения выборок:

$$\boldsymbol{\mu}^{(k+1)} \leftarrow \sum_{i=1}^{m} w_i \mathbf{x}^{(i)}. \tag{8.18}$$

где весовые коэффициенты равны единице, упорядочены по убыванию и неотрицательны:[11]

$$\sum_{i=1}^{m} w_i = 1, \qquad w_1 \geq w_2 \geq \ldots \geq w_m \geq 0. \tag{8.19}$$

Мы можем воспроизвести обновление математического ожидания по методу перекрестной энтропии, установив первые m_{elite} весов равными $1/m_{\text{elite}}$, а остальные веса — равными нулю. Метод адаптации ковариационной матрицы также присваивает вес только первым m_{elite} точкам, но распределяет веса неравномерно. Рекомендуемый вес получается путем нормализации

$$w_i' = \ln\frac{m+1}{2} - \ln i \text{ для } i \in \{1, \ldots, m\}, \tag{8.20}$$

которая приводит к весам $\mathbf{w} = \mathbf{w}' \Big/ \sum_i w_i'$. На рис. 8.8 приведены результаты сравнения обновления математических ожиданий в методах адаптации ковариационной матрицы и перекрестной энтропии.

---

[10] Для оптимизации в пространстве $\mathbb{R}^n$ рекомендуется использовать не менее $m = 4 + \lfloor 3\ln n \rfloor$ выборок на итерацию и $m_{\text{elite}} = \lfloor m/2 \rfloor$ элитных выборок.

[11] В оригинальной работе [67] были предложены обобщения, которые не подчиняются этим ограничениям, но рекомендуемая авторами реализация на самом деле их учитывает. Алгоритм 8.9 придерживается этих ограничений.

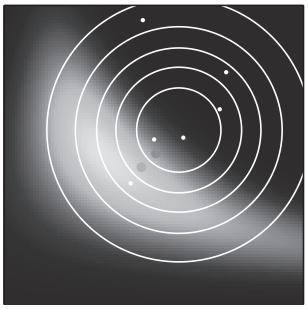


Рис. 8.8. Начальное распределение предложения (белые контуры), шесть выборок (белые точки) и новое обновленное математическое ожидание в методах адаптации ковариационной матрицы (синяя точка) и перекрестной энтропии (красная точка) с использованием трех элитных выборок. Метод адаптации ковариационной матрицы имеет тенденцию обновлять математическое ожидание чаще, чем метод перекрестной энтропии (красная точка), так как он присваивает больший вес точкам с лучшей выборкой

Шаговый множитель обновляется с использованием кумулятивной переменной $\mathbf{p}_\sigma$, которая отслеживает шаги с течением времени:

$$\begin{aligned}&\mathbf{p}_\sigma^{(1)} = 0,\\ &\mathbf{p}_\sigma^{(k+1)} \leftarrow (1-c_\sigma)\mathbf{p}_\sigma + \sqrt{c_\sigma(2-c_\sigma)\mu_{\text{eff}}}\left(\boldsymbol{\Sigma}^{(k)}\right)^{-1/2}\boldsymbol{\delta}_w,\end{aligned} \tag{8.21}$$

где параметр $c_\sigma < 1$ управляет скоростью затухания, а правое слагаемое определяет, следует ли увеличивать или уменьшать шаговый множитель на основе наблюдаемых выборок по отношению к текущему масштабу распределения. Дисперсия эффективной выборочной массы μ_{eff} имеет вид

$$\mu_{\text{eff}} = \frac{1}{\sum_i w_i^2}, \tag{8.22}$$

а $\boldsymbol{\delta}_w$ вычисляется по выборочным девиациям:

$$\boldsymbol{\delta}_w = \sum_{i=1}^{m_{\text{elite}}} w_i \boldsymbol{\delta}^{(i)}, \text{ где } \boldsymbol{\delta}^{(i)} = \frac{\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(k)}}{\sigma^{(k)}}. \tag{8.23}$$

Новый размер шага получается по формуле

$$\sigma^{(k+1)} \leftarrow \sigma^{(k)} \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\mathbf{p}_\sigma\|}{\mathbb{E}\|\mathcal{N}(\mathbf{0}, \mathbf{I})\|} - 1\right)\right), \tag{8.24}$$

где

$$\mathbb{E}\left\|\mathcal{N}\left(\mathbf{0},\mathbf{I}\right)\right\| = \sqrt{2}\frac{\Gamma\left(\frac{n+1}{2}\right)}{\Gamma\left(\frac{n}{2}\right)} \approx \sqrt{n}\left(1-\frac{1}{4n}+\frac{1}{21n^2}\right) \tag{8.25}$$

является ожидаемой длиной вектора из нормального распределения. Сравнение длины $\mathbf{p}_\sigma$ с его ожидаемой длиной при случайном выборе обеспечивает механизм, с помощью которого увеличивается или уменьшается параметр σ. Рекомендуемые значения констант c_σ и d_σ:

$$\begin{aligned} c_\sigma &= \left(\mu_{\text{eff}}+2\right)/\left(n+\mu_{\text{eff}}+5\right), \\ d_\sigma &= 1+2\max\left(0,\ \sqrt{\left(\mu_{\text{eff}}-1\right)/\left(n+1\right)}-1\right)+c_\sigma. \end{aligned} \tag{8.26}$$

Ковариационная матрица также обновляется с использованием кумулятивного вектора:

$$\begin{aligned} \mathbf{p}_\Sigma^{(1)} &= \mathbf{0}, \\ \mathbf{p}_\Sigma^{(k+1)} &= \left(1-c_\Sigma\right)\mathbf{p}_\Sigma^{(k)} + h_\sigma\sqrt{c_\Sigma\left(2-c_\Sigma\right)\mu_{\text{eff}}}\,\boldsymbol{\delta}_w, \end{aligned} \tag{8.27}$$

где

$$h_\sigma = \begin{cases} 1, \text{ если } \dfrac{\left\|\mathbf{p}_\Sigma\right\|}{\sqrt{1-\left(1-c_\sigma\right)^{2(k+1)}}} < \left(1{,}4+\dfrac{2}{n+1}\right)\mathbb{E}\left\|\mathcal{N}\left(\mathbf{0},\ \mathbf{I}\right)\right\|, \\ 0 \text{ в противном случае.} \end{cases} \tag{8.28}$$

Параметр h_σ гасит обновление $\mathbf{p}_\Sigma$, если $\|\mathbf{p}_\Sigma\|$ слишком велика, тем самым предотвращая чрезмерное увеличение $\boldsymbol{\Sigma}$, когда шаговый множитель слишком мал.

Обновление требует скорректированных весов $\mathbf{w}^\circ$:

$$w_i^\circ = \begin{cases} w_i, \text{ если } w_i \geq 0, \\ \dfrac{nw_i}{\left\|\boldsymbol{\Sigma}^{-1/2}\boldsymbol{\delta}^{(i)}\right\|^2} \text{ в противном случае.} \end{cases} \tag{8.29}$$

В таком случае формулы обновления ковариации принимают вид

$$\begin{aligned} \boldsymbol{\Sigma}^{(k+1)} \leftarrow \Bigg(1+\underbrace{c_1c_c\left(1-h_\sigma\right)\left(2-c_c\right)}_{\text{Как правило, нуль}}\Bigg)\boldsymbol{\Sigma}^{(k)} + \underbrace{c_1\mathbf{p}_\Sigma\mathbf{p}_\Sigma^{\mathrm{T}}}_{\text{Обновление первого ранга}} + \\ + \underbrace{c_\mu\sum_{i=1}^{\mu}w_i^\circ\boldsymbol{\delta}^{(i)}\left(\boldsymbol{\delta}^{(i)}\right)^{\mathrm{T}}}_{\text{Обновление ранга }\mu}. \end{aligned} \tag{8.30}$$

Рекомендуемые значения констант $c_{\boldsymbol{\Sigma}}$, c_1 и c_μ:

$$c_{\boldsymbol{\Sigma}} = \frac{4 + \mu_{\text{eff}}/n}{n + 4 + 2\mu_{\text{eff}}/n},$$
$$c_1 = \frac{2}{(n+1{,}3)^2 + \mu_{\text{eff}}}, \tag{8.31}$$
$$c_\mu = \min\left(1 - c_1,\ 2\frac{\mu_{\text{eff}} - 2 + 1/\mu_{\text{eff}}}{(n+2)^2 + \mu_{\text{eff}}}\right).$$

Обновление ковариации состоит из трех компонентов: предыдущая ковариационная матрица $\boldsymbol{\Sigma}^{(k)}$, обновление первого ранга и обновление ранга μ. Обновление первого ранга получило свое название благодаря тому факту, что матрица $\mathbf{p}_{\boldsymbol{\Sigma}}\mathbf{p}_{\boldsymbol{\Sigma}}^{\mathrm{T}}$ имеет первый ранг; у нее есть только один собственный вектор, направленный вдоль вектора $\mathbf{p}_{\boldsymbol{\Sigma}}$.

Обновления первого ранга с использованием вектора кумуляции позволяют использовать корреляции между последовательными шагами, давая возможность ковариационной матрице быстрее удлиняться вдоль подходящей оси. Обновление ранга μ получило свое название из-за того, что матрица $\sum_{i=1}^{\mu} w_i^{\circ}\boldsymbol{\delta}^{(i)}\left(\boldsymbol{\delta}^{(i)}\right)^{\mathrm{T}}$ имеет ранг $\min(\mu, n)$. Одно важное различие между обновлением эмпирической ковариационной матрицы, используемым методом перекрестной энтропии и обновлением ранга μ заключается в том, что первое оценивает ковариацию относительно нового математического ожидания $\boldsymbol{\mu}^{(k+1)}$, тогда как второе оценивает ковариацию относительно исходного математического ожидания $\boldsymbol{\mu}^{(k)}$. Таким образом, значения $\boldsymbol{\delta}^{(i)}$ помогают оценить отклонения выбранных шаговых множителей, а не дисперсию выбранных точек. Адаптация ковариационной матрицы изображена на рис. 8.9.

Алгоритм 8.9. Метод адаптации ковариационной матрицы, получающий целевую функцию `f`, начальную расчетную точку `x` и счетчик итераций `k_max`. При желании можно указать скалярный шаговый множитель `σ`, размер выборки `m` и количество элитных выборок `m_elite`. Возвращается лучшая точка-кандидат, которая является математическим ожиданием окончательного выборочного распределения. Ковариационная матрица подвергается дополнительной операции, чтобы гарантировать, что она остается симметричной; в противном случае небольшие численные несоответствия могут привести к тому, что матрица перестанет быть положительно определенной

```
function covariance_matrix_adaptation(f, x, k_max;
    σ = 1.0,
    m = 4 + floor(Int, 3 * log(length(x))),
    m_elite = div(m, 2))
```

```
    μ, n = copy(x), length(x)
    ws = normalize!(vcat(log((m + 1) / 2) .- [log(i) for i in 1 : m_elite],
                    zeros(m - m_elite)), 1)
    μ_eff = 1 / sum(ws. ^ 2)
    cσ = (μ_eff + 2)/(n + μ_eff + 5)
    dσ = 1 + 2max(0, sqrt((μ_eff - 1) / (n + 1)) - 1) + cσ
    cΣ = (4 + μ_eff / n)/(n + 4 + 2μ_eff / n)
    c1 = 2 / ((n + 1.3) ^ 2 + μ_eff)
    cμ = min(1 - c1, 2 * (μ_eff - 2 + 1 / μ_eff) / ((n + 2) ^ 2 + μ_eff))
    E = n ^ 0.5 * (1 - 1 / (4n) + 1 / (21 * n ^ 2))
    pσ, pΣ, Σ = zeros(n), zeros(n), Matrix(1.0I, n, n)
    for k in 1 : k_max
        P = MvNormal(μ, σ ^ 2 * Σ)
        xs = [rand(P) for i in 1 : m]
        ys = [f(x) for x in xs]
        is = sortperm(ys) # от лучшего значения к худшему

        # Выборка и обновление математического ожидания
        δs = [(x - μ)/σ for x in xs]
        δw = sum(ws[i] * δs[is[i]] for i in 1 : m_elite)
        μ += σ * δw

        # Управление шаговым множителем
        C = Σ ^- 0.5
        pσ = (1 - cσ) * pσ + sqrt(cσ * (2 - cσ) * μ_eff) * C * δw
        σ *= exp(cσ/dσ * (norm(pσ)/E - 1))

        # Адаптация ковариации
    hσ = norm(pσ) / sqrt(1 - (1 - cσ) ^ (2k)) < (1.4 + 2/(n + 1)) * E ? 1 : 0
        pΣ = (1 - cΣ) * pΣ + hσ * sqrt(cΣ * (2 - cΣ) * μ_eff) * δw
        w0 = [ws[i] ≥ 0 ? ws[i] : n * ws[i] / norm(C * δs[is[i]]) ^ 2
              for i in 1 : m]
        Σ = (1 - c1 - cμ) * Σ +
            c1 * (pΣ * pΣ' + (1 - hσ) * cΣ * (2 - cΣ) * Σ) +
            cμ * sum(w0[i] * δs[is[i]] * δs[is[i]]' for i in 1 : m)
        Σ = triu(Σ) + triu(Σ, 1)' # гарантия симметричности
    end
    return μ
end
```

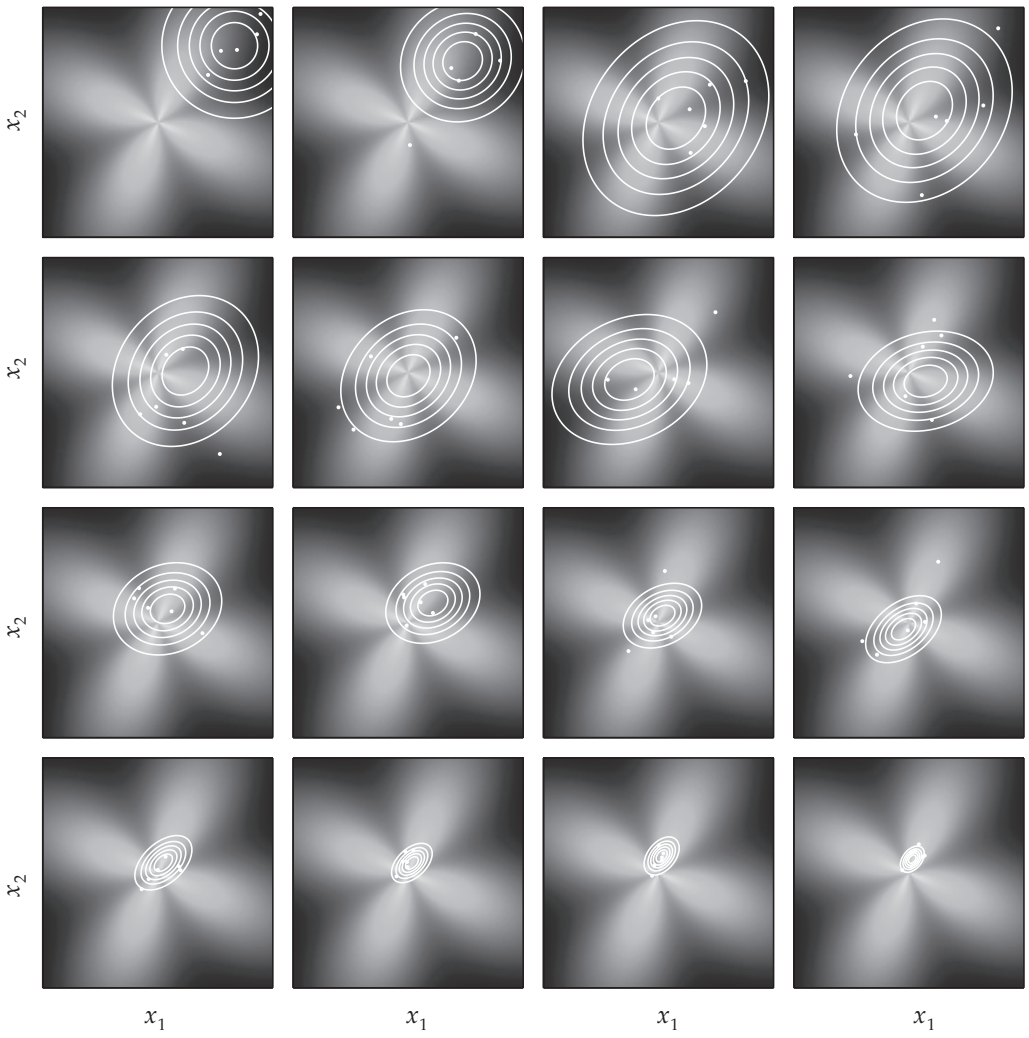


Рис. 8.9. Адаптация ковариационной матрицы с использованием многомерных нормальных распределений, примененных к функции цветка, приложение Б.4

8.7. Резюме

- Стохастические методы используют случайные числа в процессе оптимизации.
- В методе имитации отжига используется температура, которая контролирует случайный поиск и со временем уменьшается, обеспечивая сходимость к локальному минимуму.

- Метод перекрестной энтропии и стратегии эволюции поддерживают распределения предложений, из которых они отбирают выборки для формирования обновлений.
- В стратегиях естественной эволюции для обновления распределения предложений используется градиентный спуск по логарифмическому правдоподобию.
- Метод адаптации ковариационной матрицы — это надежный метод оптимизации с эффективным использованием выборок, который поддерживает многомерное нормальное распределение предложений с полной ковариационной матрицей.

8.8. Упражнения

Упражнение 8.1. Мы показали, что смешанные распределения предложений могут лучше учитывать несколько минимумов. Почему их использование в методе перекрестной энтропии может быть ограничено?

Упражнение 8.2. Каков потенциальный эффект от использования в методе перекрестной энтропии элитного размера выборки, который очень близок к общему размеру выборки?

Упражнение 8.3. Логарифмическое правдоподобие значения, выбранного из нормального распределения с математическим ожиданием значением μ и дисперсией v, задается формулой:

$$l\left(x \middle| \mu,\ v\right) = -\frac{1}{2}\ln 2\pi - \frac{1}{2}\ln v - \frac{\left(x-\mu\right)^2}{2v}.$$

Покажите, почему стратегии эволюции, использующие нормальные распределения, могут столкнуться с трудностями при применении обновления спуска для дисперсии, когда математическое ожидание находится на оптимуме, т.е. $\mu = x^*$.

Упражнение 8.4. Выведите оценку максимального правдоподобия для метода перекрестной энтропии, используя многомерное нормальное распределение:

$$\boldsymbol{\mu}^{(k+1)} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{x}^{(i)},$$

$$\boldsymbol{\Sigma}^{(k+1)} = \frac{1}{m}\sum_{i=1}^{m}\left(\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(k+1)}\right)\left(\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(k+1)}\right)^{\mathrm{T}},$$

где оценки максимального правдоподобия — это значения параметров, которые максимизируют вероятность выбора точек $\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(m)}\}$.

Глава 9

Популяционные методы

В предыдущих главах основное внимание уделялось методам, в которых единственная расчетная точка постепенно перемещается к минимуму. В этой главе представлены различные *популяционные методы*, которые включают оптимизацию с использованием набора расчетных точек, называемых *особями*. Распределение большого количества особей по пространству параметров проектирования может помочь алгоритму не застрять в локальном минимуме. Информация в разных точках пространства параметров проектирования может быть разделена между особями для глобальной оптимизации целевой функции. Большинство популяционных методов имеют стохастическую природу, и, как правило, распараллелить вычисления нетрудно.

9.1. Инициализация

Выполнение популяционных методов начинается с выбора начальной популяции, точно так же, как методы спуска требуют выбора начальной расчетной точки. Начальная популяция должна быть распределена по пространству параметров проекта, чтобы увеличить вероятность того, что выборки будут близки к лучшим областям. В этом разделе представлено несколько методов инициализации, но более продвинутые методы выбора подробно обсуждаются в главе 13.

Мы часто можем ограничить переменные проекта интересующей нас областью, состоящей из гиперпрямоугольника, определенного нижними и верхними границами `a` и `b`. Исходные популяции могут быть отобраны из равномерного распределения для каждой координаты:[1]

$$\mathbf{x}_i^{(j)} \sim U\left(a_i, b_i\right), \tag{9.1}$$

где $\mathbf{x}^{(j)}$ — это j-я особь в популяции, как видно из алгоритма 9.1.

[1] Некоторые популяционные методы требуют дополнительной информации, такой как скорость в случае оптимизации методом роя частиц. Скорость часто инициализируется в соответствии с равномерным или нормальным распределением.

Алгоритм 9.1. Метод равномерного выбора исходной совокупности `m` расчетных точек в гиперпрямоугольнике с координатами нижней границы, заданными вектором `a`, и координатами верхней границы, заданными вектором `b`

```
function endrand_population_uniform(m, a, b)
    d = length(a)
    return [a + rand(d) .* (b - a) for i in 1 : m]
end
```

Другим распространенным подходом является использование многомерного нормального распределения с центром в интересующей области. Ковариационная матрица обычно диагональная, а диагональные элементы масштабируются так, чтобы адекватно охватывать пространство поиска. Описание этого подхода приведено в алгоритме 9.2.

Алгоритм 9.2. Метод отбора исходной популяции из `m` расчетных точек с использованием многомерного нормального распределения со средним `μ` и ковариацией `Σ`

```
using Distributions
function rand_population_normal(m, μ, Σ)
    D = MvNormal(μ, Σ)
    return [rand(D) for i in 1 : m]
end
```

Равномерное и нормальное распределение ограничивает пространство параметров проекта определенной областью. *Распределение Коши* (рис. 9.1) имеет неограниченную дисперсию и может охватывать гораздо более широкое пространство. Его реализация описана в алгоритме 9.3. На рис. 9.2 сравниваются примеры исходных популяций, полученных с использованием различных методов.

Алгоритм 9.3. Метод отбора начальной популяции из `m` расчетных точек с применением распределения Коши с параметром сдвига `μ` и параметром масштаба `σ` для каждого измерения. Параметры сдвига и масштаба аналогичны математическому ожиданию и стандартному отклонению, используемому при описании нормального распределения

```
using Distributions
function rand_population_cauchy(m, μ, σ)
    n = length(μ)
    return [[rand(Cauchy(μ[j], σ[j])) for j in 1 : n] for i in 1 : m]
end
```

9.2. Генетические алгоритмы

Генетические алгоритмы (алгоритм 9.4) заимствуют идеи у биологической эволюции, в ходе которой более здоровые особи с большей вероятностью передают свои гены следующему поколению [57]. Пригодность особи к размножению обратно пропорциональна значению целевой функции в этой точке. Расчетная точка, связанная с особью, представлена в виде *хромосомы*. В каждом поколении хромосомы более приспособленных особей передаются следующему поколению после прохождения генетических операций *кроссовера* и *мутации*.

Алгоритм 9.4. Генетический алгоритм, который принимает целевую функцию `f`, начальную популяцию, число итераций `k_max`, `SelectionMethod S`, `CrossoverMethod C` и `MutationMethod M`

```
function genetic_algorithm(f, population, k_max, S, C, M)
    for k in 1 : k_max
        parents = select(S, f.(population))
        children = [crossover(C, population[p[1]], population[p[2]])
                    for p in parents]
        population .= mutate.(Ref(M), children)
    end
    population[argmin(f.(population))]
end
```

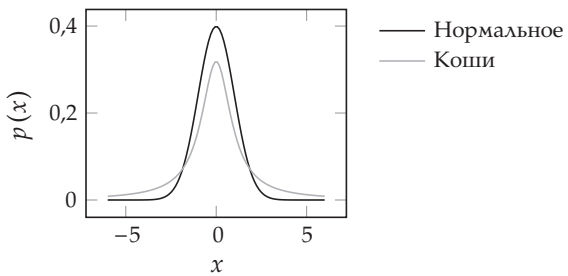


Рис. 9.1. Сравнение нормального распределения с единичным стандартным отклонением и распределения Коши с единичным параметром масштаба. Хотя для обозначения параметра масштаба в распределении Коши иногда используется символ σ, его не следует путать со стандартным отклонением, поскольку стандартное отклонение распределения Коши не определено. Распределение Коши имеет тяжелый хвост, что позволяет ему более широко охватывать пространство параметров проектирования

9.2.1. Хромосомы

Существует несколько способов представления хромосом. Простейшим является *двоичная хромосомная цепочка*, представление, похожее на способ кодирова-

ния ДНК.[2] Случайную двоичную строку длины d можно сгенерировать с помощью функции `bitrand(d)`. Двоичная цепочка хромосом изображена на рис. 9.3.

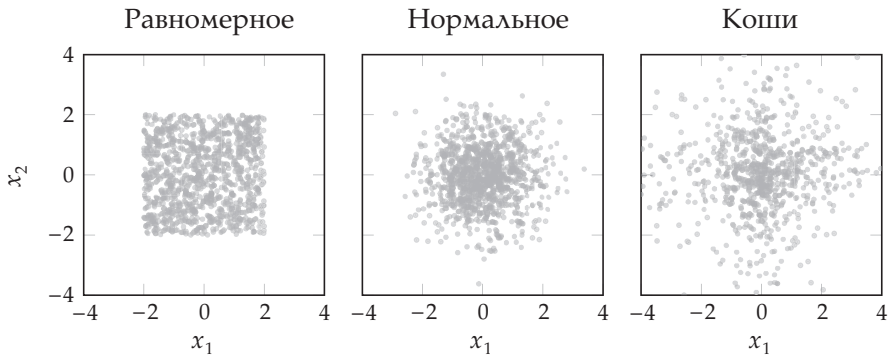


Рис. 9.2. Начальные популяции, состоящие из 1000 выборок, имеющих равномерное распределение в гиперпрямоугольнике с вершинами $\mathbf{a} = [-2, -2]$ и $\mathbf{b} = [2, 2]$, нормальное распределение и нулевым математическим ожиданием и диагональной ковариационной матрицей $\mathbf{\Sigma} = \mathbf{I}$ и распределение Коши с центром в начале координат и параметром масштаба $\sigma = 1$

Рис. 9.3. Представление хромосомы в виде двоичной цепочки

Двоичные цепочки часто используются из-за легкости выражения кроссовера и мутации. К сожалению, процесс декодирования двоичной цепочки и создания расчетной точки не всегда прост. Иногда двоичная цепочка может не представлять допустимую точку в области параметров проекта. Часто более естественно представлять хромосому, используя список действительных чисел. Такие хромосомы являются векторами в пространстве $\mathbb{R}^d$, которые непосредственно соответствуют точкам в пространстве параметров проекта.

9.2.2. Инициализация

Генетические алгоритмы начинаются со случайной начальной популяции. Двоичные хромосомные цепочки обычно инициализируются с использованием случайных битовых строк, как показано в алгоритме 9.5. Хромосомы, представляющие собой список действительных чисел, обычно инициализируются с использованием методов из предыдущего раздела.

[2] Вместо двоичного представления ДНК содержит четыре нуклеиновых основания: аденин, тимин, цитозин и гуанин, которые часто обозначаются буквами A, T, C и G.

Алгоритм 9.5. Метод отбора случайных начальных популяций `m` битовых хромосомных строк длиной `n`

```
rand_population_binary(m, n) = [bitrand(n) for i in 1 : m]
```

9.2.3. Отбор

Отбор — это процесс выбора хромосом для использования в качестве родителей для следующего поколения. Для популяции с m хромосомами метод отбора даст список из m родительских пар[3] для m потомков из следующего поколения. Выбранные пары могут содержать дубликаты.

Существует несколько подходов для смещения выбора к наиболее подходящему (алгоритм 9.6). При *отсекающем отборе* (рис. 9.4) мы выбираем родителей из числа лучших k хромосом в популяции. При *турнирном отборе* (рис. 9.5) каждый родитель конкурирует с k случайно выбранными хромосомами популяции по степени приспособленности. При отборе *методом рулетки* (рис. 9.6), также известном как *пропорциональный отбор*, каждый родитель выбирается с вероятностью, пропорциональной степени его приспособленности по отношению к популяции. Поскольку мы заинтересованы в минимизации целевой функции f, приспособленность i-й особи $x^{(i)}$ обратно связана с $y^{(i)} = f(x^{(i)})$. Существуют разные способы преобразования коллекции $y^{(1)}$, ..., $y^{(m)}$ в оценку приспособленности. Простой подход состоит в том, чтобы назначить степень приспособленности особи i в соответствии с функцией $\max\{y^{(1)}, ..., y^{(m)}\} - y^{(i)}$.

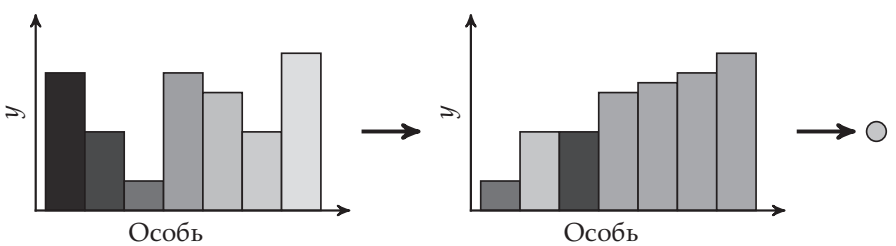


Рис. 9.4. Отсекающий выбор с размером популяции $m = 7$ и размером выборки $k = 3$. Высота столбца указывает значение его целевой функции, тогда как цвет определяет, какой особи он соответствует

[3] В некоторых случаях можно использовать группы, если вы хотите объединить более двух родителей, чтобы сформировать потомка.

Алгоритм 9.6. Несколько методов отбора для генетических алгоритмов. Вызов функции `selection` с параметром `SelectionMethod` и списком значений целевой функции `f` создает список родительских пар

```
abstract type SelectionMethod end

struct TruncationSelection <: SelectionMethod
    k # сохраняем лучшие k особей
end
function select(t :: TruncationSelection, y)
     p = sortperm(y)
     return [p[rand(1 : t.k, 2)] for i in y]
end

struct TournamentSelection <: SelectionMethod
     k
end
function select(t :: TournamentSelection, y)
     getparent() = begin
         p = randperm(length(y))
         p[argmin(y[p[1 : t.k]])]
      end
      return [[getparent(), getparent()] for i in y]
end

struct RouletteWheelSelection <: SelectionMethod end
function select( :: RouletteWheelSelection, y)
       y = maximum(y) .- y
       cat = Categorical(normalize(y, 1))
       return [rand(cat, 2) for i in y]
end
```

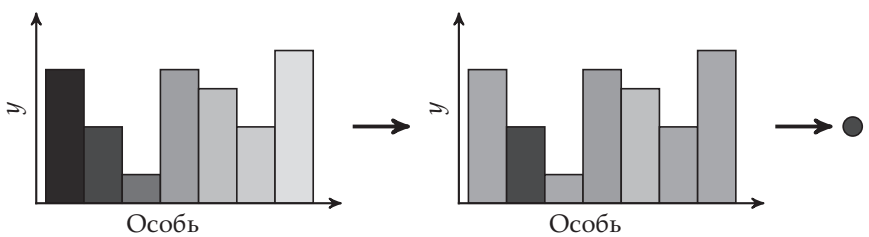


Рис. 9.5. Турнирный отбор с размером популяции $m = 7$ и размером выборки $k = 3$, который проводится отдельно для каждого родителя. Высота столбца указывает значение его целевой функции, тогда как цвет определяет, какой особи он соответствует

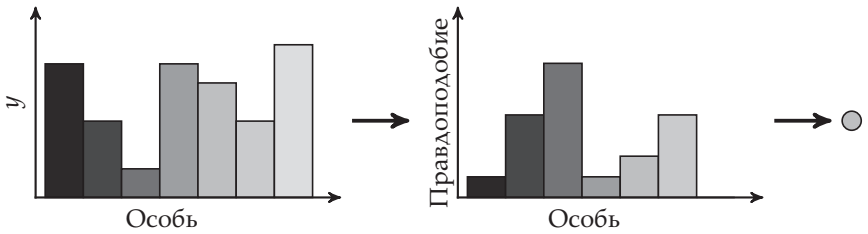


Рис. 9.6. Отбор методом рулетки с размером популяции $m = 7$, который запускается отдельно для каждого родителя. Используемый подход приводит к тому, что особь с наихудшим значением целевой функции имеет нулевую вероятность выбора. Высота столбца указывает значение его целевой функции (слева) или его вероятность (справа), тогда как цвет определяет, какой особи он соответствует

9.2.4. Кроссовер

Кроссовер объединяет хромосомы родителей, чтобы создать потомков. Как и в случае отбора, существует несколько схем кроссовера (алгоритм 9.7).

- При *одноточечном кроссовере* (рис. 9.7) первая часть родительской хромосомы A образует первую часть дочерней хромосомы, а последняя часть родительской хромосомы B образует последнюю часть дочерней хромосомы. *Точка кроссовера*, в которой происходит переход, определяется равномерно случайным образом.

Рис. 9.7. Одноточечный кроссовер

- В *двухточечном кроссовере* (рис. 9.8) мы используем две случайные точки кроссовера.

Рис. 9.8. Двухточечный кроссовер

- При *равномерном кроссовере* (рис. 9.9) каждый бит с вероятностью в 50% поступает от одного из двух родителей. Эта схема эквивалентна тому, что каждая точка имеет 50% шансов стать точкой кроссовера.

Рис. 9.9. Равномерный кроссовер

Предыдущие методы кроссовера также работают с хромосомами, представляющими собой список действительных чисел. Однако мы можем определить дополнительную процедуру кроссовера, которая интерполирует действительные значения (алгоритм 9.8). Здесь действительные значения линейно интерполируются между родительскими значениями $\mathbf{x}_a$ и $\mathbf{x}_b$:

$$\mathbf{x} \leftarrow (1-\lambda)\mathbf{x}_a + \lambda\mathbf{x}_b, \tag{9.2}$$

где λ является скалярным параметром, обычно равным половине.

Алгоритм 9.7. Несколько методов кроссовера для генетических алгоритмов. Выполнение кроссовера с помощью метода `CrossoverMethod` и двух родителей `a` и `b` приведет к образованию хромосомы потомка, которая содержит смесь генетических кодов родителей. Эти методы работают как с бинарной цепочкой, так и с хромосомами, представляюшими собой список действительных чисел

```
abstract type CrossoverMethod end
struct SinglePointCrossover <: CrossoverMethod end
function crossover( :: SinglePointCrossover, a, b)
    i = rand(1 : length(a))
    return vcat(a[1 : i], b[i + 1 : end])
end

struct TwoPointCrossover <: CrossoverMethod end
function crossover( :: TwoPointCrossover, a, b)
    n = length(a)
    i, j = rand(1 : n, 2)
    if i > j
        (i, j) = (j, i)
    end
    return vcat(a[1 : i], b[i + 1 : j], a[j + 1 : n])
end

struct UniformCrossover <: CrossoverMethod end
function crossover( :: UniformCrossover, a, b)
    child = copy(a)
    for i in 1 : length(a)
```

```
        if rand() < 0.5
            child[i] = b[i]
        end
    end
    return child
end
```

Алгоритм 9.8. Метод кроссовера для хромосом, состоящих из действительных чисел, выполняющий линейную интерполяцию между родителями

```
struct InterpolationCrossover <: CrossoverMethod
    λ
end
crossover(C :: InterpolationCrossover, a, b) = (1 - C.λ) * a + C.λ * b
```

9.2.5. Мутация

Если бы новые хромосомы были получены только путем кроссовера, то многие признаки, которые не присутствовали в исходной случайной популяции, никогда не могли бы возникнуть, и наиболее приспособленные гены могли бы заполнить всю популяцию. Мутация позволяет спонтанно появляться новым признакам, давая возможность генетическому алгоритму исследовать больше пространства состояний. Хромосомы потомков подвергаются мутации после кроссовера.

Каждый бит в бинарной хромосоме обычно имеет небольшую вероятность инвертирования (рис. 9.10). Для хромосомы с m битами эта *частота мутаций* обычно устанавливается равной $1/m$, что дает в среднем одну мутацию на хромосому среди потомков. Мутация для хромосом, состоящих из действительных чисел, может быть реализована с помощью побитового инвертирования, но чаще добавляется гауссовский шум с нулевым математическим ожиданием. Реализация этого подхода описана в алгоритме 9.9.

Алгоритм 9.9. Метод побитовой мутации для двоичных струнных хромосом и метод гауссовой мутации для хромосом, состоящих из действительных чисел

```
abstract type MutationMethod end
struct BitwiseMutation <: MutationMethod
    λ
end
function mutate(M :: BitwiseMutation, child)
    return [rand() < M.λ ? !v : v for v in child]
end
```

```
struct GaussianMutation <: MutationMethod
   σ
end
function mutate(M :: GaussianMutation, child)
   return child + randn(length(child)) * M.σ
end
```

Рис. 9.10. Мутация двоичной хромосомной цепочки присваивает каждому биту небольшую вероятность инвертирования

На рис. 9.11 показано несколько поколений генетического алгоритма. В примере 9.1 демонстрируется, как объединить стратегии отбора, кроссовера и мутации, обсуждаемые в этом разделе.

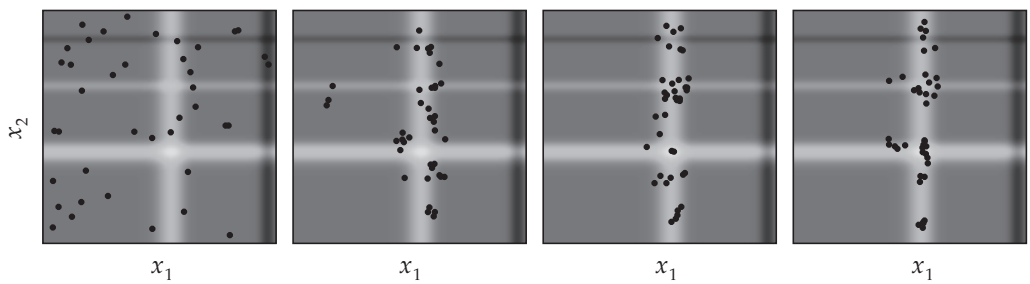


Рис. 9.11. Генетический алгоритм с выбором усечения, пересечением в одной точке и гауссовой мутацией с $\sigma = 0{,}1$, примененный к функции Михалевича, определенной в приложении Б.5

Пример 9.1. Демонстрация использования генетического алгоритма для оптимизации простой функции

```
srand(0) # задаем случайное начальное число для воспроизводимости
         # результатов
f = x -> norm(x)
m = 100      # размер популяции
k_max = 10  # количество итераций
population = rand_population_uniform(m, [-3, 3], [3, 3])
S = TruncationSelection(10)    # выбираем 10 лучших
C = SinglePointCrossover()
M = GaussianMutation(0.5)      # небольшой уровень мутаций
x = genetic_algorithm(f, population, k_max, S, C, M)
```

```
@show x

x = [-0.0130108, 0.0661564]
```

9.3. Дифференциальная эволюция

Дифференциальная эволюция (алгоритм 9.10) пытается улучшить каждую особь в популяции путем рекомбинации других особей в популяции в соответствии с простой формулой [32]. Она параметризуется вероятностью пересечения p и дифференциальным весом w. Как правило, параметр w лежит в интервале 0,4 до 1. Для каждого отдельного x выполняем следующие действия.

1. Выбираем три случайные разные особи **a**, **b** и **c**.
2. Вычисляем промежуточную расчетную точку **z** = **a** + w(**b** – **c**), как показано на рис. 9.12.

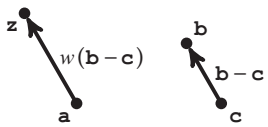


Рис. 9.12. Дифференциальная эволюция отбирает три особи **a**, **b** и **c** и объединяет их, чтобы сформировать особь-кандидат **z**

3. Выбираем случайное измерение $j \in [1, ..., n]$ для оптимизации по n измерениям.
4. Создаем особь-кандидат **x′**, используя бинарный кроссовер.

$$x_i' = \begin{cases} z_i, \text{ если } i = j \text{ с вероятностью } p, \\ x_i \text{ в противном случае.} \end{cases} \tag{9.3}$$

5. Включаем лучшую точку из **x** и **x′** в следующее поколение.

Алгоритм продемонстрирован на рис. 9.13.

Алгоритм 9.10. Дифференциальная эволюция, которая принимает целевую функцию **f**, популяцию **population**, число итераций **k_max**, вероятность кроссовера **p**, дифференциальный вес **w** и возвращает лучшую особь

```
using StatsBase
function differential_evolution(f, population, k_max; p = 0.5, w = 1)
    n, m = length(population[1]), length(population)
    for k in 1 : k_max
        for (k, x) in enumerate(population)
            weights = Weights([j != k for j in 1 : m])
            a, b, c = sample(population, weights, 3, replace = false)
             z = a + w * (b - c)
```

```
            j = rand(1 : n)
            x′ = [i == j || rand() < p ? z[i] : x[i] for i in 1 : n]
            if f(x′) < f(x)
                x[:] = x′
            end
        end
    end
    return population[argmin(f.(population))]
end
```

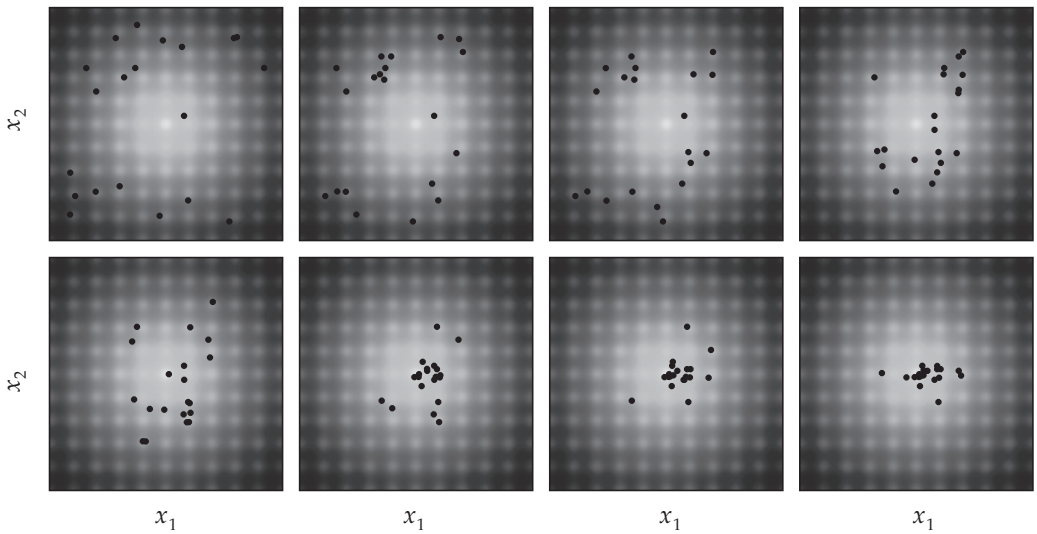


Рис. 9.13. Дифференциальная эволюция с $p = 0{,}5$ и $w = 0{,}2$ применительно к функции Экли, определенной в приложении Б.1

9.4. Оптимизация методом роя частиц

Оптимизация методом роя частиц вводит импульс для ускорения сближения с минимумами [81]. Каждая особь или частица в популяции отслеживает свое текущее положение, скорость и лучшее положение, которое она видела до сих пор (алгоритм 9.11). Импульс позволяет особи накапливать скорость в благоприятном направлении, независимо от локальных возмущений.

Алгоритм 9.11. Каждая частица в оптимизации методом роя частиц имеет положение `x` и скорость `v` в пространстве параметров проекта и отслеживает лучшую расчетную точку, найденную на данный момент, `x_best`

```
mutable struct Particle
    x
```

```
    v
    x_best
end
```

На каждой итерации каждая особь ускоряется в направлении как самой лучшей позиции, которую она видела, так и лучшей позиции, найденной до сих пор любой особью. Ускорение взвешивается случайным слагаемым, причем для каждого ускорения генерируются отдельные случайные числа. Уравнения итерационного приближения:

$$x^{(i)} \leftarrow x^{(i)} + v^{(i)}, \tag{9.4}$$

$$v^{(i)} \leftarrow wv^{(i)} + c_1 r_1 \left(x_{\text{best}}^{(i)} - x^{(i)} \right) + c_2 r_2 \left(x_{\text{best}} - x^{(i)} \right), \tag{9.5}$$

где $\mathbf{x}_{\text{best}}$ — лучшая точка, найденная до сих пор среди всех частиц; w, c_1 и c_2 — параметры; r_1 и r_2 — случайные числа, извлеченные из генеральной совокупности с распределением $U(0,1)$.[4] Реализация представлена в алгоритме 9.12. На рис. 9.14 показано несколько итераций алгоритма.

Алгоритм 9.12. Оптимизация методом роя частиц, которая принимает целевую функцию `f`, список частиц `population`, число итераций `k_max`, инерцию `w` и коэффициенты импульса `c1` и `c2`. Значения по умолчанию — это значения, рекомендованные Эберхартом и Кеннеди

```
function particle_swarm_optimization(f, population, k_max;
    w = 1, c1 = 1, c2 = 1)
    n = length(population[1].x)
    x_best, y_best = copy(population[1].x_best), Inf
    for P in population
        y = f(P.x)
        if y < y_best; x_best[:], y_best = P.x, y; end
    end
    for k in 1 : k_max
        for P in population
            r1, r2 = rand(n), rand(n)
            P.x += P.v
            P.v = w * P.v + c1 * r1. * (P.x_best - P.x) +
                            c2 * r2. * (x_best - P.x)
            y = f(P.x)
```

[4] Распространенная стратегия состоит в том, чтобы инерция w со временем затухала.

```
            if y < y_best; x_best[:], y_best = P.x, y; end
            if y < f(P.x_best); P.x_best[:] = P.x; end
        end
    end
    return population
end
```

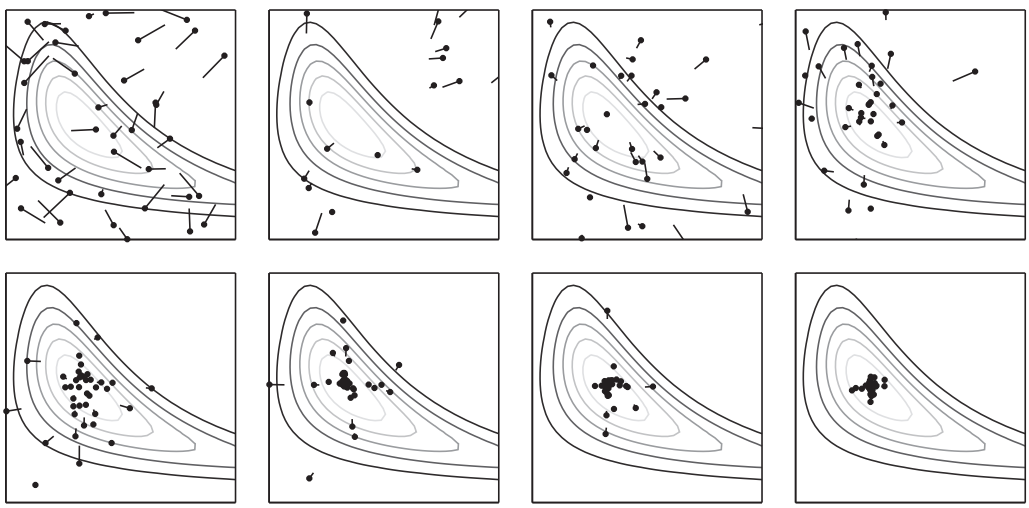


Рис. 9.14. Результаты применения метод роя частиц с $w = 0{,}1$; $c_1 = 0{,}25$ и $c_2 = 2$ к функции Уилера (приложение Б.7)

9.5. Алгоритм светлячка

Идея *алгоритма светлячка* (алгоритм 9.13) навеяна тем, как светлячки вспыхивают, чтобы привлечь партнеров [162].[5] В алгоритме светлячка каждая особь мигает, чтобы привлечь самцов других светлячков, движущихся к более привлекательным светлячкам. Светлячок **a** движется к светлячку **b**, имеющему бо́льшую привлекательность в соответствии с функцией

$$\mathbf{a} \leftarrow \mathbf{a} + \beta I(\|\mathbf{b} - \mathbf{a}\|)(\mathbf{b} - \mathbf{a}) + \alpha\varepsilon, \tag{9.6}$$

где I — интенсивность притяжения, а β — интенсивность источника. В формулу также включен компонент случайного блуждания, где ε — случайная величина, извлеченная из генеральной совокупности с многомерным нормальным распределением, имеющим нулевое математическое ожидание, единичную ковариацию,

[5] Интересно, что светлячки мужского рода вспыхивают, чтобы привлечь представителей противоположного пола, но светлячки женского рода иногда вспыхивают, чтобы привлечь мужчин других видов, которых они затем съедают.

а α — шаговый множитель. Итоговое обновление представляет собой случайное блуждание, ориентированное на более ярких светлячков.[6]

Интенсивность I уменьшается с увеличением расстояния r между двумя светлячками и становится равным единице, когда $r = 0$. Один из подходов состоит в том, чтобы моделировать интенсивность в качестве точечного источника, излучающего в пространство, и в этом случае интенсивность уменьшается в соответствии с законом обратных квадратов

$$I(r) = \frac{1}{r^2}. \tag{9.7}$$

Альтернативно, если источник находится в среде, поглощающей свет, интенсивность будет уменьшаться в соответствии с экспоненциальным затуханием

$$I(r) = e^{-\gamma r}, \tag{9.8}$$

где γ — коэффициент поглощения света.[7]

Как правило, на практике избегают формулы (9.7) из-за сингулярности при $r = 0$. Комбинация закона обратных квадратов и поглощения может быть аппроксимирована с уменьшением яркости по Гауссу:

$$I(r) = e^{-\gamma r^2}. \tag{9.9}$$

Привлекательность светлячка пропорциональна его приспособленности. Притяжение влияет только на то, привлекается ли один светлячок другим, а интенсивность определяет, на какое расстояние переместится менее привлекательный светлячок. На рис. 9.15 показаны несколько итераций алгоритма.

Алгоритм 9.13. Алгоритм светлячка, который принимает целевую функцию **f**, популяцию **flies**, состоящую из расчетных точек, число итераций **k_max**, интенсивности источника **β**, шагового множителя случайного блуждания **α** и функции интенсивности **I**. Возвращается наилучшая расчетная точка

```
using Distributions
function firefly(f, population, k_max;
    β = 1, α = 0.1, brightness = r -> exp(-r ^ 2))

    m = length(population[1])
    N = MvNormal(Matrix(1.0I, m, m))
    for k in 1 : k_max
        for a in population, b in population
```

[6] Янг (Yang) рекомендует $\beta = 1$ и $\alpha \in [0, 1]$. Если $\beta = 0$, поведение является случайным блужданием.

[7] Расстояние между светлячками перестает иметь значение, когда γ приближается к нулю.

```
            if f(b) < f(a)
                r = norm(b - a)
                a[:] += β * brightness(r) * (b - a) + α * rand(N)
            end
        end
    end
    return population[indmin([f(x) for x in population])]
end
```

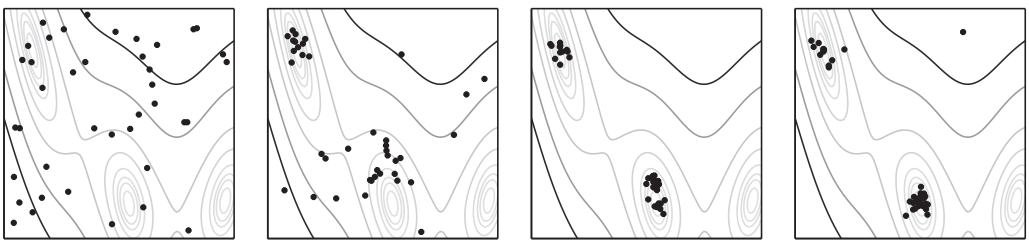


Рис. 9.15. Результаты применения метода светлячка с $\alpha = 0{,}5$, $\beta = 1$ и $\gamma = 0{,}1$ к функции Бранина (приложение Б.3)

9.6. Метод кукушки

Метод кукушки (алгоритм 9.14) — еще один вдохновленный природой алгоритм, названный в честь кукушки, которая паразитирует на чужих гнездах [164]. Кукушки откладывают яйца в гнезда других птиц, часто птиц других видов. Когда это происходит, птица-хозяин может обнаружить чужое яйцо, а затем уничтожить его или создать новое гнездо в другом месте. Однако существует также вероятность того, что яйцо будет принято и высижено птицей-хозяином.[8]

В методе кукушки каждое гнездо представляет собой расчетную точку. Новые расчетные точки могут быть получены с использованием *полетов Леви*, которые являются случайными блужданиями с длинами шагов из распределения с тяжелыми хвостами. Новая расчетная точка может заменить гнездо, если она имеет лучшее целевое значение функции, аналогично тому, как кукушка подменяет яйца птиц других видов.

Основные правила таковы.

1. Кукушка откладывает яйцо в случайно выбранном гнезде.
2. Лучшие гнезда с лучшими яйцами доживут до следующего поколения.

[8] Интересно, что инстинкт недавно вылупившихся кукушек заставляет их выталкивать из гнезда другие яйца или детенышей, принадлежащих птице-хозяину.

3. Яйца кукушки могут быть обнаружены птицей-хозяином, и в этом случае яйца уничтожаются.

Для того чтобы установить новые местоположения гнезда, в методе кукушки используются случайные полеты. Эти полеты начинаются с существующего гнезда, а затем случайным образом перемещаются в новое место. Хотя может возникнуть соблазн использовать равномерное или нормальное распределение для перемещения, это ограничило бы поиск относительно узкой областью. Вместо этого метод кукушки использует распределение Коши, которое имеет более тяжелый хвост. Кроме того, было показано, что распределение Коши более точно описывает движения других животных в дикой природе.[9] На рис. 9.16 показано несколько итераций метода кукушки.

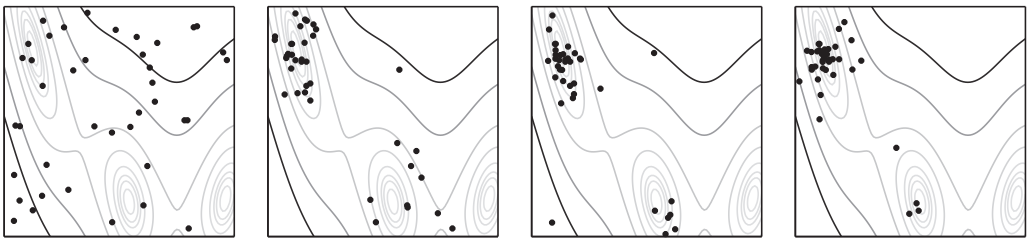


Рис. 9.16. Результаты применения метода кукушки к функции Бранина (приложение Б.3)

К другим вдохновленным природой алгоритмам относятся метод искусственной пчелиной колонии, алгоритм серого волка, алгоритм летучих мышей, метод роя светлячков, метод интеллектуальных капель воды и метод поиска гармонии [143]. Впрочем, некоторые ученые критикуют распространение методов, которые проводят аналогии с природой без ее углубленного изучения и понимания.[10]

Алгоритм 9.14. Метод кукушки, принимающий целевую функцию `f`, начальный набор гнезд `population`, число итераций `k_max`, процентное значение отвергающих гнезд `p_a` и распределение точек конца полета `C`. Распределение полетов обычно является центрированным распределением Коши

```
using Distributions
mutable struct Nest
    x  # позиция
    y  # значение, f(x)
end
```

[9] Определенная разновидность плодовых мух обследуют окрестности, выполняя повороты на 90^0, напоминающие распределение Коши. Оказывается, некоторые племена охотников-собирателей используют аналогичные шаблоны [164].

[10] Эта точка зрения выражена К. Соренсеном [147].

```
function cuckoo_search(f, population, k_max; p_a = 0.1, C = Cauchy(0, 1))
   m, n = length(population), length(population[1].x)
   a = round(Int, m * p_a)
   for k in 1 : k_max
       i, j = rand(1 : m), rand(1 : m)
       x = population[j].x + [rand(C) for k in 1 : n]
       y = f(x)
       if y < population[i].y
          population[i].x[:] = x
          population[i].y = y
       end

       p = sortperm(population, by = nest -> nest.y, rev = true)
       for i in 1 : a
           j = rand(1 : m - a) + a
               population[p[i]] = Nest(population[p[j]].x +
                                       [rand(C) for k in 1 : n],
                                       f(population[p[i]].x)
                                       )
       end
   end
   return population
end
```

9.7. Гибридные методы

Многие популяционные методы хорошо зарекомендовали себя в глобальном поиске, успешно избегая локальных минимумов и находя лучшие области пространства проектных параметров. К сожалению, эти методы не так эффективны при локальном поиске по сравнению с методами спуска. Для повышения эффективности популяционных методов при локальном поиске до уровня методов спуска было разработано несколько *гибридных методов*.[11] Существует два основных подхода к объединению методов популяций с методами локального поиска [90].

- В рамках *ламарковского подхода* популяционный метод дополняется методом локального поиска, который локально улучшает каждую особь. Исход-

[11] В литературе такие методы также называют *меметическими алгоритмами* или *генетическими алгоритмами локального поиска*.

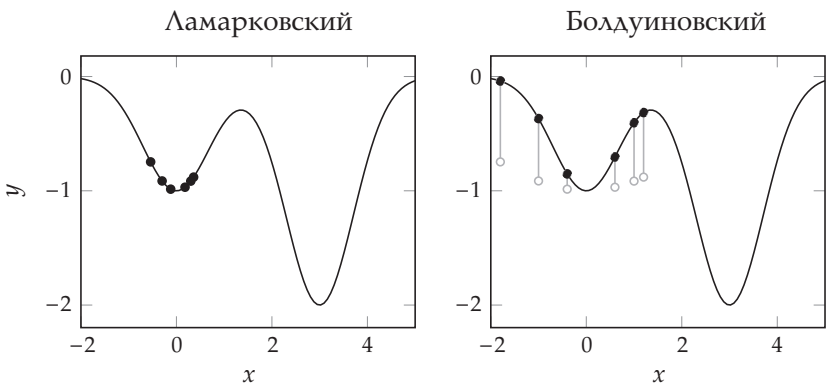
ная особь и ее значение целевой функции заменяются оптимизированным аналогом особи и ее значением целевой функции.

- В рамках *болдуиновского подхода* к каждой особи применяется один и тот же метод локального поиска, но результаты используются только для обновления значения воспринимаемой целевой функции особи. Отдельные особи не заменяются, а просто связываются с оптимизированными значениями целевой функции, которые не совпадают с их фактическим значением целевой функции. Болдуиновский подход может помочь предотвратить преждевременную сходимость.

Разница между этими подходами иллюстрируется в примере 9.2.

Пример 9.2. Сравнение гибридных методов Ламарка и Болдуина

Рассмотрим оптимизацию функции $f(x) = -e^{-x^2} - 2e^{-(x-3)^2}$ с помощью популяции особей, инициализированных в окрестности точки $x = 0$.



Обновление локального поиска по Ламарку, применяемое к этой популяции, будет перемещать особей к локальному минимуму, уменьшая вероятность того, что будущие особи сбегут и найдут глобальный оптимум в окрестности точки $x = 3$. В рамках болдуиновского подхода будет вычислено то же самое обновление, но исходные точки останутся неизменными. На этапе выбора каждая точка оценивается в соответствии с ее значением, полученным в результате локального поиска.

9.8. Резюме

- Популяционные методы используют совокупность особей в области проектирования, чтобы найти путь к оптимуму.
- Генетические алгоритмы используют отбор, кроссовер и мутации для получения лучших последующих поколений.
- Дифференциальная эволюция, оптимизация методом роя частиц, алгоритм светлячков и метод кукушки включают в себя правила и механизмы для привлечения расчетных точек к лучшим особям в популяции при сохранении подходящего пространства состояний.
- Для ускорения сходимости популяционные методы могут быть расширены с помощью методов локального поиска.

9.9. Упражнения

Упражнение 9.1. Какова цель операции отбора в генетических алгоритмах?

Упражнение 9.2. Почему мутация играет фундаментальную роль в генетических алгоритмах? Как выбрать частоту мутаций, если предположить, что существует лучшее оптимальное решение?

Упражнение 9.3. Как можно изменить параметры метода роя частиц, если он быстро сходится к неглобальному минимуму?

Глава 10

Ограничения

Предыдущие главы были сосредоточены на несложных задачах, где область изменения каждого проектного параметра — это все пространство действительных чисел. Многие задачи имеют ограничения, т.е. расчетные точки должны удовлетворять определенным условиям. В этой главе представлены различные подходы для преобразования задач с ограничениями в задачи без ограничений, что позволяет использовать алгоритмы оптимизации, которые мы уже обсуждали. Рассматриваются также аналитические методы, в том числе понятия двойственности и необходимые условия оптимальности при условной оптимизации.

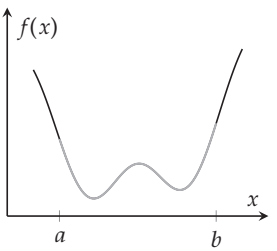
10.1. Условная оптимизация

Напомним основную формулировку задачи оптимизации (1.1):

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) \\ &\text{при условии, что } \mathbf{x} \in \mathcal{X}. \end{aligned} \tag{10.1}$$

В задачах без ограничений допустимым множеством $\mathcal{X}$ является все пространство $\mathbb{R}^n$. В задачах с ограничениями допустимое множество является подмножеством этого пространства.

Некоторые ограничения — это просто верхние или нижние границы проектных параметров, как мы видели в методе интервалов, в котором переменная x должна находиться между числами a и b. Ограничение интервала $x \in [a, b]$ можно заменить двумя ограничениями в виде неравенств: $a \leq x$ и $x \leq b$, как показано на рис. 10.1. В многомерных задачах заключение в интервал входных переменных означает, что они должны лежать в гиперпрямоугольнике, как показано на рис. 10.2.



$$\begin{aligned} &\min_{x} && f(x) \\ &\text{при условии} && x \in [a, b] \end{aligned}$$

Рис. 10.1. Простая задача оптимизации, ограниченная верхними и нижними границами

При формулировании реальных задач ограничения возникают естественным образом. Менеджер хедж-фонда не может продать больше акций, чем у него есть, у самолета не может быть крыльев нулевой толщины, а количество часов, которые можно затратить в неделю на выполнение домашней работы, не может превышать 168. Мы включаем ограничения в такие задачи, чтобы алгоритм оптимизации не предлагал невозможное решение.

Применение ограничений к задаче может повлиять на решение, но это не всегда обязательно, как показано на рис. 10.3.

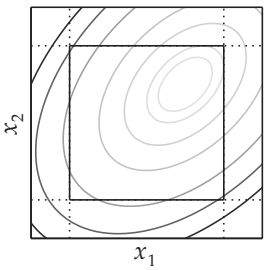


Рис. 10.2. Ограничения метода интервалов вынуждают решение находиться в гиперпрямоугольнике

Рис. 10.3. Ограничения могут изменить решение задачи, но не обязательно

10.2. Виды ограничений

Ограничения обычно не задаются через известное допустимое множество. Вместо этого допустимое множество обычно формируется из ограничений двух видов.[1]

[1] Мы приводим ограничение g в виде неравенства “меньше или равно”. Ограничение вида “больше или равно” можно преобразовать в ограничения вида “меньше или равно”, просто умножив его на –1.

1. Равенства, $h(\mathbf{x}) = 0$
2. Неравенства, $g(\mathbf{x}) \leq 0$

С помощью таких ограничений можно сформулировать любую задачу оптимизации:

$$\min_{\mathbf{x}} f(\mathbf{x})$$
$$\text{при условии, что } h_i(\mathbf{x}) = 0 \text{ для всех } i \text{ в } \{1, ..., l\}, \quad (10.2)$$
$$g_j(\mathbf{x}) \leq 0 \text{ для всех } j \text{ в } \{1, ..., m\}.$$

Конечно, ограничения можно сформулировать с использованием допустимого множества $\mathcal{X}$:

$$h(\mathbf{x}) = (x \notin \mathcal{X}), \quad (10.3)$$

где логические выражения могут принимать значения 0 или 1.

Равенства и неравенства $(h(\mathbf{x}) = 0, g(\mathbf{x}) \leq 0)$ часто используются, чтобы установить ограничения, а не принадлежность множеству $(\mathbf{x} \in \mathcal{X})$, потому что эти функции могут предоставить информацию о том, насколько далека данная точка от допустимого множества. Эта информация помогает разрабатывать методы, приближающие расчетную точку к допустимому множеству.

Ограничения в виде равенства иногда разделяются на два вида:

$$h(\mathbf{x}) = 0 \Leftrightarrow \begin{cases} h(\mathbf{x}) \leq 0, \\ h(\mathbf{x}) \geq 0. \end{cases} \quad (10.4)$$

Однако иногда ограничения в виде равенств желательно обрабатывать отдельно, как будет показано ниже в этой главе.

10.3. Преобразования для снятия ограничений

В некоторых случаях задачу удается изменить так, чтобы снять с нее ограничения. Например, ограничения снизу и сверху $a \leq x \leq b$ можно удалить, применив преобразование (рис. 10.4):

$$x = t_{a,b}(\hat{x}) = \frac{b+a}{2} + \frac{b-a}{2}\left(\frac{2\hat{x}}{1+\hat{x}^2}\right). \quad (10.5)$$

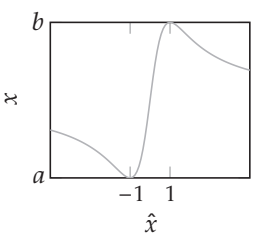


Рис. 10.4. Это преобразование гарантирует, что переменная x лежит между a и b

Этот процесс демонстрирует пример 10.1.

Пример 10.1. Снятие ограничений путем преобразования входных переменных

Рассмотрим задачу условной оптимизации

$$\min_x x \sin x$$

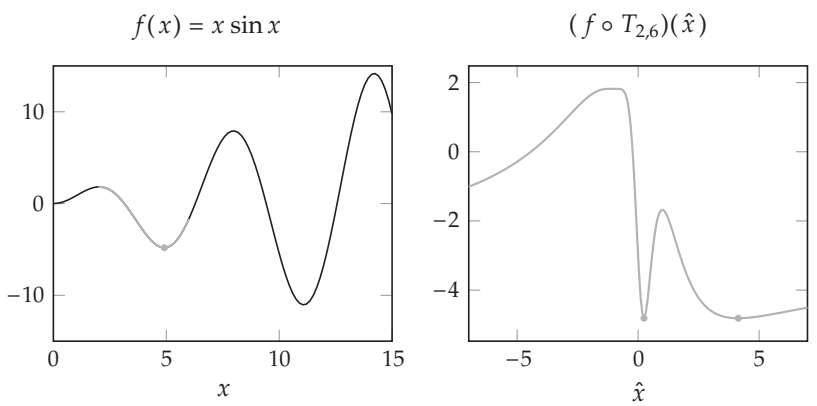
при условии, что $2 \le x \le 6$.

Преобразуем эту задачу, чтобы снять ограничения:

$$\min_x t_{2,6}(\hat{x}) \sin t_{2,6}(\hat{x}),$$

$$\min_x \left(4 + 2\left(\frac{2\hat{x}}{1+\hat{x}^2}\right)\right) \sin\left(4 + 2\left(\frac{2\hat{x}}{1+\hat{x}^2}\right)\right).$$

Для решения задачи без ограничений можно применить любой метод оптимизации. При этом мы находим два минимума: $\hat{x} \approx 0{,}242$ и $\hat{x} \approx 4{,}139$, в обоих значение функции приблизительно равно –4,814.



Решение исходной задачи получено путем перехода к переменной $\hat{x}$ через преобразование. Оба значения $\hat{x}$ приводят к значению $x = t_{2,6}(\hat{x}) \approx 4{,}914$.

Некоторые ограничения в виде равенства могут быть использованы для нахождения x_n по заданным x_1, ..., x_{n-1}. Иначе говоря, зная первые n – 1 компонентов x, мы можем использовать ограничения для вычисления x_n. В таких случаях задача оптимизации может быть переформулирована относительно x_1, ..., x_n путем удаления ограничения и исключения одной из переменных. Этот процесс демонстрирует пример 10.2.

Пример 10.2. Использование ограничений для исключения переменных

Рассмотрим ограничение

$$h(\mathbf{x}) = x_1^2 + x_2^2 + ... + x_n^2 - 1 = 0.$$

Выразим переменную x_n через первые $n - 1$ переменных:

$$x_n = \pm\sqrt{1 - x_1^2 - x_2^2 - ... - x_{n-1}^2}.$$

В таком случае задача

$$\min_{\mathbf{x}} f(\mathbf{x})$$

при условии, что $h(\mathbf{x}) = 0$

преобразуется в задачу

$$\min_{x_1, ..., x_{n-1}} f\left(\left[x_1, ..., x_{n-1}, \pm\sqrt{1 - x_1^2 - x_2^2 - ... - x_{n-1}^2}\right]\right).$$

10.4. Множители Лагранжа

Метод множителей Лагранжа используется для оптимизации функции с учетом ограничений в виде равенства. Рассмотрим задачу оптимизации с одним ограничением в виде равенства:

$$\min_{\mathbf{x}} f(\mathbf{x})$$
$$\text{при условии, что } h(\mathbf{x}) = 0, \tag{10.6}$$

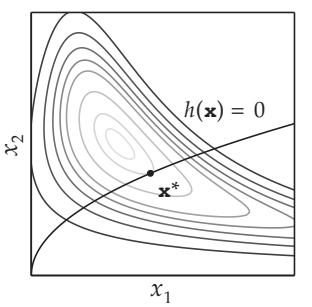
где функции f и h имеют непрерывные частные производные. Такая задача обсуждается в примере 10.3.

Пример 10.3. Иллюстрация метода множителей Лагранжа

Рассмотрим задачу минимизации:

$$\min_{\mathbf{x}} -\exp\left(-\left(x_1 x_2 - \frac{3}{2}\right)^2 - \left(x_2 - \frac{3}{2}\right)^2\right)$$

при условии, что $x_1 - x_2^2 = 0$.



Подставим ограничение $x_1 = x_2^2$ в целевую функцию для получения неограниченной целевой функции:

$$f_{\text{unc}} = -\exp\left(-\left(x_2^3 - \frac{3}{2}\right)^2 - \left(x_2 - \frac{3}{2}\right)^2\right),$$

чья производная равна

$$\frac{\partial f_{\text{unc}}}{\partial x_2} = 6\exp\left(-\left(x_2^3 - \frac{3}{2}\right)^2 - \left(x_2 - \frac{3}{2}\right)^2\right)\left(x_2^5 - \frac{3}{2}x_2^2 + \frac{1}{3}x_2 - \frac{1}{2}\right).$$

Приравнивая производную к нулю и решая это уравнение относительно x_2, получим $x_2 \approx 1{,}165$. Таким образом, решением исходной задачи оптимизации является точка $\mathbf{x}^* = [1{,}358;\ 1{,}165]$. Оптимум лежит там, где линия уровня функции f касается линии уровня функции h.

Если точка $\mathbf{x}^*$ оптимизирует f вдоль h, то ее производная в точке $\mathbf{x}^*$ по направлению вдоль h должна быть равной нулю. Иначе говоря, небольшие сдвиги $\mathbf{x}^*$ вдоль h не могут привести к улучшению.

Линии уровня функции f — это линии постоянных значений функции f. Таким образом, если линия уровня функции f касается линии уровня h, то производная по направлениям h в этой точке вдоль направления контура $h(\mathbf{x}) = 0$ должна быть равна нулю.

Метод множителей Лагранжа используется для вычисления точки, в которой изолиния функции f касается линии уровня $h(\mathbf{x}) = 0$. Поскольку градиент функции в точке перпендикулярен касательной к линии уровня функции, проходящему через эту точку, то градиент функции h перпендикулярен касательной к линии уровня $h(\mathbf{x}) = 0$. Следовательно, нам нужно найти точку, в которой градиент функции f и градиент функции h параллельны.

Мы ищем лучшую точку $\mathbf{x}$, удовлетворяющую ограничению

$$h(\mathbf{x}) = 0. \tag{10.7}$$

Градиенты функций при этом должны быть параллельны:

$$\nabla f(\mathbf{x}) = \lambda \nabla h(\mathbf{x}) \tag{10.8}$$

при некотором *множителе Лагранжа* λ. Нам нужен скаляр, потому что величины градиентов не могут быть одинаковыми.[2]

Мы можем записать *лагранжиан*, который является функцией переменных и множителя

---

[2] Если ∇f равен нулю, множитель Лагранжа λ равен нулю, независимо от ∇h.

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) - \lambda h(\mathbf{x}). \tag{10.9}$$

Решение уравнения $\nabla\mathcal{L}(\mathbf{x}, \lambda) = 0$ является решением уравнения (10.7) и (10.8). В частности, уравнение $\nabla_{\mathbf{x}}\mathcal{L} = 0$ эквивалентно условию $\nabla f = \lambda\nabla h$, а уравнение $\nabla_{\lambda}\mathcal{L} = 0$ эквивалентно условию $h(\mathbf{x}) = 0$. Любое решение считается *критической точкой*. Критическими точками могут быть локальные минимумы, глобальные минимумы или седловые точки.[3] Этот подход демонстрирует пример 10.4.

Пример 10.4. Использование метода множителей Лагранжа для решения примера 10.3

Мы можем использовать метод множителей Лагранжа для решения задачи из примера 10.3. Сформируем лагранжиан

$$\mathcal{L}(x_1, x_2, \lambda) = -\exp\left(-\left(x_1x_2 - \frac{3}{2}\right)^2 - \left(x_2 - \frac{3}{2}\right)^2\right) - \lambda\left(x_1 - x_2^2\right),$$

и вычислим градиент

$$\frac{\partial\mathcal{L}}{\partial x_1} = 2x_2 f(\mathbf{x})\left(\frac{3}{2} - x_1x_2\right) - \lambda,$$

$$\frac{\partial\mathcal{L}}{\partial x_2} = 2\lambda x_2 + f(\mathbf{x})\left(-2x_1\left(x_1x_2 - \frac{3}{2}\right) - 2\left(x_2 - \frac{3}{2}\right)\right),$$

$$\frac{\partial\mathcal{L}}{\partial\lambda} = x_2^2 - x_1.$$

Приравнивая эти производные к нулю и решая систему уравнений, получаем $x_1 \approx 1{,}385$; $x_2 \approx 1{,}165$ и $\lambda \approx 0{,}170$.

Метод множителей Лагранжа можно распространить на несколько ограничений в виде равенства. Рассмотрим задачу с двумя ограничениями в виде равенства:

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) \\ &\text{при условии, что } h_1(\mathbf{x}) = 0, \\ &\qquad\qquad\qquad\quad h_2(\mathbf{x}) = 0. \end{aligned} \tag{10.10}$$

Мы можем свести эти ограничения в одно. Новое ограничение удовлетворяется точно такими же точками, как и раньше, поэтому решение остается неизменным.

[3] Метод множителей Лагранжа дает необходимое условие первого порядка для проверки на оптимальность. Мы расширим этот метод, чтобы включить в него неравенства.

$$\min_{\mathbf{x}} f(\mathbf{x}) \quad \text{при условии, что } h_{\text{comb}}(\mathbf{x}) = h_1(\mathbf{x})^2 + h_2(\mathbf{x})^2 = 0. \tag{10.11}$$

Теперь мы можем применить метод множителей Лагранжа, как и раньше. В частности, мы вычисляем условие градиента

$$\nabla f - \lambda \nabla h_{\text{comb}} = \mathbf{0}, \tag{10.12}$$

$$\nabla f - 2\lambda(h_1 \nabla h_1 + h_2 \nabla h_2) = \mathbf{0}. \tag{10.13}$$

Наш выбор функции h_{comb} был несколько произвольным. Мы могли бы использовать другую функцию

$$h_{\text{comb}}(\mathbf{x}) = h_1(\mathbf{x})^2 + c\,h_2(\mathbf{x})^2 \tag{10.14}$$

для некоторой константы $c > 0$.

С этой более общей формулировкой мы получаем

$$\mathbf{0} = \nabla f - \lambda \nabla h_{\text{comb}} = \tag{10.15}$$

$$= \nabla f - 2\lambda h_1 \nabla h_1 + 2c\lambda h_2 \nabla h_2 = \tag{10.16}$$

$$= \nabla f - \lambda_1 \nabla h_1 + \lambda_2 \nabla h_2. \tag{10.17}$$

Таким образом, мы можем определить лагранжиан с l множителями Лагранжа для задач с l ограничениями в виде равенства

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \sum_{i=1}^{l} \lambda_i h_i(\mathbf{x}) = f(\mathbf{x}) - \boldsymbol{\lambda}^{\mathrm{T}} \mathbf{h}(\mathbf{x}). \tag{10.18}$$

10.5. Ограничения в виде неравенства

Рассмотрим задачу с одним ограничением в виде неравенства:

$$\min_{\mathbf{x}} f(\mathbf{x}) \quad \text{при условии, что } g(\mathbf{x}) \leq 0. \tag{10.19}$$

Мы уже видели, что если решение лежит на границе допустимой области, то условие Лагранжа выполнено

$$\nabla f + \mu \nabla g = 0 \tag{10.20}$$

для некоторой постоянной μ. Когда это происходит, ограничение считается *активным*, и градиент целевой функции ограничивается точно так же, как это было с ограничениями в виде равенства. Пример показан на рис. 10.5.

Если решение задачи не лежит на границе допустимой области, то ограничение считается *неактивным*. Решения f просто лежат там, где градиент f равен нулю, как при неограниченной оптимизации. В этом случае уравнение (10.20)

будет выполнено, если установить μ равным нулю. Этот пример показан на рис 10.6.

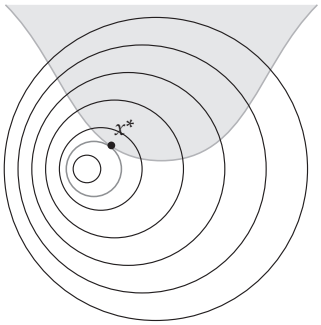


Рис. 10.5. Активное ограничение в виде неравенства. Соответствующая линия уровня показана красным цветом

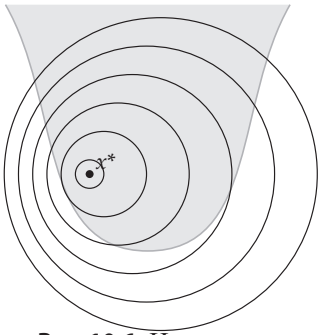


Рис. 10.6. Неактивное ограничение в виде неравенства

Мы могли бы оптимизировать задачу с ограничением в виде неравенства, введя бесконечный штраф за недопустимые точки:

$$f_{\infty-\text{step}}(\mathbf{x}) = \begin{cases} f(\mathbf{x}), \text{если } g(\mathbf{x}) \leq 0, \\ \infty \text{ в противном случае} \end{cases} = \tag{10.21}$$

$$= f(\mathbf{x}) + \infty \cdot (g(\mathbf{x}) > 0). \tag{10.22}$$

К сожалению, функцию $f_{\infty\text{-step}}$ неудобно оптимизировать.[4] Она является разрывной и недифференцируемой. Процедуры поиска не получают никакой информации о направлении, чтобы ориентироваться на допустимое множество.

Вместо этого мы можем использовать линейный штраф $\mu g(\mathbf{x})$, который формирует нижнюю границу $\infty \cdot (g(\mathbf{x}) > 0)$ и штрафует цель за отклонение от допустимого множества до тех пор, пока $\mu > 0$. Этот линейный штраф продемонстрирован на рис. 10.7.

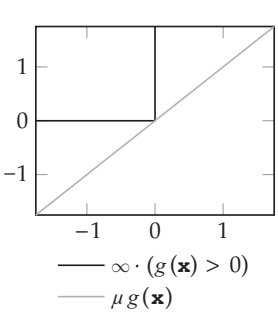


Рис. 10.7. Линейная функция $\mu g(\mathbf{x})$ является нижней границей штрафа за бесконечный шаг до тех пор, пока $\mu > 0$

[4] Такая постановка задачи может быть оптимизирована с использованием прямых методов, таких как адаптивный прямой поиск по сеткам (раздел 8.2).

Мы можем использовать этот линейный штраф для построения лагранжиана для ограничения в виде неравенства:

$$\mathcal{L}(\mathbf{x}, \mu) = f(\mathbf{x}) + \mu g(\mathbf{x}). \tag{10.23}$$

Мы можем восстановить функцию $f_{\infty\text{-step}}$, максимизируя лагранжиан по μ:

$$f_{\infty-\text{step}} = \max_{\mu \geq 0} \mathcal{L}(\mathbf{x}, \mu). \tag{10.24}$$

Для любого невозможного решения $\mathbf{x}$ мы получаем бесконечность, а для любого возможного решения $\mathbf{x}$ — $f(\mathbf{x})$.

Таким образом, новая задача оптимизации имеет вид

$$\min_{\mathbf{x}} \max_{\mu \geq 0} \mathcal{L}(\mathbf{x}, \mu). \tag{10.25}$$

Эта формулировка называется *прямой задачей*.

Оптимизация прямой задачи требует нахождения критических точек $\mathbf{x}^*$, таких что:

1. $g(\mathbf{x}^*) \leq 0$

Точка является допустимой.

2. $\mu \geq 0$

Штраф должен указывать правильное направление. Это требование иногда называют *двойственной допустимостью* (dual feasibility).

3. $\mu g(\mathbf{x}^*) = 0$

Допустимая точка на границе будет удовлетворять условию $g(\mathbf{x}) = 0$, тогда как для допустимой точки, удовлетворяющей условию $g(\mathbf{x}) < 0$, при восстановлении $f(\mathbf{x}^*)$ из лагранжиана выполняется условие $\mu = 0$.

4. $\nabla f(\mathbf{x}^*) + \mu \nabla g(\mathbf{x}^*) = 0$

Если ограничение активно, мы требуем, чтобы линии уровня функций f и g касались друг друга, что эквивалентно параллельности их градиентов. Если ограничение не активно, оптимальная точка будет удовлетворять условиям $\nabla f(\mathbf{x}^*) = 0$ и $\mu = 0$.

Эти четыре требования можно обобщить для задач оптимизации с любым количеством ограничений в виде равенств и неравенств:[5]

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) \\ &\text{при условии, что } g(\mathbf{x}) \leq 0, \\ &\qquad\qquad\qquad\quad h(\mathbf{x}) = 0, \end{aligned} \tag{10.26}$$

[5] Если $\mathbf{u}$ и $\mathbf{v}$ — векторы одинаковой длины, то мы говорим $\mathbf{u} \leq \mathbf{v}$, если $u_i \leq v_i$ для всех i. Аналогично для векторов определяются неравенства $\geq$, $<$ и $>$.

где каждый компонент вектора $\mathbf{g}$ является ограничением в виде неравенства, а каждый компонент вектора $\mathbf{h}$ является ограничением в виде равенства. Следующие четыре условия называются условиями KKT (Каруша–Куна–Таккера).[6]

1. **Допустимость:** все ограничения выполнены.

$$\mathbf{g}(\mathbf{x}^*) \leq 0, \tag{10.27}$$

$$\mathbf{h}(\mathbf{x}^*) = 0. \tag{10.28}$$

2. **Двойственная допустимость:** Штраф указывает направление к допустимому множеству.

$$\boldsymbol{\mu} \geq 0 \tag{10.29}$$

3. **Дополняющая нежесткость:** множители Лагранжа вводят нежесткость: либо μ_i равны нулю, либо $g_i(\mathbf{x}^*)$ равны нулю.[7]

$$\boldsymbol{\mu} \odot \mathbf{g} = 0 \tag{10.30}$$

4. **Стационарность:** линия уровня целевой функции является касательной ко всем активным ограничениям.

$$\nabla f\left(\mathbf{x}^*\right) + \sum_i \mu_i \nabla g_i\left(\mathbf{x}^*\right) + \sum_j \lambda_j \nabla h_j\left(\mathbf{x}^*\right) = \mathbf{0}. \tag{10.31}$$

Эти четыре условия являются необходимыми условиями оптимальности первого порядка и, таким образом, являются условиями FONC для задач с гладкими ограничениями. Как и в случае с условиями FONC для безусловной оптимизации, особое внимание следует уделить тому, чтобы идентифицированные критические точки были настоящими локальными минимумами.

10.6. Двойственность

При выводе условий FONC для оптимизации с ограничениями мы также находим более общий вид лагранжиана. Этот *обобщенный лагранжиан* имеет вид[8]

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_i \mu_i g_i(\mathbf{x}) + \sum_j \lambda_j h_j(\mathbf{x}). \tag{10.32}$$

---

[6] Эти условия были названы в честь Гарольда В. Куна и Альберта В. Таккера, которые опубликовали их в 1951 г. Позже было обнаружено, что Уильям Каруш изучал эти условия в неопубликованной магистерской диссертации в 1939 г. История вопроса изложена в [85].

[7] Операция $a \odot b$ означает поэлементное произведение векторов $\mathbf{a}$ и $\mathbf{b}$.

[8] Так как знак λ может быть любым, мы можем поменять на противоположный знак в ограничениях в виде равенств, полученных с помощью метода множителей Лагранжа.

Прямой задачей оптимизации является исходная задача оптимизации, сформулированная с использованием обобщенного лагранжиана

$$\min_{\mathbf{x}} \max_{\boldsymbol{\mu}\geq 0,\boldsymbol{\lambda}} \mathcal{L}(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}). \tag{10.33}$$

Прямая задача эквивалентна исходной и ее так же трудно оптимизировать.

Двойственная форма задачи оптимизации меняет порядок минимизации и максимизации в уравнении (10.33):

$$\max_{\boldsymbol{\mu}\geq 0,\boldsymbol{\lambda}} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}). \tag{10.34}$$

Неравенство max-min утверждает, что для любой функции $f(\mathbf{a}, \mathbf{b})$:

$$\max_{\mathbf{a}} \min_{\mathbf{b}} f(\mathbf{a}, \mathbf{b}) \leq \min_{\mathbf{b}} \max_{\mathbf{a}} f(\mathbf{a}, \mathbf{b}). \tag{10.35}$$

Таким образом, решение двойственной задачи является нижней границей решения прямой задачи. Иначе говоря, $d^* \leq p^*$, где d^* — это *двойственное решение*, а p^*— это *прямое решение*.

Для удобства обозначений внутренняя максимизация в двойственной задаче часто сворачивается в *двойственную функцию*:

$$\mathcal{D}(\boldsymbol{\mu} \geq 0, \boldsymbol{\lambda}) = \min_{\mathbf{x}} \mathcal{L}(x, \boldsymbol{\mu}, \boldsymbol{\lambda}). \tag{10.36}$$

Двойственная функция является вогнутой [110]. Градиентный подъем по вогнутой функции всегда сходится к глобальному максимуму. Оптимизировать двойственную задачу легко, если легко минимизировать лагранжиан по $\mathbf{x}$.

Мы знаем, что $\max_{\boldsymbol{\mu}\geq 0,\boldsymbol{\lambda}} \mathcal{D}(\boldsymbol{\mu}, \boldsymbol{\lambda}) \leq p^*$. Отсюда следует, что двойственная функция всегда является нижней границей решения прямой задачи (см. пример 10.5). Для любого $\boldsymbol{\mu} \geq 0$ и любого $\boldsymbol{\lambda}$ выполняется неравенство

$$\mathcal{D}(\boldsymbol{\mu} \geq 0, \boldsymbol{\lambda}) \leq p^*. \tag{10.37}$$

Разница $p^* - d^*$ между двойственными и прямыми решениями называется *разрывом двойственности*. В некоторых случаях двойственная задача гарантированно будет иметь то же решение, что и прямая задача, — разрыв двойственности в таком случае равен нулю [22]. В таких случаях двойственность обеспечивает альтернативный подход к оптимизации нашей задачи. Этот подход продемонстрирован в примере 10.6.

Пример 10.5. Двойственная функция является нижней границей решения прямой задачи

Рассмотрим задачу оптимизации:

$$\min_{x} \sin x$$

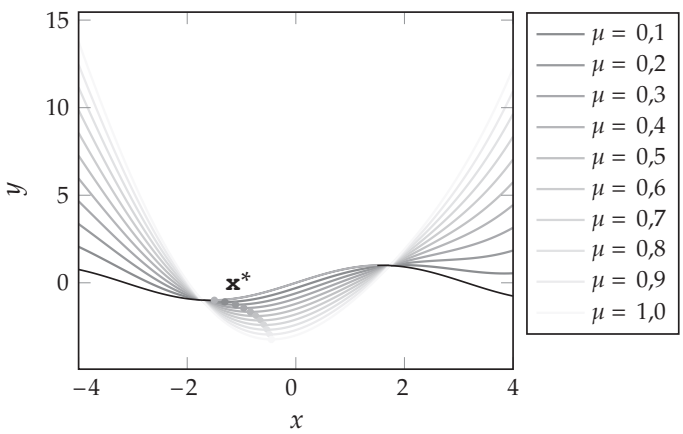
при условии, что $x^2 \leq 3$.

Обобщенный лагранжиан имеет вид $\mathcal{L}(x, \mu) = \sin x + \mu(x^2 - 3)$, что приводит нас к прямой задаче

$$\min_x \max_{\mu \geq 0} \sin x + \mu\left(x^2 - 3\right)$$

и двойственной задаче

$$\max_{\mu \geq 0} \min_x \sin x + \mu\left(x^2 - 3\right).$$



Целевая функция изображена черным цветом, а допустимая область — синим. Минимум достигается в точке $x^* = -\sqrt{3}$ с $p^* \approx -0{,}997$. Фиолетовые линии — это лагранжиан $\mathcal{L}(x, \mu)$ для $\mu = 0{,}1; 0{,}2; \ldots; 1$, каждый из которых имеет минимум меньше, чем p^*.

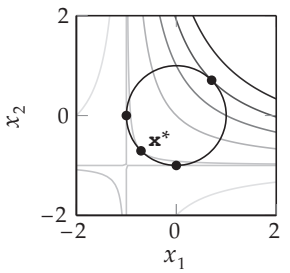
Пример 10.6. Пример применения лагранжевой двойственности к задаче с ограничением в виде равенства. На верхнем рисунке показан контур целевой функции и ограничение с четырьмя критическими точками, отмеченными маркерами. На нижнем рисунке показана двойственная функция

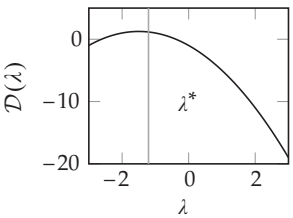
Рассмотрим задачу

$$\min_{\mathbf{x}} x_1 + x_2 + x_1 x_2$$

при условии, что $x_1^2 + x_2^2 = 1$.

Лагранжиан имеет вид $\mathcal{L}\left(x_1, x_2, \lambda\right) = x_1 + x_2 + x_1 x_2 + \lambda\left(x_1^2 + x_2^2 - 1\right)$.





Применим метод множителей Лагранжа:

$$\frac{\partial \mathcal{L}}{\partial x_1} = 1 + x_2 + 2\lambda x_1 = 0,$$
$$\frac{\partial \mathcal{L}}{\partial x_2} = 1 + x_1 + 2\lambda x_2 = 0,$$
$$\frac{\partial \mathcal{L}}{\partial \lambda} = x_1^2 + x_2^2 - 1 = 0.$$

Решение этой системы приводит к четырем критическим точкам:

x_1	x_2	λ
–1	0	1/2
0	–1	1/2
$\frac{\sqrt{2}+1}{\sqrt{2}+2}$	$\frac{\sqrt{2}+1}{\sqrt{2}+2}$	$\frac{1}{2}\left(-1-\sqrt{2}\right)$
$\frac{\sqrt{2}-1}{\sqrt{2}-2}$	$\frac{\sqrt{2}-1}{\sqrt{2}-2}$	$\frac{1}{2}\left(-1+\sqrt{2}\right)$

Двойственная функция имеет вид

$$\mathcal{D}(\lambda) = \min_{x_1, x_2} x_1 + x_2 + x_1 x_2 + \lambda\left(x_1^2 + x_2^2 - 1\right).$$

Выполним подстановку $x_1 = x_2 = x$, приравняем к нулю производную по x и получим $x = -1 - \lambda$. После подстановки, получаем

$$\mathcal{D}(\lambda) = -1 - 3\lambda - \lambda^2.$$

Решение двойственной задачи $\max\limits_{\lambda} \mathcal{D}(\lambda)$ имеет вид $\lambda = \left(-1-\sqrt{2}\right)/2$.

10.7. Методы штрафных функций

Чтобы преобразовать задачи условной оптимизации в задачу безусловной оптимизации, можно использовать *методы штрафных функций*, добавив штрафные члены к целевой функции, что позволит нам использовать методы, разработанные в предыдущих главах.

Рассмотрим общую задачу оптимизации:

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) \\ &\text{при условии, что} \quad g(\mathbf{x}) \leq 0, \\ &\qquad\qquad\qquad\quad h(\mathbf{x}) = 0, \end{aligned} \tag{10.38}$$

Простой метод штрафных функций подсчитывает количество нарушенных ограничений:

$$p_{\text{count}}(\mathbf{x}) = \sum_i \left(g_i(\mathbf{x}) > 0\right) + \sum_j \left(h_j(\mathbf{x}) \neq 0\right), \tag{10.39}$$

что приводит к задаче безусловной оптимизации, которая штрафует за недопустимость

$$\min_{\mathbf{x}} f(\mathbf{x}) + \rho p_{\text{count}}(\mathbf{x}), \tag{10.40}$$

где $\rho > 0$ регулирует величину штрафа. Пример показан на рис. 10.8.

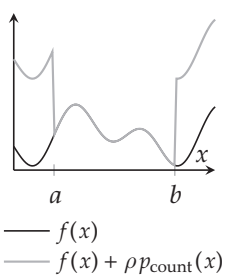


Рис. 10.8. Исходные и штрафные целевые функции для минимизации f с условием $x \in [a, b]$

Выполнение *методов штрафных функций* начинается с точки $\mathbf{x}$ и небольшого значения для ρ. Затем решается задача безусловной оптимизации (10.40). Полученное решение затем используется в качестве отправной точки для другой оптимизации с увеличенным штрафом. Мы продолжаем эту процедуру до тех пор, пока результирующая точка не перестанет быть допустимой или не будет превышено установленное максимальное количество итераций. Реализация описана в алгоритме 10.1.

Алгоритм 10.1. Метод штрафных функций для целевой функции `f`, функции штрафа `ρ`, начальной точки `x`, количества итераций `k_max`, начального штрафа `ρ > 0` и штрафного множителя `γ > 1`. Метод `minimize` должен быть заменен подходящим методом безусловной минимизации

```
function penalty_method(f, p, x, k_max; ρ = 1, γ = 2)
    for k in 1 : k_max
        x = minimize(x -> f(x) + ρ * p(x), x)
        ρ *= γ
        if p(x) == 0
            return x
        end
    end
    return x
end
```

Этот штраф сохранит решение задачи для больших значений ρ, но оно вносит резкий разрыв. Точки, не входящие в допустимое множество, не имеют информации о градиенте, чтобы переместиться ближе к допустимому множеству.

Мы можем использовать *квадратичные штрафы* для получения гладкой целевой функции (рис. 10.9):

$$p_{\text{quadratic}}(\mathbf{x}) = \sum_i \max\left(g_i(\mathbf{x}), 0\right)^2 + \sum_j h_j(\mathbf{x})^2. \tag{10.41}$$

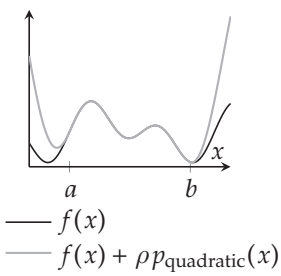


Рис. 10.9. Использование квадратичной штрафной функции для минимизации f с условием $x \in [a, b]$

Квадратичные штрафы вблизи границы ограничения очень малы, и может потребоваться устремить ρ к бесконечности, прежде чем решение перестает нарушать ограничения.

Также можно смешивать функцию подсчета и квадратичного штрафа (рис. 10.10):

$$p_{\text{mixed}}(\mathbf{x}) = \rho_1 p_{\text{count}}(\mathbf{x}) + \rho_2 p_{\text{quadratic}}(\mathbf{x}). \tag{10.42}$$

Такая смесь штрафов проводит четкую границу между допустимой и недопустимой областями, в то же время предоставляя информацию о градиенте.

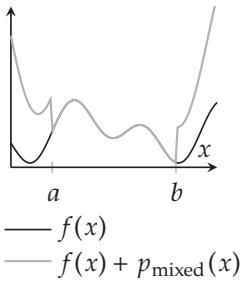


Рис. 10.10. Использование как квадратичной, так и дискретной штрафной функции для минимизации f с условием $x \in [a, b]$

На рис. 10.11 показан прогресс функции штрафа при увлечении ρ. Квадратичные штрафные функции не могут обеспечить допустимость, как показано в примере 10.7.

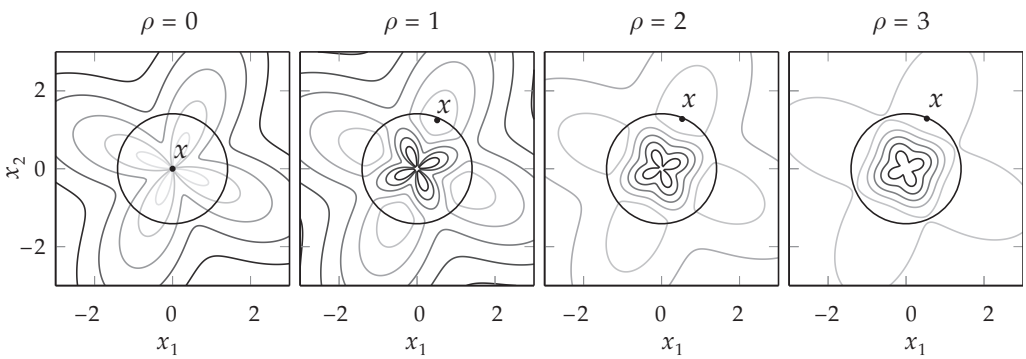


Рис. 10.11. Метод штрафных функций, примененный к функции цветка (см. приложение Б.4), и круговое ограничение $x_1^2 + x_2^2 \geq 2$

Пример 10.7. Пример, показывающий, как квадратичные штрафы не могут обеспечить допустимость

Рассмотрим задачу

$$\min_x x$$

при условии, что $x \geq 5$,

используя квадратичную штрафную функцию.

Целевая функция безусловной оптимизации имеет вид

$$f(x) = x + \rho \max(5 - x, 0)^2.$$

Минимум целевой функции безусловной оптимизации имеет вид

$$x^* = 5 - \frac{1}{2\rho}.$$

В то время как минимум задачи условной оптимизации явно равен $x = 5$, минимум задачи оптимизации со штрафом просто приближается к точке $x = 5$, требуя бесконечного штрафа для достижения допустимого множества.

10.8. Расширенный метод Лагранжа

Расширенный метод Лагранжа[9] является адаптацией метода штрафных функций специально для ограничений в виде равенств. В отличие от метода штрафных функций, где штраф ρ должен иногда стремиться к бесконечности, прежде чем будет найдено допустимое решение, расширенный метод Лагранжа будет работать с меньшими значениями ρ. Он использует как квадратичный, так и линейный штраф для каждого ограничения.

Для задачи оптимизации с ограничениями в виде равенства $\mathbf{h}(\mathbf{x}) = 0$ функция штрафа имеет вид:

$$p_{\text{Lagrange}}(\mathbf{x}) = \frac{1}{2}\rho\sum_i \left(h_i(\mathbf{x})\right)^2 - \sum_j \lambda_j h_j(\mathbf{x}), \tag{10.43}$$

где $\boldsymbol{\lambda}$ сходится к множителю Лагранжа.

В дополнение к увеличению с каждой итерацией вектор линейного штрафа обновляется по формуле:

$$\boldsymbol{\lambda}^{(k+1)} = \boldsymbol{\lambda}^{(k)} - \rho\mathbf{h}(\mathbf{x}). \tag{10.44}$$

Реализация описана в алгоритме 10.2.

Алгоритм 10.2. Расширенный метод Лагранжа для целевой функции `f`, функции ограничения в виде равенства `h`, начальной точки `x`, числа итераций `k_max`, скаляра начального штрафа `ρ > 0` и штрафного множителя `γ > 1`. Функция `minimize` должна быть заменена любым выбранным методом минимизации

```
function augmented_lagrange_method(f, h, x, k_max; ρ = 1, γ = 2)
    λ = zeros(length(h(x)))
    for k in 1 : k_max
        p = x -> f(x) + λ⋅h(x) - ρ / 2 * sum(h(x) .^ 2)
        x = minimize(x -> f(x) + p(x), x)
        ρ *= γ
```

[9] Не путайте с множителями Лагранжа.

```
            λ -= ρ * h(x)
        end
        return x
    end
```

10.9. Методы внутренних точек

Методы внутренних точек (алгоритм 10.3), иногда называемые *барьерными методами*, являются методами оптимизации, которые гарантируют, что точки поиска всегда остаются в пределах допустимой области.[10] Методы внутренних точек используют барьерную функцию, которая приближается к бесконечности при приближении к границе допустимой области. Эта барьерная функция, $p_{\text{barrier}}(\mathbf{x})$, должна удовлетворять нескольким свойствам:

1. $p_{\text{barrier}}(\mathbf{x})$ непрерывна;
2. $p_{\text{barrier}}(\mathbf{x})$ неотрицательна $\left(p_{\text{barrier}}(\mathbf{x}) \geq 0\right)$;
3. $p_{\text{barrier}}(\mathbf{x})$ приближается к бесконечности, когда $\mathbf{x}$ приближается к любой границе допустимой области.

Рассмотрим некоторые примеры барьерных функций.

Обратный барьер:

$$p_{\text{barrier}}(\mathbf{x}) = -\sum_i \frac{1}{g_i(\mathbf{x})}. \tag{10.45}$$

Логарифмический барьер:

$$p_{\text{barrier}}(\mathbf{x}) = -\sum_i \begin{cases} \log\left(-g_i(\mathbf{x})\right), \text{ если } g_i(\mathbf{x}) \geq -1, \\ 0 \text{ в противном случае.} \end{cases} \tag{10.46}$$

Задача с ограничениями в виде неравенств может быть преобразована в задачу безусловной оптимизации:

$$\min_{\mathbf{x}} f(\mathbf{x}) + \frac{1}{\rho} p_{\text{barrier}}(\mathbf{x}). \tag{10.47}$$

Когда ρ увеличивается, штраф за приближение к границе уменьшается (рис. 10.12).

Особое внимание должно быть уделено тому, чтобы линейный поиск не покинул допустимой области. Линейный поиск $f(\mathbf{x} + \alpha\mathbf{d})$ ограничен интервалом $\alpha = [0, \alpha_u]$, где α_u — шаг к ближайшей границе. На практике α_u выбирается так,

[10] Методы внутренних точек, остановленные на ранней стадии, дают почти оптимальные, хотя и допустимые, расчетные точки. Методы могут быть остановлены рано из-за временных или технологических ограничений.

чтобы точка $\mathbf{x} + \alpha \mathbf{d}$ находилось исключительно внутри допустимой области, чтобы избежать сингулярности границы.

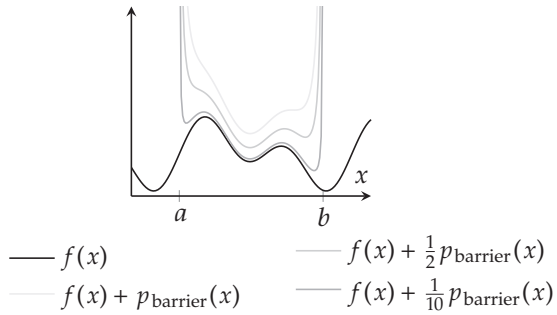


Рис. 10.12. Применение метода внутренней точки с обратным барьером для минимизации f при условии $x \in [a, b]$

Как и метод штрафных функций, метод внутренней точки начинается с низкого значения ρ и медленно увеличивает его до выполнения критерия сходимости. Метод внутренней точки обычно завершается, когда разница между последующими точками становится меньше определенного порога. Пример эффекта постепенного увеличения ρ показан на рис. 10.13.

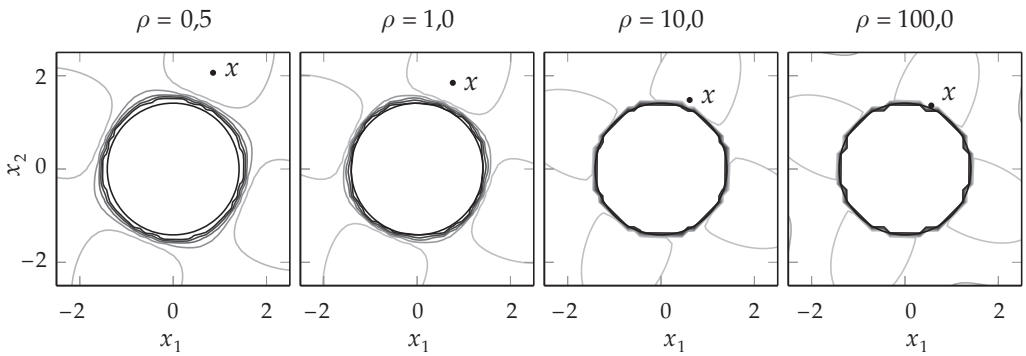


Рис. 10.13. Метод внутренней точки с обратным барьером, примененным к функции цветка (см. приложение Б.4), и ограничением $x_1^2 + x_2^2 \geq 2$

Метод внутренней точки требует допустимой точки, с которой начинается поиск. Один из удобных методов поиска допустимой точки состоит в том, чтобы оптимизировать квадратичную штрафную функцию и минимизировать ее функцию квадратичного штрафа

$$\min_{\mathbf{x}} p_{\text{quadratic}}(\mathbf{x}). \tag{10.48}$$

Алгоритм 10.3. Метод внутренней точки для целевой функции `f`, барьерной функции `p`, начальной точки `x`, начального штрафа `ρ` > 0, штрафного множителя `γ` > 1 и допуска `ε` > 0

```
function interior_point_method(f, p, x; ρ = 1, γ = 2, ε = 0.001)
    delta = Inf
    while delta > ε
      x′ = minimize(x -> f(x) + p(x) / ρ, x)
      delta = norm(x′ - x)
      x = x′
      ρ *= γ
    end
    return x
end
```

10.10. Резюме

- Ограничения — это требования, которым должно удовлетворять решение.
- Некоторые ограничения могут быть преобразованы или заменены, что приводит к безусловной задаче оптимизации.
- Аналитические методы с использованием множителей Лагранжа дают обобщенный лагранжиан и необходимые условия оптимальности с ограничениями.
- Задача условной оптимизации имеет двойственную формулировку задачи, которую легче решить, и решение которой является нижней границей решения прямой задачи.
- Методы штрафных функций штрафуют недопустимые решения и часто предоставляют оптимизатору информацию о градиенте, чтобы направлять недопустимые точки в сторону допустимой области.
- Методы внутренних точек обеспечивают допустимость, но используют барьерные функции, чтобы не выходить за пределы допустимой области.

10.11. Упражнения

Упражнение 10.1. Решите задачу

$$\begin{aligned} &\min_x x \\ &\text{при условии, что } x \geq 0, \end{aligned} \tag{10.49}$$

используя метод квадратичного штрафа $\rho > 0$. Решите задачу в замкнутом виде.

Упражнение 10.2. Решите задачу, поставленную выше, используя метод подсчета штрафов с $\rho > 1$ и сравните его с методом квадратичного штрафа.

Упражнение 10.3. Предположим, что вы решаете сложную задачу с помощью метода штрафных функций. Вы замечаете, что итерации остаются недопустимыми, и решаете остановить алгоритм. Что вы можете сделать, чтобы быть более успешным при следующей попытке?

Упражнение 10.4. Рассмотрим простую одномерную задачу минимизации, где минимизируется функция $f(x)$, при условии, что $x \geq 0$. Предположим, что ограничение активно, т.е. $x^* = 0$, где x^* является точкой минимума и $f'(x^*) > 0$ из условий оптимальности. Покажите, что, решая ту же задачу методом штрафных функций

$$f(x) + (\min(x, 0))^2, \tag{10.50}$$

вы получите недопустимое решение прямой задачи.

Упражнение 10.5. В чем преимущество расширенного метода Лагранжа по сравнению с методом квадратичного штрафа?

Упражнение 10.6. В каких ситуациях барьерный метод предпочтительнее метода штрафных функций?

Упражнение 10.7. Приведите такой пример задачи гладкой оптимизации, что для любого штрафного параметра $\rho > 0$ существует начальная точка $x^{(1)}$, для которой метод наискорейшего спуска расходится.

Упражнение 10.8. Рассмотрим задачу оптимизации

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) \\ &\text{при условии, что } \mathbf{h}(\mathbf{x}) = 0, \\ &\qquad\qquad\qquad \mathbf{g}(\mathbf{x}) \leq 0 \end{aligned} \tag{10.51}$$

без начальной расчетной точки. Как выбрать допустимую точку с учетом указанных ограничений?

Упражнение 10.9. Решите задачу условной оптимизации

$$\begin{aligned} &\min_{x} \sin\frac{4}{x} \\ &\text{при условии, что } x \in [1, 10], \end{aligned} \tag{10.52}$$

используя преобразование $x = t_{a,b}(\hat{x})$ и сигмоидальное преобразование

$$x = s(\hat{x}) = a + \frac{b-a}{1+e^{-\hat{x}}}, \quad x \in [a, b]. \tag{10.53}$$

Почему t-преобразование лучше, чем сигмоидальное?

Упражнение 10.10. Приведите пример квадратичной целевой функции, включающей две переменные, в котором добавление линейного ограничения приводит к другому оптимуму.

Упражнение 10.11. Предположим, мы хотим минимизировать функцию $x_1^3 + x_2^2 + x_3$ при условии, что $x_1 + 2x_2 + 3x_3 = 6$. Как преобразовать эту задачу в задачу безусловной минимизации с той же точкой минимума?

Упражнение 10.12. Предположим, мы хотим минимизировать функцию $-x_1 - 2x_2$ при условии, что $ax_1 + x_2 \leq 5$ и $x_1, x_2 \geq 0$. Если a — ограниченная постоянная, то какой диапазон значений a приведет к бесконечному числу оптимальных решений?

Упражнение 10.13. Рассмотрите возможность использования метода штрафных функций для решения задачи

$$\begin{aligned} &\min_x 1 - x^2 \\ &\text{при условии, что } |x| \leq 2. \end{aligned} \tag{10.54}$$

Оптимизация с использованием метода штрафа обычно включает несколько оптимизаций с увеличением веса штрафа. Нетерпеливые инженеры могут захотеть решить задачу за один прием, используя очень большой штрафной вес. Объясните, какие задачи возникают как для метода подсчета штрафа, так и для метода квадратичного штрафа.

Глава 11

Оптимизация с линейными ограничениями

Линейное программирование включает в себя решение задач оптимизации с помощью линейных целевых функций и линейных ограничений. Многие задачи естественным образом описываются линейными уравнениями и неравенствами, в том числе задачи в таких разных областях, как транспорт, связь, производство, экономика и исследование операций. Многие задачи, которые не являются линейными по природе, часто могут быть аппроксимированы линейными задачами. Для эксплуатации линейной структуры были разработаны несколько методов. Современные методы и оборудование могут решать задачи минимизации с миллионами переменных и ограничений.[1]

11.1. Условная оптимизация

Задача линейного программирования, или *линейная программа*,[2] может быть выражена в нескольких формах. Каждая задача линейного программирования состоит из линейной целевой функции и набора линейных ограничений:

---

[1] Эта глава представляет собой краткое введение в задачи линейного программирования и один из вариантов симплексного алгоритма, используемого для их решения. Линейному программированию полностью посвящено несколько учебников, в том числе [157]. Для решения задач линейного программирования разработано множество пакетов, таких как `Convex.jl` и `JuMP.jl`, которые имеют интерфейсы для открытых и коммерческих приложений.

[2] *Задача квадратичного программирования* — это обобщение задачи линейного программирования, где целевая функция квадратичная, а ограничения являются линейными. Общие подходы к решению таких задач включают некоторые алгоритмы, которые обсуждались в предыдущих главах, в частности метод внутренних точек, расширенный метод Лагранжа и метод сопряженных градиентов. Симплексный метод, описанный в этой главе, также был адаптирован для решения задач квадратичного программирования [114].

$$\begin{aligned}
&\min_{\mathbf{x}} \mathbf{c}^{\mathrm{T}}\mathbf{x} \\
&\text{при условии, что } \mathbf{w}_{\mathrm{LE}}^{(i)\mathrm{T}}\mathbf{x} \le b_i \text{ для } i \in \{1, 2, ...\}, \\
&\qquad\qquad \mathbf{w}_{\mathrm{GE}}^{(j)\mathrm{T}}\mathbf{x} \le b_j \text{ для } j \in \{1, 2, ...\}, \\
&\qquad\qquad \mathbf{w}_{\mathrm{EQ}}^{(k)\mathrm{T}}\mathbf{x} = b_k \text{ для } k \in \{1, 2, ...\},
\end{aligned} \tag{11.1}$$

где индексы i, j и k пробегают конечное множество значений. Такая задача оптимизации приведена в примере 11.1. Преобразование реальных задач в такую математическую форму часто бывает нетривиальным. Этот учебник фокусируется на алгоритмах для получения решений, а вопросы моделирования реальных ситуаций обсуждаются в других книгах (см., например, [159]). Несколько интересных преобразований приведены в примере 11.2.

Пример 11.1. Пример задачи линейного программирования

Следующая задача имеет линейную цель и линейные ограничения, что делает ее задачей линейного программирования.

$$\begin{aligned}
&\min_{x_1,x_2,x_3} \quad 2x_1 - 3x_2 + 7x_3 \\
&\text{при условии, что} \quad 2x_1 + 3x_2 - 8x_3 \le 5, \\
&\qquad\qquad\qquad\quad 4x_1 + \ x_2 + 3x_3 \le 9, \\
&\qquad\qquad\qquad\quad x_1 - 5x_2 - 3x_3 \ge -4, \\
&\qquad\qquad\qquad\quad x_1 + \ x_2 + \ 2x_3 = 1.
\end{aligned}$$

Пример 11.2. Общие задачи минимизации норм, которые можно преобразовать в задачи линейного программирования

Многие задачи могут быть преобразованы в задачи линейного программирования, имеющие одинаковое решение. Два примера — это задачи минимизации норм L_1 и L_∞:

$$\min \|\mathbf{Ax} - \mathbf{b}\|_1 \qquad\qquad \min \|\mathbf{Ax} - \mathbf{b}\|_\infty$$

Первая проблема эквивалентна решению задачи

$$\begin{aligned}
&\min_{\mathbf{x},\,\mathbf{s}} \mathbf{1}^{\mathrm{T}}\mathbf{s} \\
&\text{при условии, что} \\
&\qquad\qquad \mathbf{Ax} - \mathbf{b} \le \mathbf{s}, \\
&\qquad\qquad \mathbf{Ax} - \mathbf{b} \ge -\mathbf{s}.
\end{aligned}$$

с дополнительными переменными $\mathbf{s}$.

Вторая задача эквивалентна решению

$$\min_{\mathbf{x},t} t$$

при условии, что

$$\mathbf{Ax} - \mathbf{b} \leq t\mathbf{1},$$
$$\mathbf{Ax} - \mathbf{b} \geq -t\mathbf{1}.$$

с дополнительной переменной t.

11.1.1 Общий вид

Мы можем записать задачи линейного программирования более компактно, используя матрицы:[3]

$$\begin{aligned} &\min_{\mathbf{x}} \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ &\text{при условии, что } \mathbf{A}_{\mathrm{LE}}\mathbf{x} \leq \mathbf{b}_{\mathrm{LE}}, \\ &\qquad\qquad\qquad\quad \mathbf{A}_{\mathrm{GE}}\mathbf{x} \geq \mathbf{b}_{\mathrm{GE}}, \\ &\qquad\qquad\qquad\quad \mathbf{A}_{\mathrm{EQ}}\mathbf{x} = \mathbf{b}_{\mathrm{EQ}}. \end{aligned} \tag{11.2}$$

11.1.2. Стандартный вид

Общая задача линейного программирования (11.2) может быть преобразована в *стандартный вид*, где все ограничения имеют вид неравенства "меньше или равно", а расчетные переменные неотрицательны

$$\begin{aligned} &\min_{\mathbf{x}} \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ &\text{при условии, что } \mathbf{Ax} \leq \mathbf{b}, \\ &\qquad\qquad\qquad\quad \mathbf{x} \geq \mathbf{0}. \end{aligned} \tag{11.3}$$

Неравенства "больше или равно" инвертируются, а ограничения в виде равенств делятся на две части.

$$\mathbf{A}_{\mathrm{GE}}\mathbf{x} \geq \mathbf{b}_{\mathrm{GE}} \rightarrow -\mathbf{A}_{\mathrm{GE}}\mathbf{x} \leq -\mathbf{b}_{\mathrm{GE}}$$
$$\mathbf{A}_{\mathrm{EQ}}\mathbf{x} = \mathbf{b}_{\mathrm{EQ}} \rightarrow \begin{cases} \mathbf{A}_{\mathrm{EQ}}\mathbf{x} \leq \mathbf{b}_{\mathrm{EQ}}, \\ -\mathbf{A}_{\mathrm{EQ}}\mathbf{x} \leq -\mathbf{b}_{\mathrm{EQ}}. \end{cases} \tag{11.4}$$

Мы должны убедиться, что все точки $\mathbf{x}$ также неотрицательны. Предположим, мы начинаем с задачи линейного программирования, где $\mathbf{x}$ не обязательно должен быть неотрицательной переменной:

$$\begin{aligned} &\min_{\mathbf{x}} \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ &\text{при условии, что } \mathbf{Ax} \leq \mathbf{b}. \end{aligned} \tag{11.5}$$

[3] Здесь каждое ограничение является поэлементным. Например, запись $\mathbf{a} \leq \mathbf{b}$ означает $a_i \leq b_i$ для всех i.

Изменим $\mathbf{x}$ на $\mathbf{x}^+ - \mathbf{x}^-$ и ограничим $\mathbf{x}^+ \geq \mathbf{0}$ и $\mathbf{x}^- \geq \mathbf{0}$:

$$\begin{aligned} \min_{\mathbf{x}^+,\mathbf{x}^-} \quad & \begin{bmatrix} \mathbf{c}^T & -\mathbf{c}^T \end{bmatrix} \begin{bmatrix} \mathbf{x}^+ \\ \mathbf{x}^- \end{bmatrix} \\ \text{при условии, что} \quad & \begin{bmatrix} \mathbf{A} & -\mathbf{A} \end{bmatrix} \begin{bmatrix} \mathbf{x}^+ \\ \mathbf{x}^- \end{bmatrix} \leq \mathbf{b}, \\ & \begin{bmatrix} \mathbf{x}^+ \\ \mathbf{x}^- \end{bmatrix} \geq \mathbf{0}. \end{aligned} \tag{11.6}$$

Линейная целевая функция $\mathbf{c}^T\mathbf{x}$ образует пологий откос. Функция увеличивается в направлении $\mathbf{c}$, и, как следствие, все линии уровня перпендикулярны $\mathbf{c}$ и параллельны друг другу, как показано на рис. 11.1.

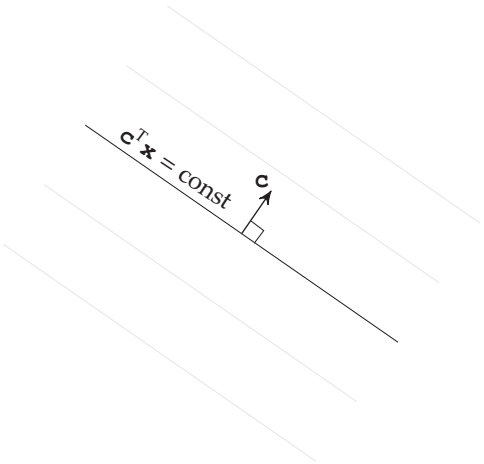


Рис. 11.1. Линии уровня линейной целевой функции $\mathbf{c}^T\mathbf{x}$, которые увеличиваются в направлении $\mathbf{c}$

Одно ограничение в виде неравенства $\mathbf{w}^T\mathbf{x} \leq b$ образует *полупространство*, или область, расположенную по одну сторону от гиперплоскости. Гиперплоскость перпендикулярна вектору $\mathbf{w}$ и определяется как $\mathbf{w}^T\mathbf{x} > b$ (рис. 11.2). Область $\mathbf{w}^T\mathbf{x} > b$ находится на стороне $+\mathbf{w}$, а $\mathbf{w}^T\mathbf{x} < b$ — на стороне вектора $-\mathbf{w}$.

Полупространства — это выпуклые множества (см. приложение В.3), а пересечение выпуклых множеств является выпуклым, как показано на рис. 11.3. Таким образом, допустимое множество задачи линейного программирования всегда будет образовывать выпуклое множество. Выпуклость допустимого множества, наряду с выпуклостью целевой функции, подразумевает, что если мы найдем локальный допустимый минимум, то это будет глобальный допустимый минимум.

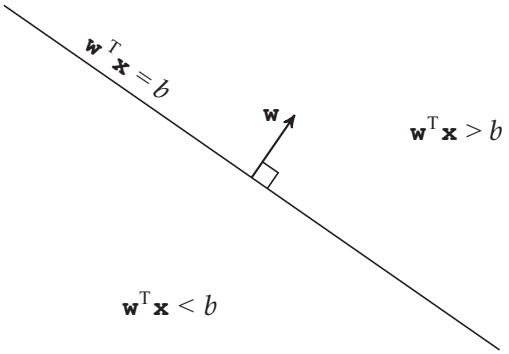


Рис. 11.2. Линейное ограничение

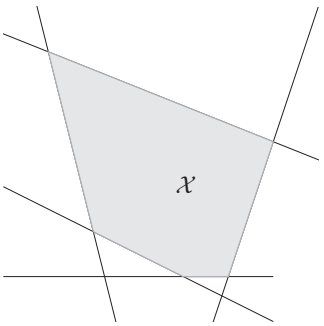


Рис. 11.3. Пересечение линейных ограничений является выпуклым множеством

Допустимое множество — это выпуклая область, окруженная плоскими гранями. В зависимости от конфигурации области решение может лежать в вершине, на ребре или на всей грани. Если задача недостаточно ограничена, то решение может быть неограниченным, а если система *чрезмерно ограничена*, то допустимого решения не существует. Примеры таких случаев показаны на рис. 11.4.

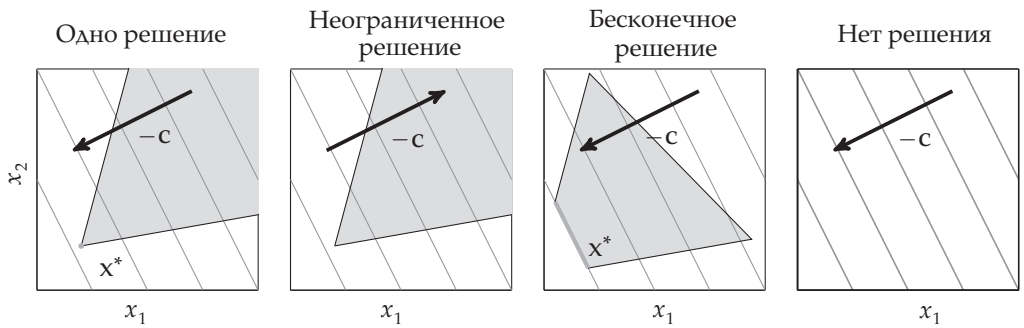


Рис. 11.4. Несколько различных линейных форм задач с различными решениями

11.1.3. Форма с ограничениями в виде равенства

Задачи линейного программирования часто решаются в третьей форме, *в виде равенства*

$$\min_{\mathbf{x}} \quad \mathbf{c}^T\mathbf{x} \tag{11.7}$$
$$\text{при условии, что } \mathbf{A}\mathbf{x} = \mathbf{b},$$
$$\mathbf{x} \geq \mathbf{0},$$

где векторы $\mathbf{x}$ и $\mathbf{c}$ содержат n элементов, $\mathbf{A}$ — матрица $m \times n$, а вектор $\mathbf{b}$ содержит m компонентов. Иначе говоря, у нас есть n неотрицательных переменных и система из m уравнений, определяющих ограничения в виде равенства.

Ограничения в виде равенства состоят из двух частей. Первое ограничение, $\mathbf{Ax} = \mathbf{b}$, требует, чтобы решение находилось в аффинном подпространстве.[4] Такое ограничение удобно, потому что методы поиска могут оставаться в пределах ограниченного аффинного подпространства, чтобы оставаться допустимыми. Вторая часть ограничений требует, чтобы выполнялось неравенство $\mathbf{x} \geq 0$, т.е. решение должно находиться в положительном квадранте. Таким образом, допустимое множество является неотрицательной частью аффинного подпространства. Пример 11.3 предоставляет визуализацию для простой линейной программы.

Пример 11.3. Допустимые множества для формы с равенствами — это гиперплоскости

Рассмотрим задачу линейного программирования в стандартном виде:

$$\min_{x} x$$
$$\text{при условии, что } x \geq 1$$

Преобразовывая ее в форму равенства, получаем

$$\min_{x,s} x$$
$$\text{при условии, что } x - s = 1,$$
$$x, s \geq 0.$$

Ограничение в виде равенств требует, чтобы допустимые точки попадали на прямую $x - s = 1$. Эта прямая является одномерным аффинным подпространством двумерного евклидова пространства.

Любая задача линейного программирования в стандартной форме может быть преобразована в форму с равенствами. С помощью ввода *дополнительных переменных* $\mathbf{s}$ ограничения преобразуются в следующий вид:

$$\mathbf{Ax} \leq \mathbf{b} \rightarrow \mathbf{Ax} + \mathbf{s} = \mathbf{b}, \mathbf{s} \geq 0. \tag{11.8}$$

Дополнительные переменные вводятся, чтобы обеспечить выполнение равенства.

Начнем с задачи линейного программирования

$$\min_{\mathbf{x}} \quad \mathbf{c}^{\mathrm{T}}\mathbf{x} \tag{11.9}$$
$$\text{при условии, что } \mathbf{Ax} \leq \mathbf{b},$$
$$\mathbf{x} \geq \mathbf{0}.$$

[4] Говоря неформально, *аффинное подпространство* — это векторное пространство, которое было параллельно перенесено так, что его начало координат в многомерном пространстве не обязательно совпадает с нулем.

Введем дополнительные переменные

$$\begin{aligned} &\min_{\mathbf{x},\mathbf{s}} && \begin{bmatrix} \mathbf{c}^{\mathrm{T}} & \mathbf{0}^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{s} \end{bmatrix} \\ &\text{при условии, что} && \begin{bmatrix} \mathbf{A} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{s} \end{bmatrix} = \mathbf{b}, \\ & && \begin{bmatrix} \mathbf{x} \\ \mathbf{s} \end{bmatrix} \geq \mathbf{0}. \end{aligned} \tag{11.10}$$

Пример 11.4 демонстрирует переход от стандартного вида к виду с ограничениями в виде равенств.

Пример 11.4. Преобразование задачи линейного программирования в форму равенства

Рассмотрим задачу линейного программирования

$$\begin{aligned} &\min_{\mathbf{x}} 5x_1 + 4x_2 \\ &\text{при условии, что} \quad 2x_1 + 3x_2 \leq 5, \\ &\qquad\qquad\qquad\quad 4x_1 + \ x_2 \leq 11. \end{aligned}$$

Чтобы преобразовать программу в форму с ограничениями в виде равенства, сначала введем две дополнительные переменные:

$$\begin{aligned} &\min_{\mathbf{x},\mathbf{s}} 5x_1 + 4x_2 \\ &\text{при условии, что} \quad 2x_1 + 3x_2 + s_1 = 5, \\ &\qquad\qquad\qquad\quad 4x_1 + \ x_2 \ + s_2 = 11, \\ &\qquad\qquad\qquad\qquad\quad s_1, s_2 \geq 0. \end{aligned}$$

Затем разобъем вектор $\mathbf{x}$:

$$\begin{aligned} &\min_{\mathbf{x}^+,\mathbf{x}^-} && 5\left(x_1^+ - x_1^-\right) + 4\left(x_2^+ - x_2^-\right) \\ &\text{при условии, что} && 2\left(x_1^+ - x_1^-\right) + 3\left(x_2^+ - x_2^-\right) + s_1 = 5, \\ & && 4\left(x_1^+ - x_1^-\right) + \left(x_2^+ - x_2^-\right) + s_2 = 11, \\ & && x_1^+, x_1^-, x_2^+, x_2^-, s_1, s_2 \geq 0. \end{aligned}$$

11.2. Симплекс-метод

Симплекс-метод решает задачи линейного программирования, перемещаясь от вершины к вершине допустимого множества.[5] Метод гарантированно достигнет оптимального решения, пока задача линейного программирования является допустимой и ограниченной.

Симплекс-метод применяется к задачам линейного программирования с ограничениями в форме равенства ($\mathbf{Ax} = \mathbf{b}$, $\mathbf{x} \geq 0$). Предполагается, что строки матрицы $\mathbf{A}$ линейно независимы.[6] Мы также предполагаем, что число ограничений в виде равенства меньше, чем количество переменных ($m \leq n$). Это гарантирует, что задача не является чрезмерно ограниченной. Выполнение данных условий гарантируется с помощью предварительной обработки матрицы $\mathbf{A}$.

11.2.1. Вершины

Допустимые множества для задач линейного программирования в форме равенства имеют вид выпуклых *многогранников*, которые являются геометрическими объектами с плоскими гранями. Эти многогранники образованы пересечением ограничений в форме равенства с положительным квадрантом. С многогранником связаны *вершины*, являющиеся точками в допустимом множестве, которые не лежат между любыми другими точками в допустимом множестве.

Допустимое множество состоит из нескольких различных типов расчетных точек. Точки во внутренней части допустимого множества никогда не бывают оптимальными, потому что их можно улучшить, двигаясь вдоль вектора $-\mathbf{c}$. Точки на гранях могут быть оптимальными, только если грань перпендикулярна вектору $\mathbf{c}$. Точки на гранях, не перпендикулярных вектору $\mathbf{c}$, можно улучшить, скользя вдоль грани в направлении проекции вектора $\mathbf{c}$ на грани. Точно так же точки на ребрах могут быть оптимальными, только если ребро перпендикулярно вектору $\mathbf{c}$, в противном случае их можно улучшить, скользя вдоль проекции вектора $\mathbf{c}$ на ребро. Наконец, вершины также могут быть оптимальными.

Симплекс-метод создает оптимальную вершину. Если задача линейного программирования содержит допустимые точки, она также содержит хотя бы одну вершину. Кроме того, если задача линейного программирования имеет решения, то хотя бы одно решение должно лежать в вершине. Если все ребро или грань оптимальны, решение в вершине так же приемлемо, как и любое другое.

---

[5] Симплекс-метод был первоначально разработан в 1940-х годах Джорджем Данцигом. Историю развития можно найти в [31].

[6] Если строки матрицы линейно независимы, то говорят, что она имеет полный строковый ранг. Линейная независимость достигается за счет устранения избыточных ограничений в форме равенства.

Каждая вершина задачи линейного программирования с ограничениями в форме равенства может быть однозначно определена с помощью $n - m$ компонентов вектора $\mathbf{x}$, которые равны нулю. Эти компоненты активно ограничены неравенством $x_i \geq 0$. На рис. 11.5 представлены активные ограничения, необходимые для идентификации вершины.

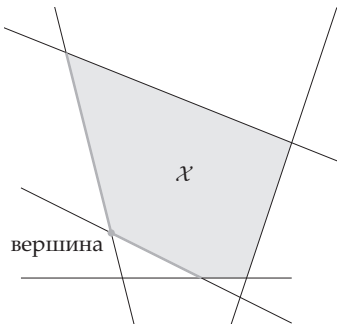


Рис. 11.5. В двух измерениях любая вершина будет иметь как минимум два активных ограничения

Ограничение в виде равенства $\mathbf{Ax} = \mathbf{b}$ имеет единственное решение, если $\mathbf{A}$ — квадратная матрица. Мы предположили, что $m \leq n$, поэтому выбор m переменных и приравнивание оставшихся переменных к нулю, по существу, приводит к удалению $n - m$ столбцов матрицы $\mathbf{A}$ и появлению матрицы ограничений $m \times m$ (см. пример 11.5).

Пример 11.5. Приравнивание к нулю $n - m$ компонентов вектора $\mathbf{x}$ может однозначно определить точку

В задаче с пятью переменными и тремя ограничениями приравнивание двух переменных к нулю однозначно определяет точку.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \end{bmatrix} \begin{bmatrix} x_1 \\ 0 \\ x_3 \\ x_4 \\ 0 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{13} & a_{14} \\ a_{21} & a_{23} & a_{24} \\ a_{31} & a_{33} & a_{34} \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}.$$

Множество индексов любой вершины $\{1, ..., n\}$ можно разбить на два подмножества, $\mathcal{B}$ и $\mathcal{V}$, так что выполняются следующие условия.

- Расчетные значения, связанные с индексами в подмножестве $\mathcal{V}$, равны нулю:

$$i \in \mathcal{V} \Rightarrow x_i = 0 \tag{11.11}$$

- Расчетные значения, связанные с индексами в $\mathcal{B}$, равное или не равные нулю:

$$i \in \mathcal{B} \Rightarrow x_i \geq 0. \tag{11.12}$$

- Подмножество $\mathcal{B}$ содержит ровно m элементов, а множество $\mathcal{V}$ — в точности $n - m$ элементов

Мы используем обозначение $\mathbf{x}_\mathcal{B}$ для ссылки на вектор, состоящий из компонентов $\mathbf{x}$, находящихся в подмножестве $\mathcal{B}$, и $\mathbf{x}_\mathcal{V}$ — для ссылки на вектор, состоящий из компонентов $\mathbf{x}$, находящихся в подмножестве $\mathcal{V}$. Обратите внимание на то, что $\mathbf{x}_\mathcal{V} = \mathbf{0}$.

Вершина, связанная с разбиением $(\mathcal{B}, \mathcal{V})$, может быть получена с использованием $m \times m$ матрицы $\mathbf{A}_\mathcal{B}$, образованной m столбцами $\mathbf{A}$, выбранными с помощью подмножества $\mathcal{B}$:[7]

$$\mathbf{A}\mathbf{x} = \mathbf{A}_\mathcal{B}\mathbf{x}_\mathcal{B} = \mathbf{b} \rightarrow \mathbf{x}_\mathcal{B} = \mathbf{A}_\mathcal{B}^{-1}\mathbf{b}. \tag{11.13}$$

Знания вектора $\mathbf{x}_\mathcal{B}$ достаточно для построения $\mathbf{x}$, остальные переменные равны нулю. Алгоритм 11.1 реализует эту процедуру, а пример 11.6 демонстрирует проверку того, что данная расчетная точка является вершиной.

Алгоритм 11.1. Метод выделения вершины, связанной с разбиением `B` и задачей линейного программирования `LP` с ограничениями в форме равенства. В программе введен специальный тип `LinearProgram` для задач линейного программирования

```
mutable struct LinearProgram
    A
    b
    c
end
function get_vertex(B, LP)
    A, b, c = LP.A, LP.b, LP.c
    b_inds = sort!(collect(B))
    AB = A[:, b_inds]
    xB = AB \ b
    x = zeros(length(c))
    x[b_inds] = xB
    return x
end
```

[7] Если множества $\mathcal{B}$ и $\mathcal{V}$ идентифицируют вершину, то столбцы матрицы $\mathbf{A}_\mathcal{B}$ должны быть линейно независимыми, поскольку система $\mathbf{A}\mathbf{x} = \mathbf{b}$ должна иметь ровно одно решение. Следовательно, система $\mathbf{A}_\mathcal{B}\mathbf{x}_\mathcal{B} = \mathbf{b}$ должно иметь ровно одно решение. Эта линейная независимость гарантирует, что матрица $\mathbf{A}_\mathcal{B}$ обратима.

Пример 11.6. Проверка того, что расчетная точка является вершиной для ограничений в виде равенства

Рассмотрим ограничения:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & -1 & 2 & 3 \\ 2 & 1 & 2 & -1 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix}, \quad \mathbf{x} \geq \mathbf{0}.$$

Рассмотрим расчетную точку $\mathbf{x} = [1, 1, 0, 0]$. Мы можем проверить, что вектор $\mathbf{x}$ является допустимым и что он имеет не более трех ненулевых элементов. Мы можем выбрать $\mathcal{B} = \{1, 2, 3\}$ или $\mathcal{B} = \{1, 2, 4\}$. Обе матрицы

$$\mathbf{A}_{\{1,2,3\}} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & -1 & 2 \\ 2 & 1 & 2 \end{bmatrix}$$

и

$$\mathbf{A}_{\{1,2,4\}} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & -1 & 3 \\ 2 & 1 & -1 \end{bmatrix}$$

являются обратимыми. Таким образом, $\mathbf{x}$ — вершина допустимого многогранника.

Хотя каждая вершина имеет ассоциированное с ней разбиение $(\mathcal{B}, \mathcal{V})$, не каждому разбиению соответствует вершина. Разбиение соответствует вершине только в том случае, если матрица $\mathbf{A}_{\mathcal{B}}$ невырождена и точка, полученная с помощью уравнения (11.13), является допустимой.[8] Идентификация разделов, соответствующих вершинам, нетривиальна, и в разделе 11.2.4 мы показываем, что нахождение такого разбиения предполагает решение задачи линейного программирования! Симплекс-метод состоит из двух этапов: этапа инициализации, которая идентифицирует разбиение вершин, и этапа оптимизации, которая перебирает разбиения вершин в направлении разбиения, соответствующего оптимальной вершине. Мы обсудим оба эти этапа позже в этом разделе.

[8] Например, $\mathcal{B} = \{1, 2\}$ для ограничений $\begin{bmatrix} 1 & 2 & 0 \\ 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ соответствует уравнению $\begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, в котором матрица $\mathbf{A}_{\mathcal{B}}$ не является обратимой, поэтому это уравнение не имеет единственного решения.

11.2.2. Необходимые условия первого порядка

Необходимые условия оптимальности первого порядка (FONC) позволяют определить, когда вершина является оптимальной, и сообщить, как перейти к более благоприятной вершине. Построим лагранжиан для задачи линейного программирования в форме равенства:[9]

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}) = \mathbf{c}^{\mathrm{T}}\mathbf{x} - \boldsymbol{\mu}^{\mathrm{T}}\mathbf{x} - \boldsymbol{\lambda}^{\mathrm{T}}(\mathbf{A}\mathbf{x} - \mathbf{b}) \tag{11.14}$$

со следующими условиями FONC.

1. **Выполнимость:** $\mathbf{A}\mathbf{x} = \mathbf{b}$, $\mathbf{x} \geq \mathbf{0}$.
2. **Двойственная допустимость:** $\boldsymbol{\mu} \geq 0$.
3. **Дополнительная нежесткость:** $\boldsymbol{\mu} \odot \mathbf{x} = \mathbf{0}$.
4. **Стационарность:** $\mathbf{A}^{\mathrm{T}}\boldsymbol{\lambda} + \boldsymbol{\mu} = \mathbf{c}$.

Условия FONC являются достаточными условиями оптимальности для задач линейного программирования. Таким образом, если $\boldsymbol{\mu}$ и $\boldsymbol{\lambda}$ могут быть вычислены для данной вершины и все четыре уравнения FONC удовлетворены, то вершина является оптимальной.

Мы можем разложить условие стационарности на компоненты $\mathcal{B}$ и $\mathcal{V}$:

$$\mathbf{A}^{\mathrm{T}}\boldsymbol{\lambda} + \boldsymbol{\mu} = \mathbf{c} \rightarrow \begin{cases} \mathbf{A}_{\mathcal{B}}^{\mathrm{T}}\boldsymbol{\lambda} + \boldsymbol{\mu}_{\mathcal{B}} = \mathbf{c}_{\mathcal{B}}, \\ \mathbf{A}_{\mathcal{V}}^{\mathrm{T}}\boldsymbol{\lambda} + \boldsymbol{\mu}_{\mathcal{V}} = \mathbf{c}_{\mathcal{V}}. \end{cases} \tag{11.15}$$

Мы можем выбрать $\boldsymbol{\mu}_{\mathcal{B}} = \mathbf{0}$, чтобы удовлетворить условие дополнительной нежесткости. Значение $\boldsymbol{\lambda}$ может быть вычислено по множеству $\mathcal{B}$:[10]

$$\begin{aligned} \mathbf{A}^{\mathrm{T}}\boldsymbol{\lambda} + \underbrace{\boldsymbol{\mu}_{\mathcal{B}}}_{=\mathbf{0}} &= \mathbf{c}_{\mathcal{B}}, \\ \boldsymbol{\lambda} &= \mathbf{A}_{\mathcal{B}}^{-\mathrm{T}}\mathbf{c}_{\mathcal{B}}. \end{aligned} \tag{11.16}$$

Мы можем использовать его, чтобы получить

$$\begin{aligned} \mathbf{A}_{\mathcal{V}}^{\mathrm{T}}\boldsymbol{\lambda} + \boldsymbol{\mu}_{\mathcal{V}} &= \mathbf{c}_{\mathcal{V}}, \\ \boldsymbol{\mu}_{\mathcal{V}} &= \mathbf{c}_{\mathcal{V}} - \mathbf{A}_{\mathcal{V}}^{\mathrm{T}}\boldsymbol{\lambda}, \\ \boldsymbol{\mu}_{\mathcal{V}} &= \mathbf{c}_{\mathcal{V}} - \left(\mathbf{A}_{\mathcal{B}}^{-1}\mathbf{A}_{\mathcal{V}}\right)^{\mathrm{T}}\mathbf{c}_{\mathcal{B}}, \end{aligned} \tag{11.17}$$

---

[9] Обратите внимание на то, что в неравенство $\mathbf{x} \geq \mathbf{0}$ необходимо инвертировать путем умножения обеих сторон на –1, что приводит к появлению отрицательного знака перед $\boldsymbol{\mu}$. Лагранжиан может быть определен как с положительным, так и отрицательным множителем $\boldsymbol{\mu}$.

[10] Мы используем $\mathbf{A}^{-\mathrm{T}}$ для обозначения транспонированной обратной матрицы $\mathbf{A}$.

Знание множества $\boldsymbol{\mu}_{\mathcal{V}}$ позволяет оценить оптимальность вершин. Если $\boldsymbol{\mu}_{\mathcal{V}}$ содержит отрицательные компоненты, то двойственная допустимость не выполняется и вершина неоптимальна.

11.2.3. Этап оптимизации

Симплекс-метод поддерживает разбиение $(\mathcal{B}, \mathcal{V})$, которое соответствует вершине многогранника допустимого множества. Разбиение можно обновить, поменяв местами индексы между $\mathcal{B}$ и $\mathcal{V}$. Такой обмен означает перемещение от одной вершины вдоль ребра допустимого многогранника к другой вершине. Если начальное разбиение соответствует вершине и задача имеет ограничения, то симплекс-метод гарантированно сходится к оптимальному решению.

Переход $\mathbf{x} \rightarrow \mathbf{x}'$ между вершинами должен удовлетворять уравнению $\mathbf{A}\mathbf{x}' = \mathbf{b}$. Начиная с разбиения, определенного множеством $\mathcal{B}$, мы выбираем *входной индекс* $q \in \mathcal{V}$, который определяет входное множество $\mathcal{B}$ с помощью одной из эвристик, описанных в конце этого раздела. Новая вершина $\mathbf{x}'$ должна удовлетворять уравнению:

$$\mathbf{A}\mathbf{x}' = \mathbf{A}_{\mathcal{B}}\mathbf{x}'_{\mathcal{B}} + \mathbf{A}_{\{q\}}x'_q = \mathbf{A}_{\mathcal{B}}\mathbf{x}_{\mathcal{B}} = \mathbf{A}\mathbf{x} = \mathbf{b}. \tag{11.18}$$

Во время перехода один *выходной индекс* $p \in \mathcal{V}$ в $\mathbf{x}'_{\mathcal{B}}$ становится нулевым и заменяется столбцом, соответствующим индексу q. Это действие называется *выбором главного элемента.*

Мы можем найти новую расчетную точку

$$\mathbf{x}'_{\mathcal{B}} = \mathbf{x}_{\mathcal{B}} - \mathbf{A}_{\mathcal{B}}^{-1}\mathbf{A}_{\{q\}}x'_q. \tag{11.19}$$

Определенный выходной индекс $p \in \mathcal{B}$ становится активным, когда выполняется условие

$$\left(\mathbf{x}'_{\mathcal{B}}\right)_p = 0 = \left(\mathbf{x}_{\mathcal{B}}\right)_p - \left(\mathbf{A}_{\mathcal{B}}^{-1}\mathbf{A}_{\{q\}}\right)_p x'_q \tag{11.20}$$

и, таким образом, происходит увеличение от $x_q = 0$ до x'_q, где

$$x'_q = \frac{\left(\mathbf{x}_{\mathcal{B}}\right)_p}{\left(\mathbf{A}_{\mathcal{B}}^{-1}\mathbf{A}_{\{q\}}\right)_p}. \tag{11.21}$$

Выходной индекс получается с использованием *критерия минимального отношения*, который рассчитывается для каждого потенциального выходного индекса и выбирает индекс с минимальным значением x'_q. Затем индексы p и q меняются местами в множествах $\mathcal{B}$ и $\mathcal{V}$. Переход по ребру реализован в алгоритме 11.2.

Алгоритм 11.2. Способ вычисления индекса `p` и нового значения координаты x'_q, полученного путем увеличения индекса `q` вершины, определенной разбиением `B` в задаче линейного программирования `LP` с ограничениями в виде равенства

```
function edge_transition(LP, B, q)
    A, b, c = LP.A, LP.b, LP.c
    n = size(A, 2)
    b_inds = sort(B)
    n_inds = sort!(setdiff(1 : n, B))
    AB = A[:,b_inds]
    d, xB = AB \ A[:, n_inds[q]], AB \ b

    p, xq' = 0, Inf
    for i in 1 : length(d)
        if d[i] > 0
           v = xB[i] / d[i]
           if v < xq'
              p, xq' = i, v
           end
        end
    end
    return (p, xq')
end
```

Влияние перехода на целевую функцию может быть вычислено с использованием x'_q. Значение целевой функции в новой вершине равно[11]

$$\mathbf{c}^{\mathrm{T}}\mathbf{x}' = \mathbf{c}_{\mathcal{B}}^{\mathrm{T}}\mathbf{x}'_{\mathcal{B}} + c_q x'_q = \tag{11.22}$$

$$= \mathbf{c}_{\mathcal{B}}^{\mathrm{T}}\left(\mathbf{x}_{\mathcal{B}} - \mathbf{A}_{\mathcal{B}}^{-1}\mathbf{A}_{\{q\}}x'_q\right) + c_q x'_q = \tag{11.23}$$

$$= \mathbf{c}_{\mathcal{B}}^{\mathrm{T}}\mathbf{x}_{\mathcal{B}} - \mathbf{c}_{\mathcal{B}}^{\mathrm{T}}\mathbf{A}_{\mathcal{B}}^{-1}\mathbf{A}_{\{q\}}x'_q + c_q x'_q = \tag{11.24}$$

$$= \mathbf{c}_{\mathcal{B}}^{\mathrm{T}}\mathbf{x}_{\mathcal{B}} - \left(c_q - \mu_q\right)x'_q + c_q x'_q = \tag{11.25}$$

$$= \mathbf{c}^{\mathrm{T}}\mathbf{x} + \mu_q x'_q. \tag{11.26}$$

Выбор входного индекса q уменьшает значение целевой функции на

$$\mathbf{c}^{\mathrm{T}}\mathbf{x}' - \mathbf{c}^{\mathrm{T}}\mathbf{x} = \mu_q x'_q. \tag{11.27}$$

[11] Здесь мы использовали тот факт, что $\boldsymbol{\lambda} = \mathbf{A}_{\mathcal{B}}^{-\mathrm{T}}\mathbf{c}_{\mathcal{B}}$ и что $\mathbf{A}_{\{q\}}^{\mathrm{T}}\boldsymbol{\lambda} = \mathbf{c}_q - \boldsymbol{\mu}_q$.

Целевая функция уменьшается, только если μ_q отрицательно. Чтобы двигаться к оптимальности, мы должны выбрать индекс q в множестве $\mathcal{V}$ такой, что μ_q будет отрицательным. Если все компоненты $\boldsymbol{\mu}_\mathcal{V}$ не отрицательные, мы нашли глобальный оптимум.

Поскольку в множестве $\boldsymbol{\mu}_\mathcal{V}$ может быть несколько отрицательных элементов, для выбора входного индекса можно использовать разные эвристики.[12]

- *Жадная эвристика*, которая выбирает индекс q, максимально уменьшающий $\mathbf{c}^\mathrm{T}\mathbf{x}$.
- *Правило Данцига*, которое выбирает индекс q самого большого по модулю отрицательного элемента в векторе $\boldsymbol{\mu}$. Это правило легко вычислить, но оно не гарантирует максимального уменьшения $\mathbf{c}^\mathrm{T}\mathbf{x}$. Оно также чувствительно к масштабированию ограничений.[13]
- *Правило Блэнда*, которое выбирает первый индекс q, соответствующий отрицательному элементу. При самостоятельном использовании правило Блэнда обычно снижает производительность алгоритма на практике. Однако это правило может помочь предотвратить *циклы*, которые возникают, когда мы возвращаемся к вершине, которую посетили ранее, без уменьшения целевой функции. Это правило обычно используется только после того, как не было сделано никаких улучшений для нескольких итераций другого правила, чтобы выйти из цикла и обеспечить сходимость.

Одна итерация этапа оптимизации симплекс-метода перемещает разбиение вершины в соседнюю вершину на основе эвристики для входного индекса. Алгоритм 11.3 реализует такую итерацию с жадной эвристикой. Пример 11.7 демонстрирует использование симплекс-метода, начиная с известного разбиения вершин, для решения задачи линейного программирования.

Алгоритм 11.3. Одна итерация симплексного алгоритма, в котором множество $\mathcal{B}$ перемещается из одной вершины в соседнюю, при этом максимально уменьшая целевую функцию. Функция **`step_lp!`** получает разбиение, определенное множеством **`B`** и задачей линейного программирования **`LP`**

```
function step_lp!(B, LP)
    A, b, c = LP.A, LP.b, LP.c
    n = size(A, 2)
    b_inds = sort!(B)
```

[12] Современные реализации используют более сложные правила (например, см. [51]).

[13] Для ограничения $\mathbf{A}^\mathrm{T}\mathbf{x} = \mathbf{b} \rightarrow \alpha\mathbf{A}^\mathrm{T}\mathbf{x} = \alpha\mathbf{b}$, $\alpha > 0$, мы не меняем решение, но множители Лагранжа масштабируются $\boldsymbol{\lambda} \rightarrow \alpha^{-1}\boldsymbol{\lambda}$.

```
    n_inds = sort!(setdiff(1 : n, B))
    AB, AV = A[:, b_inds], A[:, n_inds]
    xB = AB \ b
    cB = c[b_inds]
    λ = AB' \ cB
    cV = c[n_inds]
    μV = cV - AV' * λ

    q, p, xq', D = 0, 0, Inf, Inf
    for i in 1 : length(μV)
        if μV[i] < 0
           pi, xi' = edge_transition(LP, B, i)
           if μV[i] * xi' < Δ
              q, p, xq', Δ = i, pi, xi', μV[i] * xi'
           end
        end
    end

    if q == 0
      return (B, true) # оптимальная точка найдена
    end

    if isinf(xq')
       error("unbounded")
    end

    j = findfirst(isequal(b_inds[p]), B)
    B[j] = n_inds[q]       # перестановка индексов
    return (B, false)      # новая вершина, но не оптимальная
end
```

Пример 11.7. Решение задачи линейного программирования с помощью симплекс-метода

Рассмотрим задачу линейного программирования с ограничениями в виде равенства:

$$\mathbf{A}=\begin{bmatrix}1 & 1 & 1 & 0\\ -4 & 2 & 0 & 1\end{bmatrix},\quad \mathbf{b}=\begin{bmatrix}9\\ 2\end{bmatrix},\quad \mathbf{c}=\begin{bmatrix}3\\ -1\\ 0\\ 0\end{bmatrix},$$

а начальная вершина определена множеством $\mathcal{B} = \{3, 4\}$. Убедившись, что множество $\mathcal{B}$ определяет допустимую вершину, мы можем начать итерацию симплекс-метода.

Извлечем точку $\mathbf{x}_{\mathcal{B}}$:

$$\mathbf{x}_{\mathcal{B}} = \mathbf{A}_{\mathcal{B}}^{-1}\mathbf{b} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 9 \\ 2 \end{bmatrix} = \begin{bmatrix} 9 \\ 2 \end{bmatrix}.$$

Затем вычислим $\boldsymbol{\lambda}$:

$$\boldsymbol{\lambda} = \mathbf{A}_{\mathcal{B}}^{-\mathrm{T}}\mathbf{c}_{\mathcal{B}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^{-\mathrm{T}} \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \mathbf{0}$$

и $\boldsymbol{\mu}_{\mathcal{V}}$:

$$\boldsymbol{\mu}_{\mathcal{V}} = \mathbf{c}_{\mathcal{V}} - \left(\mathbf{A}_{\mathcal{B}}^{-1}\mathbf{A}_{\mathcal{V}}\right)^{\mathrm{T}} \mathbf{c}_{\mathcal{B}} = \begin{bmatrix} 3 \\ -1 \end{bmatrix} - \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 1 \\ -4 & 2 \end{bmatrix} \right)^{\mathrm{T}} \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ -1 \end{bmatrix}.$$

Выясняется, что $\boldsymbol{\mu}_{\mathcal{V}}$ содержит отрицательные элементы, поэтому текущая вершина $\mathcal{B}$ является неоптимальной. Сделаем ведущим индекс единственного отрицательного элемента, $q = 2$. Переход выполняется от $\mathbf{x}_{\mathcal{B}}$ в направлении $-\mathbf{A}_{\mathcal{B}}^{-1}\mathbf{A}_{\{q\}} = [1, 2]$.

Используя формулу (11.19), мы увеличиваем x'_q, пока новое ограничение не станет активным. В этом случае условие $x'_q = 1$ приводит к тому, что координата x_4 становится равной нулю. Обновим набор базовых индексов до $\mathcal{B} = \{2, 3\}$.

На второй итерации находим:

$$\mathbf{x}_{\mathcal{B}} = \begin{bmatrix} 1 \\ 8 \end{bmatrix}, \quad \boldsymbol{\lambda} = \begin{bmatrix} 0 \\ -1/2 \end{bmatrix}, \boldsymbol{\mu}_{\mathcal{V}} = \begin{bmatrix} 1 \\ 1/2 \end{bmatrix}.$$

Вершина является оптимальной, потому что вектор $\boldsymbol{\mu}_{\mathcal{V}}$ не имеет отрицательных элементов. Таким образом, наш алгоритм заканчивается в вершине $\mathcal{B} = \{2, 3\}$, для которой расчетная точка равна $\mathbf{x}^* = [0, 1, 8, 0]$.

11.2.4. Этап инициализации

Этап оптимизации симплекс-метода реализован в алгоритме 11.4. К сожалению, он требует указания начального разбиения, которое соответствует вершине.

Обычно для этого создается и решается вспомогательная задача линейного программирования с известным разбиением вершин, чтобы получить разбиение вершин для исходной задачи линейного программирования. Эта процедура называется *фазой* 1, а процесс решения получающейся задачи линейного програм-

мирования — *фазой* 2. Эта вспомогательная задача линейного программирования содержит дополнительные переменные $\mathbf{z} \in \mathbb{R}^m$, которые мы стремимся обнулить:[14]

$$\begin{aligned} &\min_{\mathbf{x},\mathbf{z}} && \begin{bmatrix}\mathbf{0}^\mathrm{T} & \mathbf{1}^\mathrm{T}\end{bmatrix}\begin{bmatrix}\mathbf{x}\\ \mathbf{z}\end{bmatrix} \\ &\text{при условии, что} && \begin{bmatrix}\mathbf{A} & \mathbf{Z}\end{bmatrix}\begin{bmatrix}\mathbf{x}\\ \mathbf{z}\end{bmatrix} = \mathbf{b}, \\ & && \begin{bmatrix}\mathbf{x}\\ \mathbf{z}\end{bmatrix} \geq 0, \end{aligned} \tag{11.28}$$

где $\mathbf{Z}$ — диагональная матрица, элементы которой равны

$$Z_{ii} = \begin{cases} +1, \text{ если } b_i \geq 0, \\ -1 \text{ в противном случае}. \end{cases} \tag{11.29}$$

Вспомогательная задача линейного программирования решается при разбиении, определенном множеством $\mathcal{B}$, которое выбирает только значения z. Каждой вершине соответствует точка $\mathbf{x} = \mathbf{0}$, и каждый элемент вектора z является абсолютным значением соответствующего значения вектора $\mathbf{b}$: $z_j = |b_j|$. Можно легко показать, что эта начальная вершина j является допустимой.

Алгоритм 11.4. Решение задачи линейного программирования с учетом вершинного разбиения, определенного множеством `B`, и задачи линейного программирования `LP`

```
function minimize_lp!(B, LP)
    done = false
    while !done
        B, done = step_lp!(B, LP)
    end
    return B
end
```

Решение вспомогательной задачи линейного программирования для получения допустимой вершины демонстрирует пример 11.8.

[14] Значения $\mathbf{z}$ представляют собой величину нарушения равенства $\mathbf{Ax} = \mathbf{b}$. Обнуляя $\mathbf{z}$, мы находим допустимую точку. Если при решении вспомогательной задачи мы не найдем вершину с обнуленным вектором $\mathbf{z}$, можно сделать вывод, что задача неразрешима. Кроме того, не всегда необходимо добавлять все m дополнительных переменных, особенно когда при преобразовании между стандартной формой и формой ограничений в виде равенств вводятся дополнительные переменные нежесткости.

Пример 11.8. Решение вспомогательной задачи линейного программирования для получения допустимой вершины

Рассмотрим линейную программу в форме равенства:

$$\begin{aligned} \min_{x_1,x_2,x_3} \quad & c_1x_1 + c_2x_2 + c_3x_3 \\ \text{при условии, что} \quad & 2x_1 - 1x_2 + 2x_3 = 1, \\ & 5x_1 + 1x_2 - 3x_3 = -2, \\ & x_1, x_2, x_3 \geq 0. \end{aligned}$$

Мы можем определить допустимую вершину, решив задачу:

$$\begin{aligned} \min_{x_1,x_2,x_3,z_1,z_2} \quad & z_1 + z_2 \\ \text{при условии, что} \quad & 2x_1 - 1x_2 + 2x_3 + z_1 = 1, \\ & 5x_1 + 1x_2 - 3x_3 - z_2 = -2, \\ & x_1, x_2, x_3, z_1, z_2 \geq 0, \end{aligned}$$

с начальной вершиной, определенной вершиной $\mathcal{B} = \{4, 5\}$.

Начальная вершина имеет координаты

$$x_{\mathcal{B}}^{(1)} = \mathbf{A}_{\mathcal{B}}^{-1}\mathbf{b}_{\mathcal{B}} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ -2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

и, следовательно, $\mathbf{x}^{(1)} = [0, 0, 0, 1, 2]$. Решением вспомогательной задачи является точка $\mathbf{x}^* \approx [0{,}045; 1{,}713; 1{,}312; 0; 0]$. Таким образом, вершина $[0{,}045; 1{,}713; 1{,}312]$ является допустимой вершиной в исходной задаче.

Разбиение, полученное в результате решения вспомогательной задачи линейного программирования, даст возможную расчетную точку, $\mathbf{Ax} = \mathbf{b}$, потому что вектор $\mathbf{z}$ будет обнулен. Если $\mathbf{z}$ не равен нулю, то исходная задача линейного программирования не имеет решения. Если вектор $\mathbf{z}$ равен нулю, то результирующее разбиение может использоваться в качестве начального разбиения для фазы оптимизации симплекс-метода. Исходная задача должна быть слегка изменена, чтобы включить новые переменные $\mathbf{z}$:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{z}} \quad & \begin{bmatrix} \mathbf{c}^{\mathrm{T}} & \mathbf{0}^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{z} \end{bmatrix} \\ \text{при условии, что} \quad & \begin{bmatrix} \mathbf{A} & \mathbf{I} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{z} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix}, \\ & \begin{bmatrix} \mathbf{x} \\ \mathbf{z} \end{bmatrix} \geq \mathbf{0}. \end{aligned} \tag{11.30}$$

Значения $\mathbf{z}$ должны быть включены. Несмотря на то что их векторные аналоги равны нулю, возможно, некоторые индексы в элементах вектора $\mathbf{z}$ включены в начальное разбиение $\mathcal{B}$. Можно проверить начальное разбиение и включить только необходимые элементы.

Решение $(\mathbf{x}^*, \mathbf{z}^*)$, полученное решением второй задачи линейного программирования, будет иметь $\mathbf{z}^* = \mathbf{0}$. Таким образом, $\mathbf{x}^*$ будет решением исходной задачи линейного программирования.

Реализация полного симплекс-метода приведена в алгоритме 11.5.

Алгоритм 11.5. Симплексный алгоритм для решения линейных программ в виде равенства, когда начальное разбиение неизвестно

```
function minimize_lp(LP)
    A, b, c = LP.A, LP.b, LP.c
    m, n = size(A)
    z = ones(m)
    Z = Matrix(Diagonal([j ≥ 0 ? 1 : -1 for j in b]))

    A′ = hcat(A, Z)
    b′ = b
    c′ = vcat(zeros(n), z)
    LP_init = LinearProgram(A′, b′, c′)
    B = collect(1 : m) .+ n
    minimize_lp!(B, LP_init)

    if any(i-> i > n, B)
        error("infeasible")
    end

    A′′ = [A           Matrix(1.0I, m, m);
           zeros(m, n) Matrix(1.0I, m, m)]
    b′′ = vcat(b, zeros(m))
    c′′ = c′
    LP_opt = LinearProgram(A′′, b′′, c′′)
    minimize_lp!(B, LP_opt)
    return get_vertex(B, LP_opt)[1 : n]
end
```

11.3. Двойственные сертификаты

Предположим, у нас есть подходящее решение, и мы хотим убедиться, что оно оптимально. Проверка оптимальности с использованием *двойственных сертификатов* (алгоритм 11.6) полезна в таких случаях, как отладка кода для решения задачи линейного программирования.

Алгоритм 11.6. Метод проверки, позволяющий убедиться, что расчетная точка `x` задает допустимое решение и дуальная точка `μ` для задачи линейного программирования `LP` в форме равенства оптимальна. Параметр контролирует допуск для ограничения в виде равенства

```
function dual_certificate(LP, x, μ, ε = 1e-6)
    A, b, c = LP.A, LP.b, LP.c
    primal_feasible = all(x .≥ 0) && A * x ≈ b
    dual_feasible = all(A' * μ .≥ c)
    return primal_feasible && dual_feasible &&
           isapprox(c⋅x, b⋅μ, atol = ε)
end
```

Из условий FONC для условной оптимизации следует, что оптимальное решение двойственной задачи d^* является нижней границей оптимального решения прямой задачи p^*. Задачи линейного программирования являются линейными и выпуклыми, и можно показать, что оптимальное значение двойственной задачи также является оптимальным значением прямой задачи. т.е. $d^* = p^*$.

Прямая задача линейного программирования может быть преобразована в двойственную форму следующим образом.[15]

Прямая задача (равенство)

$$\min_{\mathbf{x}} \quad \mathbf{c}^T\mathbf{x}$$

при условии, что $\mathbf{A}\mathbf{x} = \mathbf{b}$,

$$\mathbf{x} \geq 0.$$

Двойственная задача

$$\max_{\boldsymbol{\mu}} \quad \mathbf{b}^T\boldsymbol{\mu}$$

при условии, что $\mathbf{A}^T\boldsymbol{\mu} \geq \mathbf{c}$.

Если прямая задача имеет n переменных и m ограничений в виде равенства, то двойственная задача имеет m переменных и n ограничений. Кроме того, двойственная к двойственной является прямой задачей.

[15] Альтернатива симплекс-методу, *самодвойственный симплекс-метод*, на практике, как правило, быстрее. Для него не требуется, чтобы матрица $\mathbf{A}_\mathcal{B}$ удовлетворяла условию $\mathbf{x}_\mathcal{B} = \mathbf{A}_\mathcal{B}^{-1}\mathbf{b} \geq \mathbf{0}$. Самодвойственный симплекс-метод является модификацией симплекс-метода для двойственной задачи линейного программирования в стандартном виде.

Оптимальность может быть оценена путем проверки трех свойств. Если утверждается, что $(\mathbf{x}^*, \boldsymbol{\mu}^*)$ является оптимальным решением, мы можем быстро проверить утверждение, убедившись, что выполняются все три следующих условия:

1. $\mathbf{x}^*$ допустимо в прямой задаче;
2. $\boldsymbol{\mu}^*$ допустимо в двойственной задаче;
3. $p^* = \mathbf{c}^T\mathbf{x}^* = \mathbf{b}^T\boldsymbol{\mu}^* = d^*$.

Двойственные сертификаты используются в примере 11.9 для проверки решения задачи линейного программирования.

Пример 11.9. Проверка решения с помощью двойственных сертификатов

Рассмотрим задачу линейного программирования в стандартной форме:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & -1 \\ -1 & 2 & 0 \\ 1 & 2 & 3 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ -2 \\ 5 \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}.$$

Мы хотели бы проверить, является ли пара $\mathbf{x}^* = [2, 0, 1]$ и $\boldsymbol{\mu}^* = [1, 0, 0]$ оптимальной. Сначала убедимся, что точка $\mathbf{x}^*$ является допустимой:

$$\mathbf{A}\mathbf{x}^* = [1, -2{,}5] = \mathbf{b}, \quad \mathbf{x}^* \geq 0.$$

Затем проверим, является ли точка $\boldsymbol{\mu}^*$ двойственно допустимой:

$$\mathbf{A}^T\boldsymbol{\mu}^* \approx [1, 1, -1] \geq \mathbf{c}.$$

В заключение проверяем, совпадают ли решения p^* и q^*:

$$p^* = \mathbf{c}^T\mathbf{x}^* = 1 = \mathbf{b}^T\boldsymbol{\mu}^* = d^*.$$

11.4. Резюме

- Задачи линейного программирования — это задачи, состоящие из линейной целевой функции и линейных ограничений.
- Симплекс-метод может эффективно оптимизировать задачи линейного программирования в глобальном масштабе.
- Двойственные сертификаты позволяют нам проверить, оптимальна ли пара решений прямой и двойственной задач.

11.5. Упражнения

Упражнение 11.1. Предположим, вы не знаете ни одного алгоритма оптимизации для решения задачи линейного программирования. Вы решаете проверить все вершины и выяснить, какая из них достигает наименьшего значения для целевой функции. Укажите свободную верхнюю границу количества возможных точек минимума.

Упражнение 11.2. Докажите, что симплекс-метод должен сходиться, если задача линейного программирования в примере 11.1 ограничена снизу.

Упражнение 11.3. Предположим, мы хотим минимизировать $6x_1 + 5x_2$ с учетом ограничения $3x_1 - 2x_2 \geq 5$. Как перевести эту задачу в задачу линейного программирования с ограничениями в виде равенства с той же точкой минимума?

Упражнение 11.4. Предположим, ваш алгоритм оптимизации нашел направление поиска, и вы хотите провести линейный поиск. Однако вы знаете, что существует линейное ограничение $\mathbf{w}^T\mathbf{x} \geq 0$. Как бы вы изменили линейный поиск, чтобы учесть это ограничение? Вы можете предположить, что ваша текущая проектная точка является допустимой.

Упражнение 11.5. Переформулируйте задачу линейного программирования

$$\begin{aligned} &\min_{\mathbf{x}} && \mathbf{c}^T\mathbf{x} \\ &\text{при условии, что} && \mathbf{A}\mathbf{x} \geq 0 \end{aligned} \tag{11.31}$$

в задачу безусловной оптимизации со штрафом в виде логарифмического барьера.

Глава 12

Многокритериальная оптимизация

В предыдущих главах были разработаны методы оптимизации однокритериальных функций, но эта глава посвящена *многокритериальной*, или *векторной, оптимизации*, при которой необходимо выполнять оптимизацию по нескольким критериям одновременно. Инженерное дело часто является компромиссом между стоимостью, производительностью и временем выхода на рынок, и зачастую сложно для столь разных целей расставить приоритеты. Мы обсудим различные методы преобразования векторнозначных целевых функций в скалярнозначные целевые функции, позволяющие использовать алгоритмы, рассмотренные в предыдущих главах, для достижения оптимального результата. Кроме того, будут рассмотрены алгоритмы для определения набора расчетных точек, которые представляют собой наилучший компромисс между целями, без необходимости установки конкретных приоритетов. Эти расчетные точки могут быть затем представлены экспертам для выбора оптимального проекта.[1]

12.1. Оптимальность по Парето

При обсуждении задач, имеющих несколько критериев, полезно использовать понятие *оптимальности по Парето*. Проект является оптимальным по Парето, если невозможно улучшить одну целевую функцию, не ухудшая хотя бы еще одну целевую функцию. При многокритериальной оптимизации проекта мы, как правило, можем сосредоточить свои усилия на проектах, которые являются оптимальными по Парето, без необходимости искать определенный компромисс между критериями. В этом разделе представлены некоторые определения и концепции, которые полезны при обсуждении подходов к определению оптимальных по Парето проектов.

[1] Дополнительные методы рассматриваются в [100]. Учебник [104] целиком посвящен многокритериальной оптимизации.

12.1.1. Доминирование

При однокритериальной оптимизации две расчетные точки $\mathbf{x}$ и $\mathbf{x}'$ могут быть ранжированы объективно на основе соответствующих значений скалярных функций. Точка $\mathbf{x}'$ лучше, если $f(\mathbf{x}')$ меньше, чем $f(\mathbf{x})$.

При многокритериальной оптимизации целевая функция создает m-мерный вектор значений, которые вычисляются в расчетной точке $\mathbf{x}$. Различные измерения соответствуют различным целям, которые иногда также называют *метриками* или *критериями*. Мы можем объективно оценить две расчетные точки $\mathbf{x}$ и $\mathbf{x}'$, только когда одна из них лучше по крайней мере по одному критерию и не хуже по любому другому. Говорят, что $\mathbf{x}$ *доминирует* над $\mathbf{x}'$, если и только если

$$f_i(\mathbf{x}) \leq f_i(\mathbf{x}') \text{ для всех } i \text{ в } \{1, ..., m\} \text{ и}$$
$$f_i(\mathbf{x}) < f_i(\mathbf{x}') \text{ для некоторого } i. \qquad (12.1)$$

Это сравнение компактно реализовано в алгоритме 12.1.

Алгоритм 12.1. Метод, позволяющий проверить, доминирует ли $\mathbf{x}$ над $\mathbf{x}'$, где $\mathbf{y}$ — вектор целевых значений для $\mathbf{f(x)}$ и $\mathbf{y}'$ — вектор целевых значений для $\mathbf{f(x')}$

```
dominates(y, y') = all(y .≤ y') && any(y .< y')
```

На рис. 12.1 показано, что в нескольких измерениях существуют области с неоднозначностью доминирования. Эта двусмысленность возникает всякий раз, когда $\mathbf{x}$ лучше по одним критериям, а $\mathbf{x}'$ — по другим. Существует несколько методов разрешения этих неоднозначностей.

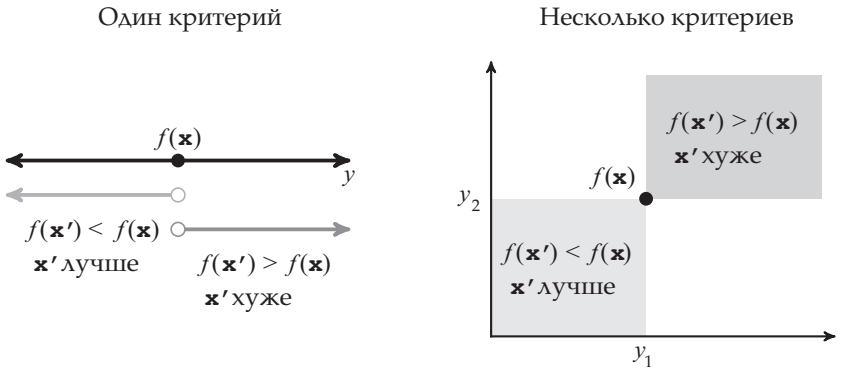


Рис. 12.1. Расчетные точки в однокритериальной оптимизации можно объективно упорядочить, но в многокритериальной оптимизации это возможно только в некоторых случаях

12.1.2. Граница Парето

В математике *образ* множества входных данных относительно некоторой функции представляет собой набор всех возможных значений этой функции на элементах данного множества. Обозначим образ $\mathcal{X}$ при отображении `f` как $\mathcal{Y}$ и будем называть $\mathcal{Y}$ *критериальным пространством*. На рис. 12.2 показаны примеры критериальных пространств для задач с одним и несколькими критериями. Как показано на рисунке, критериальное пространство в однокритериальной оптимизации является одномерным. Все глобальные оптимумы имеют одно целевое значение функции, y^*. При многокритериальной оптимизации критериальное пространство m-мерно, где m — количество критериев. Как правило, не существует глобально лучшего значения целевой функции, потому что может возникнуть неоднозначность, если не определены компромиссы между критериями.

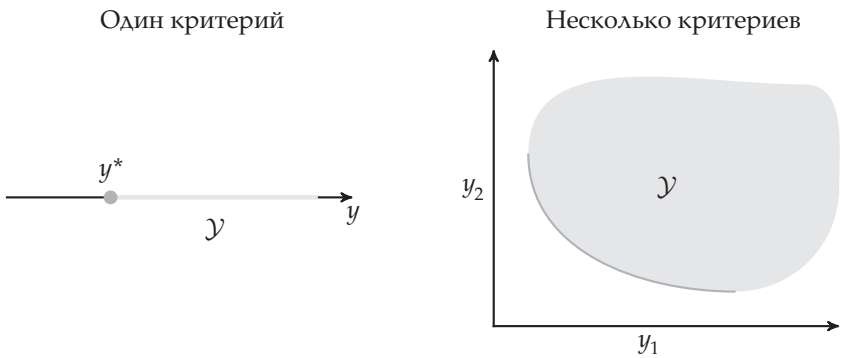


Рис. 12.2. Критериальное пространство — это совокупность всех целевых значений, полученных с помощью возможных расчетных точек. У хорошо поставленных задач есть критериальные пространства, ограниченные снизу, но они не должны быть ограничены сверху. Граница Парето выделена темно-синим цветом

В многокритериальной оптимизации мы можем определить понятие *оптимальности по Парето*. Расчетная точка $\mathbf{x}$ является оптимальной по Парето, когда над ней не доминирует ни одна точка. Иначе говоря, точка $\mathbf{x} \in \mathcal{X}$ является оптимальной по Парето, если не существует $\mathbf{x}' \in \mathcal{X}$, которая доминировала бы над $\mathbf{x}$. Множество точек, оптимальных по Парето, образует *границу Парето*. Граница Парето позволяет лицам, принимающим решения, оценивать компромиссы, как показано в примере 12.1. В двумерном пространстве граница Парето также называется *кривой Парето*.

Все точки, оптимальные по Парето, лежат на границе критериального пространства. Некоторые методы многокритериальной оптимизации также находят *точки, слабо оптимальные по Парето*. Тогда как оптимальными по Парето точками являются такие, что ни одна другая точка не улучшает хотя бы один критерий, слабыми по Парето точками являются точки, в которых ни одна другая точка

не улучшает все критерии (рис. 12.3). Иначе говоря, точка $\mathbf{x} \in \mathcal{X}$ является слабо оптимальной по Парето, если не существует точки $\mathbf{x}' \in \mathcal{X}$ такой, что $\mathbf{f}(\mathbf{x}') < \mathbf{f}(\mathbf{x})$. Точки, оптимальные по Парето, также являются слабо оптимальными по Парето. В то же время точки, слабо оптимальные по Парето, не обязательно являются оптимальными по Парето.

Несколько методов, обсуждаемых ниже, используют другой особый момент. Мы определяем *точку утопии* как точку в критериальном пространстве, состоящую из покомпонентных оптимумов:

$$y_i^{\text{utopia}} = \min_{\mathbf{x} \in \mathcal{X}} f_i(\mathbf{x}). \tag{12.2}$$

Точка утопии часто недостижима; оптимизация одного компонента обычно требует компромисса с другим компонентом.

12.1.3. Создание границы Парето

Существует несколько методов создания границ Парето. Наивным подходом является выборка расчетных точек во всем пространстве проектирования, а затем идентификация недоминированных точек (алгоритм 12.2). Такой подход, как правило, является расточительным, что приводит к появлению многочисленных доминирующих расчетных точек, как показано на рис. 12.4. Кроме того, этот подход не гарантирует гладкой, или правильной, границы Парето. В оставшейся части этой главы обсуждаются более эффективные способы создания границ Парето.

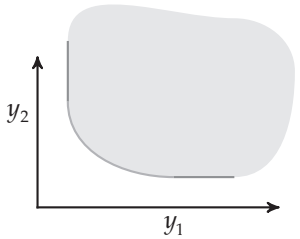


Рис. 12.3. Слабо оптимальные по Парето точки, показанные красным, нельзя улучшить одновременно по всем целевым функциям

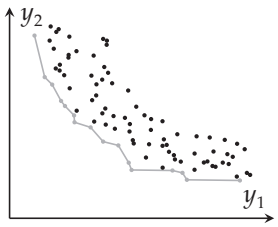


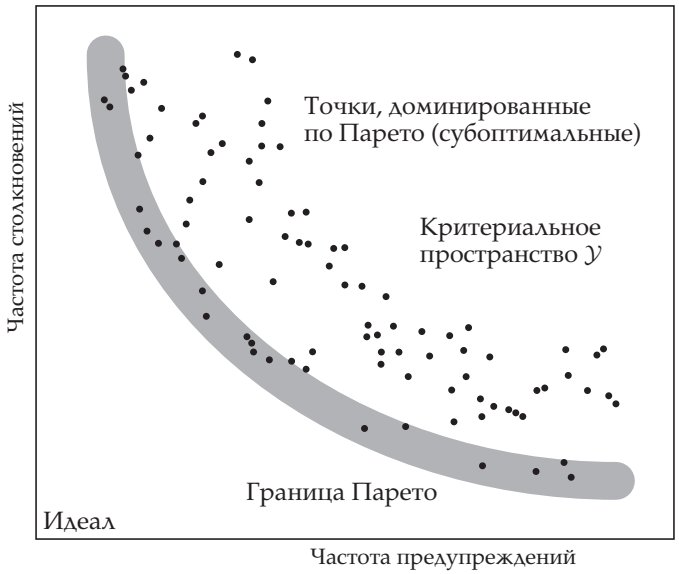
Рис. 12.4. Генерация границ Парето с наивно разбросанными точками проста, но неэффективна и приблизительна

Пример 12.1. Приблизительная граница Парето, полученная в результате оценки множества различных расчетных точек для системы предотвращения столкновений воздушных судов

При создании системы предотвращения столкновений для воздушных судов необходимо минимизировать как частоту столкновений, так и частоту преду-

преждений. Хотя большее количество предупреждений может привести к предотвращению новых коллизий, если последуют эти предупреждения, слишком большое количество предупреждений может подорвать доверие пилотов к системе и снизить ее надежность. Следовательно, разработчикам системы придется балансировать между излишними предупреждениями и риском столкновения.

Изменяя параметры конструкции системы предотвращения столкновений, мы можем получить множество различных систем предотвращения столкновений, но, как показано на рисунке, некоторые из них будут лучше, чем другие. Границу Парето можно извлечь, чтобы помочь экспертам в области и регуляторам понять влияние, которое компромиссы между критериями окажут на оптимизированную систему.



Алгоритм 12.2. Метод генерации границы Парето с использованием произвольно выбранных расчетных точек **xs** и их многокритериальных значений **ys**, возвращающий расчетные точки, оптимальные по Парето, и их целевые значения

```
function naive_pareto(xs,ys)
    pareto_xs, pareto_ys = similar(xs, 0), similar(ys, 0)
    for(x,y) in zip(xs, ys)
      if !any(dominates(y', y) for y' in ys)
          push!(pareto_xs, x)
          push!(pareto_ys, y)
      end
```

```
    end
    return (pareto_xs, pareto_ys)
end
```

12.2. Методы ограничения

Ограничения могут быть использованы для вырезания участков границы Парето и получения единственной оптимальной точки в пространстве критериев. Ограничения могут быть либо предоставлены постановщиком задачи, либо автоматически получены на основе упорядочения целей.

12.2.1. Метод ограничений

Метод ограничений ограничивает все, кроме одной из целей. Здесь мы выбираем f_1 без потери общности:

$$\begin{aligned} \min_{\mathbf{x}} \quad & f_1(\mathbf{x}) \\ \text{при условии, что} \quad & f_2(\mathbf{x}) \le c_2, \\ & f_3(\mathbf{x}) \le c_3, \\ & \ldots \\ & f_m(\mathbf{x}) \le c_m, \\ & \mathbf{x} \in \mathcal{X}. \end{aligned} \tag{12.3}$$

При заданном векторе `c` метод ограничений создает единственную оптимальную точку в критериальном пространстве, при условии, что ограничения являются допустимыми. Метод ограничений может быть использован для создания границ Парето путем изменения `c`, как показано на рис. 12.5.

12.2.2. Лексикографический метод

Лексикографический метод ранжирует цели в порядке важности. Для достижения целей в порядке важности решается серия задач однокритериальной оптимизации. Каждая задача оптимизации включает в себя ограничения для сохранения оптимальности по отношению к ранее оптимизированным целям, как показано на рис. 12.6.

Итерации всегда допустимы, потому что минимальная точка из предыдущей оптимизации всегда является допустимой. Ограничения также могут быть заменены равенствами, но зачастую легче реализовать неравенства. Кроме того, если используемый метод оптимизации не является оптимальным, то при последующей оптимизации могут возникнуть лучшие решения, которые в противном слу-

чае были бы отклонены. Лексикографический метод чувствителен к упорядоченности целевых функций.

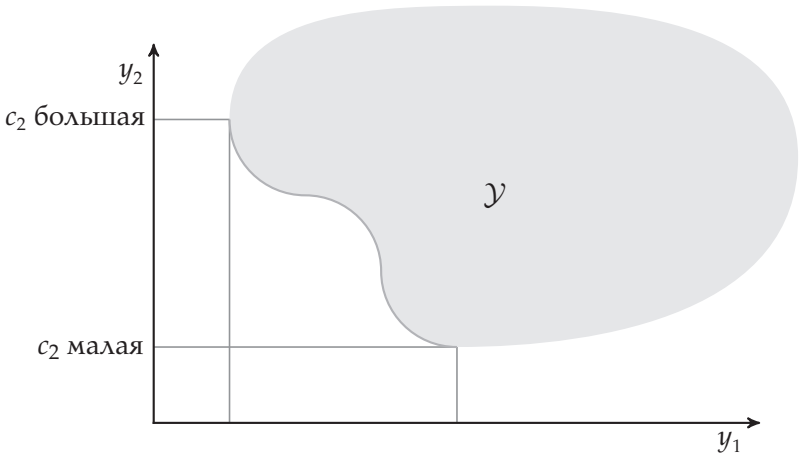


Рис. 12.5. Метод ограничения для генерации границы Парето. Этот метод может идентифицировать точки в вогнутой области границы Парето

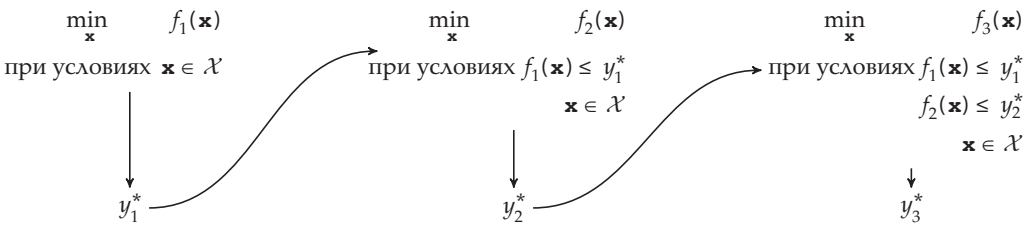


Рис. 12.6. Лексикографический метод для задачи оптимизации с тремя целями

12.3. Весовые методы

Конструктор может иногда определять предпочтения между целями и кодировать эти предпочтения как вектор весов. В тех случаях, когда выбор весов не очевиден, мы можем создать границу Парето, переместившись в пространство весов. В этом разделе также обсуждаются различные альтернативные методы преобразования многокритериальных функций в однокритериальные.

12.3.1. Метод взвешенной суммы

Метод взвешенной суммы (алгоритм 12.3) использует вектор весов $\mathbf{w}$ для преобразования $\mathbf{f}$ в однокритериальную функцию f [167]:

$$f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{f}(\mathbf{x}), \tag{12.4}$$

где веса неотрицательны, а их сумма равна единице. Веса могут быть интерпретированы как затраты, связанные с каждым критерием. Границу Парето можно извлечь, изменив **w** и решив соответствующую задачу оптимизации с критерием, заданным уравнением (12.4). В двумерном пространстве мы меняем w_1 от нуля до единицы, устанавливая $w_2 = 1 - w_1$. Этот подход иллюстрируется на рис. 12.7.

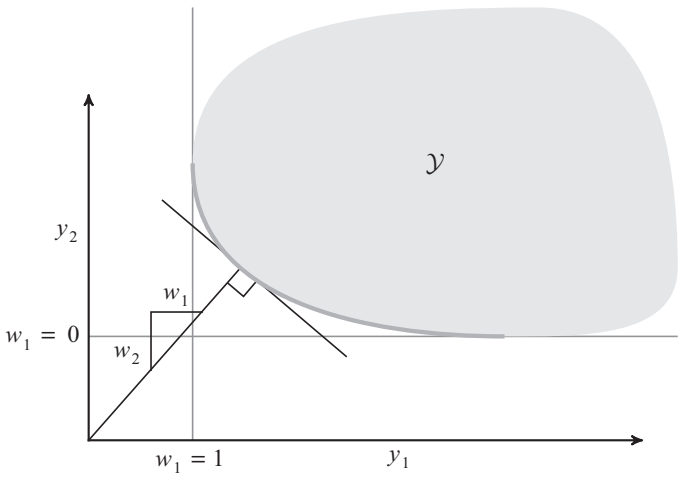


Рис. 12.7. Метод взвешенной суммы, используемый для генерации границы Парето. Изменение весов позволяет нам проследить границу Парето.

В отличие от метода ограничений, метод взвешенной суммы не может найти точки в невыпуклых областях границы Парето, как показано на рис. 12.8.

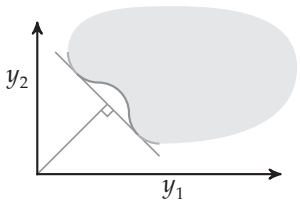


Рис. 12.8. Красные точки являются оптимальными по Парето, но не могут быть найдены с использованием метода взвешенной суммы

Данный набор весов образует линейную целевую функцию с параллельными линиями уровня, идущими от начала координат. Если допустимое множество отклоняется от начала координат, оно будет иметь другие оптимальные по Парето точки на границе, которые не могут быть восстановлены путем минимизации уравнения (12.4).

Алгоритм 12.3. Метод взвешенной суммы для генерации границы Парето, который принимает целевые функции `f1` и `f2` и количество точек Парето `npts`

```
function weight_pareto(f1, f2, npts)
    return[
       optimize(x -> w1 * f1(x) + (1 - w1) * f2(x))
       for w1 in range(0, stop = 1, length = npts)
    ]
end
```

12.3.2. Целевое программирование

Целевое программирование[2] — это метод преобразования многокритериальной функции в однокритериальную функцию путем минимизации L_p-нормы разности между $\mathbf{f}(\mathbf{x})$ и целевой точкой:

$$\min_{\mathbf{x}\in\mathcal{X}} \left\| \mathbf{f}(\mathbf{x}) - \mathbf{y}^{\text{goal}} \right\|_p, \tag{12.5}$$

где целью является, как правило, точка утопии. Приведенное выше уравнение не включает вектор весов, но другие методы, обсуждаемые в этой главе, можно рассматривать как обобщения целевого программирования. Этот подход иллюстрируется на рис. 12.9.

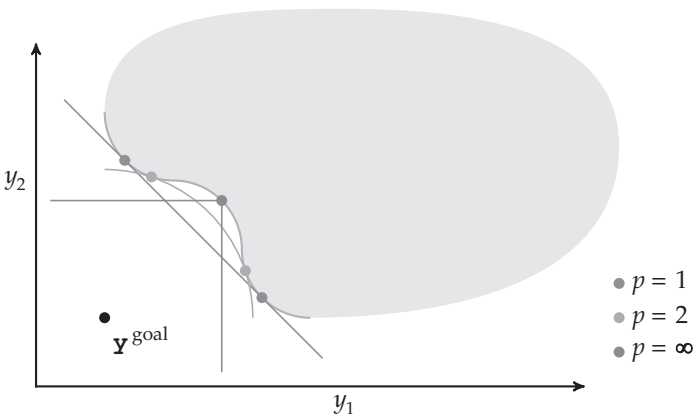


Рис. 12.9. Решения для программирования целей при изменении значений p

12.3.3. Метод взвешенной экспоненциальной суммы

Метод взвешенной экспоненциальной суммы объединяет целевое программирование и метод взвешенной суммы [165]:

[2] Целевое программирование обычно относится к случаю $p = 1$. Обзор см. в [75].

$$f(\mathbf{x}) = \sum_{i=1}^{m} w_i \left(f_i(\mathbf{x}) - y_i^{\text{goal}} \right)^p, \tag{12.6}$$

где $\mathbf{w}$ — вектор положительных весов, сумма которых равна единице, а $p \geq 1$ — показатель степени, аналогичный используемому в нормах L_p. Как и прежде, нулевые веса могут привести к слабо оптимальным по Парето точкам.

Метод взвешенной экспоненциальной суммы взвешивает каждый компонент расстояния между точкой решения и точкой цели в критериальном пространстве. Увеличение p приводит к увеличению относительного штрафа наибольшего отклонения координат между $\mathbf{f}(\mathbf{x})$ и точкой цели. В то время как части оптимального по Парето набора можно получить путем непрерывного изменения p, мы не гарантируем получение полной границы Парето, и, как правило, предпочтительно изменять $\mathbf{w}$ с использованием постоянного значения p.

12.3.4. Взвешенный метод min-max

Использование более высоких значений p в методе взвешенной экспоненциальной суммы приводит к лучшему охвату границы Парето, потому что контуры расстояний могут входить в невыпуклые области границы Парето. *Взвешенный минимаксный метод*, также называемый *взвешенным методом Чебышева*, является пределом при приближении p к бесконечности:[3]

$$f(\mathbf{x}) = \max_i \left[w_i \left(f_i(\mathbf{x}) - y_i^{\text{goal}} \right) \right]. \tag{12.7}$$

Взвешенный минимаксный метод может обеспечить полное минимизацию по парето-оптимальному набору, проверяя веса, но также находит слабо оптимальные по Парето точки. Метод может быть дополнен для получения только границы Парето

$$f(\mathbf{x}) = \max_i \left[w_i \left(f_i(\mathbf{x}) - y_i^{\text{goal}} \right) \right] + \rho \mathbf{f}(\mathbf{x})^{\mathrm{T}} \mathbf{y}^{\text{goal}}, \tag{12.8}$$

где ρ — небольшой положительный скаляр со значениями обычно от 0,0001 до 0,01. Добавленный член гарантирует, что все компоненты вектора $\mathbf{y}^{\text{goal}}$ будут положительными. Это может быть достигнуто путем смещения целевой функции. По определению, $\mathbf{f}(\mathbf{x}) \geq \mathbf{y}^{\text{goal}}$ для всех $\mathbf{x}$. В любой точке, слабо оптимальной по Парето, значение $\mathbf{f}(\mathbf{x})^{\mathrm{T}} \mathbf{y}^{\text{goal}}$ больше, чем расстояние от точки, сильно оптимальной точки по Парето, до $\mathbf{y}^{\text{goal}}$.

---

[3] Максимизацию можно исключить с помощью включения дополнительного параметра λ:

$$\min_{\mathbf{x},\lambda} \lambda$$

при условии, что $\mathbf{x} \in \mathcal{X}$,

$$\mathbf{w} \odot \left(\mathrm{f}(\mathbf{x}) - \mathbf{y}^{\text{goal}} \right) - \lambda \mathbf{1} \leq \mathbf{0}.$$

12.3.5. Экспоненциально-взвешенный критерий

Экспоненциально-взвешенный критерий [7] компенсирует неспособность метода взвешенной суммы находить точки на невыпуклых участках границы Парето. Он строит скалярную целевую функцию в соответствии с формулой

$$f(\mathbf{x}) = \sum_{i=1}^{m} \left(e^{pw_i} - 1 \right) e^{pf_i(\mathbf{x})}. \tag{12.9}$$

Каждая цель индивидуально трансформируется и пересматривается. Высокие значения p могут привести к переполнению.

12.4. Популяционные многокритериальные методы

Популяционные методы также применялись для многокритериальной оптимизации (см. главу 9). Мы можем адаптировать стандартные алгоритмы, чтобы стимулировать распространение популяций через границу Парето.

12.4.1. Подпопуляции

Популяционные методы могут разделить свое внимание на несколько потенциально конкурирующих целей. Популяция может быть разделена на подгруппы, где каждая подгруппа оптимизирована с учетом различных целей. Например, традиционный генетический алгоритм можно модифицировать, чтобы сместить выбор особей для рекомбинации в сторону наиболее приспособленных особей в каждой подгруппе. Отобранные особи могут сформировать потомство с особями из разных подпопуляций.

Одной из первых адаптаций популяционных методов к многокритериальной оптимизации является *векторнозначный генетический алгоритм* (алгоритм 12.4) [138]. На рис. 12.10 показано, как субпопуляции используются в векторнозначном генетическом алгоритме для поддержания разнообразия по нескольким целям. Выполнение векторнозначного генетического метода показано на рис. 12.11.

Алгоритм 12.4. Генетический векторнозначный алгоритм, который принимает векторную целевую функцию `f`, начальную популяцию, число итераций `k_max`, методы `SelectionMethod S`, `CrossoverMethod C` и `MutationMethod M` и возвращает результирующую популяцию

```
function vector_evaluated_genetic_algorithm(f, population,
    k_max, S, C, M)
    m = length(f(population[1]))
    m_pop = length(population)
    m_subpop = m_pop ÷ m
    for k in 1 : k_max
```

```
        ys = f.(population)
        parents = select(S, [y[1] for y in ys])[1 : m_subpop]
        for i in 2 : m
            subpop = select(S,[y[i] for y in ys])[1 : m_subpop]
            append!(parents, subpop)
        end

        p = randperm(2m_pop)
        p_ind=i->parents[mod(p[i] - 1, m_pop) + 1][(p[i] - 1) ÷ m_pop + 1]
        parents = [[p_ind(i), p_ind(i + 1)] for i in 1 : 2 : 2m_pop]
        children = [crossover(C, population[p[1]], population[p[2]])
                    for p in parents]
        population = [mutate(M, c) for c in children]
    end
    return population
end
```

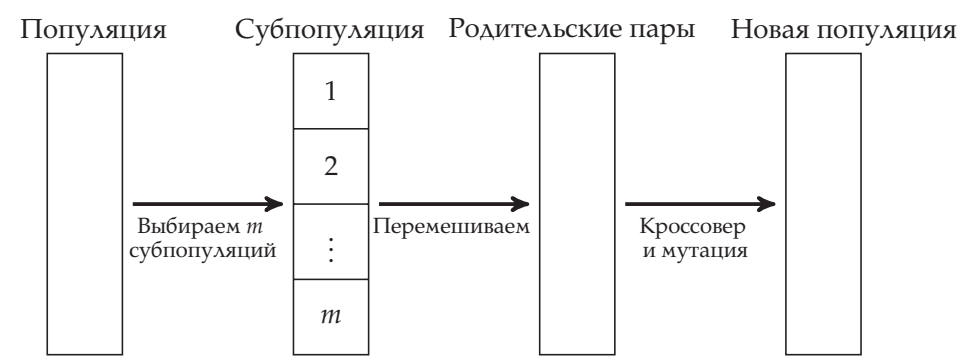


Рис. 12.10. Применение подпопуляций в векторнозначном генетическом алгоритме

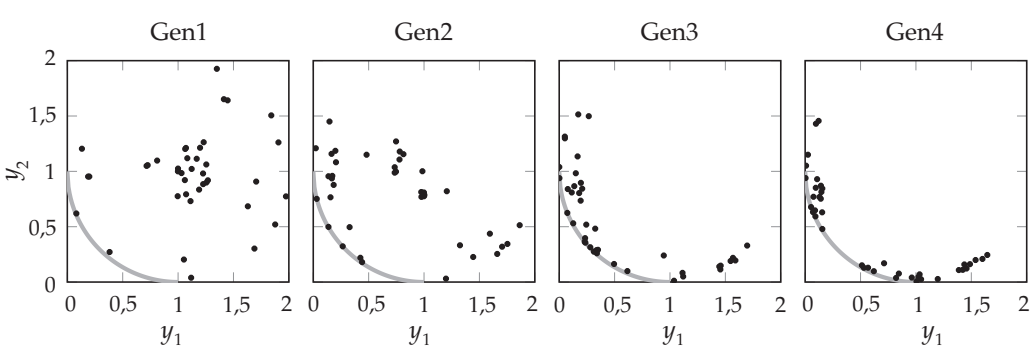


Рис. 12.11. Векторнозначный генетический алгоритм, примененный к функции круга, определенной в приложении Б.8. Граница Парето показана синим цветом

12.4.2. Ранги недоминирования

Можно вычислить наивные границы Парето, используя особей в популяции. Расчетная точка, лежащая на приближенной границе Парето, считается лучшей, чем значение в глубине критериального пространства. Мы можем использовать ранжирование по степени недоминирования (алгоритм 12.5) для ранжирования особей по следующим уровням [36].

Уровень 1. Особи в популяции, над которыми никто не доминирует.

Уровень 2. Особи в популяции, над которыми никто не доминирует, за исключением особей уровня 1.

Уровень 3. Особи в популяции, над которыми никто не доминирует, за исключением особей уровня 1 или 2.

...

Уровень k. Особи в популяции, над которыми никто не доминирует, за исключением особей уровней от 1 до $k - 1$.

Уровень 1 получается путем применения алгоритма 12.2 ко всей популяции. Последующие уровни генерируются путем удаления всех предыдущих уровней из совокупности и последующего применения алгоритма 12.2. Этот процесс повторяется, пока все особи не будут ранжированы. Значение целевой функции особи пропорционально его рангу.

Ранги недоминирования на примере популяции показаны на рис. 12.12.

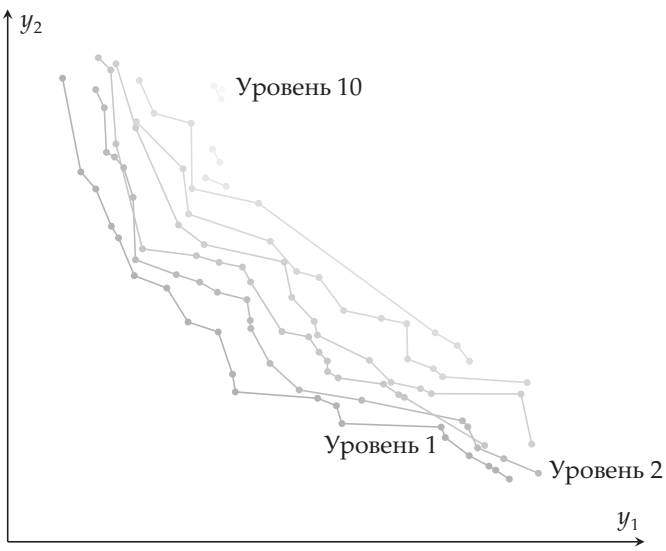


Рис. 12.12. Ранги недоминирования в популяции. Более темные уровни соответствуют более низкому (а значит, лучшему) рангу

Алгоритм 12.5. Функция для вычисления рангов недоминирования массива многокритериальной функции

```
function get_non_domination_levels(ys)
   L, m = 0, length(ys)
   levels = zeros(Int, m)
   while minimum(levels) == 0
       L += 1
       for (i, y) in enumerate(ys)
          if levels[i] == 0 &&
             !any((levels[i] == 0 || levels[i] == L) &&
                   dominates(ys[i], y) for i in 1 : m)
             levels[i] = L
          end
       end
   end
   return levels
end
```

12.4.3. Фильтры Парето

Методы популяций могут быть дополнены *фильтром Парето*, создающим популяцию, которая аппроксимирует границу Парето [72]. Фильтр, как правило, обновляется с каждым новым поколением (алгоритм 12.7). В фильтр добавляются особи, над которыми никто не доминирует. Все доминируемые точки из фильтра удаляются. Особи из фильтра Парето могут быть включены в популяцию, тем самым уменьшая вероятность того, что части границы Парето будут потеряны между поколениями.

Фильтр часто имеет максимальную емкость.[4] Фильтры с избыточной емкостью можно уменьшить, найдя ближайшую пару расчетных точек в критериальном пространстве и удалив одну особь из этой пары. Этот метод сокращения реализован в алгоритме 12.6. Фильтр Парето, полученный с использованием генетического алгоритма, показан на рис. 12.13.

Алгоритм 12.6. Метод **discard_closest_pair!** используется для удаления одной особи из фильтра, который превышает емкость. Метод принимает список расчетных точек **xs**, принадлежащих фильтру и соответствующие значения целевой функции **ys**

```
function discard_closest_pair!(xs, ys)
    index, min_dist = 0, Inf
```

[4] Как правило, размер популяции.

```
    for(i, y) in enumerate(ys)
       for(j, y′) in enumerate(ys[i + 1 : end])
           dist = norm(y - y′)
           if dist < min_dist
              index, min_dist = rand([i, j]), dist
           end
       end
    end
    deleteat!(xs, index)
    deleteat!(ys, index)
    return (xs, ys)
end
```

Алгоритм 12.7. Метод обновления фильтра Парето с расчетными точками **filter_xs**, соответствующими значениями целевой функции **filter_ys**, популяцией с расчетными точками **xs**, значениями целевой функции **ys** и емкостью фильтра **capacity**, которая по умолчанию равна размеру популяции

```
function update_pareto_filter!(filter_xs, filter_ys, xs, ys;
          capacity = length(xs),
         )
         for (x,y) in zip(xs, ys)
              if !any(dominates(y′, y) for y′ in filter_ys)
                 push!(filter_xs, x)
                 push!(filter_ys, y)
              end
         end
         filter_xs, filter_ys = naive_pareto(filter_xs, filter_ys)
         while length(filter_xs) > capacity
              discard_closest_pair!(filter_xs, filter_ys)
         end
         return (filter_xs, filter_ys)
end
```

12.4.4. Нишевые методы

Термин *ниша* относится к сосредоточенному кластеру точек, обычно в критериальном пространстве, как показано на рис. 12.14. Популяционные методы могут сходиться на нескольких нишах, что ограничивает их распространение на

границе Парето. *Нишевые методы* помогают стимулировать равномерное распределение точек.

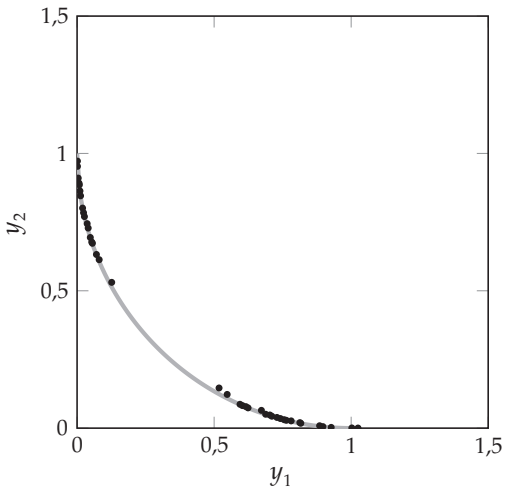


Рис. 12.13. Фильтр Парето, используемый в генетическом алгоритме на рис. 12.11 для аппроксимации границы Парето

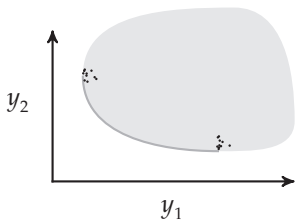


Рис. 12.14. Две четкие ниши для популяции в двумерном критериальном пространстве

При *совместном использовании приспособленности*[5], показанном на рис. 12.15, целевые значения особи штрафуются с коэффициентом, равным количеству других точек в пределах указанного расстояния в критериальном пространстве [57]. Эта схема заставляет все точки в локальной области совместно использовать приспособленность других точек в пределах локальной области. Совместное использование приспособленности может использоваться вместе с рангом недоминирования и оценкой субпопуляции.

Совместное использование класса эквивалентности может применяться к ранжированию недоминирования. При сравнении двух особей сначала на основе рейтинга недоминирования определяется более подходящая особь. Если они равны, то лучшая особь — та, у которой на указанном расстоянии в критериальном пространстве находится наименьшее количество особей.

Другой нишевый метод был предложен для генетических алгоритмов, в которых родители, выбранные для кроссовера, не могут быть расположены слишком близко друг к другу в критериальном пространстве. Также рекомендуется выбирать только недоминированных особей [109].

[5] Приспособленность обратно пропорциональна минимизируемой целевой функции.

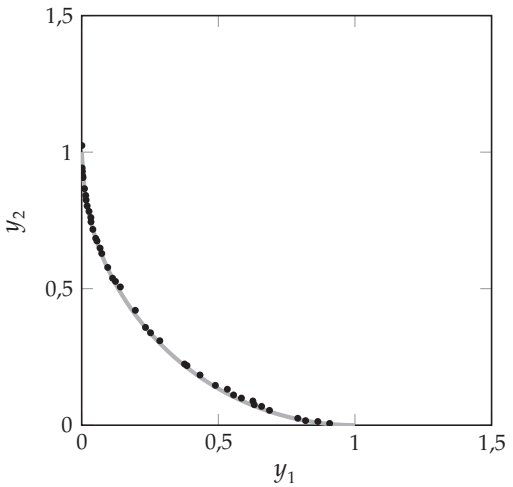


Рис. 12.15. Результаты применения совместного использования приспособленности к фильтру Парето на рис. 12.13, улучшающие его охват

12.5. Выявление предпочтений

Выявление предпочтений включает в себя вывод скалярной целевой функции из предпочтений экспертов относительно компромиссов между целями.[6] Существует много различных способов представления скалярной целевой функции, но в этом разделе основное внимание будет уделено модели взвешенной суммы, где $f(\mathbf{x}) = \mathbf{w}^T\mathbf{f}(\mathbf{x})$. Как только мы определим подходящее значение $\mathbf{w}$, мы можем использовать эту скалярную целевую функцию, чтобы найти оптимальное решение.

12.5.1. Идентификация модели

Общий подход к идентификации вектора весовых коэффициентов $\mathbf{w}$ в нашей модели предпочтений включает в себя обращение к экспертам с просьбой указать их предпочтение между двумя точками $\mathbf{a}$ и $\mathbf{b}$ в критериальном пространстве $\mathcal{Y}$ (рис. 12.16). Каждая из этих точек является результатом оптимизации для точки на границе Парето с использованием соответствующих весовых векторов $\mathbf{w}_\mathbf{a}$ и $\mathbf{w}_\mathbf{b}$. Ответ эксперта — это либо предпочтение $\mathbf{a}$, либо предпочтение $\mathbf{b}$. Существуют и другие схемы извлечения информации о предпочтениях, такие как ранжирование точек в критериальном пространстве, но было показано, что этот бинарный запрос о предпочтениях создает минимальную когнитивную нагрузку на эксперта [27].

---

[6] В этой главе дается обзор небайесовских подходов к выявлению предпочтений. Относительно байесовских подходов см. [63] и [96].

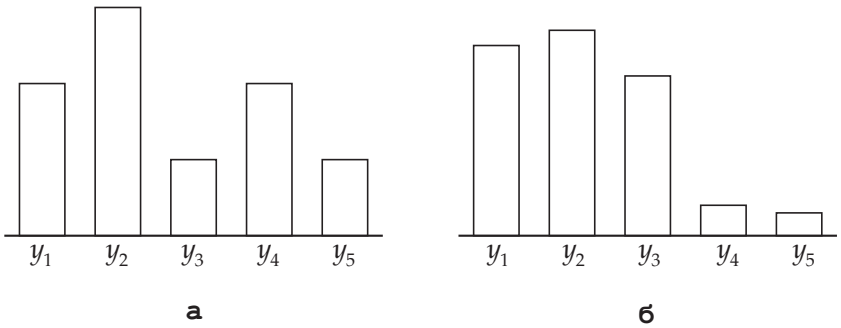


Рис. 12.16. Схемы выявления предпочтений часто включают в себя выяснение у экспертов их предпочтений между двумя точками в критериальном пространстве

Предположим, что результаты опросов экспертов привели к следующему набору пар критериев

$$\{(\mathbf{a}^{(1)}, \mathbf{b}^{(1)}), ..., (\mathbf{a}^{(n)}, \mathbf{b}^{(n)})\}, \tag{12.10}$$

где $\mathbf{a}^{(i)}$ предпочтительнее, чем $\mathbf{b}^{(i)}$, в каждой паре. Для каждого из этих предпочтений весовой вектор должен удовлетворять условию

$$\mathbf{w}^{\mathrm{T}}\mathbf{a}^{(i)} < \mathbf{w}^{\mathrm{T}}\mathbf{b}^{(i)} \Rightarrow (\mathbf{a}^{(i)} - \mathbf{b}^{(i)})^{\mathrm{T}}\mathbf{w} < 0. \tag{12.11}$$

Чтобы соответствовать данным, весовой вектор должен удовлетворять условию

$$\begin{cases} \left(\mathbf{a}^{(i)} - \mathbf{b}^{(i)}\right)^{\mathrm{T}} \mathbf{w} < 0 \text{ для всех } i \text{ в } \{1, ..., n\}, \\ \mathbf{1}^{\mathrm{T}}\mathbf{w} = 1 \\ \mathbf{w} \geq \mathbf{0}. \end{cases} \tag{12.12}$$

Множество различных весовых векторов могут потенциально удовлетворять вышеуказанному уравнению. Один из подходов состоит в том, чтобы выбрать вектор $\mathbf{w}$, который лучше всего отделяет $\mathbf{w}^{\mathrm{T}}\mathbf{a}^{(i)}$ от $\mathbf{w}^{\mathrm{T}}\mathbf{b}^{(i)}$:

$$\begin{aligned} &\min_{\mathbf{w}} \quad \sum_{i=1}^{n}\left(\mathbf{a}^{(i)} - \mathbf{b}^{(i)}\right)^{\mathrm{T}} \mathbf{w} \\ &\text{при условии, что } \left(\mathbf{a}^{(i)} - \mathbf{b}^{(i)}\right)^{\mathrm{T}} \mathbf{w} < 0 \text{ для } i \in \{1, ..., n\}, \\ &\qquad\qquad \mathbf{1}^{\mathrm{T}}\mathbf{w} = 1, \qquad \mathbf{w} \geq \mathbf{0}. \end{aligned} \tag{12.13}$$

Часто желательно выбрать следующий весовой вектор так, чтобы он минимизировал расстояние от предыдущего весового вектора. Мы можем изменить целе-

вую функцию в уравнении (12.13) на $\|\mathbf{w} - \mathbf{w}^{(n)}\|_1$, тем самым гарантируя, что новый весовой вектор $\mathbf{w}^{(n+1)}$ будет как можно ближе к текущему.[7]

12.5.2. Выбор парных запросов

Как правило, мы хотим выбрать две точки в критериальном пространстве, чтобы результат запроса был максимально информативным. Существует много разных подходов для определения этих точек, но мы сосредоточимся на методах, которые пытаются уменьшить пространство весов в соответствии с *ответами экспертов*, т.е. информацией о предпочтениях, предоставленной экспертом в области.

Мы будем обозначать множество весов, соответствующих экспертным ответам, как $\mathcal{W}$. Оно определяется линейными ограничениями в уравнении (12.12). Поскольку весовые коэффициенты ограничены между 0 и 1, допустимое множество представляет собой замкнутую область в виде выпуклого многогранника с конечным объемом. Как правило, мы хотим уменьшить объем области $\mathcal{W}$, чтобы она содержала как можно меньше запросов.

Метод Q-Eval [73], показанный на рис. 12.17, является жадной стратегией выявления предпочтений, которая эвристически стремится уменьшить объем $\mathcal{W}$ как можно быстрее с каждой итерацией. Он выбирает запрос, который ближе всего разбивает область $\mathcal{W}$ на две равные части. Метод работает на конечной выборке оптимальных по Парето расчетных точек. Процедура выбора пары запросов состоит из следующих действий.

1. Вычислить *главный аналитический центр* $\mathbf{c}$ в области $\mathcal{W}$, который является точкой, максимизирующей сумму логарифмов расстояний между собой и ближайшей точкой на каждом неизбыточном ограничении в $\mathcal{W}$:

$$\mathbf{c} = \arg\max_{\mathbf{w}\in\mathcal{W}} \sum_{i=1}^{n} \ln\left(\left(\mathbf{b}^{(i)} - \mathbf{a}^{(i)}\right)^{\mathrm{T}} \mathbf{w}\right). \tag{12.14}$$

2. Определить нормальное расстояние от секущей гиперплоскости для каждой пары точек и центра.
3. Упорядочить пары расчетных точек в порядке увеличения расстояния.
4. Для каждой из k гиперплоскостей, ближайших к $\mathbf{c}$, вычислить соотношение объемов двух многогранников, образованных расщеплением области $\mathcal{W}$ вдоль гиперплоскости.
5. Выбрать пару расчетных точек с коэффициентом разделения, ближайшим к единице.

---

[7] Предыдущий весовой вектор может как соответствовать, так и не соответствовать дополнительному ограничению $(\mathbf{a}^{(n)} - \mathbf{b}^{(n)})^{\mathrm{T}}\mathbf{w} < 0$.

Рис. 12.17. Визуализации стратегии жадного поиска Q-Eval. Последовательность фигур показывает начальный набор весов $\mathcal{W}$, соответствующий предыдущим предпочтениям, пару векторов весов и их соответствующую секущую гиперплоскость, а также два многогранника, образованные расщеплением вдоль секущей гиперплоскости

Метод многогранников [24] работает путем аппроксимации $\mathcal{W}$ ограничивающим эллипсоидом с центром в аналитическом центре области $\mathcal{W}$, как показано на рис. 12.18. Запросы предназначены для разбиения ограничивающего эллипсоида на приблизительно равные части и для выбора сечений, которые перпендикулярны самой длинной оси эллипсоида, чтобы уменьшить неопределенность и сбалансировать ширину в каждом измерении.

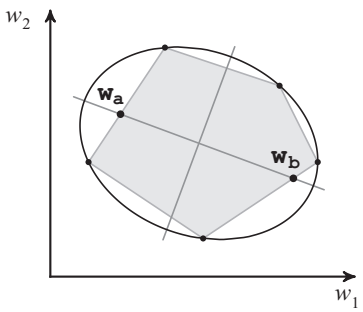


Рис. 12.18. Метод многогранников использует ограничивающий эллипсоид для $\mathcal{W}$

Алгоритм рассматривает все возможные пары из конечной выборки оптимальных по Парето точек проектирования и выбирает запрос, который наиболее равномерно разбивает множество $\mathcal{W}$.

12.5.3. Выбор проекта

В предыдущем разделе обсуждались методы запросов, которые выбирают пары запросов для эффективного сокращения пространства поиска. Завершив выбор запроса, нужно еще выбрать окончательный проект.

Один из таких методов, *улучшение качества решений* [23], основан на идее, что, если нам нужно зафиксировать определенный вес, мы должны использовать тот, для которого целевое значение наихудшего случая является наименьшим:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathcal{X}} \max_{\mathbf{w}\in\mathcal{W}} \mathbf{w}^\mathrm{T}\mathbf{f}(\mathbf{x}). \tag{12.15}$$

Это минимаксное решение является надежным, поскольку оно обеспечивает верхнюю границу для объективного значения.

Метод минимаксного сожаления [20] вместо этого сводит к минимуму максимальное количество сожалений, которое пользователь может получить при выборе определенного проекта:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathcal{X}} \underbrace{\max_{\mathbf{w}\in\mathcal{W}} \max_{\mathbf{x}'\in\mathcal{X}} \mathbf{w}^\mathrm{T}\mathbf{f}(\mathbf{x}) - \mathbf{w}^\mathrm{T}\mathbf{f}(\mathbf{x}')}_{\text{Максимальное сожаление}}, \tag{12.16}$$

где $\mathbf{w}^\mathrm{T}\mathbf{f}(\mathbf{x}) - \mathbf{w}^\mathrm{T}\mathbf{f}(\mathbf{x}')$ — *сожаление*, связанное с выбором расчетной точки $\mathbf{x}$ вместо $\mathbf{x}'$ при векторе весов предпочтения $\mathbf{w}$. Минимаксное сожаление можно рассматривать как учет неопределенности системы принятия решений в отношении истинной функции полезности.

Минимаксное сожаление можно использовать в качестве критерия остановки для стратегий выявления предпочтений. Мы можем прекратить процедуру выявления предпочтений, как только минимаксное сожаление упадет ниже определенного порога.

12.6. Резюме

- Задачи проектирования с несколькими критериями часто связаны с эффективностью балансирования между различными критериями.
- Граница Парето представляет собой набор потенциально оптимальных решений.
- Векторнозначные целевые функции могут быть преобразованы в скалярные целевые функции с использованием методов на основе ограничений или весов.
- Популяционные методы могут быть расширены для получения особей, которые охватывают границу Парето.
- Знание предпочтений экспертов между парами точек в пространстве критериев может помочь при выводе скалярной целевой функции.

12.7. Упражнения

Упражнение 12.1. Метод взвешенной суммы — очень простой подход, и он действительно используется инженерами для многокритериальной оптимизации. Назовите недостаток этой процедуры, используемой для вычисления границы Парето.

Упражнение 12.2. Почему популяционные методы хорошо подходят для многокритериальной оптимизации?

Упражнение 12.3. Предположим, у вас есть точки {[1, 2], [2, 1], [2, 2], [1, 1]} в критериальном пространстве, и вы хотите аппроксимировать границу Парето. Какие точки являются оптимальными по Парето относительно остальных точек? Существуют ли слабо оптимальные по Парето точки?

Упражнение 12.4. Многоцелевую оптимизацию нелегко выполнить с помощью методов второго порядка. Почему?

Упражнение 12.5. Рассмотрим квадратное критериальное пространство $\mathcal{Y}$ с $y_1 \in [0, 1]$ и $y_2 \in [0, 1]$. Постройте критериальное пространство, укажите оптимальные по Парето и слабо оптимальные по Парето точки.

Упражнение 12.6. Условий $\mathbf{w} \geq 0$ и $\|\mathbf{w}\|_1 = 1$ в методе взвешенных сумм недостаточно для оптимальности по Парето. Приведите пример, когда координаты с нулевыми весами находят слабо оптимальные по Парето точки.

Упражнение 12.7. Приведите пример, когда программирование целей не находит оптимальной по Парето точки.

Упражнение 12.8. Используя метод ограничений, получите кривую Парето для задачи оптимизации:

$$\min_x \left[x^2, (x-2)^2 \right]. \tag{12.17}$$

Упражнение 12.9. Предположим, у нас есть многокритериальная задача оптимизации с двумя критериями:

$$f_1(x) = -(x-2)\sin x, \tag{12.18}$$

$$f_2(x) = -(x+3)^2 \sin x, \tag{12.19}$$

где $x \in \{-5, -3, -1, 1, 3, 5\}$. Постройте точки в критериальном пространстве. Сколько точек лежат на границе Парето?

Глава 13

Планы выбора

Для многих задач оптимизации вычисление функции может оказаться довольно дорогим. Например, для оценки конструкции аппаратного обеспечения может потребоваться длительный процесс изготовления, для конструкции самолета может потребоваться испытание в аэродинамической трубе, а для новых гиперпараметров с глубоким обучением может потребоваться неделя работы графических процессоров. Обычный подход к оптимизации в тех случаях, когда оценка расчетных точек является дорогостоящей, заключается в построении суррогатной модели, представляющей собой модель задачи оптимизации, которая может быть эффективно оптимизирована вместо истинной целевой функции. Дальнейшие вычисления истинной целевой функции могут быть использованы для улучшения модели. Настройка таких моделей требует начального набора точек (в идеале — точек, которые заполняют пространство); иначе говоря, необходимы точки, которые распределены по области как можно шире. В этой главе рассматриваются различные планы выбора для охвата пространства поиска при ограниченных ресурсах.[1]

13.1. Полный факторный план

Полный факторный план выбора (алгоритм 13.1) размещает сетку равномерно расположенных точек в пространстве поиска. Этот подход прост в реализации, не основан на случайности и охватывает пространство, но использует большое количество точек. Сетка равномерно расположенных точек распределена по пространству поиска, как показано на рис. 13.1. Оптимизация по точкам при использовании факторного плана называется *поиском по сетке*.

Алгоритм 13.1. Функция для получения всех местоположений выборки для полной факторной сетки. Здесь `a` — вектор переменных нижних границ, `b` — вектор переменных верхних границ, а `m` — вектор счетчика для каждого измерения

```
function samples_full_factorial(a, b, m)
    ranges = [range(a[i], stop = b[i], length = m[i])
```

[1] Существуют источники, в которых темы этой главы обсуждаются более подробно, например: [21], [35] и [106].

```
            for i in 1 : length(a)]
    collect.(collect(product(ranges...)))
end
```

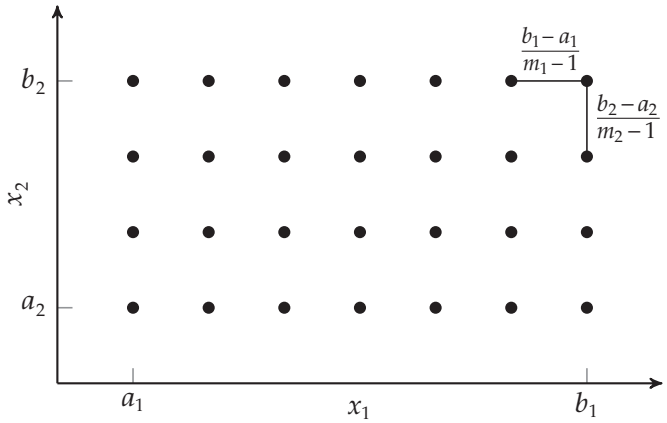


Рис. 13.1. Полный факторный поиск охватывает пространство поиска на сетке точек

Выборочная сетка определяется векторами нижней границы **a** и верхней границы **b** таким образом, что $a_i \le x_i \le b_i$ для каждого компонента i. Для сетки с m_i выборками в i-м измерении ближайшие точки разделены расстоянием $(b_i - a_i)/$ $/(m_i - 1)$.

Полный факторный метод требует количества подсчетов, экспоненциально зависящего от количества измерений.[2] Для n измерений с m выборками на измерение общее количество выборок равно m^n. Этот экспоненциальный рост слишком велик, чтобы его можно было использовать на практике, когда существует несколько переменных. Даже когда можно использовать полный факторный выбор, точки сетки, как правило, вынуждены быть достаточно грубыми и, следовательно, могут легко пропускать небольшие локальные особенности ландшафта оптимизации.

13.2. Случайный выбор

Непосредственной альтернативой методу полного факторного выбора является *случайный выбор*, который просто извлекает m случайных выборок из области параметров проектирования с использованием генератора псевдослучайных

[2] Название полного факторного метода происходит от проектирования с двумя или более *дискретными факторами*. Здесь коэффициентами являются m дискретизированных уровней, связанных с каждой переменной.

чисел. Чтобы сгенерировать случайную выборку $\mathbf{x}$, мы можем выбирать каждую переменную независимо от распределения. Если у нас есть границы для переменных, таких как $a_i \leq x_i \leq b_i$, общий подход заключается в использовании равномерного распределения по $[a_i, b_i]$, хотя могут использоваться и другие распределения. Для некоторых переменных может иметь смысл использовать логарифмически равномерное распределение.[3] Выборки расчетных точек не связаны друг с другом. Надежда состоит в том, что случайность сама по себе приведет к адекватному покрытию пространства параметров.

13.3. Планы равномерной проекции

Предположим, у нас есть двумерная задача оптимизации, дискретизированная на $m \times m$ выборочной сетке, как в полном факторном методе, но вместо того, чтобы брать все m^2 выборки, мы хотим выбрать только m позиций. Мы могли бы извлекать выборки наугад, но не все схемы одинаково полезны. Мы хотим, чтобы выборки распределялись по пространству, причем по каждому отдельному компоненту.

План равномерной проекции — это схема выбора по дискретной сетке, где распределение по каждому измерению является равномерным. Например, в крайней справа схеме на рис. 13.2 каждая строка и каждый столбец имеют ровно одну запись.

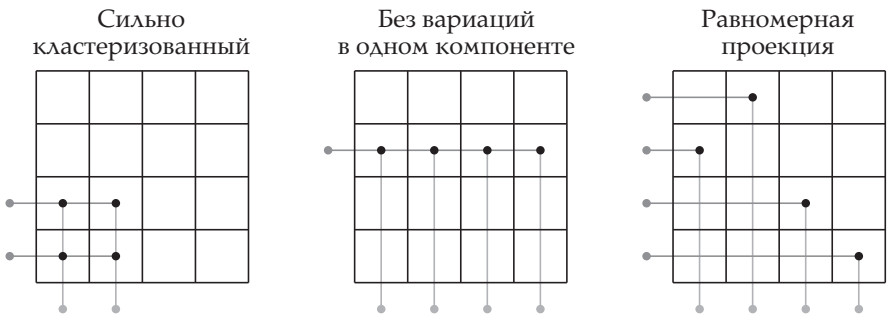


Рис. 13.2. Несколько способов выбрать m выборок из двумерной сетки. Как правило, мы предпочитаем схемы выбора, которые охватывают пространство и варьируются по каждому компоненту

План равномерной проекции с m выборками на сетке $m \times m$ может быть построен с использованием перестановки m элементов, как показано на рис. 13.3. Следовательно, существует $m!$ возможных планов равномерной проекции.[4]

[3] Некоторые параметры, такие как скорость обучения для глубоких нейронных сетей, лучше всего искать в логарифмическом пространстве.

[4] При $m = 5$ уже существуют $5! = 120$ возможных схем. При $m = 10$ таких схем уже 3 628 800.

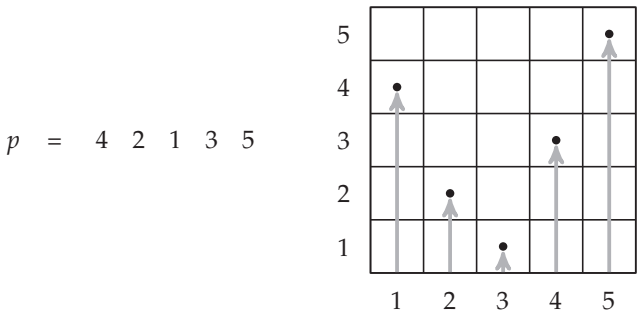


Рис. 13.3. Построение плана равномерной проекции с использованием перестановки

Выбор по плану равномерной проекции иногда называется *выбором из латинского гиперкуба* из-за связи с латинскими квадратами (рис. 13.4). Латинский квадрат — это сетка $m \times m$, где каждая строка и столбец содержат все целые числа от 1 до m. Латинские гиперкубы являются обобщением латинского квадрата на любое количество измерений.

Планы равномерных проекций для n измерений могут быть построены с использованием перестановки для каждого измерения (алгоритм 13.2).

4	1	3	2
1	4	2	3
3	2	1	4
2	3	4	1

Рис. 13.4. Латинский квадрат 4×4. План равномерной проекции может быть построен путем выбора значения $i \in \{1, 2, 3, 4\}$ и выбора всех ячеек с этим значением

Алгоритм 13.2. Функция для построения плана равномерных проекций для `n`-мерного гиперкуба с `m` выборками на измерение. Возвращает вектор индексных векторов

```
function uniform_projection_plan(m, n)
   perms = [randperm(m) for i in 1 : n]
   [[perms[i][j] for i in 1 : n] for j in 1 : m]
end
```

13.4. Стратифицированный выбор

Многие выборочные схемы, включая равномерное проецирование и полные факторные планы, основаны на сетке $m \times m$. Такая сетка, даже если она полностью заполнена, может пропустить важную информацию из-за систематических закономерностей, как показано на рис. 13.5. Один из способов обеспечить возможность попадания в каждую точку — использовать *стратифицированный выбор*.

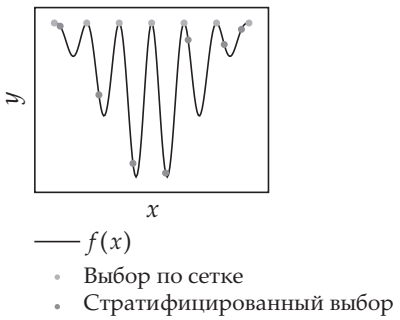


Рис. 13.5. Использование равномерно распределенной сетки в функции с систематическими закономерностями может пропустить важную информацию

Стратифицированный выбор изменяет любой план выбора на основе сетки, включая полные факторные планы и планы равномерной проекции. Ячейки отбираются по точке, выбранной равномерно случайным образом внутри ячейки, а не в ее центре, как показано на рис. 13.6.

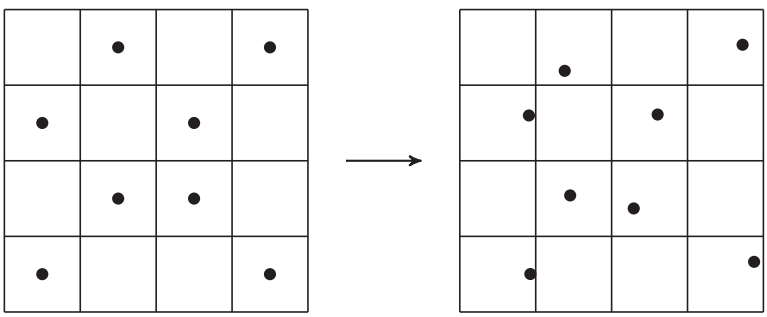


Рис. 13.6. Применение стратифицированного выбора к плану равномерной проекции

13.5. Параметры, заполняющие пространство

Хороший план выбора заполняет пространство параметров проекта, поскольку способность суррогатной модели обобщать на основе выборок уменьшается с расстоянием от этих выборок. Не все планы, даже одинаковые проекционные планы, одинаково хороши для покрытия пространства поиска. Например, диаго-

наль сетки (рис. 13.7) представляет собой план равномерной проекции, но охватывает только узкую полосу. В этом разделе рассматриваются различные метрики для измерения степени, до которой план выбора $X \subseteq \mathcal{X}$ заполняет пространство параметров проекта.

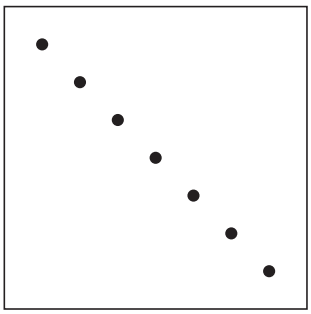


Рис. 13.7. План равномерной проекции, который не заполняет пространство

13.5.1. Несоответствие

Способность схемы выбора заполнять гиперпрямоугольное пространство можно измерить по ее *несоответствию* [91]. Если схема X имеет низкую степень несоответствия, то случайно выбранное помножество пространства параметров проектирования должно содержать долю выборок, пропорциональную объему подмножества.[5] Несоответствие, связанное со схемой X, является максимальной разницей между долей выборок в гиперпрямоугольном подмножестве $\mathcal{H}$ и объемом этого подмножества:

$$d(X) = \sup_{\mathcal{H}} \left| \frac{\#(X \cap \mathcal{H})}{\#X} - \lambda(\mathcal{H}) \right|, \tag{13.1}$$

где $\#X$ и $\#(X \cap \mathcal{H})$ — количество точек в X и количество точек, лежащих в пересечении X и $\mathcal{H}$ соответственно. Значение $\lambda(\mathcal{H})$ представляет собой n-мерный объем гиперпрямоугольника $\mathcal{H}$. Понятие *supremum* очень похоже на максимум, но оно позволяет найти решение для задач, где $\mathcal{H}$ лишь приближается к определенному прямоугольному подмножеству, как показано в примере 13.1.[6]

---

[5] В многомерном пространстве мы можем использовать меру Лебега, которая является обобщением объема (13.1) любого подмножества n-мерного евклидова пространства. Это длина в одномерном пространстве, площадь в двумерном пространстве и объем в трехмерном пространстве.

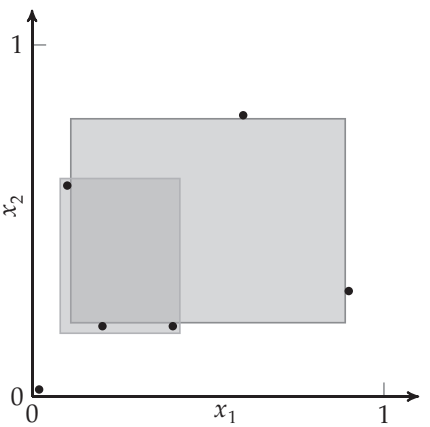
[6] Определение несоответствия требует использования гиперпрямоугольников и обычно предполагает, что X является конечным подмножеством единичного гиперкуба. Понятие несоответствия может быть расширено, чтобы позволить $\mathcal{H}$ включать другие множества, такие как выпуклые многогранники.

Пример 13.1. Вычисление несоответствия для плана выбора на единицу площади. Размеры прямоугольников слегка преувеличены, чтобы ясно показать, какие точки они содержат

Рассмотрим множество

$$X = \left\{ \left[\frac{1}{5}, \frac{1}{5}\right], \left[\frac{2}{5}, \frac{1}{5}\right], \left[\frac{1}{10}, \frac{3}{5}\right], \left[\frac{9}{10}, \frac{3}{10}\right], \left[\frac{1}{50}, \frac{1}{50}\right], \left[\frac{3}{5}, \frac{4}{5}\right] \right\}.$$

Расхождение по отношению единичного квадрата определяется прямоугольным подмножеством $\mathcal{H}$, которое либо имеет очень маленькую площадь, но содержит очень много точек, либо имеет очень большую площадь и содержит очень мало точек.



Синий прямоугольник, $x_1 \in \left[\frac{1}{10}, \frac{2}{5}\right]$, $x_2 \in \left[\frac{1}{5}, \frac{3}{5}\right]$ имеет объем, равный 0,12, и содержит три точки. Соответствующая мера расхождения составляет 0,38.

Фиолетовый прямоугольник $x_1 \in \left[\frac{1}{10}+\varepsilon, \frac{9}{10}-\varepsilon\right]$, $x_2 \in \left[\frac{1}{5}+\varepsilon, \frac{4}{5}-\varepsilon\right]$ дает еще большее расхождение. По мере приближения к нулю объем и расхождение приближаются, потому что прямоугольник не содержит точек. Отметим, что предел был нужен, чтобы отразить необходимость использования супремума в определении расхождения.

Вычисление несоответствия плана выбора по единичному гиперпрямоугольнику часто бывает трудным, и не всегда понятно, как мы можем вычислить расхождение для непрямоугольных выполнимых множеств.

13.5.2. Попарные расстояния

Альтернативный метод определения того, какой из двух планов выбора m точек является более плотным, заключается в сравнении *попарных расстояний* между всеми точками в каждом плане выбора. Планы выбора, которые сильнее разрежены, будут иметь бо́льшие попарные расстояния.

Сравнение обычно выполняется путем сортировки попарных расстояний каждого множества в порядке возрастания. План с первым попарным расстоянием, которое превышает другое, считается более плотно заполняющим пространство.

Алгоритм 13.3 вычисляет все попарные расстояния между точками в плане выборки. Алгоритм 13.4 сравнивает, насколько хорошо два плана выборки заполняют пространство, используя их соответствующие попарные расстояния.

Алгоритм 13.3. Функция для получения списка попарных расстояний между точками в плане выборки `X` с использованием нормы L_p, заданной параметром `p`

```
function pairwise_distances(X, p = 2)
    m = length(X)
    [norm(X[i] - X[j], p) for i in 1 : (m - 1) for j in (i + 1) : m]
end
```

Алгоритм 13.4. Функция для сравнения степени заполненности пространства для двух планов выбора `A` и `B` с использованием нормы L_p, заданной параметром `p`. Функция возвращает `-1`, если `A` больше заполняет пространство, чем `B`. Она возвращает `1`, если `B` больше заполняет пространство, чем `A`. Она возвращает `0`, если они эквивалентны

```
function compare_sampling_plans(A, B, p = 2)
    pA = sort(pairwise_distances(A, p))
    pB = sort(pairwise_distances(B, p))
    for (dA, dB) in zip(pA, pB)
        if dA < dB
            return 1
        elseif dA > dB
            return -1
        end
    end
    return 0
end
```

Один из способов создания плана равномерной проекции, заполняющего пространство, состоит в том, чтобы создать несколько случайных кандидатов, а затем использовать тот, который больше всего заполняет пространство.

Мы можем искать заполняющий пространство план равномерной проекции, многократно изменяя его таким образом, чтобы сохранить свойство равномерной проекции (алгоритм 13.5). Например, для плана выбора с хорошим охватом может быть использована имитация отжига.

Алгоритм 13.5. Функция мутирования плана равномерной проекции `X` при сохранении его свойства равномерного проецирования

```
function mutate!(X)
    m,n = length(X), length(X[1])
    j = rand(1 : n)
    i = randperm(m)[1 : 2]
    X[i[1]][j], X[i[2]][j] = X[i[2]][j], X[i[1]][j]
    return X
end
```

13.5.3. Критерий Морриса–Митчелла

Схема сравнения, описанная в разделе 13.5.2, обычно приводит к сложной задаче оптимизации со многими локальными минимумами. Альтернативой является оптимизация по *критерию Морриса–Митчелла* (алгоритм 13.6) [107]:

$$\Phi_q(X) = \left(\sum_i d_i^{-q} \right)^{1/q}, \tag{13.2}$$

где d_i — это i-е попарное расстояние между точками в X, а $q > 0$ — настраиваемый параметр.[7] Моррис и Митчелл рекомендуют оптимизировать:

$$\min_X \max_{q \in \{1,2,3,10,20,50,100\}} \Phi_q(X). \tag{13.3}$$

Алгоритм 13.6. Реализация критерия Морриса–Митчелла, который принимает список расчетных точек `X`, параметр критерия `q > 0` и параметр нормы `p ≥ 1`

```
function phiq(X, q = 1, p = 2)
    dists = pairwise_distances(X, p)
    return sum(dists .^ (-q)) ^ (1/q)
end
```

[7] Большие значения q дают более высокие штрафы за бо́льшие расстояния.

На рис. 13.8 показан критерий Морриса–Митчелла, вычисленный для нескольких случайно сгенерированных планов равномерной проекции.

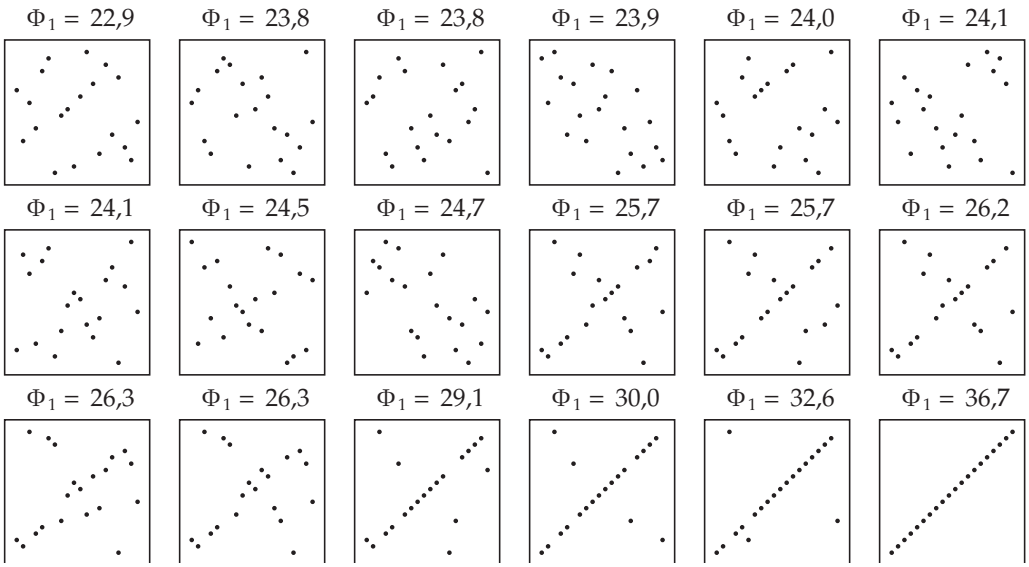


Рис. 13.8. Планы равномерной проекции отсортированы от лучших к худшим согласно показателю Φ_1

13.6. Подмножества, заполняющие пространство

Допустим, у нас есть набор точек X, и мы хотим найти подмножество точек $S \subset X$, которое по-прежнему максимально заполняет X. Необходимость идентификации *подмножеств, заполняющих пространство X*, возникает в контексте *моделей разного уровня точности* [52]. Например, предположим, что мы использовали план выбора X, чтобы идентифицировать различные конструкции крыла самолета для оценки с использованием вычислительных гидродинамических моделей при моделировании. Мы можем выбрать только подмножество этих проектных точек $\mathcal{S}$, чтобы построить и протестировать модель в аэродинамической трубе. При этом желательно, чтобы множество $\mathcal{S}$ плотно заполняло пространство.

Степень, до которой S заполняет пространство параметров проектов, может быть определена количественно с использованием максимального расстояния между точкой в X и ближайшей точкой в $\mathcal{S}$. Эта метрика обобщается на любые два конечных множества A и B (алгоритм 13.7). Мы можем применить любую норму L_p, но мы обычно используем L_2, т.е. евклидово расстояние:

$$d_{\max}\left(X, \mathcal{S}\right) = \max_{\mathbf{x} \in X} \min_{\mathbf{s} \in \mathcal{S}} \left\| \mathbf{s} - \mathbf{x} \right\|_p . \tag{13.4}$$

Алгоритм 13.7. Набор метрик расстояния L_p между двумя дискретными наборами, где **A** и **B** — списки расчетных точек, а **p** — параметр нормы L_p

```
min_dist(a, B, p) = minimum(norm(a - b, p) for b in B)
d_max(A, B, p = 2) = maximum(min_dist(a, B, p) for a in A)
```

План выбора, заполняющий пространство, — тот, который минимизирует этот показатель.[8] Поиск плана выбора, заполняющего пространства с помощью m элементов, представляет собой задачу оптимизации

$$\min_{\mathcal{S}} d_{\max}(X, \mathcal{S}) \quad \text{при условии, что } \mathcal{S} \subseteq X, \ \#\mathcal{S} = m. \tag{13.5}$$

Оптимизация уравнения (13.5), как правило, трудно поддается вычислению. Метод полного перебора будет использовать все $d!/m!(d-m)!$ подмножества размера m из множества, содержащего d точек. И жадный локальный поиск (алгоритм 13.8), и *алгоритм обмена* (алгоритм 13.9) являются эвристическими стратегиями для преодоления этой трудности. Они обычно находят приемлемые подмножества X для заполнения пространства.

Алгоритм 13.8. Жадный локальный поиск для нахождения плана выбора m элементов, который минимизирует метрику **d** для дискретного набора **X**

```
function greedy_local_search(X, m, d = d_max)
    S = [X[rand(1 : m)]]
    for i in 2 : m
        j = argmin([x ∈ S ? Inf : d(X, push!(copy(S), x))
                for x in X])
        push!(S, X[j])
    end
    return S
end
```

Алгоритм 13.9. Алгоритм обмена для нахождения плана выбора m элементов, который минимизирует метрику расстояния **d** для дискретного набора **X**

```
function exchange_algorithm(X, m, d = d_max)
    S = X[randperm(m)]
    δ, done = d(X, S), false
    while !done
```

[8] Для этого можно также использовать критерий Морриса–Митчелла для $\mathcal{S}$.

```
        best_pair = (0, 0)
        for i in 1 : m
            s = S[i]
            for (j, x) in enumerate(X)
                if !in(x, S)
                    S[i] = x
                    δ′ = d(X, S)
                    if δ′ < δ
                       δ = δ′
                       best_pair = (i, j)
                    end
                end
            end
            S[i] = s
        end
        done = best_pair == (0, 0)
        if !done
           i, j = best_pair
           S[i] = X[j]
        end
    end
    return S
end
```

Жадный локальный поиск начинается с точки, выбранной случайным образом из X, и постепенно добавляет следующую лучшую точку, которая минимизирует метрику расстояния. Очки добавляются, пока не будет достигнуто желаемое количество очков. Поскольку точки инициализируются случайным образом, наилучшие результаты получают, если несколько раз запустить жадный локальный поиск и сохранить лучший план выбора (алгоритм 13.10).

Алгоритм 13.10. Локальный поиск **Multistart** запускает определенный алгоритм поиска несколько раз и возвращает лучший результат. Здесь **X** — список точек, **m** — размер желаемого плана выборки, **alg** — **exchange_algorithm** или **greedy_local_search**, **k_max** — количество итераций, которые нужно выполнить, а **d** — метрика

```
function multistart_local_search(X, m, alg, k_max, d = d_max)
    sets = [alg(X, m, d)for i in 1 : k_max]
    return sets[argmin([d(X, S) for S in sets])]
end
```

Алгоритм обмена инициализирует $\mathcal{S}$ случайным подмножеством X и многократно заменяет точки в $\mathcal{S}$ на другую точку в X, которой еще нет в $\mathcal{S}$, для улучшения расстояния. Алгоритм обмена также обычно запускается несколько раз.

Подмножества, заполняющие пространство, полученные с использованием жадного локального поиска и алгоритма обмена, сравниваются на рис. 13.9.

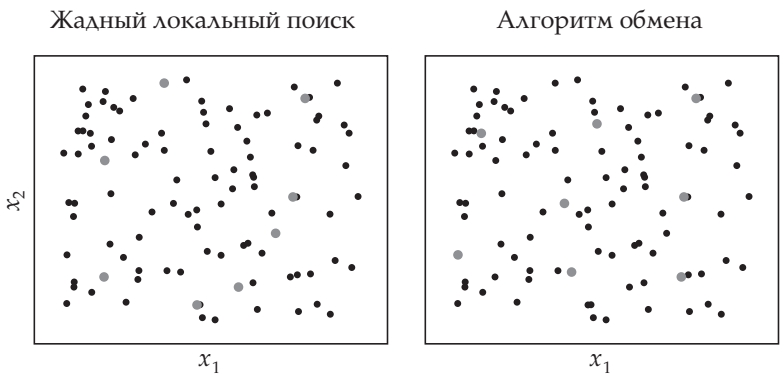


Рис. 13.9. Подмножества, заполняющие пространство, полученные с помощью жадного локального поиска и алгоритма обмена

13.7. Квазислучайные последовательности

Квазислучайные последовательности [95] также называемые *последовательностями с малым расхождением*, часто используются при попытках аппроксимировать интеграл по многомерному пространству:

$$\int_{\mathcal{X}} f(\mathbf{x})\, d\mathbf{x} \approx \frac{v}{m} \sum_{i=1}^{m} f\left(\mathbf{x}^{(i)}\right), \tag{13.6}$$

где каждая точка $\mathbf{x}^{(i)}$ случайным образом равномерно выбирается из области $\mathcal{X}$, а v — объем области $\mathcal{X}$. Это приближение известно как *интегрирование методом Монте-Карло*.

Квазислучайные последовательности содержат детерминированные, а не случайные или псевдослучайные точки интегрирования, и систематически заполняют пространство, так что интеграл сходится настолько быстро, насколько это возможно при количестве точек m.[9] *Методы квази-Монте-Карло* имеют сходимость по ошибке $O(1/m)$, в отличие от $O\left(1/\sqrt{m}\right)$ при традиционном методе интегрирования Монте-Карло, как показано на рис. 13.10.

[9] Последовательности псевдослучайных чисел, например созданные в результате последовательности вызовов функции `rand`, являются детерминированными для конкретного начального значения, но кажутся случайными. Квазислучайные числа также являются детерминированными, но не кажутся случайными.

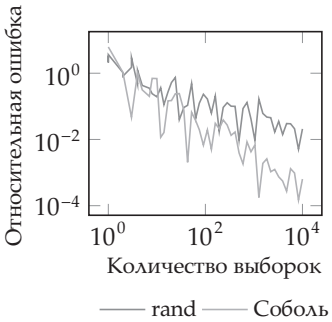


Рис. 13.10. Ошибка при вычислении $\int_0^1 \sin 10x\,dx$ с помощью метода интегрирования Монте-Карло со случайными числами из $\mathcal{U}(0, 1)$ и последовательностью Соболя. Последовательность Соболя, описанная в разделе 13.7.3, сходится быстрее

Квазислучайные последовательности обычно строятся для единичного n-мерного гиперкуба $[0, 1]^n$. Любая многомерная функция с границами для каждой переменной может быть преобразована в такой гиперкуб. Это преобразование реализовано в алгоритме 7.9.

Существуют различные методы генерации квазислучайных последовательностей. Несколько таких методов сравниваются со случайным выбором на рис. 13.12.

13.7.1. Аддитивная рекурсия

Простые рекуррентные отношения вида

$$x^{(k+1)} = x^{(k)} + c \pmod 1 \tag{13.7}$$

генерируют множества, заполняющие пространства при условии, что c иррационально. Значение c, приводящее к наименьшему расхождению, равно

$$c = 1 - \varphi = \frac{\sqrt{5} - 1}{2} \approx 0{,}618034, \tag{13.8}$$

где φ — это золотое сечение [139].

Мы можем построить множество, заполняющее n-мерное пространство, используя аддитивную последовательность повторений для каждой координаты, каждую со своим собственным значением c. Известно, что квадратные корни простых чисел иррациональны, и поэтому их можно использовать для получения различных последовательностей для каждой координаты:

$$c_1 = \sqrt{2}, c_2 = \sqrt{3}, c_3 = \sqrt{5}, c_4 = \sqrt{7}, c_5 = \sqrt{11}, \ldots \tag{13.9}$$

Методы аддитивного повторения реализованы в алгоритме 13.11.

Алгоритм 13.11. Аддитивное повторение для построения m-элементных последовательностей, заполняющих n-мерные единичные гиперкубы. Для генерации первых n простых чисел, таких что k-е простое число при $k > 6$ ограничено числом $k(\log k + \log\log k)$, используется пакет `Primes`

```
using Primes
function get_filling_set_additive_recurrence(m; c = φ - 1)
   X = [rand()]
   for i in 2 : m
       push!(X, mod(X[end] + c, 1))
   end
   return X
end
function get_filling_set_additive_recurrence(m, n)
    ps = primes(max(ceil(Int, n * (log(n) + log(log(n)))), 13))
    seqs = [get_filling_set_additive_recurrence(m, c = sqrt(p))
           for p in ps[1 : n]]
    return [collect(x) for x in zip(seqs...)]
end
```

13.7.2. Последовательность Холтона

Последовательность Холтона является многомерным квазислучайным множеством, заполняющим пространство [66]. Одномерная версия, называемая *последовательностью Ван дер Корпута*, генерирует последовательности, в которых единичный интервал делится на степени основания b. Например, $b = 2$ производит последовательность

$$X = \left\{\frac{1}{2'}, \frac{1}{4'}, \frac{3}{4'}, \frac{1}{8'}, \frac{5}{8'}, \frac{3}{8'}, \frac{7}{8'}, \frac{1}{16'}, \ldots\right\}, \tag{13.10}$$

а при $b = 5$ получается:

$$X = \left\{\frac{1}{5'}, \frac{2}{5'}, \frac{3}{5'}, \frac{4}{5'}, \frac{1}{25'}, \frac{6}{25'}, \frac{11}{25'}, \ldots\right\}. \tag{13.11}$$

В многомерных последовательностях, заполняющих пространство, используется одна последовательность Ван дер Корпута для каждого измерения, каждая со своей базой b. Основания, однако, должны быть взаимно простыми[10], чтобы быть некоррелированными. Методы построения последовательностей Холтона реализованы в алгоритме 13.12.

[10] Два целых числа являются взаимно простыми, если единственное положительное целое число, которое является их общим делителем, равно единице.

Алгоритм 13.12. Квазислучайные m-элементные последовательности Холтона, заполняющие **n**-мерный единичный гиперкуб, где **b** — база. Основания **bs** должны быть взаимно простыми.

```
using Primes
function halton(i, b)
    result, f = 0.0, 1.0
    while i > 0
       f = f / b;
       result = result + f * mod(i, b)
       i = floor(Int, i / b)
    end
    return result
end
get_filling_set_halton(m; b = 2) = [halton(i, b) for i in 1 : m]
function get_filling_set_halton(m, n)
    bs = primes(max(ceil(Int, n * (log(n) + log(log(n)))), 6))
    seqs = [get_filling_set_halton(m, b = b) for b in bs[1 : n]]
    return [collect(x) for x in zip(seqs...)]
end
```

Для больших простых чисел мы можем получить корреляцию в первых нескольких числах. Такая корреляция показана на рис. 13.11. Корреляции можно избежать с помощью *скачкообразного метода Холтона* [86], который принимает каждую p-ю точку, где p — простое число, отличное от всех координатных баз.

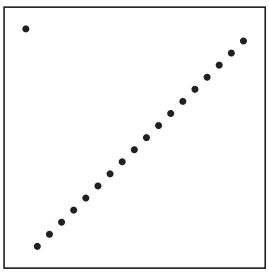


Рис. 13.11. Последовательность Холтона при **b** = [19, 23], в которой первые 18 выборок идеально коррелируют друг с другом

13.7.2. Последовательность Соболя

Последовательность Соболя — это квазислучайная последовательность, заполняющая n-мерных гиперкубов. Они генерируются путем применения операции

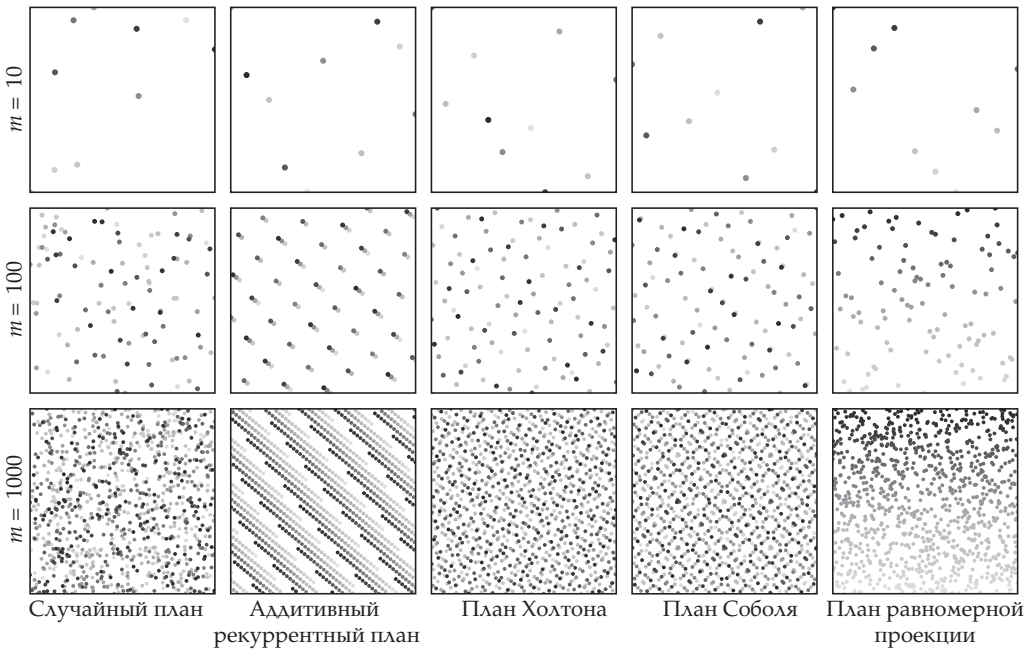


Рис. 13.12. Сравнение планов выбора в двумерном пространстве. Выборки окрашиваются в соответствии с порядком их отбора. План равномерной проекции был создан случайным образом и не оптимизирован

"исключительное или" к предыдущему числу Соболя и набору чисел по направлениям [145]:[11]

$$X_j^{(i)} = X_j^{(i-1)} \veebar v_j^{(k)}, \tag{13.12}$$

где $v_j^{(k)}$ — это j-й бит числа, соответствующего k-му направлению. Таблицы хороших чисел, соответствующих направлениям, были предоставлены различными авторами.[12]

Сравнение этих и предыдущих подходов показано на рис. 13.12. Для высоких значений несколько методов имеют четкую структуру.

13.8. Резюме

- Планы выбора используются для покрытия областей поиска с помощью ограниченного количества точек.

[11] Символ $\veebar$ обозначает операцию "исключительное или", которая возвращает истину тогда и только тогда, когда операнды не равны друг другу.

[12] Пакет `Sobol.jl` обеспечивает реализацию до 1111 измерений.

- Полный факторный выбор, который включает выбор по вершинам равномерно дискретизированной сетки, требует числа точек, экспоненциально зависящего от количества измерений.
- Планы равномерной проекции, которые проецируются равномерно по каждому измерению, могут быть эффективно сгенерированы и оптимизированы для заполнения пространства.
- Жадный локальный поиск и алгоритм обмена могут быть использованы для поиска подмножества точек, максимально заполняющих пространство.
- Квазислучайные последовательности — это детерминированные процедуры, с помощью которых могут быть сгенерированы выборки, заполняющие пространство.

13.9. Упражнения

Упражнение 13.1. Заполнение многомерного пространства по мере увеличения числа измерений требует экспоненциально большого количества точек. Чтобы обосновать это утверждение, определите длины сторон n-мерного гиперкуба таким образом, чтобы он заполнил половину объема n-мерного гиперкуба.

Упражнение 13.2. Предположим, что выборки случайным образом извлекаются из единичной сферы в n-мерном пространстве. Вычислите вероятность того, что случайно выбранная точка находится на расстоянии ε от поверхности сферы при $n \to \infty$. Подсказка: объем сферы равен $C(n)r^n$, где r — радиус, а $C(n)$ — функция, зависящая только от размера n.

Упражнение 13.3. Предположим, у нас есть план выбора $X = \{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(10)}\}$, где

$$\mathbf{x}^{(i)} = [\cos(2\pi i/10), \sin(2\pi i/10)]. \tag{13.13}$$

Вычислите критерий Морриса–Митчелла для X с помощью нормы L_2, если параметр q установлен равным 2. Иначе говоря, оцените $\Phi_2(X)$. Изменится ли $\Phi_2(X)$, если мы добавим [2, 3] к каждому $\mathbf{x}^{(i)}$? Обоснуйте свой ответ.

Упражнение 13.4. Аддитивая рекурсия требует, чтобы мультипликативный коэффициент c в уравнении (13.7) был иррациональным. Почему он не может быть рациональным?

Глава 14

Суррогатные модели

В предыдущей главе обсуждались методы составления плана выбора. В этой главе показано, как использовать такие выборки для построения моделей целевой функции, которые можно использовать вместо реальной целевой функции. Описанные суррогатные модели можно легко и недорого вычислить и эффективно оптимизировать. Суррогатная модель помогает направить поиск оптимальной функции к реальной цели.

14.1. Обучение суррогатных моделей

Суррогатная модель $\hat{f}$ зависит от параметров $\boldsymbol{\theta}$ и предназначена для имитации истинной целевой функции f. Параметры $\boldsymbol{\theta}$ могут быть подобраны для обучения модели на основе выборок, взятых из модели f. Пример суррогатной модели показан на рис. 14.1.

Рис. 14.1. Суррогатные модели приближают истинную целевую функцию. Модель настраивается по вычисленным расчетным точкам, но вдали от них отклоняется от истинных значений

Предположим, у нас есть m расчетных точек

$$X = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(m)}\} \tag{14.1}$$

и соответствующие значения функции

$$\mathbf{y} = \{y^{(1)}, y^{(2)}, ..., y^{(m)}\}. \tag{14.2}$$

При определенном наборе параметров модель будет предсказывать значения

$$\hat{\mathbf{y}} = \left\{ \hat{f}_{\boldsymbol{\theta}}\left(\mathbf{x}^{(1)}\right), \hat{f}_{\boldsymbol{\theta}}\left(\mathbf{x}^{(2)}\right), \ldots, \hat{f}_{\boldsymbol{\theta}}\left(\mathbf{x}^{(m)}\right) \right\}. \tag{14.3}$$

Обучение модели по набору точек требует подбора параметров, минимизирующих разницу между истинными значениями и значениями, предсказанными моделью, как правило, в соответствии с нормой L_p:[1]

$$\min_{\boldsymbol{\theta}} \left\| \mathbf{y} - \hat{\mathbf{y}} \right\|_p . \tag{14.4}$$

Формула (14.4) штрафует отклонение модели только в точках данных. Нет никаких гарантий, что модель будет по-прежнему хорошо соответствовать данным наблюдений, и точность модели, как правило, уменьшается по мере удаления от точек выборки.

Эта форма настройки модели называется *регрессией*. Существует большой объем работ по решению проблем регрессии, и он широко изучается в области машинного обучения [108].

В оставшейся части этой главы рассматривается несколько популярных суррогатных моделей и алгоритмов для обучения суррогатных моделей по данным, а в заключение приводятся методы выбора между типами моделей.

14.2. Линейные модели

Простая суррогатная модель — это *линейная модель*, которая имеет вид[2]

$$\hat{f} = w_0 + \mathbf{w}^{\mathrm{T}}\mathbf{x}, \quad \boldsymbol{\theta} = \left(w_0, \mathbf{w}\right). \tag{14.5}$$

Для n-мерного пространства проектирования линейная модель имеет $n+1$ параметров и, таким образом, требует наличия по крайней мере $n+1$ выборок, чтобы аппроксимация была однозначной.

Вместо того, чтобы иметь $\mathbf{w}$ и w_0 в качестве параметров, обычно строят один вектор $\boldsymbol{\theta} = [w_0, \mathbf{w}]$ и добавляют единицу к вектору $\mathbf{x}$, получая

$$\hat{f} = \boldsymbol{\theta}^{\mathrm{T}}\mathbf{x}. \tag{14.6}$$

Поиск оптимального $\boldsymbol{\theta}$ требует решения задачи *линейной регрессии*:

$$\min_{\boldsymbol{\theta}} \left\| \mathbf{y} - \hat{\mathbf{y}} \right\|_2^2 , \tag{14.7}$$

что эквивалентно решению задачи

$$\min_{\boldsymbol{\theta}} \left\| \mathbf{y} - \mathbf{X}\boldsymbol{\theta} \right\|_2^2 , \tag{14.8}$$

где $\mathbf{X}$ — матрица плана, сформированная из m точек исходных данных

[1] Обычно используется норма L_2. Минимизация этой функции по норме L_2 эквивалентна минимизации среднеквадратической ошибки в этих точках данных.

[2] Это хорошо известное уравнение гиперплоскости.

$$\mathbf{X} = \begin{bmatrix} \left(\mathbf{x}^{(1)}\right)^{\mathrm{T}} \\ \left(\mathbf{x}^{(2)}\right)^{\mathrm{T}} \\ \cdots \\ \left(\mathbf{x}^{(m)}\right)^{\mathrm{T}} \end{bmatrix}. \tag{14.9}$$

Алгоритм 14.1. Метод построения матрицы плана из списка точек `X` проекта и способ обучения суррогатной модели с использованием линейной регрессии по списку расчетных точек `X` и вектора значений целевой функции `y`

```
function design_matrix(X)
   n, m = length(X[1]), length(X)
   return [j == 0 ? 1.0 : X[i][j] for i in 1 : m, j in 0 : n]
end
function linear_regression(X, y)
   θ = pinv(design_matrix(X)) * y
   return x -> θ⋅[1; x]
end
```

Алгоритм 14.1 реализует методы для вычисления матрицы плана и для решения задачи линейной регрессии. Несколько случаев линейной регрессии показаны на рис. 14.2.

Линейная регрессия имеет аналитическое решение

$$\boldsymbol{\theta} = \mathbf{X}^{+}\mathbf{y}, \tag{14.10}$$

где $\mathbf{X}^{+}$ — *псевдообращение Мура–Пенроуза* матрицы $\mathbf{X}$.

Если матрица $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ обратима, то псевдообратная матрица может быть вычислена как

$$\mathbf{X}^{+} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}. \tag{14.11}$$

Если матрица $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ обратима, то псевдообратная матрица может быть вычислена как

$$\mathbf{X}^{+} = \mathbf{X}^{\mathrm{T}}\left(\mathbf{X}\mathbf{X}^{\mathrm{T}}\right)^{-1}. \tag{14.12}$$

Функция `pinv` вычисляет псевдообратную матрицу заданной матрицы.[3]

[3] Функция `pinv` использует сингулярное разложение $\mathbf{X}^{+} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{*}$ для вычисления псевдообратной матрицы $\mathbf{X}^{+} = \mathbf{V}\boldsymbol{\Sigma}^{+}\mathbf{U}^{*}$, где псевдообратная диагональная матрица получается путем вычисления обратной величины каждого ненулевого элемента диагонали с последующим транспонированием результата.

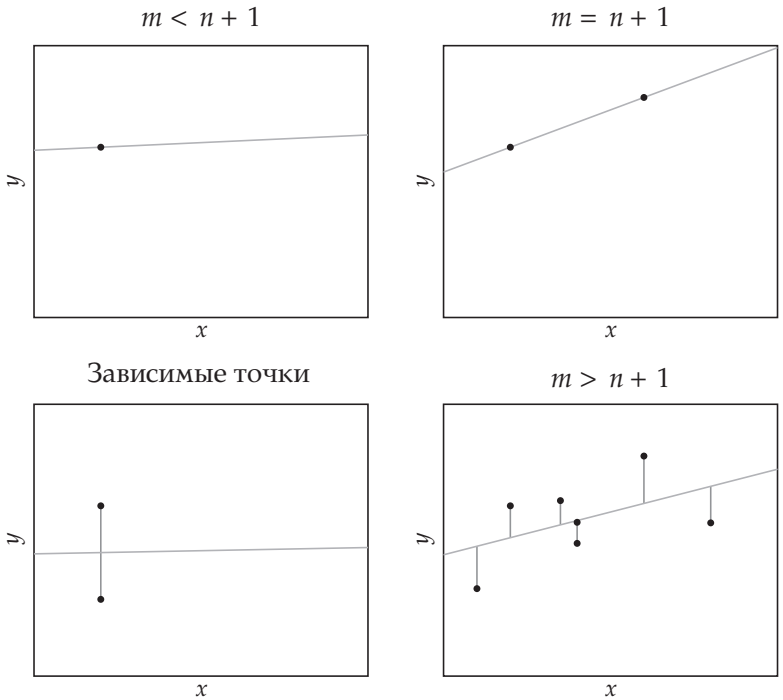


Рис. 14.2. Модели, полученные в результате линейной регрессии, которая минимизирует квадрат вертикального расстояния модели от каждой точки. Псевдообращение находит единственное решение для любой непустой конфигурации точек. В нижней левой части рисунка показана модель, полученная для двух повторяющихся точек, в данном случае $m = n + 1$. Поскольку две точки повторяются, матрица $\mathbf{X}$ не является невырожденной. Хотя в этом случае матрица $\mathbf{X}$ не является обратимой, псевдообращение дает единственное решение, которое проходит между двумя точками

14.3. Базисные функции

Линейная модель представляет собой линейную комбинацию компонентов вектора $\mathbf{x}$:

$$\hat{f}(\mathbf{x}) = \theta_1 x_1 + \ldots + \theta_n x_n = \sum_{i=1}^{n} \theta_i x_i = \boldsymbol{\theta}^{\mathrm{T}} \mathbf{x}, \tag{14.13}$$

которая является конкретным примером более общей линейной комбинации базисных функций

$$\hat{f}(\mathbf{x}) = \theta_1 b_1(\mathbf{x}) + \ldots + \theta_n b_q(\mathbf{x}) = \sum_{i=1}^{q} \theta_i b_i(\mathbf{x}) = \boldsymbol{\theta}^{\mathrm{T}} \mathbf{b}(\mathbf{x}). \tag{14.14}$$

В случае линейной регрессии базисные функции просто извлекают каждый компонент, $b_i(\mathbf{x}) = x_i$.

Любая суррогатная модель, представленная в виде линейной комбинации базисных функций, может быть настроена с использованием регрессии:

$$\min_{\boldsymbol{\theta}} \left\| \mathbf{y} - \mathbf{B}\boldsymbol{\theta} \right\|_2^2, \tag{14.15}$$

где $\mathbf{B}$ — базисная матрица, сформированная из m точек исходных данных:

$$\mathbf{B} = \begin{bmatrix} \mathbf{b}\left(\mathbf{x}^{(1)}\right)^{\mathrm{T}} \\ \mathbf{b}\left(\mathbf{x}^{(2)}\right)^{\mathrm{T}} \\ \cdots \\ \mathbf{b}\left(\mathbf{x}^{(m)}\right)^{\mathrm{T}} \end{bmatrix}. \tag{14.16}$$

Весовые параметры могут быть получены с использованием псевдообращения

$$\boldsymbol{\theta} = \mathbf{B}^{+}\mathbf{y} \tag{14.17}$$

Эту более общую процедуру регрессии реализует алгоритм 14.2.

Алгоритм 14.2. Способ обучения суррогатной модели по списку расчетных точек `X` и соответствующих значений целевой функции `y` с использованием регрессии с базисными функциями, содержащимися в массиве `bases`

```
function regression(X, y, bases)
    B = [b(x) for x in X, b in bases]
    θ = pinv(B) * y
    return x -> sum(θ[i] * bases[i](x) for i in 1 : length(θ))
end
```

Линейные модели не могут отражать нелинейные отношения. Существует множество других семейств базовых функций, которые позволяют построить более выразительные суррогатные модели. В оставшейся части этого раздела обсуждается несколько общих семейств базисных функций.

14.3.1. Полиномиальные базисные функции

Полиномиальные базисные функции состоят из произведения компонентов вектора параметров, каждый из которых возведен в степень. Линейные базисные функции являются частным случаем полиномиальных базисных функций.

Из разложения в ряд Тейлора (см. приложение В.2) мы знаем, что любая бесконечно дифференцируемая функция может быть сколь угодно точно аппроксимирована полиномом достаточно высокой степени. Мы можем построить эти базисы, используя алгоритм 14.3.

Алгоритм 14.3. Метод построения массива полиномиальных базисных функций с точностью до степени `k` для `i`-го компонента расчетной точки и метод построения списка `n`-мерных полиномиальных базисов для слагаемых до степени `k`

```
polynomial_bases_1d(i, k) = [x -> x[i] ^ p for p in 0 : k]
function polynomial_bases(n, k)
    bases = [polynomial_bases_1d(i, k) for i in 1 : n]
    terms = Function[]
    for ks in product([0 : k for i in 1 : n]...)
        if sum(ks) ≤ k
           push!(terms,
                 x -> prod(b[j + 1](x) for (j, b) in zip(ks, bases)))
        end
    end
    return terms
end
```

В одномерном пространстве полиномиальная модель степени k имеет вид

$$\hat{f}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \ldots + \theta_k x^k = \sum_{i=0}^{k} \theta_i x^i. \tag{14.18}$$

Следовательно, мы имеем набор базисных функций $b_i(x) = x^i$ для перехода от 0 до k.

В двух измерениях полиномиальная модель степени k имеет базисные функции вида

$$b_{ij}(\mathbf{x}) = x_1^i x_2^j \text{ для } i, j \in \{0, \ldots, k\},\ i + j \le k. \tag{14.19}$$

Настройка полиномиальной суррогатной модели является задача регрессии, поэтому полиномиальная модель является линейной в многомерном пространстве (рис. 14.3). Любая линейная комбинация базисных функций может рассматриваться как линейная регрессия в многомерном пространстве.

14.3.2. Синусоидальные базисные функции

Любая непрерывная функция на конечной области может быть представлена с использованием бесконечного множества синусоид.[4] *Ряд Фурье* можно построить для любой интегрируемой одномерной функции f на отрезке $[a, b]$:

[4] Ряд Фурье так же точно приближает периодическую функцию f, определенную на всей вещественной прямой.

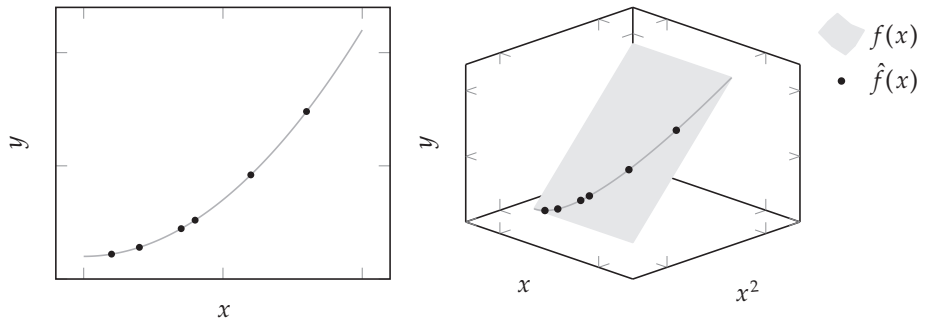


Рис. 14.3. Полиномиальная модель в пространствах более высоких измерений является линейной. Функция существует в плоскости, образованной из базисов, но она не занимает всю плоскость, поскольку члены базиса не являются независимыми

$$f(x) = \frac{\theta_0}{2} + \sum_{i=1}^{\infty}\left(\theta_i^{(\sin)} \sin\frac{2\pi i x}{b-a} + \theta_i^{(\cos)} \cos\frac{2\pi i x}{b-a}\right), \tag{14.20}$$

где

$$\theta_0 = \frac{2}{b-a}\int_a^b f(x)dx, \tag{14.21}$$

$$\theta_i^{(\sin)} = \frac{2}{b-a}\int_a^b f(x)\sin\frac{2\pi i x}{b-a}dx, \tag{14.22}$$

$$\theta_i^{(\cos)} = \frac{2}{b-a}\int_a^b f(x)\cos\frac{2\pi i x}{b-a}dx. \tag{14.23}$$

Аналогично тому, как в полиномиальных моделях используются первые несколько членов ряда Тейлора, в синусоидальных моделях используются первые несколько членов ряда Фурье. Базис в одномерной области $x \in [a, b]$ имеет вид

$$\begin{cases} b_0(x) = \dfrac{1}{2}, \\ b_i^{(\sin)} = \sin\dfrac{2\pi i x}{b-a}, \\ b_i^{(\cos)} = \cos\dfrac{2\pi i x}{b-a}. \end{cases} \tag{14.24}$$

Мы можем объединить слагаемые для многомерных синусоидальных моделей так же, как мы объединяем слагаемые в полиномиальных моделях. Для построения синусоидальных базисных функций может быть использован алгоритм 14.4. Несколько случаев синусоидальной регрессии показаны на рис. 14.4.

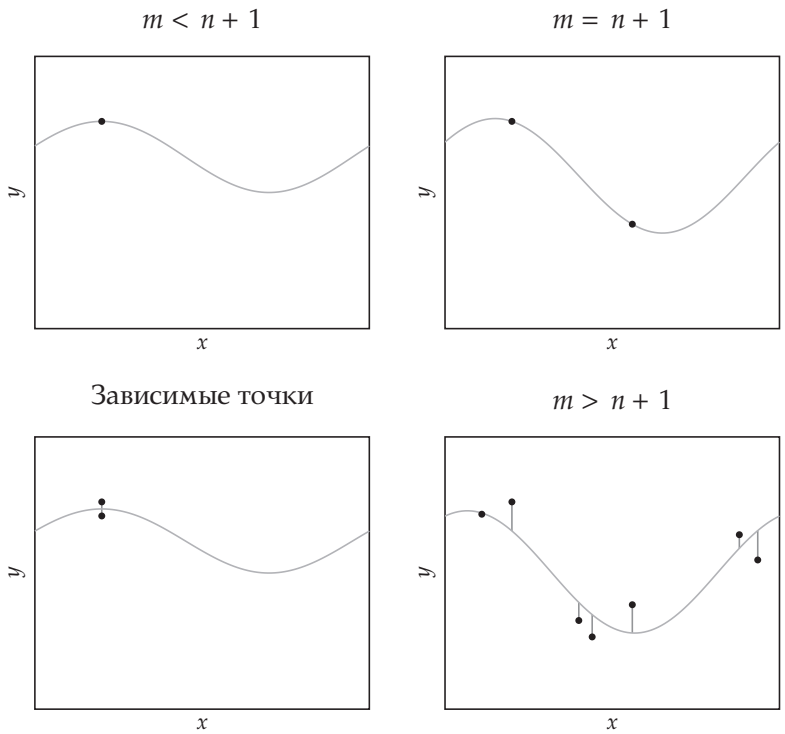


Рис. 14.4. Подгонка синусоидальных моделей к зашумленным точкам

Алгоритм 14.4. Метод **sinusoidal_bases_ld** создает список базовых функций с точностью до степени **k** для **i**-го компонента расчетного вектора с учетом нижней границы **a** и верхней границы **b**. Метод **sinusoidal_bases** создает все комбинации функций вплоть до степени **k** для нижнего вектора **a** и верхнего вектора **b**

```
function sinusoidal_bases_ld(j, k, a, b)
   T = b[j] - a[j]
   bases = Function[x -> 1/2]
   for i in 1 : k
       push!(bases, x -> sin(2π * i * x[j] / T))
       push!(bases, x -> cos(2π * i * x[j] / T))
   end
   return bases
end
function sinusoidal_bases(k, a, b)
    n = length(a)
    bases = [sinusoidal_bases_ld(i, k, a, b) for i in 1 : n]
    terms = Function[]
```

```
    for ks in product([0 : 2k for i in 1 : n]...)
        powers = [div(k + 1, 2) for k in ks]
        if sum(powers) ≤ k
           push!(terms,
                 x -> prod(b[j + 1](x) for (j, b) in zip(ks, bases)))
        end
    end
    return terms
end
```

14.3.3. Радиальные базовые функции

Радиальная функция φ — это функция, которая зависит только от расстояния от точки до некоторой центральной точки $\mathbf{c}$, так что ее можно записать в виде $\psi(\mathbf{x}, \mathbf{c}) = \psi(\|\mathbf{x} - \mathbf{c}\|) = \psi(r)$. Радиальные функции — это удобные базисные функции, потому что включение радиальной функции вносит вклад в холм или долину. Некоторые общие радиальные базисные функции показаны на рис. 14.5.

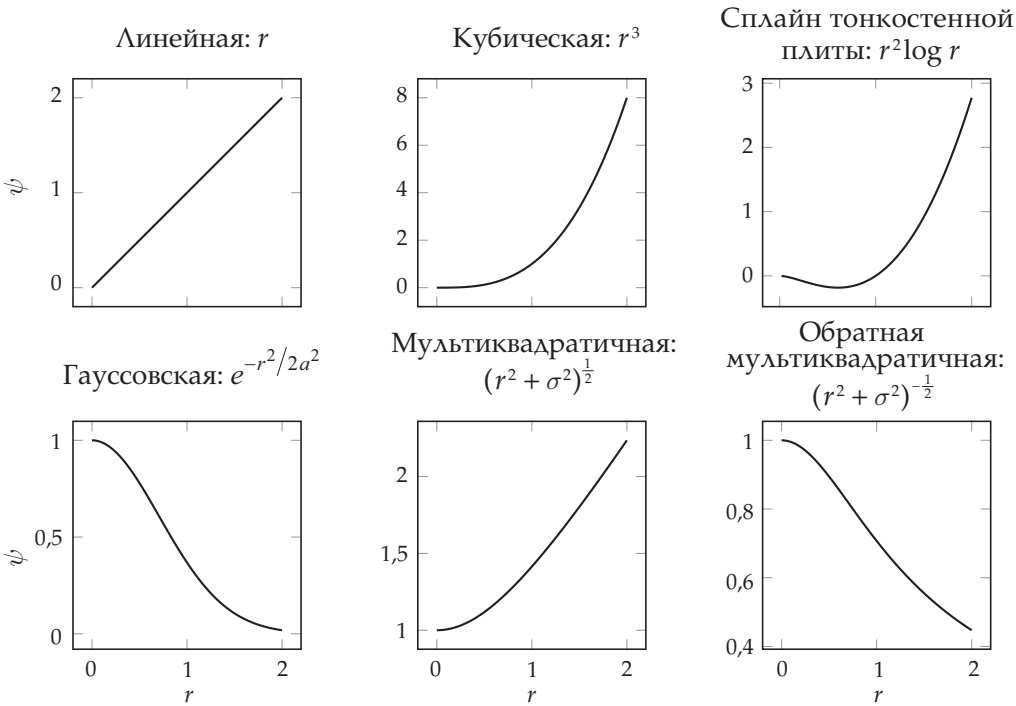


Рис. 14.5. Примеры радиальных базисных функций

Радиальные базисные функции требуют указания центральных точек. Одним из подходов при подборе радиальных базисных функций по множеству точек

исходных данных является использование этих точек в качестве центров. Таким образом, для набора из m точек строится m радиальных базисных функций.

$$b_i(\mathbf{x}) = \psi\left(\|\mathbf{x} - \mathbf{x}^{(i)}\|\right) \quad \text{для } i \in \{1, ..., m\}. \tag{14.25}$$

Соответствующая базисная матрица $m \times m$ всегда является полуопределенной. Для построения радиальных базисных функций с известными центральными точками можно использовать алгоритм 14.5. Суррогатные модели с различными радиальными базисными функциями показаны на рис. 14.6.

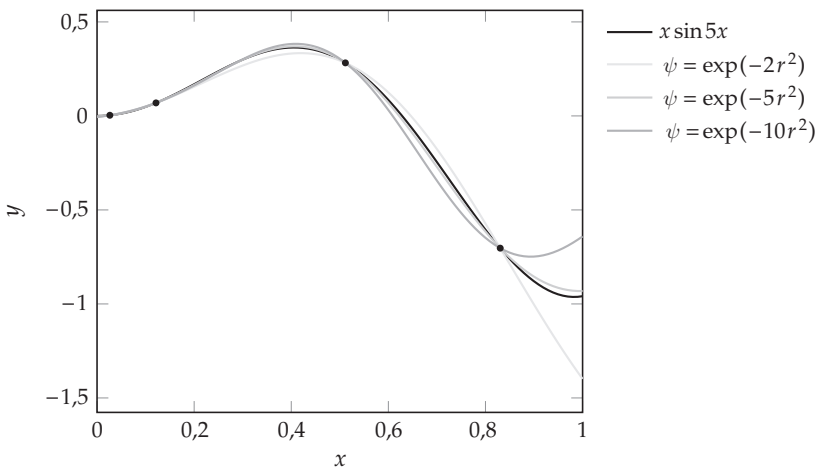


Рис. 14.6. Пример приближения функции $x \sin 5x$ по четырем выборкам без шума с помощью нескольких гауссовых радиальных базисных функций

Алгоритм 14.5. Метод получения списка базисных функций для заданной радиальной базисной функции `φ`, списка центров `C` и параметра `p` нормы L_p

```
radial_bases(ψ, C, p = 2) = [x -> ψ(norm(x - c, p)) for c in C]
```

14.4. Приближение целевых функций с шумом

Настроенная регрессионная модель будет проходить максимально близко к каждой расчетной точке. Если значения целевой функции содержат шум, то сложные модели могут оказаться чрезмерно извилистыми, стремясь пройти через каждую точку. Однако более гладкие приближения часто являются лучшими предикторами истинной целевой функции.

Задача базисной регрессии, указанная в уравнении (14.15), может быть дополнена, чтобы отдавать предпочтение более гладким решениям. К ошибке предсказания добавляется член регуляризации, отдающий предпочтение решениям

с меньшим весом. Результирующая проблема базисной регрессии с *регуляризацией* L_2 имеет вид:[5]

$$\min_{\boldsymbol{\theta}} \left\| \mathbf{y} - \mathbf{B}\boldsymbol{\theta} \right\|_2^2 + \lambda \left\| \boldsymbol{\theta} \right\|_2^2, \tag{14.26}$$

где $\lambda \geq 0$ — параметр сглаживания, а $\lambda = 0$ не приводит к сглаживанию.

Оптимальный вектор параметров определяется как:[6]

$$\boldsymbol{\theta} = \left(\mathbf{B}^{\mathrm{T}}\mathbf{B} + \lambda \mathbf{I} \right)^{-1} \mathbf{B}^{\mathrm{T}}\mathbf{y}, \tag{14.27}$$

где $\mathbf{I}$ — это единичная матрица.

Регрессию с регуляризацией L_2 реализует алгоритм 14.6. Суррогатные модели с различными радиальными базисными функциями, настроенными для зашумленных выборок, показаны на рис. 14.7.

Алгоритм 14.6. Метод регрессии при наличии шума, где `λ` — сглаживающий член. Метод возвращает суррогатную модель, настроенную по расчетным точкам `X` и значениям соответствующей целевой функции `y` с помощью регрессии на основе базисных функций

```
function regression(X, y, bases, λ)
    B = [b(x) for x in X, b in bases]
    θ = (B'B + λ * I) \ B'y
    return x -> sum(θ[i] * bases[i](x) for i in 1 : length(θ))
end
```

14.5. Выбор модели

До сих пор мы обсуждали, как обучить конкретную модель по данным. В этом разделе объясняется, как выбрать модель для использования. Как правило, мы хотим минимизировать ошибку обобщения, которая является мерой ошибки модели во всем пространстве параметров проектирования, включая точки, которые могут не входить в множество данных, используемых для обучения модели. Один из способов измерения ошибки обобщения состоит в том, чтобы использовать ожидаемую квадратичную ошибку своих прогнозов:

$$\varepsilon_{\text{gen}} = \mathbb{E}_{\mathbf{x} \sim \mathcal{X}} \left[\left(f(\mathbf{x}) - \hat{f}(\mathbf{x}) \right)^2 \right]. \tag{14.28}$$

[5] Другие L_p-нормы, описанные в приложении В.4, также могут быть использованы. Использование нормы L_1 будет поощрять разреженные решения с менее влиятельными весами компонентов, равными нулю, что может быть полезно при определении важных базовых функций.

[6] Матрица $\mathbf{B}^{\mathrm{T}}\mathbf{B} + \lambda\mathbf{I}$ не всегда обратима. При достаточно большом λ мы всегда можем получить обратимую матрицу.

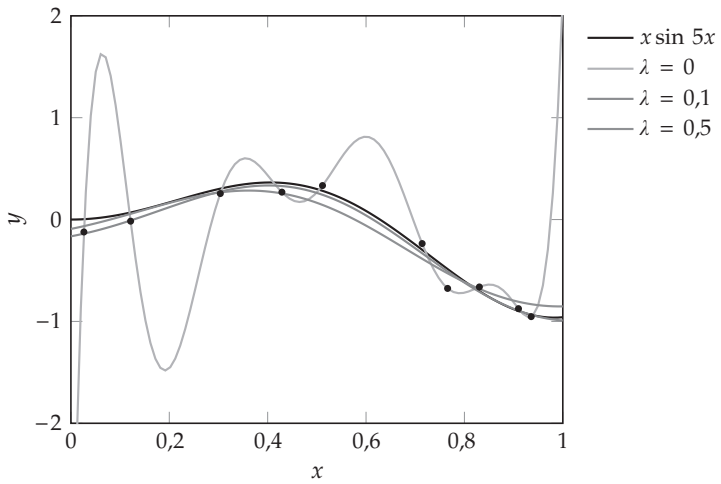


Рис. 14.7. Несколько различных гауссовых радиальных базисных функций с нулевым математическим ожиданием и среднеквадратическим отклонением, равным 0,1, используемых для приближения функции $x\sin 5x$ на основе 10 выборок с шумом и радиальной базисной функцией $y = \exp(-5r^2)$

Конечно, мы не можем вычислить эту ошибку обобщения точно, потому что она требует знания функции, которую мы пытаемся аппроксимировать. Заманчиво оценить ошибку обобщения модели по ошибке обучения. Одним из способов измерения ошибки обучения является использование среднеквадратичной ошибки (MSE) модели, оцененной по m выборкам, использованным для обучения:

$$\varepsilon_{\text{train}} = \frac{1}{m}\sum_{i=1}^{m}\left(f\left(\mathbf{x}^{(i)}\right) - \hat{f}\left(\mathbf{x}^{(i)}\right)\right)^2. \tag{14.29}$$

Тем не менее хорошие результаты на обучающих данных не обязательно соответствуют малой ошибке обобщения. Сложные модели могут уменьшить ошибку в обучающем множестве, но они могут не обеспечивать хорошие прогнозы в других точках пространства параметров проектирования, как показано в примере 14.1.[7]

Пример 14.1. Сравнение ошибки обучения и обобщения при варьировании степени полиномиальной суррогатной модели

Рассмотрите возможность аппроксимации целевой функции с помощью полиномов различной степени

$$f(x) = x/10 + \sin(x)/4 + \exp(-x^2).$$

[7] Основная задача машинного обучения — балансирование сложности модели с целью избежать *переобучения* на обучающих данных [108].

Ниже мы строим полиномиальные суррогатные модели различной степени, используя те же девять результатов вычислений, равномерно распределенных по [–4, 4]. Также показаны ошибки обучения и обобщения, где обобщение рассчитывается по отрезку [–5, 5].

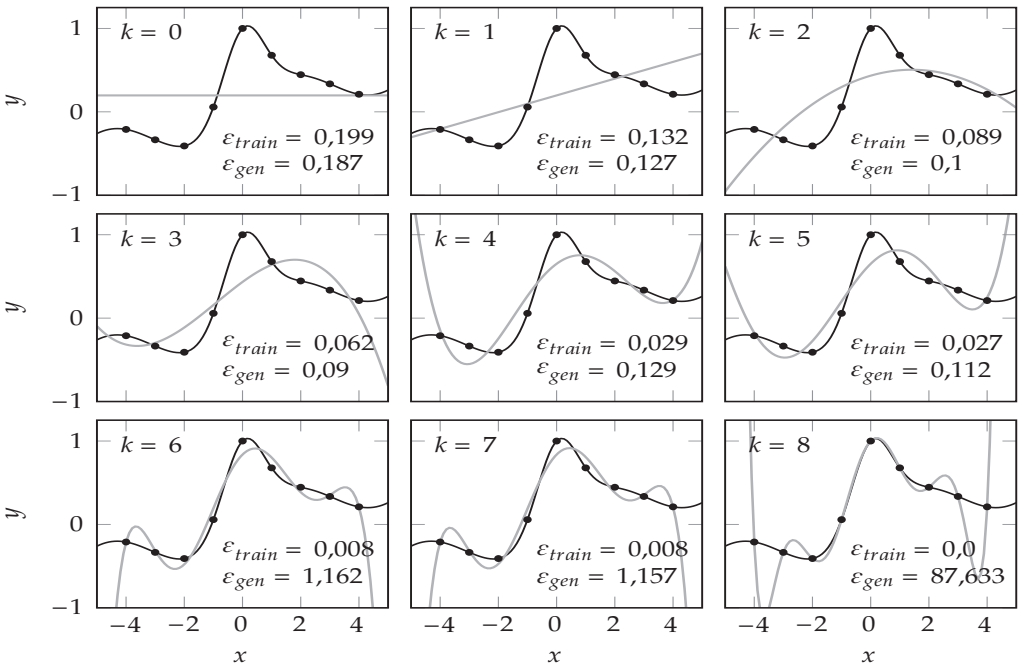


График показывает, что ошибка обобщения высока как для очень низких, так и для высоких значений k, а что ошибка обучения уменьшается при увеличении степени полинома. Полиномы высокой степени являются особенно плохими предикторами для параметра за пределами отрезка [–4, 4].

В этом разделе рассматриваются несколько методов оценки ошибки обобщения. Эти методы обучают и тестируют подмножества данных. Введем тип **`Train Test`** (алгоритм 14.7), который содержит список обучающих индексов и список тестовых заданий. Метод **`fit`** принимает обучающее множество и создает модель. Метрика **`metric`** принимает модель и набор тестов и создает метрику, такую как среднеквадратическая ошибка. Метод **`train_and_validate`** (алгоритм 14.7) является вспомогательной функцией для обучения, а затем оценки модели. Хотя при оценке ошибки обобщения мы оцениваем подмножества данных, после выбора модели мы можем обучаться на полном наборе данных.

Алгоритм 14.7. Вспомогательный тип и метод для обучения модели, а затем проверки ее по метрике. Здесь `train` и `test` — это списки индексов в обучающих данных, `X` — это список расчетных точек, `y` — вектор соответствующих оценок функций, `tt` — параметр, задающий разделение выборок на обучающие и тестовые, `fit` — функция обучения модели, а функция `metric` оценивает модель на тестовом наборе для получения оценки ошибки обобщения

```
struct TrainTest
    train
    test
end
function train_and_validate(X, y, tt, fit, metric)
    model = fit(X[tt.train], y[tt.train])
    return metric(model, X[tt.test], y[tt.test])
end
```

14.5.1. Отложенные множества

Простым подходом к оценке ошибки обобщения является метод отложенных множеств, при котором имеющиеся данные разбиваются на *тестовое множество* $\mathcal{D}_h$ с h выборками и *обучающее множество* $\mathcal{D}_t$, состоящее из всех оставшихся $m - h$ выборок, как показано на рис. 14.8. Обучающее множество используется для настроек параметров модели. Отложенное тестовое множество не используется во время обучения модели и, следовательно, может использоваться для оценки ошибки обобщения.

Используются разные коэффициенты разделения, обычно в диапазоне от 50% для обучения и 50% для тестирования до 90% для обучения и 10% для тестирования, в зависимости от размера и характера набора данных. Использование слишком малого количества выборок для обучения может привести к плохим приближениям (рис. 14.9), тогда как использование слишком большого количества выборок приведет к плохим оценкам обобщения.

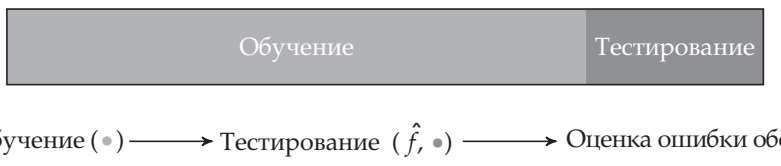


Рис. 14.8. Метод отложенных множеств (слева) разбивает данные на обучающее и тестовое множества

Контрольная ошибка модели $\hat{f}$ на обучающем множестве составляет

$$\varepsilon_{\text{holdout}} = \frac{1}{h} \sum_{(\mathbf{x},y)\in\mathcal{D}_h} \left(y - \hat{f}(\mathbf{x})\right)^2. \tag{14.30}$$

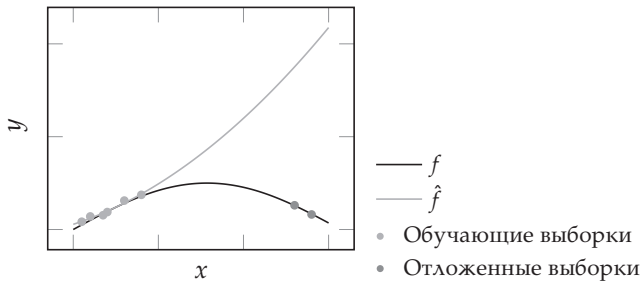


Рис. 14.9. Неудачное разделение на обучающее и тестовое множества может привести к снижению качества модели

Алгоритм 14.8. Метод случайного разделения m выборок на обучающее и отложенное множества, где h выборок назначаются в отложенное множество

```
function holdout_partition(m, h = div(m, 2))
    p = randperm(m)
    train = p[(h + 1) : m]
    holdout = p[1 : h]
    return TrainTest(train, holdout)
end
```

Даже если соотношение между множествами является фиксированным, контрольная ошибка будет зависеть от конкретного выбранного разделения на обучающее и отложенное множества. Выбор этих множеств наугад (алгоритм 14.8) даст только точечную оценку. При случайном выборе (алгоритм 14.9) мы применяем метод отложенного множества несколько раз со случайно выбранными разделениями на об учающее и тестовое множества.

Предполагаемая ошибка обобщения является средним значением для всех прогонов.[8] Поскольку множества для перекрестной проверки выбираются случайным образом, этот метод не гарантирует, что мы проверим все точки данных.

Алгоритм 14.9. Метод случайного выбора множеств, используемый для получения оценок математического ожидания и стандартного отклонения для ошибки обобщения модели с использованием **k_max** прогонов метода отложенного множества

```
function random_subsampling(X, y, fit, metric;
    h = div(length(X), 2), k_max = 10)
    m = length(X)
```

[8] Для оценки стандартного отклонения расчетной ошибки обобщения может использоваться стандартное отклонение по всем прогонам.

```
    mean(train_and_validate(X, y, holdout_partition(m, h),
        fit, metric) for k in 1 : k_max)
end
```

14.5.2. Перекрестная проверка

Разделение на обучающее и тестовое множества бывает расточительным, поскольку настройка модели может использовать только часть данных. Лучшие результаты часто достигаются с помощью *k-групповой перекрестной проверки* (k-fold cross-validation).[9] Здесь исходное множество данных $\mathcal{D}$ случайным образом разбивается на k подмножеств $\mathcal{D}_1$, $\mathcal{D}_2$, ..., $\mathcal{D}_k$ одинакового или приблизительно равного размера, как показано на рис. 14.10 и реализовано в алгоритме 14.10. Затем мы обучаем k моделей, по одной на каждом из $k-1$ подмножеств, и используем отложенное множество для оценки ошибки обобщения. Оценка ошибки обобщения на основе перекрестной проверки является средней ошибкой обобщения по всем группам:[10]

$$\begin{aligned}\varepsilon_{\text{cross-validation}} &= \frac{1}{k}\sum_{i=1}^{k}\varepsilon^{(i)}_{\text{cross-validation}},\\ \varepsilon^{(i)}_{\text{cross-validation}} &= \frac{1}{\left|\mathcal{D}^{(i)}_{\text{test}}\right|}\sum_{(\mathbf{x},y)\in D^{(i)}_{\text{test}}}\left(y-\hat{f}^{(i)}(\mathbf{x})\right)^2,\end{aligned} \tag{14.31}$$

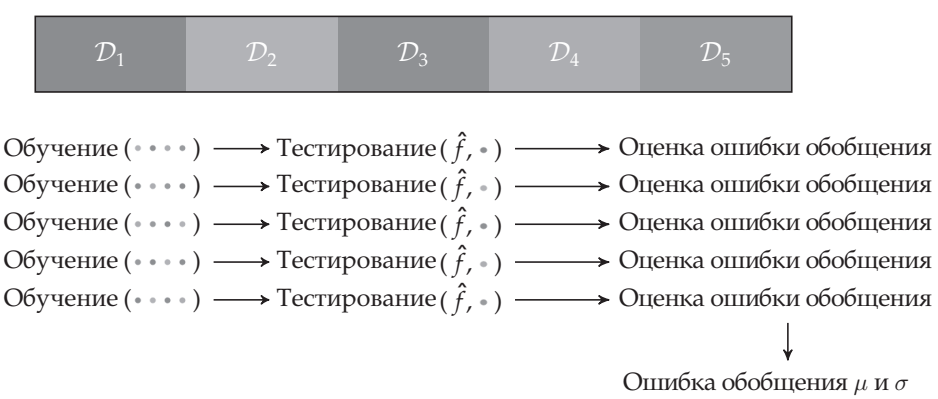


Рис. 14.10. Перекрестная проверка разделяет данные на множества одинакового размера. Каждое из этих множеств один раз используется в качестве отложенного множества. Здесь мы показываем 5-групповую перекрестную проверку

[9] Эту величину также называют *оценкой вращения* (rotation estimation).

[10] Как и в случае выбора случайного подмножества, оценку дисперсии можно получить из стандартного отклонения.

Алгоритм 14.10. Метод **k_fold_cross_validation_sets** создает наборы, необходимые для *k*-групповой перекрестной проверки по **m** выборкам. Метод **cross_validation_estimate** вычисляет среднее значение оценки ошибки обобщения на основе обучающего и контрольного множества, входящих в список обучающих и контрольных множеств **sets**. Другими переменными являются список расчетных точек **X**, соответствующие значения целевой функции **y**, функция **fit**, которая обучает суррогатную модель, и функция **metric**, которая оценивает модель на наборе данных

```
function k_fold_cross_validation_set(m, k)
   perm = randperm(m)
   sets = TrainTest[]
   for i = 1 : k
       validate = perm[i : k : m];
       train = perm[setdiff(1 : m, i : k : m)]
       push!(sets, TrainTest(train, validate))
    end
    return sets end
function cross_validation_estimate(X, y, sets, fit, metric)
    mean(train_and_validate(X, y, tt, fit, metric)
          for tt in sets)
end
```

Перекрестная проверка также зависит от конкретного разделения данных. Исключением является *скользящая перекрестная проверка* (leave-one-out cross-validation), разделение которой является детерминированным. Она осуществляет обучение на максимально возможном количестве данных, но требует обучения m моделей.[11] Усреднение по всем возможным разделам, известное как полная перекрестная проверка, часто слишком дорого. Хотя можно выполнить усреднение нескольких прогонов перекрестной проверки, более распространено усреднение моделей из одного раздела перекрестной проверки [150].

Перекрестная проверка продемонстрирована в примере 14.2.

Пример 14.2. Перекрестная проверка, используемая для настройки гиперпараметров

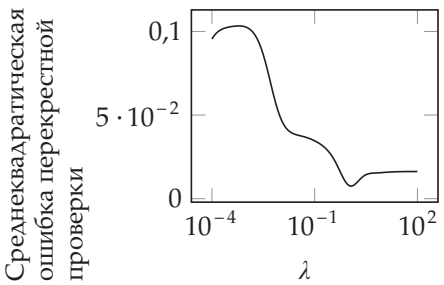
Предположим, мы хотим аппроксимировать целевую функцию с шумом, используя радиальные базисные функции с гиперпараметром шума λ (раздел 14.4). Для определения λ мы можем использовать перекрестную проверку.

[11] Как и в случае выбора случайного подмножества, оценку дисперсии можно получить из стандартного отклонения.

Дано десять выборок из целевой функции с шумом. На практике целевая функция обычно неизвестна, но в этом примере используется функция

$$f(x) = \sin 2x \cos 10x + \varepsilon/10,$$

где $x \in [0, 1]$ и ε — случайный шум с нулевым математическим ожиданием и единичной дисперсией, $\varepsilon \sim \mathcal{N}(0, 1)$.




```
srand(0)
f = x -> sin(2x) * cos(10x)
X = rand(10)
y = f.(X) + randn(length(X)) / 10
```

Мы будем использовать три раза, назначенные случайным образом:

```
sets = k_fold_cross_validation_sets(length(X), 3)
```

Далее реализуем метрику. Используем среднеквадратичную ошибку:

```
metric = (f, X, y) -> begin
   m = length(X)
   return sum((f(X[i]) - y[i]) ^ 2 for i in m) / m
end
```

Теперь перебираем различные значения λ и аппроксимируем различные радиальные базисные функции. Будем использовать гауссовский радиальный базис. Перекрестная проверка применяется, чтобы получить оценку MSE для каждого значения:

```
λs = logspace(-4, 2, 101)
es = []
for λ in λs
    fit = (X, y) -> regression(X, y, radial_bases(r -> exp(-5r ^ 2), X), λ)
    push!(es, cross_validation_estimate(X, y, sets, fit, metric)[1])
end
```

Полученная кривая имеет минимум при $\lambda \approx 0{,}2$.

14.5.3. Бутстрэп

Бутстрэп (bootstrap) [44] использует несколько выборок начальной загрузки, которые состоят из m индексов в наборе данных размера m, независимо выбираемых случайным образом. Индексы выбираются с заменой, поэтому некоторые индексы могут выбираться несколько раз, а некоторые индексы могут не выбираться вообще, как показано на рис. 14.11. Бутстрэп-выборка используется для обучения модели, которая затем оценивается на исходном обучающем множестве. Метод получения бутстрэп-выборок приведен в алгоритме 14.11.

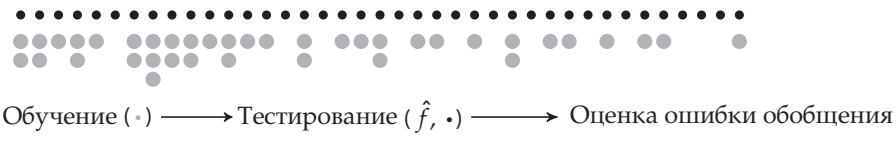


Рис. 14.11. Одна бутстрэп-выборка состоит из m индексов в наборе данных, извлеченных с заменой. Бутстрэп-выборка используется для обучения модели, которая оценивается по полному множеству данных для получения оценки ошибки обобщения

Алгоритм 14.11. Метод для получения `b` бутстрэп-выборок, каждая из которых извлечена из множества данных размера `m`

```
bootstrap_sets(m, b) = [TrainTest(rand(1 : m, m), 1 : m) for i in 1 : b]
```

Если создано b бутстрэп-выборок, оценка ошибки обобщения метолда бутстрэп является средним значением соответствующих оценок ошибок обобщения

$$\varepsilon_{\text{boot}} = \frac{1}{b}\sum_{i=1}^{b}\varepsilon_{\text{test}}^{(i)} = \tag{14.32}$$

$$= \frac{1}{m}\sum_{j=1}^{m}\frac{1}{b}\sum_{i=1}^{b}\left(y^{(j)} - \hat{f}^{(i)}\left(\mathbf{x}^{(j)}\right)\right)^2, \tag{14.33}$$

где $\hat{f}^{(i)}$ — модель, соответствующая i-й бутстрэп-выборки. Метод бутстрэп реализован в алгоритме 14.12.

Алгоритм 14.12. Метод вычисления оценки погрешности обобщения метода бутстрэп по списку множеств **`trainvalidate`**, содержащихся в аргументе **`sets`**. Другими переменными являются список расчетных точек **`X`**, соответствующие значения целевой функции **`y`**, функция **`fit`**, которая обучает суррогатную модель, и функция **`metric`**, оценивающая модель на наборе данных

```
function bootstrap_estimate(X, y, sets, fit, metric)
    mean(train_and_validate(X, y, tt, fit, metric) for tt in sets)
end
```

Ошибка метода бутстрэп в уравнении (14.32) определяет точность модели на точках данных, по которым она обучалась. *Скользящая оценка бутстрэп-метода* (leave-one-out bootstrap estimate) —устраняет этот источник систематической ошибки, оценивая только обученные модели на отложенных данных:

$$\varepsilon_{\text{leave-one-out}} = \frac{1}{m}\sum_{j=1}^{m}\frac{1}{c_{-j}}\sum_{i=1}^{b}\begin{cases}\left(y^{(j)} - \hat{f}^{(i)}\left(\mathbf{x}^{(j)}\right)\right)^2, \text{ если } j\text{-й индекс} \\ \quad \text{принадлежит } i\text{-й бутстрэп-выборке} \\ 0 \text{ — в противном случае.}\end{cases} \quad (14.34)$$

Здесь c_{-j} — количество бутстрэп-выборок, которые не содержат индекс j. Скользящий бутстрэп реализован в алгоритме 14.13.

Алгоритм 14.13. Способ вычисления оценки ошибки обобщения скользящего метода бутстрэп с помощью перекрестной проверки на основе обучающих множеств **sets**. Другими переменными являются список расчетных точек **X**, соответствующие значения целевой функции **y**, функция **fit**, которая обучает суррогатную модель, и функция **metric**, оценивающая модель на множестве данных

```
function leave_one_out_bootstrap_estimate(X, y, sets, fit, metric)
    m, b = length(X), length(sets)
    ε = 0.0
    models = [fit(X[tt.train], y[tt.train]) for tt in sets]
    for j in 1 : m
        c = 0
        δ = 0.0
        for i in 1 : b
            if j ∉ sets[i].train
                c += 1
                δ += metric(models[i], [X[j]], [y[j]])
            end
        end
        ε += δ/c
    end
    return ε/m
end
```

Вероятность того, что конкретный индекс не будет принадлежать бутстрэп-выборке, равна:

$$\left(1-\frac{1}{m}\right)^m \approx e^{-1} \approx 0{,}368. \tag{14.35}$$

Таким образом, ожидается, что бутстрэп-выборка будет иметь в среднем $0{,}632m$ разных индексов из исходного множества данных.

К сожалению, скользящая оценка бутстрэп привносит новое смещение из-за варьирования размеров тестовых множеств. Оценка *метода бутстрэп 0,632* (алгоритм 14.14) устраняет это смещение:[12]

$$\varepsilon_{0{,}632\text{-boot}} = 0{,}632\varepsilon_{\text{leave-one-out-boot}} + 0{,}368\varepsilon_{\text{boot}}. \tag{14.36}$$

Алгоритм 14.14. Метод получения оценки бутстрэп-метода 0,632 для точек данных `X`, значений целевой функции `y`, количества бутстрэп-выборок `b`, а также функций `fit` и `metric`

```
function bootstrap_632_estimate(X, y, sets, fit, metric)
    models = [fit(X[tt.train], y[tt.train]) for tt in sets]
    ε_loob = leave_one_out_bootstrap_estimate(X, y, sets, fit, metric)
    ε_boot = bootstrap_estimate(X, y, sets, fit, metric)
    return 0.632ε_loob + 0.368ε_boot
end
```

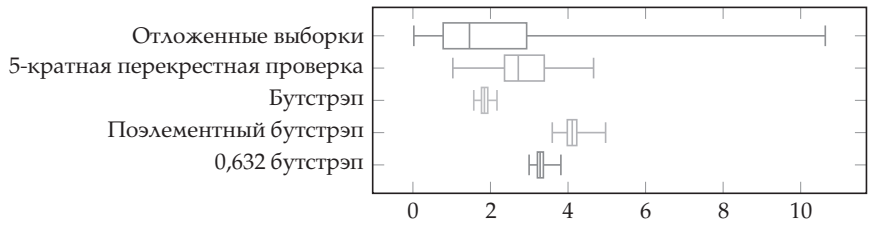
Результаты сравнения нескольких методов оценки ошибки обобщения приводятся в примере 14.3.

Пример 14.3. Сравнение методов оценки ошибки обобщения. Вертикальные линии на графике прямоугольника и усы указывают минимальный, максимальный, первый и третий квартили и медиану каждого метода оценки ошибки обобщения по 50 испытаниям

Рассмотрим десять равномерно распределенных выборок $f(x) = x^2 + \varepsilon/2$ по $x \in [3, 3]$, где ε — гауссовский шум с нулевым математическим ожиданием и единичной дисперсией. Мы хотим протестировать несколько различных методов оценки ошибок обобщения при обучении линейной модели по этим данным. Наша метрика — это *корень квадратный из среднеквадратической ошибки*.

Используемые методы — это метод отложенного множества с восемью обучающими выборками, пятикратная перекрестная проверка и бустраэп-методы, каждый с десятью бутстрэп-выборками. Каждый метод был выполнен 100 раз. Полученные статистические данные показаны ниже.

[12] Оценка метода бутстрэп 0,632 была введена в работе [45]. Ее вариант, оценка метода бутстрэп 0,632+, был представлен в работе [46].



14.6. Резюме

- Суррогатные модели представляют собой аппроксимации функций, которые можно оптимизировать вместо истинной, потенциально дорогой целевой функции.
- Многие суррогатные модели могут быть представлены с использованием линейной комбинации базисных функций.
- Выбор модели включает в себя компромиссное отклонение между моделями с низкой сложностью, которые не могут охватить важные тренды, и моделям с высокой сложностью, которые соответствуют шуму.
- Ошибка обобщения может быть оценена с использованием таких методов, как отложенное множество, *k*-кратная перекрестная проверка и бутстрэп.

14.7. Упражнения

Упражнение 14.1. Получите выражение, удовлетворяемое оптимумом регрессионного уравнения (14.8), приравняв градиент к нулю. Не инвертируйте никакие матрицы. Результирующее соотношение называется *нормальным уравнением*.

Упражнение 14.2. Когда следует использовать более сложную модель, например с полиномиальными признаками, по сравнению с более простой моделью, такой как линейная регрессия?

Упражнение 14.3. Задача линейной регрессии вида в уравнении (14.8) не всегда решается аналитически, и вместо этого используются методы оптимизации. Почему?

Упражнение 14.4. Предположим, что мы вычисляем нашу целевую функцию в четырех точках 1, 2, 3 и 4 и получаем 0, 5, 4 и 6. Мы хотим подобрать полиномиальную модель $f(x) = \sum_{i=0}^{k} \theta_i x^i$. Вычислите оценку скользящей перекрестной проверки для среднеквадратичной ошибки, когда k изменяется от 0 до 4. Какое значение k является лучшим и каковы наилучшие значения элементов $\boldsymbol{\theta}$ согласно этой метрике?

Глава 15

Вероятностные суррогатные модели

В предыдущей главе рассказывалось, как построить суррогатные модели по расчетным точкам. Используя суррогатные модели для оптимизации, иногда возникает необходимость количественно оценить уверенность в предсказаниях этих моделей. Один из способов — использовать вероятностный подход к суррогатному моделированию. Распространенной вероятностной суррогатной моделью является гауссовский процесс, который представляет распределение вероятностей по функциям. В этой главе будет объяснено, как использовать *гауссовские процессы*, чтобы вывести распределение по значениям различных проектных точек, учитывая значения ранее вычисленных расчетных точек. Мы обсудим, как включить информацию о градиенте, а также зашумленные измерения целевой функции. Поскольку предсказания, сделанные гауссовским процессом, регулируются набором параметров, мы покажем, как вывести эти параметры непосредственно из данных.

15.1. Нормальное распределение

Прежде чем вводить гауссовские процессы, рассмотрим некоторые соответствующие свойства многомерного нормального распределения.[1] Многомерное нормальное распределение имеет параметры — математическое ожидание $\boldsymbol{\mu}$ и ковариационную матрицу $\boldsymbol{\Sigma}$ Плотность вероятности в точке $\mathbf{x}$ задается формулой

$$N\left(\mathbf{x}\middle|\boldsymbol{\mu},\boldsymbol{\Sigma}\right)=\left(2\pi\right)^{-n/2}\left|\boldsymbol{\Sigma}\right|^{-1/2}\exp\left(-\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}\right)^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}\right)\right). \tag{15.1}$$

На рис. 15.1 показаны линии уровня функций плотности с различной ковариационной матрицей. Ковариационные матрицы всегда являются положительно полуопределенными.

Значение, выбранное из нормального распределения, записывается в виде

$$\mathbf{x}\sim\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma}). \tag{15.2}$$

[1] Одномерное нормальное распределение обсуждается в приложении В.7.

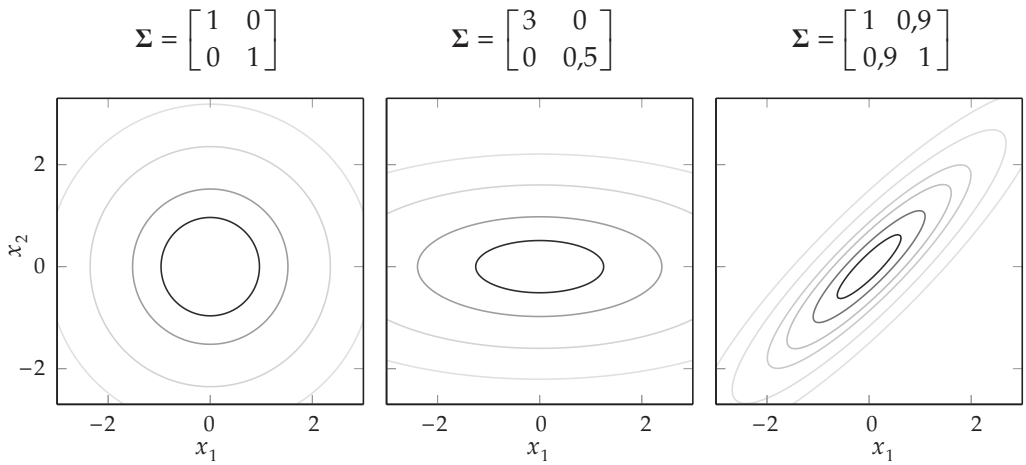


Рис. 15.1. Многомерные нормальные распределения с разными ковариационными матрицами

Две случайные величины $\mathbf{a}$ и $\mathbf{b}$, имеющие совместное нормальное распределение, можно записать в виде

$$\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_\mathbf{a} \\ \boldsymbol{\mu}_\mathbf{b} \end{bmatrix}, \begin{bmatrix} \mathbf{A} & \mathbf{C} \\ \mathbf{C}^\mathrm{T} & \mathbf{B} \end{bmatrix} \right). \tag{15.3}$$

Маргинальное распределение[2] для вектора случайных величин задается соответствующими математическим ожиданием и ковариацией:

$$\mathbf{a} \sim \mathcal{N}(\boldsymbol{\mu}_\mathbf{a}, \mathbf{A}),\ \mathrm{a} \sim \mathcal{N}(\boldsymbol{\mu}_\mathbf{b}, \mathbf{B}). \tag{15.4}$$

Условное распределение многомерного нормального распределения также можно представить в замкнутом виде:

$$\mathbf{a} \mid \mathbf{b} = \mathcal{N}(\boldsymbol{\mu}_{\mathbf{a}|\mathbf{b}}, \boldsymbol{\Sigma}_{\mathbf{a}|\mathbf{b}}), \tag{15.5}$$

$$\boldsymbol{\mu}_{\mathbf{a}|\mathbf{b}} = \boldsymbol{\mu}_\mathbf{a} + \mathbf{C}\mathbf{B}^{-1}(\mathbf{b} - \boldsymbol{\mu}_\mathbf{b}), \tag{15.6}$$

$$\boldsymbol{\Sigma}_{\mathbf{a}|\mathbf{b}} = \mathbf{A} - \mathbf{C}\mathbf{B}^{-1}\mathbf{C}^\mathrm{T}. \tag{15.7}$$

Пример 15.1 иллюстрирует, как определить маргинальное и условное распределения многомерного нормального распределения.

Пример 15.1. Маргинальное и условное распределения многомерного нормального распределения

[2] Маргинальное распределение — это распределение подмножества переменных при условии, что остальные исключены путем интегрирования, т.е. маргинализированы. Для распределения по двум переменным a и b маргинальное распределение по a имеет вид:

$$p(a) = \int p(a, b)\, db.$$

Например, рассмотрим

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{pmatrix} 3 & 1 \\ 1 & 2 \end{pmatrix}\right).$$

Маргинальное распределение по x_1 равно $\mathcal{N}(0, 3)$, а маргинальное распределение по x_2 — $\mathcal{N}(1, 2)$.

Условное распределение по x_1 при $x_2 = 2$ определяется следующим образом.

$$\mu_{x_1|x_2=2} = 0 + 1 \cdot 2^{-1} \cdot (2 - 1) = 0{,}5;$$

$$\Sigma_{x_1|x_2=2} = 3 - 1 \cdot 2^{-1} \cdot 1 = 2{,}5;$$

$$x_1 \big| (x_2 = 2) \sim \mathcal{N}(0{,}5;\ 2{,}5).$$

15.2. Гауссовские процессы

В предыдущей главе мы аппроксимировали целевую функцию f, используя суррогатную модельную функцию $\hat{f}$, приближенную по ранее найденным расчетным точкам. Специальный тип суррогатной модели, известный как *гауссовский процесс*, позволяет не только прогнозировать f, но и количественно определять неопределенность в этом прогнозе, используя распределение вероятностей.[3]

Гауссовский процесс — это распределение по функциям. Для любого конечного множества точек $\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(m)}\}$ ассоциированные с ними значения функций $\{y_1, ..., y_m\}$ распределяются по формуле

$$\begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} \sim N\left(\begin{bmatrix} m(\mathbf{x}^{(1)}) \\ \vdots \\ m(\mathbf{x}^{(m)}) \end{bmatrix}, \begin{bmatrix} k(\mathbf{x}^{(1)}, \mathbf{x}^{(1)}) & \cdots & k(\mathbf{x}^{(1)}, \mathbf{x}^{(m)}) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}^{(m)}, \mathbf{x}^{(1)}) & \cdots & k(\mathbf{x}^{(m)}, \mathbf{x}^{(m)}) \end{bmatrix}\right), \tag{15.8}$$

где $m(\mathbf{x})$ — *функция математического ожидания*, а $k(\mathbf{x}, \mathbf{x}')$ — *ковариационная функция*, или *ядро*.[4] Функция математического ожидания может представлять предварительные знания о функции. Ядро контролирует гладкость функций. Способы построения вектора математических ожиданий и ковариационной матрицы с использованием функции математического ожидания и ковариационной функции приведены в алгоритме 15.1.

---

[3] Более подробное введение в гауссовские процессы изложено в [126].

[4] Значением функции математического ожидания является математическое ожидание: $m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]$, а значением ковариационной функции является ковариация:

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))].$$

Алгоритм 15.1. Функция `μ` для построения вектора математических ожиданий при заданном списке расчетных точек и функция математического ожидания `m`, а также функция `Σ` для построения ковариационной матрицы при заданных одном или двух списках расчетных точек и ковариационной функции `k`

```
μ(X, m) = [m(x) for x in X]
Σ(X, k) = [k(x, x′) for x in X, x′ in X]
K(X, X′, k) = [k(x, x′) for x in X, x′ in X′]
```

Широко распространенной функцией ядра является *квадрат экспоненциального ядра* (squared exponential kernel), где

$$k(x, x') = \exp\left(-\frac{(x - x')^2}{2l^2}\right). \tag{15.9}$$

Параметр l соответствует так называемому *характеристическому масштабу длины*, рассматриваемой как расстояние, которое мы должны пройти в пространстве проектирования до тех пор, пока значение целевой функции существенно не изменится. Следовательно, большие значения l приводят к более гладким функциям. На рис. 15.2 показаны функции, выбранные из процесса Гаусса с функцией с нулевым математическим ожиданием и квадратом экспоненциального ядра с различными характеристическими масштабами длины.[5]

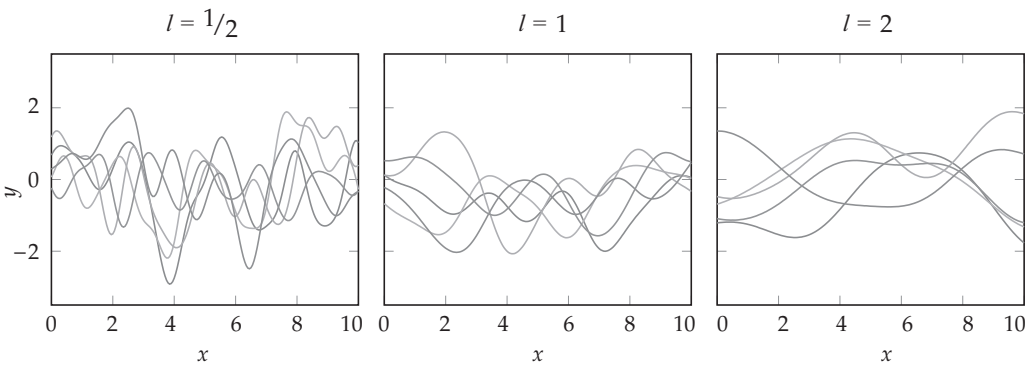


Рис. 15.2. Функции, извлеченные из гауссовских процессов с квадратами экспоненциальных ядер

Помимо квадрата экспоненты существует много других функций ядра,. Некоторые из них показаны на рис. 15.3. Многие функции ядра используют параметр r, который является расстоянием между $\mathbf{x}$ и $\mathbf{x}'$. Обычно используется евклидово расстояние. *Ядро Матерна* (Matérn kernel) применяет *гамма-функцию* Γ, реализо-

[5] Математическое определение характерной шкалы длины предоставлено в [126].

ванную функцией `gamma`, а $K_\nu(x)$ — *модифицированную функцию Бесселя второго рода*, реализованную функцией **`besselk(ν, x)`**. *Ядро нейронной сети* дополняет каждый расчетный вектор одним для простоты обозначения: $\bar{\mathbf{x}} = [1, x_1, x_2, ...]$ и $\bar{\mathbf{x}}' = [1, x_1', x_2', ...]$.

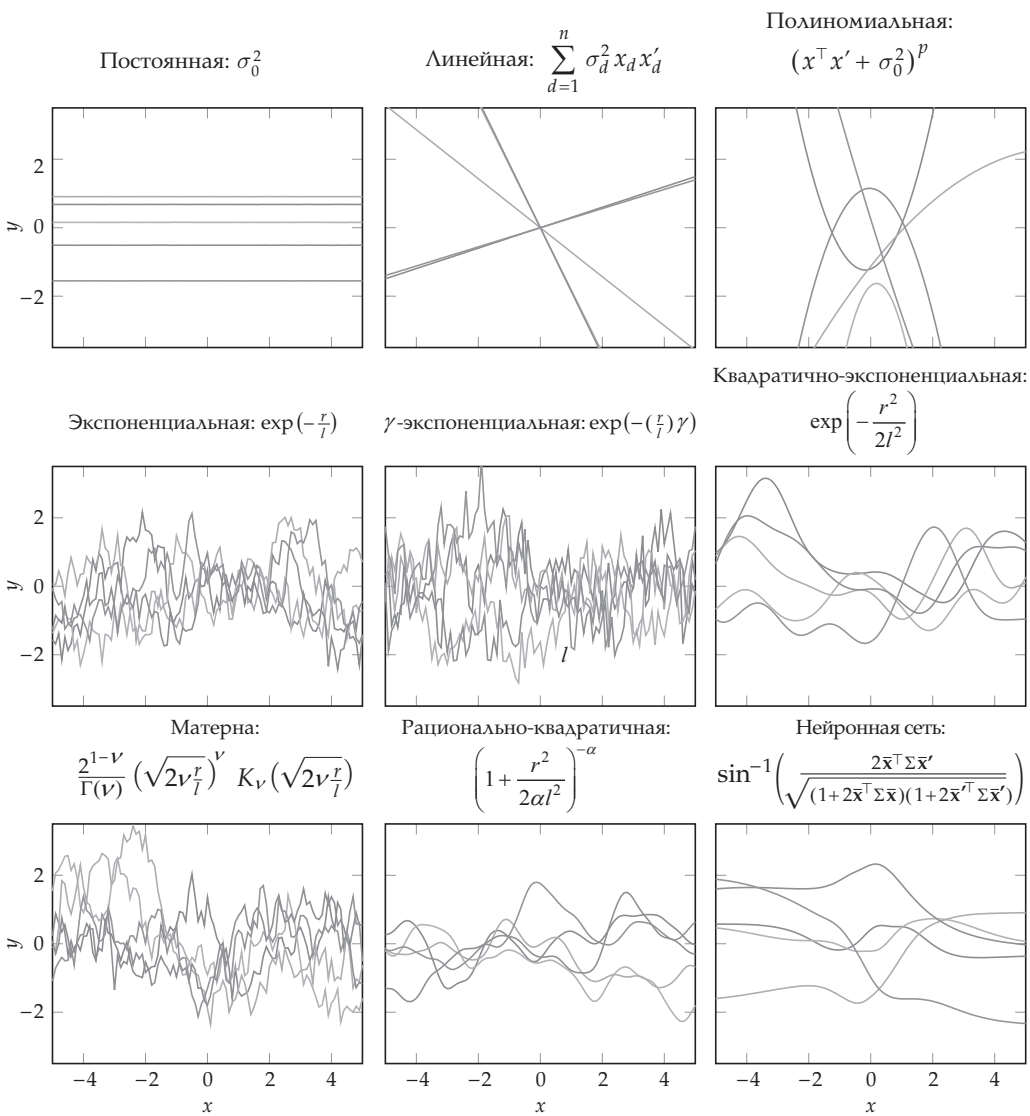


Рис. 15.3. Функции, извлеченные из гауссовских процессов с различными функциями ядра. Показаны функции для $\sigma_0^2 = \sigma_d^2 = l = 1$, $p = 2$, $\gamma = \upsilon = \alpha = 0{,}5$ и $\boldsymbol{\Sigma} = \mathbf{I}$

Для простоты эта глава будет сосредоточена на примерах гауссовских процессов в одномерных пространствах проектирования. Однако гауссовские процессы

могут быть определены в многомерном пространстве проектирования, как показано на рис. 15.4.

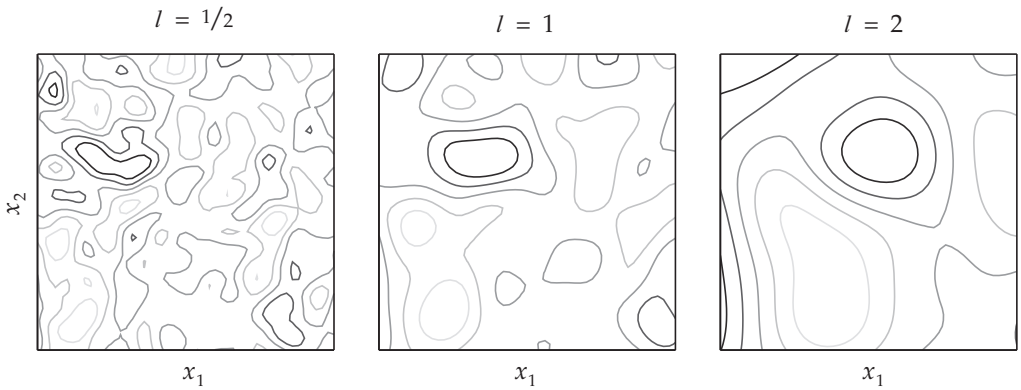


Рис. 15.4. Функции, извлеченные из гауссовских процессов

Как мы увидим в разделе 15.5, гауссовские процессы также могут включать в себя априорную зависимую дисперсию шума, обозначаемую как v. Таким образом, гауссовский процесс определяется функцией математического ожидания и ковариационной функцией, предыдущими расчетными точками и оценками их функций, а также дисперсией шума. Ассоциированный с ним тип приведен в алгоритме 15.2.

Алгоритм 15.2. Гауссовский процесс определяется функцией математического ожидания `m`, ковариационной функцией `k`, выборочными расчетными векторами `X` и их соответствующими значениями целевой функции `y` и дисперсией шума `ν`

```
mutable struct GaussianProcess
    m    # математическое ожидание
    k    # функция ковариации
    X    # расчетные точки
    y    # значения целевой точки
    ν    # дисперсия шума
end
```

15.3. Предсказание

Гауссовские процессы могут представлять распределения по функциям с использованием условных вероятностей. Предположим, у нас уже есть набор точек X и соответствующий вектор $\mathbf{y}$, но мы хотим предсказать значения $\hat{\mathbf{y}}$ в точках X^*. Совместное распространение таково:

$$\begin{bmatrix}\hat{\mathbf{y}}\\ \mathbf{y}\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\mathbf{m}\left(X^*\right)\\ \mathbf{m}(X)\end{bmatrix}, \begin{pmatrix}\mathbf{K}\left(X^*, X^*\right) & \mathbf{K}\left(X^*, X\right)\\ \mathbf{K}\left(X, X^*\right) & \mathbf{K}(X, X)\end{pmatrix}\right). \tag{15.10}$$

В приведенном выше уравнении мы используем функции $\mathbf{m}$ и $\mathbf{K}$, которые определены следующим образом:

$$\mathbf{m}(X) = \left[m\left(\mathbf{x}^{(1)}\right), ..., m\left(\mathbf{x}^{(n)}\right)\right], \tag{15.11}$$

$$\mathbf{K}(X, X') = \begin{bmatrix} k\left(\mathbf{x}^{(1)}, \mathbf{x}'^{(1)}\right) & ... & k\left(\mathbf{x}^{(1)}, \mathbf{x}'^{(m)}\right)\\ \vdots & \ddots & \vdots\\ k\left(\mathbf{x}^{(n)}, \mathbf{x}'^{(1)}\right) & ... & k\left(\mathbf{x}^{(n)}, \mathbf{x}'^{(m)}\right)\end{bmatrix}. \tag{15.12}$$

Условное распределение дается формулой

$$\hat{\mathbf{y}}|\mathbf{y} \sim \mathcal{N}\left(\underbrace{\mathbf{m}\left(X^*\right) + \mathbf{K}\left(X^*, X\right)\mathbf{K}(X,X)^{-1}\left(\mathbf{y} - \mathbf{m}(X)\right)}_{\text{Математическое ожидание}}, \underbrace{\mathbf{K}\left(X^*, X^*\right) - \mathbf{K}\left(X^*, X\right)\mathbf{K}(X, X)^{-1}\mathbf{K}\left(X, X^*\right)}_{\text{Ковариация}}\right). \tag{15.13}$$

Обратите внимание на то, что ковариация не зависит от $\mathbf{y}$. Это распределение часто называют апостериорным распределением.[6] Метод вычисления и выборки из апостериорного распределения, определенного гауссовским процессом, приведен в алгоритме 15.3.

Алгоритм 15.3. Функция `mvnrand` осуществляет выборку из многомерного нормального распределения с добавленным коэффициентом инфляции для предотвращения вычислительных проблем. Метод `rand` позволяет выбрать гауссовский процесс `GP` в заданных расчетных точках в матрице `X`

```
function mvnrand(μ, Σ, inflation = 1e-6)
   N = MvNormal(μ, Σ + inflation * I)
   return rand(N)
end
Base.rand(GP, X) = mvnrand(μ(X, GP.m), Σ(X, GP.k))
```

[6] На языке байесовской статистики апостериорное распределение — это распределение возможных ненаблюдаемых значений, обусловленных наблюдаемыми значениями.

Прогнозируемое математическое ожидание можно записать как функцию от $\mathbf{x}$:

$$\hat{\mu}(\mathbf{x}) = m(\mathbf{x}) + \mathbf{K}(\mathbf{x}, X)\mathbf{K}(X, X)^{-1}(\mathbf{y} - \mathbf{m}(X)) = \tag{15.14}$$

$$= m(\mathbf{x}) + \boldsymbol{\theta}^{\mathrm{T}}\mathbf{K}(X, \mathbf{x}), \tag{15.15}$$

где $\boldsymbol{\theta} = \mathbf{K}(X, X)^{-1}(\mathbf{y} - \mathbf{m}(X))$ можно вычислить один раз и повторно использовать для различных значений $\mathbf{x}$. Обратите внимание на сходство с суррогатными моделями в предыдущей главе. Ценность гауссовского процесса за пределами суррогатных моделей, обсуждавшихся ранее, состоит в том, что он также количественно определяет нашу неуверенность в предсказаниях.

Дисперсия предсказанного математического ожидания также может быть получена как функция от $\mathbf{x}$:

$$\hat{v}(\mathbf{x}) = \mathbf{K}(\mathbf{x}, \mathbf{x}) - \mathbf{K}(\mathbf{x}, X)\mathbf{K}(X, X)^{-1}\mathbf{K}(X, \mathbf{x}). \tag{15.16}$$

В некоторых случаях удобнее составлять уравнения в терминах стандартного отклонения, которое является квадратным корнем из дисперсии:

$$\hat{\sigma}(\mathbf{x}) = \sqrt{\hat{v}(\mathbf{x})}. \tag{15.17}$$

Стандартное отклонение имеет те же единицы измерения, что и математическое ожидание. По стандартному отклонению мы можем вычислить 95%-ную *доверительную область*, которая представляет собой интервал, содержащий 95% вероятностной массы, связанной с распределением по y при заданном $\mathbf{x}$. Для конкретного $\mathbf{x}$ 95%-ная доверительная область определяется как $\hat{\mu}(\mathbf{x}) \pm 1{,}96\hat{\sigma}(\mathbf{x})$. Можно использовать доверительный уровень, отличный от 95%, но для графиков в этой главе мы будем использовать 95%. На рис. 15.5 показан график доверительной области, связанной с гауссовым процессом, подходящей для четырех оценок функций.

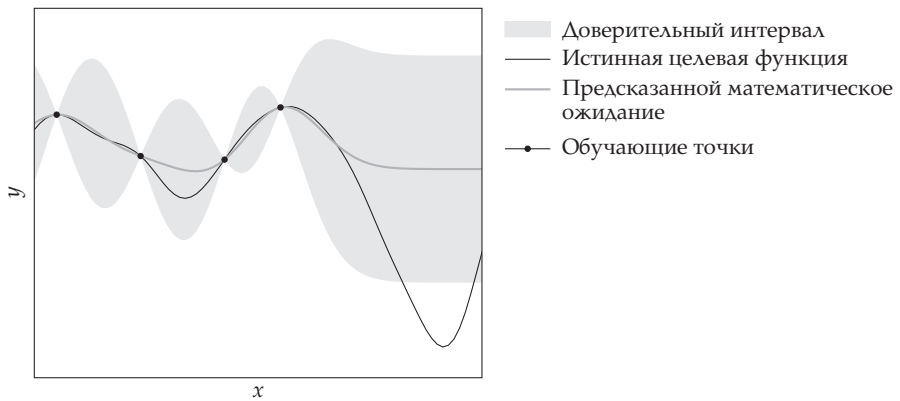


Рис. 15.5. Гауссовский процесс, использующий квадрат экспоненциального ядра и его 95%-ный доверительный интервал. Неопределенность возрастает по мере удаления от точки данных, и ожидаемое значение функции приближается к нулю, когда точка удаляется от данных

15.4. Информация о градиенте

В гауссовские процессы можно включить информацию о градиенте (см., например, [116]), расширив его определение, чтобы учесть как значение функции, так и ее градиент:

$$\begin{bmatrix} \mathbf{y} \\ \nabla \mathbf{y} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{m}_f \\ \mathbf{m}_\nabla \end{bmatrix}, \begin{bmatrix} \mathbf{K}_{ff} & \mathbf{K}_{f\nabla} \\ \mathbf{K}_{\nabla f} & \mathbf{K}_{\nabla\nabla} \end{bmatrix} \right), \tag{15.18}$$

где $\mathbf{y} \sim \mathcal{N}(\mathbf{m}_f, \mathbf{K}_{ff})$ — это традиционный гауссовский процесс, $\mathbf{m}_\nabla$ является ковариационной матрицей между значениями функции,[7] $\mathbf{K}_{\nabla f}$ — ковариационная матрица между градиентами функции и значениями функции, а $\mathbf{K}_{\nabla\nabla}$ — ковариационная матрица между градиентами функций.

Эти ковариационные матрицы строятся с использованием ковариационных функций. Линейность нормальных распределений обеспечивает следующие зависимости между ковариационными функциями:

$$k_{ff}(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}, \mathbf{x}'), \tag{15.19}$$

$$k_{\nabla f}(\mathbf{x}, \mathbf{x}') = \nabla_{\mathbf{x}} k(\mathbf{x}, \mathbf{x}'), \tag{15.20}$$

$$k_{f\nabla}(\mathbf{x}, \mathbf{x}') = \nabla_{\mathbf{x}'} k(\mathbf{x}, \mathbf{x}') \tag{15.21}$$

$$k_{\nabla\nabla}(\mathbf{x}, \mathbf{x}') = \nabla_{\mathbf{x}} \nabla_{\mathbf{x}'} k(\mathbf{x}, \mathbf{x}') \tag{15.22}$$

В примере 15.2 эти отношения используются для получения ковариационных функций высшего порядка для конкретного ядра.

Пример 15.2. Вывод ковариационных функций для гауссовского процесса с градиентными наблюдениями

Рассмотрим квадрат экспоненциальной ковариационной функции

$$k_{ff}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2} \left\| \mathbf{x} - \mathbf{x}' \right\|^2 \right).$$

Мы можем использовать уравнения (15.19)–(15.22), чтобы получить другие ковариационные функции, необходимые для использования гауссовских процессов с градиентной информацией:

$$k_{\nabla f}(\mathbf{x}, \mathbf{x}') = -(\mathbf{x}_i - \mathbf{x}_i') \exp\left(-\frac{1}{2} \left\| \mathbf{x} - \mathbf{x}' \right\|^2 \right),$$

$$k_{\nabla\nabla}(\mathbf{x}, \mathbf{x}')_{ij} = -\left((i = j) - (\mathbf{x}_i - \mathbf{x}_i')(\mathbf{x}_j - \mathbf{x}_j') \right) \exp\left(-\frac{1}{2} \left\| \mathbf{x} - \mathbf{x}' \right\|^2 \right).$$

---

[7] Аналогично среднему значению функции, значение $\mathbf{m}_\nabla$ часто равно нулю.

Напоминаем, что логические выражения, такие как $(i = j)$, возвращают единицу, если их значение — истина, и нуль, если ложь.

Прогнозирование может быть выполнено так же, как и при традиционном гауссовском процессе. Сначала мы строим совместное распределение

$$\begin{pmatrix} \hat{\mathbf{y}} \\ \mathbf{y} \\ \nabla\mathbf{y} \end{pmatrix} \sim N\left(\begin{bmatrix} \mathbf{m}_f\left(X^*\right) \\ \mathbf{m}_f\left(X\right) \\ \mathbf{m}_\nabla\left(X\right) \end{bmatrix}, \begin{bmatrix} \mathbf{K}_{ff}\left(X^*, X^*\right) & \mathbf{K}_{ff}\left(X^*, X\right) & \mathbf{K}_{f\nabla}\left(X^*, X\right) \\ \mathbf{K}_{ff}\left(X^*, X\right) & \mathbf{K}_{ff}\left(X, X\right) & \mathbf{K}_{f\nabla}\left(X, X\right) \\ \mathbf{K}_{\nabla f}\left(X, X^*\right) & \mathbf{K}_{\nabla f}\left(X, X\right) & \mathbf{K}_{\nabla\nabla}\left(X, X\right) \end{bmatrix} \right). \quad (15.23)$$

Для гауссовского процесса над n-мерными расчетными векторами при заданных значениях функций и градиентов и l тестовых точках блоки ковариантной матрицы имеют следующие размерности:

$$\begin{matrix} l \times l & l \times m & l \times nm \\ m \times l & m \times m & m \times nm \\ nm \times l & nm \times m & nm \times nm \end{matrix} \quad (15.24)$$

Построение такой ковариационой матрицы описано в примере 15.3.

Пример 15.3. Построение ковариационной матрицы для гауссовского процесса с градиентными наблюдениями

Предположим, что мы вычислили функцию и ее градиент в двух точках, $\mathbf{x}^{(1)}$ и $\mathbf{x}^{(2)}$, и хотим предсказать значение функции в точке $\hat{\mathbf{x}}$. Мы можем вывести совместное распределение по $\hat{\mathbf{y}}$, $\mathbf{y}$ и $\nabla\mathbf{y}$, используя гауссовский процесс. Ковариационная матрица имеет вид:

$$\begin{bmatrix} k_{ff}\left(\hat{\mathbf{x}}, \hat{\mathbf{x}}\right) & k_{ff}\left(\hat{\mathbf{x}}, \hat{\mathbf{x}}^{(1)}\right) & k_{ff}\left(\hat{\mathbf{x}}, \hat{\mathbf{x}}^{(2)}\right) & k_{f\nabla}\left(\hat{\mathbf{x}}, \hat{\mathbf{x}}^{(1)}\right)_1 & k_{f\nabla}\left(\hat{\mathbf{x}}, \hat{\mathbf{x}}^{(1)}\right)_2 & k_{f\nabla}\left(\hat{\mathbf{x}}, \hat{\mathbf{x}}^{(2)}\right)_1 & k_{f\nabla}\left(\hat{\mathbf{x}}, \hat{\mathbf{x}}^{(2)}\right)_2 \\ k_{ff}\left(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}\right) & k_{ff}\left(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}^{(1)}\right) & k_{ff}\left(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}^{(2)}\right) & k_{f\nabla}\left(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}^{(1)}\right)_1 & k_{f\nabla}\left(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}^{(1)}\right)_2 & k_{f\nabla}\left(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}^{(2)}\right)_1 & k_{f\nabla}\left(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}^{(2)}\right)_2 \\ k_{ff}\left(\hat{\mathbf{x}}^{(2)}, \hat{\mathbf{x}}\right) & k_{ff}\left(\hat{\mathbf{x}}^{(2)}, \hat{\mathbf{x}}^{(1)}\right) & k_{ff}\left(\hat{\mathbf{x}}^{(2)}, \hat{\mathbf{x}}^{(2)}\right) & k_{f\nabla}\left(\hat{\mathbf{x}}^{(2)}, \hat{\mathbf{x}}^{(1)}\right)_1 & k_{f\nabla}\left(\hat{\mathbf{x}}^{(2)}, \hat{\mathbf{x}}^{(1)}\right)_2 & k_{f\nabla}\left(\hat{\mathbf{x}}^{(2)}, \hat{\mathbf{x}}^{(2)}\right)_1 & k_{f\nabla}\left(\hat{\mathbf{x}}^{(2)}, \hat{\mathbf{x}}^{(2)}\right)_2 \\ k_{\nabla f}\left(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}\right)_1 & k_{\nabla f}\left(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}^{(1)}\right)_1 & k_{\nabla f}\left(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}^{(2)}\right)_1 & k_{\nabla\nabla}\left(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}^{(1)}\right)_{11} & k_{\nabla\nabla}\left(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}^{(1)}\right)_{12} & k_{\nabla\nabla}\left(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}^{(2)}\right)_{11} & k_{\nabla\nabla}\left(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}^{(2)}\right)_{12} \\ k_{\nabla f}\left(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}\right)_2 & k_{\nabla f}\left(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}^{(1)}\right)_2 & k_{\nabla f}\left(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}^{(2)}\right)_2 & k_{\nabla\nabla}\left(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}^{(1)}\right)_{21} & k_{\nabla\nabla}\left(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}^{(1)}\right)_{22} & k_{\nabla\nabla}\left(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}^{(2)}\right)_{21} & k_{\nabla\nabla}\left(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}^{(2)}\right)_{22} \\ k_{\nabla f}\left(\hat{\mathbf{x}}^{(2)}, \hat{\mathbf{x}}\right)_1 & k_{\nabla f}\left(\hat{\mathbf{x}}^{(2)}, \hat{\mathbf{x}}^{(1)}\right)_1 & k_{\nabla f}\left(\hat{\mathbf{x}}^{(2)}, \hat{\mathbf{x}}^{(2)}\right)_1 & k_{\nabla\nabla}\left(\hat{\mathbf{x}}^{(2)}, \hat{\mathbf{x}}^{(1)}\right)_{11} & k_{\nabla\nabla}\left(\hat{\mathbf{x}}^{(2)}, \hat{\mathbf{x}}^{(1)}\right)_{12} & k_{\nabla\nabla}\left(\hat{\mathbf{x}}^{(2)}, \hat{\mathbf{x}}^{(2)}\right)_{11} & k_{\nabla\nabla}\left(\hat{\mathbf{x}}^{(2)}, \hat{\mathbf{x}}^{(2)}\right)_{12} \\ k_{\nabla f}\left(\hat{\mathbf{x}}^{(2)}, \hat{\mathbf{x}}\right)_2 & k_{\nabla f}\left(\hat{\mathbf{x}}^{(2)}, \hat{\mathbf{x}}^{(1)}\right)_2 & k_{\nabla f}\left(\hat{\mathbf{x}}^{(2)}, \hat{\mathbf{x}}^{(2)}\right)_2 & k_{\nabla\nabla}\left(\hat{\mathbf{x}}^{(2)}, \hat{\mathbf{x}}^{(1)}\right)_{21} & k_{\nabla\nabla}\left(\hat{\mathbf{x}}^{(2)}, \hat{\mathbf{x}}^{(1)}\right)_{22} & k_{\nabla\nabla}\left(\hat{\mathbf{x}}^{(2)}, \hat{\mathbf{x}}^{(2)}\right)_{21} & k_{\nabla\nabla}\left(\hat{\mathbf{x}}^{(2)}, \hat{\mathbf{x}}^{(2)}\right)_{22} \end{bmatrix}$$

Условное распределение подчиняется тем же гауссовским соотношениям, что и в уравнении (15.13):

$$\hat{\mathbf{y}} \big| \mathbf{y}, \nabla\mathbf{y} \sim N\left(\boldsymbol{\mu}_\nabla, \boldsymbol{\Sigma}_\nabla\right), \quad (15.25)$$

где:

$$\boldsymbol{\mu}_{\nabla} = \mathbf{m}_f\left(X^*\right) + \begin{bmatrix} \mathbf{K}_{ff}\left(X, X^*\right) \\ \mathbf{K}_{\nabla f}\left(X, X^*\right) \end{bmatrix}^{\mathrm{T}} \times \\ \times \begin{bmatrix} \mathbf{K}_{ff}(X, X) & \mathbf{K}_{f\nabla}(X, X) \\ \mathbf{K}_{\nabla f}(X, X) & \mathbf{K}_{\nabla\nabla}(X, X) \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{y} - \mathbf{m}_f(X) \\ \nabla\mathbf{y} - \mathbf{m}_{\nabla}(X) \end{bmatrix}, \tag{15.26}$$

$$\boldsymbol{\Sigma}_{\nabla} = \mathbf{K}_{ff}\left(X^*, X^*\right) - \begin{bmatrix} \mathbf{K}_{ff}\left(X, X^*\right) \\ \mathbf{K}_{\nabla f}\left(X, X^*\right) \end{bmatrix}^{\mathrm{T}} \times \\ \times \begin{bmatrix} \mathbf{K}_{ff}(X, X) & \mathbf{K}_{f\nabla}(X, X) \\ \mathbf{K}_{\nabla f}(X, X) & \mathbf{K}_{\nabla\nabla}(X, X) \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{K}_{ff}\left(X, X^*\right) \\ \mathbf{K}_{\nabla f}\left(X, X^*\right) \end{bmatrix}. \tag{15.27}$$

На рис. 15.6 области, полученные при включении градиентных наблюдений, сравниваются с областями без градиентных наблюдений.

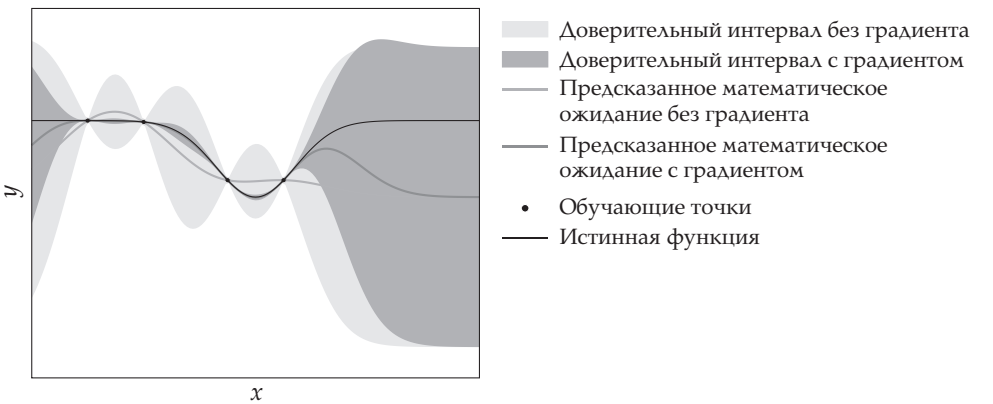


Рис. 15.6. Гауссовские процессы с информацией о градиенте и без нее с использованием квадратов экспоненциальных ядер. Включение информации о градиенте может значительно уменьшить доверительные интервалы

15.5. Информация о шуме

До сих пор мы предполагали, что целевая функция f является детерминированной. На практике, однако, оценки f могут включать в себя шум измерения, экспериментальную ошибку или числовое округление.

Мы можем смоделировать оценки шума как $y = f(\mathbf{x}) + z$, где f является детерминированной функцией, но z является гауссовским шумом с нулевым математи-

ческим ожиданием, т.е. $z \sim \mathcal{N}(0, \nu)$. Дисперсию шума можно настроить для контроля неопределенности.[8]

$$\begin{bmatrix} \hat{\mathbf{y}} \\ \mathbf{y} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{m}\left(X^*\right) \\ \mathbf{m}\left(X^*\right) \end{bmatrix}, \begin{bmatrix} \mathbf{K}\left(X^*, X^*\right) & \mathbf{K}\left(X^*, X\right) \\ \mathbf{K}\left(X, X^*\right) & \mathbf{K}(X, X) + \nu\mathbf{I} \end{bmatrix} \right) \tag{15.28}$$

с условным распределением:

$$\hat{\mathbf{y}} | \mathbf{y}, \nu \sim \mathcal{N}\left(\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*\right), \tag{15.29}$$

$$\boldsymbol{\mu}^* = \mathbf{m}\left(X^*\right) + \mathbf{K}\left(X^*, X\right)\left(\mathbf{K}(X, X) + \nu\mathbf{I}\right)^{-1}\left(\mathbf{y} - \mathbf{m}(X)\right), \tag{15.30}$$

$$\boldsymbol{\Sigma}^* = \mathbf{K}\left(X^*, X^*\right) - \mathbf{K}\left(X^*, X\right)\left(\mathbf{K}(X, X) + \nu\mathbf{I}\right)^{-1} \mathbf{K}\left(X, X^*\right). \tag{15.31}$$

Как показывают приведенные выше уравнения, учет гауссовского шума является простым, а последующее распределение может быть вычислено аналитически. Зашумленный гауссовский процесс показан на рис. 15.7. Прогнозирование для гауссовских процессов с зашумленными измерениями реализует алгоритм 15.4.

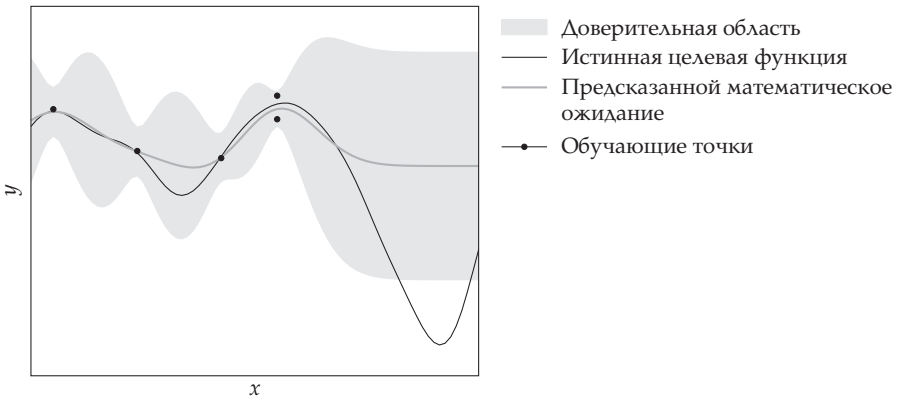


Рис. 15.7. Зашумленный гауссовский процесс с использованием квадрата экспоненциального ядра

Алгоритм 15.4. Метод получения предсказанных средних и стандартных отклонений по f при гауссовском процессе. Метод принимает гауссовский процесс `GP` и список точек `X_pred`, в которых можно оценить прогноз. Он возвращает среднее значение и дисперсию в каждой точке оценки

```
function predict(GP, X_pred)
    m, k, ν = GP.m, GP.k, GP.ν
    tmp = K(X_pred, GP.X, k) / (K(GP.X, GP.X, k) + ν * I)
```

[8] Для настройки дисперсии шума можно использовать методы, описанные в разделе 14.5.

```
    μp = μ(X_pred, m) + tmp * (GP.y - μ(GP.X, m))
    S = K(X_pred, X_pred, k) - tmp * K(GP.X, X_pred, k)
    νp = diag(S) .+ eps() # значение eps предотвращает вычислительные
                          # проблемы
    return (μp, νp)
end
```

15.6. Подгонка гауссовских процессов

Выбор ядра и параметров оказывает большое влияние на форму гауссовского процесса между расчетными точками. Ядра и их параметры могут быть выбраны с помощью перекрестной проверки, представленной в предыдущей главе. Вместо того чтобы минимизировать квадрат ошибки на тестовых данных, мы максимизируем вероятность данных.[9] Иначе говоря, мы ищем параметры, которые максимизируют вероятность значений функции, $p(\mathbf{y}|X, \boldsymbol{\theta})$. Правдоподобность данных — это вероятность того, что наблюдаемые точки были взяты из модели. Эквивалентно мы можем максимизировать логарифмическую вероятность, которая обычно предпочтительнее, потому что умножение малых вероятностей при вычислении вероятности может привести к чрезвычайно малым значениям. При заданном наборе данных $\mathcal{D}$ с n элементами логарифмическое правдоподобие определяется как

$$\begin{aligned}\log p(\mathbf{y}|X, \nu, \boldsymbol{\theta}) = &-\frac{n}{2}\log 2\pi - \frac{1}{2}\log\left|K_{\boldsymbol{\theta}}(X, X) + \nu\mathbf{I}\right| - \\ &-\frac{1}{2}\left(\mathbf{y} - \mathbf{m}_{\boldsymbol{\theta}}(X)\right)^{\mathrm{T}}\left(K_{\boldsymbol{\theta}}(X, X) + \nu\mathbf{I}\right)^{-1}\left(\mathbf{y} - \mathbf{m}_{\boldsymbol{\theta}}(X)\right),\end{aligned} \tag{15.32}$$

где математическое ожидание и ковариационные функции параметризованы параметром $\boldsymbol{\theta}$.

Предположим, что математическое значение равно нулю, т.е. $\mathbf{m}_{\boldsymbol{\theta}}(X) = \mathbf{0}$ и параметр $\boldsymbol{\theta}$ относится только к параметрам для ковариационной функции гауссовского процесса. Мы можем прийти к оценке *максимального правдоподобия* путем градиентного подъема. Тогда градиент определяется как

$$\frac{\partial}{\partial \boldsymbol{\theta}_j}\log p(\mathbf{y}|X, \boldsymbol{\theta}) = \frac{1}{2}\mathbf{y}^{\mathrm{T}}\mathbf{K}^{-1}\frac{\partial \mathbf{K}}{\partial \boldsymbol{\theta}_j}\mathbf{K}^{-1}\mathbf{y} - \frac{1}{2}\mathrm{tr}\left(\Sigma_{\boldsymbol{\theta}}^{-1}\frac{\partial \mathbf{K}}{\partial \boldsymbol{\theta}_j}\right), \tag{15.33}$$

где $\Sigma_{\boldsymbol{\theta}} = \mathbf{K}_{\boldsymbol{\theta}}(X, X) + \nu\mathbf{I}$. Выше мы использовали матричные производные отношения

[9] В качестве альтернативы можно максимизировать псевдослучайность, как показано в [126].

$$\frac{\partial \mathbf{K}^{-1}}{\partial \boldsymbol{\theta}_j} = -\mathbf{K}^{-1} \frac{\partial \mathbf{K}}{\partial \boldsymbol{\theta}_j} \mathbf{K}^{-1}, \tag{15.34}$$

$$\frac{\partial \log |\mathbf{K}|}{\partial \boldsymbol{\theta}_j} = \operatorname{tr}\left(\mathbf{K}^{-1} \frac{\partial \mathbf{K}}{\partial \boldsymbol{\theta}_j} \right), \tag{15.35}$$

где $\operatorname{tr}(\mathbf{A})$ обозначает *след* матрицы $\mathbf{A}$, определенной как сумма элементов на главной диагонали.

15.7. Резюме

- Гауссовские процессы — это распределения вероятностей по функциям.
- Выбор ядра влияет на гладкость функций, выбранных из гауссовского процесса.
- Многомерное нормальное распределение имеет аналитические условные и маргинальные распределения.
- Можно рассчитать математическое и стандартное отклонение прогноза целевой функции в конкретной расчетной точке с учетом ряда прошлых оценок.
- Можно включить информацию о градиенте, чтобы улучшить прогнозы значений целевой функции и ее градиента.
- Можно включить измерительный шум в гауссовский процесс.
- Можно подобрать параметры гауссовского процесса, используя максимальное правдоподобие.

15.8. Упражнения

Упражнение 15.1. Гауссовские процессы будут усложняться в процессе оптимизации по мере накопления большего количества выборок. Как это может стать преимуществом перед регрессионными моделями?

Упражнение 15.2. Как вычислительная сложность предсказания с гауссовским процессом увеличивается с числом точек данных m?

Упражнение 15.3. Рассмотрим функцию $f(x) = \sin x / (x^2 + 1)$ на отрезке $[-5, 5]$. Постройте 95%-ные доверительные границы для гауссовского процесса с информацией о производной, соответствующей оценкам в точках $\{-5; -2{,}5; 0; 2{,}5; 5\}$. Каково максимальное стандартное отклонение прогнозируемого распределения в диапазоне [–5, 5]? Сколько оценок функций, равномерно распределенных по

области, необходимо для того, чтобы гауссовский процесс без информации о производной достигал одинакового максимального прогнозирующего стандартного отклонения?

Предположим, что функции имеют нулевое математическое ожидание и наблюдения без шума, а также функции ковариации:

$$k_{ff}\left(x, x'\right) = \exp\left(-\frac{1}{2}\left\|x - x'\right\|_2^2\right),$$

$$k_{\nabla f}\left(x, x'\right) = \left(x' - x\right)\exp\left(-\frac{1}{2}\left\|x - x'\right\|_2^2\right),$$

$$k_{\nabla\nabla}\left(x, x'\right) = \left(\left(x - x'\right)^2 - 1\right)\exp\left(-\frac{1}{2}\left\|x - x'\right\|_2^2\right).$$

Упражнение 15.4. Выведите отношение

$$k_{f\nabla}\left(\mathbf{x}, \mathbf{x}'\right)_i = \operatorname{cov}\left(f\left(\mathbf{x}\right), \frac{\partial}{\partial x_i'} f\left(\mathbf{x}'\right)\right) = \frac{\partial}{\partial x_i'} k_{ff}\left(\mathbf{x}, \mathbf{x}'\right).$$

Упражнение 15.5. Предположим, что мы имеем многомерное нормальное распределение по двум переменным *a* и *b*. Покажите, что дисперсия условного распределения *a* при условии *b* не больше дисперсии маргинального распределения по *a*. Имеет ли это интуитивный смысл?

Упражнение 15.6. Допустим, что мы наблюдаем много выбросов, т.е. наблюдаем выборки, которые не попадают в доверительный интервал, заданный гауссовским процессом. Это означает, что выбранная нами вероятностная модель не подходит. Что можно сделать?

Упражнение 15.7. Рассмотрим выбор модели для пар оценки функций (x, y):

$$\{(1, 0), (2, -1), (3, -2), (4, 1), (5, 0)\}$$

Используйте скользящую перекрестную проверку, чтобы выбрать ядро, которое максимизирует вероятность предсказания отложенной пары с учетом гауссовского процесса, определенного на других парах в группе. Предположим, что функция имеет нулевое математическое ожидание и не имеет шума. Выберите одно из ядер:

$$\exp\left(-\left\|x - x'\right\|\right) \quad \exp\left(-\left\|x - x'\right\|^2\right) \quad \left(1 + \left\|x - x'\right\|\right)^{-1} \quad \left(1 + \left\|x - x'\right\|^2\right)^{-1} \quad \left(1 + \left\|x - x'\right\|\right)^{-2}.$$

Глава 16

Суррогатная оптимизация

В предыдущей главе объяснялось, как использовать вероятностную суррогатную модель, в частности гауссовский процесс, для вывода вероятностных распределений по истинной целевой функции. Эти распределения позволяют направить процесс оптимизации к более точным расчетным точкам [52]. В этой главе описываются распространенные методы выбора следующей расчетной точки. Методы, которые мы здесь обсуждаем, жадно оптимизируют различные метрики.[1] Мы также обсудим, как суррогатные модели могут быть использованы для безопасной оптимизации целевого показателя.

16.1. Исследование, основанное на предсказании

В исследовании, основанном на предсказаниях, выбирается точка минимума суррогатной функции. Примером такого подхода является поиск квадратичной аппроксимации, обсуждаемый в разделе 3.5. При поиске квадратичной аппроксимации мы используем квадратичную суррогатную модель для подбора последних трех точек метода интервалов, а затем выбираем точку минимума квадратичной функции.

Если мы используем суррогатную модель гауссовского процесса, оптимизация, основанная на прогнозировании, заставит нас выбрать точку минимума функции математического ожидания:

$$\mathbf{x}^{(m+1)} = \arg\min_{\mathbf{x}\in\mathcal{X}} \hat{\mu}(\mathbf{x}), \tag{16.1}$$

где $\hat{\mu}(\mathbf{x})$ — это прогнозируемое математическое ожидание гауссовского процесса в расчетной точке $\mathbf{x}$, полученное на основе предыдущих m расчетных точек. Этот процесс иллюстрирует рис. 16.1.

[1] Альтернатива жадной оптимизации состоит в том, чтобы сформулировать задачу как *частично наблюдаемый марковский процесс принятия решений* и запланировать заранее некоторое количество шагов, описанных в [93, 156].

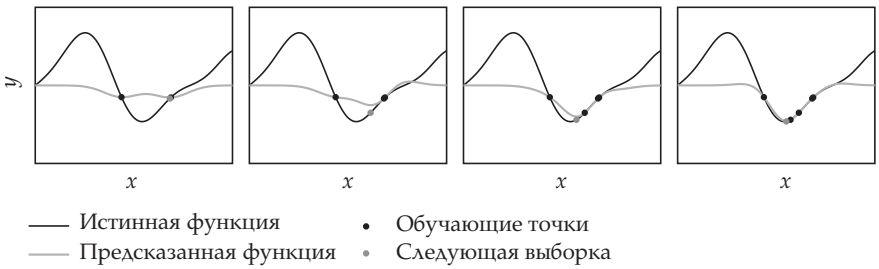


Рис. 16.1. Оптимизация на основе прогноза выбирает точку, которая минимизирует математическое ожидание целевой функции

Оптимизация на основе прогнозирования не учитывает неопределенность, и новые выборки могут быть созданы очень близко к существующим. Отбор выборок в местах, где мы уже уверены в значении целевой функции, является пустой тратой ресурсов.

16.2. Исследование на основе ошибок

Исследование на основе ошибок стремится повысить уверенность в истинной функции. Гауссовский процесс позволяет определить как математическое ожидание, так и стандартное отклонение в каждой точке. Большое стандартное отклонение указывает на низкую достоверность, т.е. сообщает о том, что выборки, полученные с помощью исследования на основе ошибок в расчетных точках, обладают максимальной неопределенностью.

Следующая выборочная точка отсчета определятся следующим образом:

$$\mathbf{x}^{(m+1)} = \arg\min_{\mathbf{x}\in\mathcal{X}} \hat{\sigma}(\mathbf{x}), \tag{16.2}$$

где $\hat{\sigma}(\mathbf{x})$ — стандартное отклонение гауссовского процесса в расчетной точке x, вычисленное на основе предыдущих m расчетных точек. Этот процесс иллюстрирует рис. 16.2.

Гауссовские процессы часто определены на всем пространстве $\mathbb{R}^n$. Задачи оптимизации с неограниченными допустимыми множествами всегда будут иметь высокую неопределенность вдали от выборочных точек, что делает невозможным уверенность в истинной базовой функции во всей области. Таким образом, исследование на основе ошибок должно быть ограничено замкнутой областью.

16.3. Исследование на основе нижнего доверительной границы

Хотя исследование, основанное на ошибках, уменьшает неопределенность целевой функции в целом, ее выборки часто находятся в областях, которые вряд ли

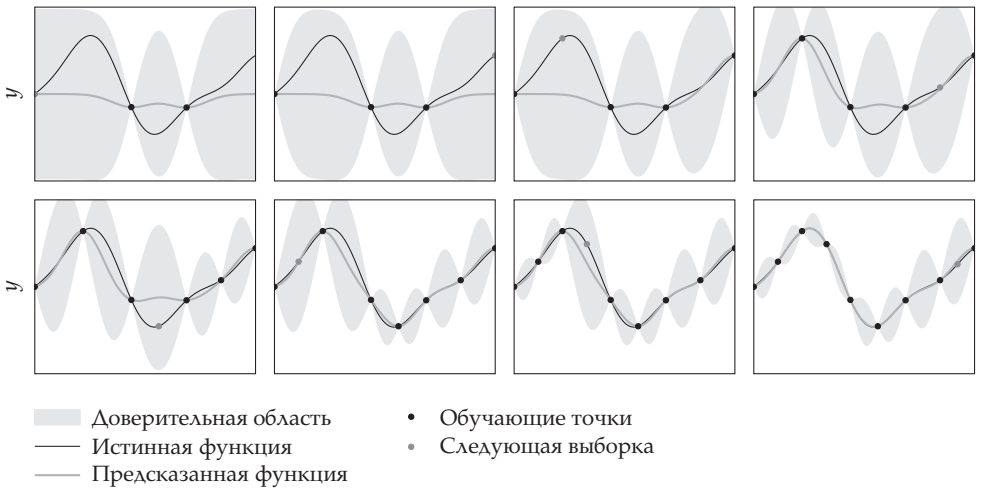


Рис. 16.2. Исследование на основе ошибок выбирает точку с максимальной неопределенностью

содержат глобальный минимум. *Исследование на основе нижней доверительной границы* ищет компромисс между жадной минимизацией, используемой оптимизацией на основе прогнозов, и снижением неопределенности, применяемым в исследовании на основе ошибок. Следующий пример минимизирует нижнюю доверительную границу целевой функции

$$LB(\mathbf{x}) = \hat{\mu}(\mathbf{x}) - \alpha\hat{\sigma}(\mathbf{x}), \tag{16.3}$$

где $\alpha \geq 0$ — константа, которая контролирует компромисс между *исследованием* и *эксплуатацией*. Исследование предполагает минимизацию неопределенности, а эксплуатация — минимизацию прогнозируемого математического ожидания. Оптимизации на основе прогноза соответствует значение $\alpha = 0$, а исследованию на основе ошибок — значение α, стремящееся к бесконечности. Этот процесс иллюстрирует рис. 16.3.

16.4. Исследование на основе вероятности улучшения

Иногда мы можем добиться более быстрой сходимости, выбирая расчетную точку, максимизируя вероятность того, что новая точка будет лучше, чем выборки, которые мы извлекали до сих пор. *Улучшение* функции, выбранной в точке $\mathbf{x}$ со значением $\mathbf{y} = f(\mathbf{x})$, составляет

$$I(y) = \begin{cases} y_{\min} - y, \text{ если } y < y_{\min}, \\ 0 \text{ в противном случае,} \end{cases} \tag{16.4}$$

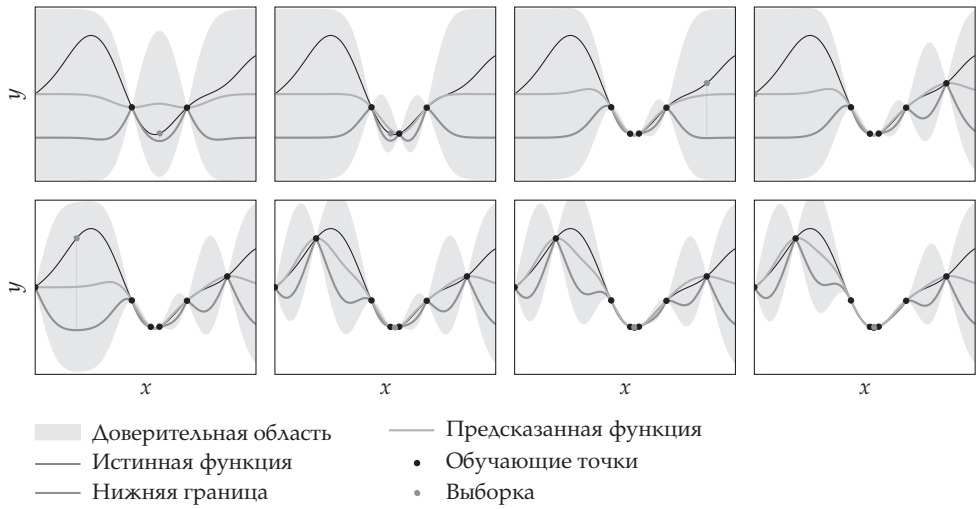


Рис. 16.3. Исследование на основе нижней доверительной границы связано с минимизацией неопределенности и минимизацией прогнозируемой функции

где $y_{\min}$ — минимальное значение, выбранное до сих пор.

Вероятность улучшения в точках, где $\hat{\sigma} > 0$

$$P\left(y < y_{\min}\right) = \int_{-\infty}^{y_{\min}} \mathcal{N}\left(y \middle| \hat{\mu}, \hat{\sigma}\right) dy = \tag{16.5}$$

$$= \Phi\left(\frac{y_{\min} - \hat{\mu}}{\hat{\sigma}}\right), \tag{16.6}$$

где Φ — кумулятивная функция стандартного нормального распределения (см. приложение В.7). Эти вычисления (алгоритм 16.1) показаны на рис. 16.4, а соответствующий процесс — на рис. 16.5. Если $\hat{\sigma} = 0$ (что происходит в точках, где измерения проведены без шума), вероятность улучшения равна нулю.

Алгоритм 16.1. Вычисление вероятности улучшения для заданного наилучшего значения `y_min`, математического ожидания `μ` и дисперсии `n`

```
prob_of_improvement (y_min, μ, σ) = cdf (Normal (μ, σ), y_min)
```

16.5. Исследование на основе ожидаемого улучшения

Оптимизация связана с поиском минимума целевой функции. Хотя максимизация вероятности улучшения будет иметь тенденцию уменьшать целевую функцию с течением времени, она не будет значительно повышаться с каждой итерацией.

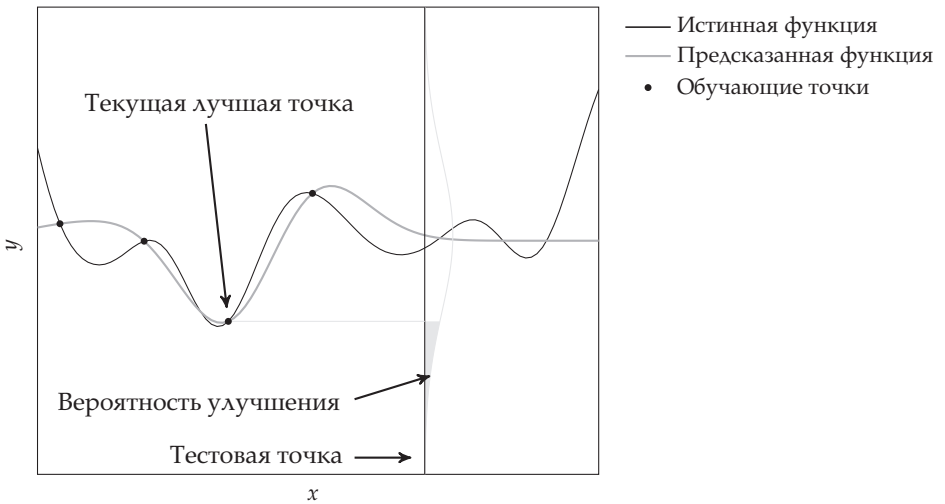


Рис. 16.4. Вероятность улучшения — это вероятность того, что вычисления в конкретной точке дадут лучший результат, чем до сих пор. На этом рисунке показана функция плотности вероятности, предсказанная в тестовой точке. Заштрихованная область соответствует кумулятивной вероятности меньше $y_{\min}$, соответствующей вероятности улучшения

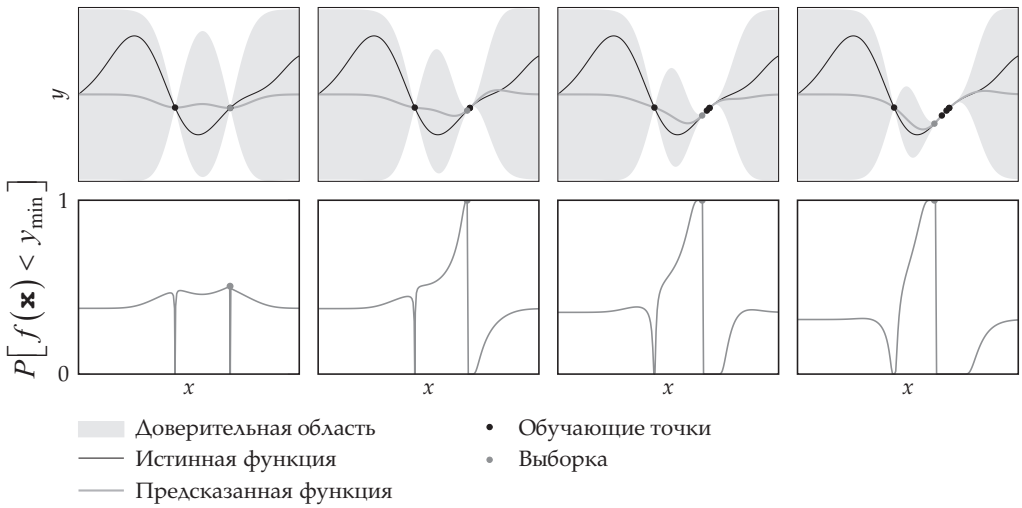


Рис. 16.5. Максимизация вероятности улучшения отбирает выборки, которые с наибольшей вероятностью дают более низкие значения целевой функции

Сосредоточим исследование на точках, которые максимизируют ожидаемое улучшение по сравнению с текущим лучшим значением функции.

Используя замены

$$z = \frac{y - \hat{\mu}}{\hat{\sigma}}, \quad y'_{\min} = \frac{y_{\min} - \hat{\mu}}{\hat{\sigma}}, \tag{16.7}$$

запишем улучшение в уравнении (16.4) как

$$I(y) = \begin{cases} \hat{\sigma}\left(y'_{\min} - z\right), \text{если } z < y'_{\min} \text{ и } \hat{\sigma} > 0, \\ 0 \text{ в противном случае,} \end{cases} \tag{16.8}$$

где $\hat{\mu}$ и $\hat{\sigma}$ — прогнозируемые математическое ожидание и стандартное отклонение в выборочной точке x.

Вычислим ожидаемое улучшение, используя распределение, предсказанное гауссовским процессом:

$$\mathbb{E}\left[I(y)\right] = \hat{\sigma} \int_{-\infty}^{y'_{\min}} \left(y'_{\min} - z\right) \mathcal{N}\left(z \middle| 0,1\right) dz = \tag{16.9}$$

$$= \hat{\sigma}\left[y'_{\min} \int_{-\infty}^{y'_{\min}} \mathcal{N}\left(z \middle| 0,1\right) dz - \int_{-\infty}^{y'_{\min}} z\mathcal{N}\left(z \middle| 0,1\right) dz \right] = \tag{16.10}$$

$$= \hat{\sigma}\left[y'_{\min} P\left(z \le y'_{\min}\right) + \mathcal{N}\left(y'_{\min} \middle| 0,1\right) - \underbrace{\mathcal{N}\left(-\infty \middle| 0,1\right)}_{=0} \right] = \tag{16.11}$$

$$= \left(y_{\min} - \hat{\mu}\right) P\left(y \le y_{\min}\right) + \hat{\sigma}\mathcal{N}\left(y_{\min} \middle| \hat{\mu}, \hat{\sigma}\right). \tag{16.12}$$

Этот процесс иллюстрируют рис. 16.6 и алгоритм 16.2.

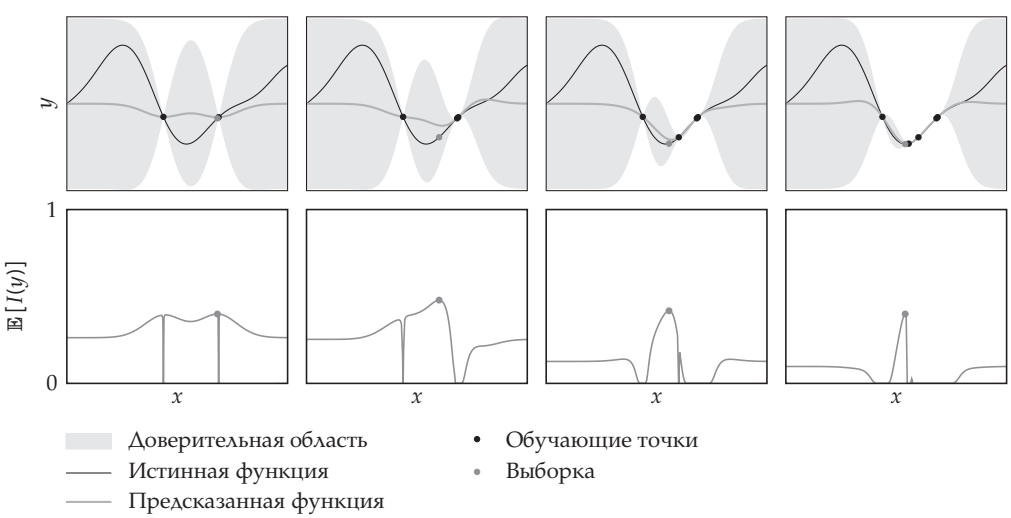


Рис. 16.6. Максимизация ожидаемого улучшения отбирает выборки, которые с наибольшей вероятностью улучшают нижнюю доверительную границу

Алгоритм 16.2. Вычисление ожидаемого улучшения для заданного наилучшего значения `y_min`, математического ожидания `μ` и стандартного отклонения `σ`

```
function expected_improvement(y_min, μ, σ)
   p_imp = prob_of_improvement(y_min, μ, σ)
   p_ymin = pdf(Normal(μ, σ), y_min)
   return (y_min - μ) * p_imp + σ * p_ymin
end
```

16.6. Безопасная оптимизация

В некоторых контекстах слишком дорого оценивать точки, которые считаются небезопасными, это может соответствовать неэффективным или недопустимым точкам. Такие задачи, как настройка контроллера или видеокамеры беспилотного летательного аппарата, требуют *безопасного исследования* — найти оптимальную расчетную точку, тщательно избегая небезопасной выборки.

В этом разделе описывается *алгоритм SafeOpt* [152], который решает класс задач безопасного исследования. Мы выбираем ряд расчетных точек $\mathbf{x}^{(1)}$, ..., $\mathbf{x}^{(m)}$ в погоне за минимумом, но без вычисления значений $f(\mathbf{x}^{(i)})$, превышающих критический порог безопасности $y_{\max}$. Кроме того, мы получаем только зашумленные измерения целевой функции, где шум равен нулю с дисперсией v. Такая целевая функция и связанные с ней безопасные области показаны на рис. 16.7.

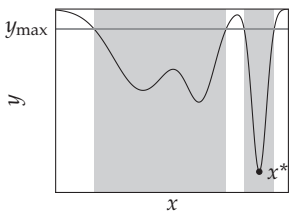


Рис. 16.7. Алгоритм SafeOpt решает проблемы безопасного исследования, которые минимизируют функцию f, оставаясь в безопасных областях, определенных максимальными значениями целевой функции

Алгоритм SafeOpt использует суррогатные модели гауссовского процесса для прогнозирования. На каждой итерации мы подгоняем гауссовский процесс к зашумленным выборкам из значений f. После i-й выборки алгоритм SafeOpt вычисляет верхнюю и нижнюю доверительные границы значения x гауссовского процесса:

$$u_i(\mathbf{x}) = \hat{\mu}_{i-1}(\mathbf{x}) + \sqrt{\beta \hat{v}_{i-1}(\mathbf{x})}, \tag{16.13}$$

$$l_i(\mathbf{x}) = \hat{\mu}_{i-1}(\mathbf{x}) - \sqrt{\beta \hat{v}_{i-1}(\mathbf{x})}, \tag{16.14}$$

где большие значения β приводят к более широкой доверительной области. Такие границы показаны на рис. 16.8.

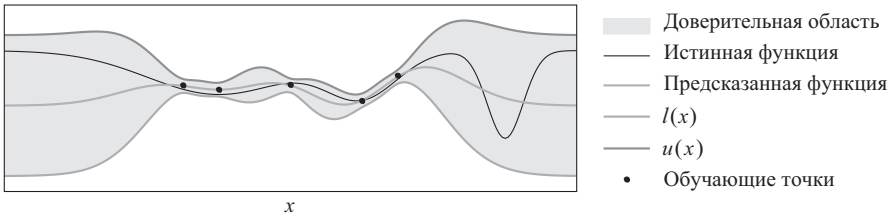


Рис. 16.8. Иллюстрация функций, основанных на предсказаниях гауссовского процесса, используемого в алгоритме SafeOpt

Гауссовский процесс предсказывает распределение $f(\mathbf{x})$ для любой расчетной точки. Будучи гауссовскими, эти прогнозы могут обеспечить только вероятностную гарантию безопасности до произвольного фактора:[2]

$$P\left(f\left(\mathbf{x}\right) \le y_{\max}\right) = \Phi\left(\frac{y_{\max} - \hat{\mu}\left(\mathbf{x}\right)}{\sqrt{\hat{\nu}\left(\mathbf{x}\right)}}\right) \ge P_{\text{safe}}. \tag{16.15}$$

Предсказанная безопасная область $\mathcal{S}$ состоит из расчетных точек, которые обеспечивают вероятность безопасности, превышающую требуемый уровень P_{safe}, как показано на рис. 16.9. Безопасная область также может быть определена в терминах верхних оценок Липшица, построенных из верхних оценок, вычисленных в ранее выбранных точках.

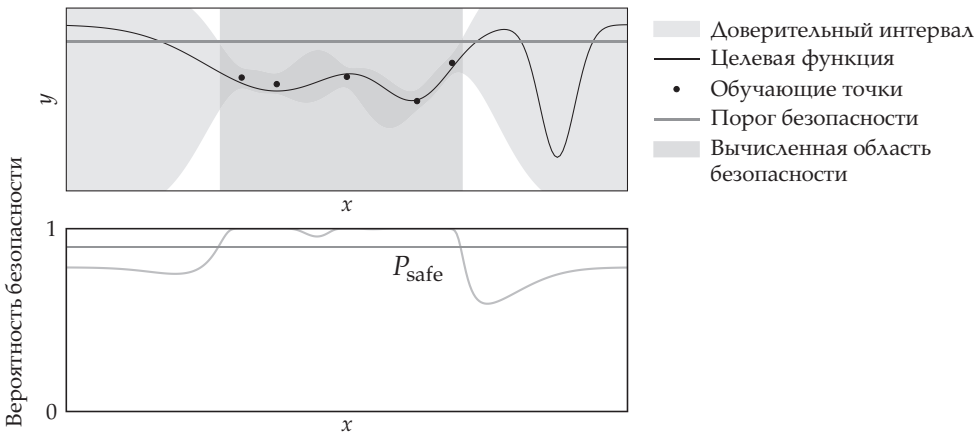


Рис. 16.9. Целевая функция безопасных областей (зеленая), предсказанная гауссовским процессом

[2] Обратите внимание на сходство с вероятностью улучшения.

Алгоритм SafeOpt выбирает безопасную точку выборки, которая уравновешивает желание локализовать допустимую точку минимума функции f и расширить безопасную область. Множество потенциальных точек минимума функции f обозначено как $\mathcal{M}$ (рис. 16.10), а множество точек, которые потенциально могут привести к расширению безопасных областей, обозначено как $\mathcal{E}$ (рис. 16.11). Для компромисса между исследованием и эксплуатацией мы выбираем расчетную точку $\mathbf{x}$ с наибольшей прогностической дисперсией среди множеств $\mathcal{M}$ и $\mathcal{E}$.[3]

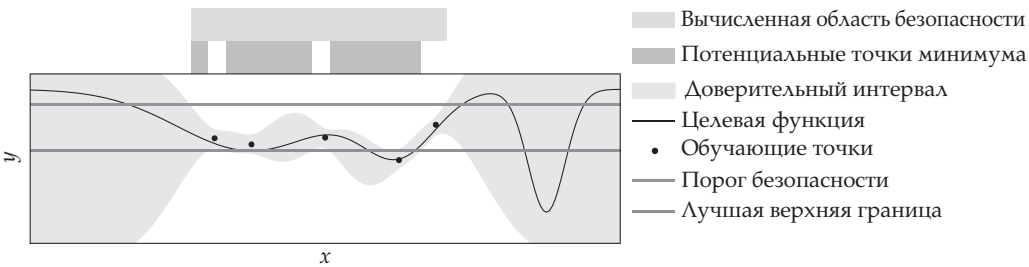


Рис. 16.10. Потенциальные точки минимума — это безопасные точки, нижние границы которых ниже лучшей безопасной верхней границы

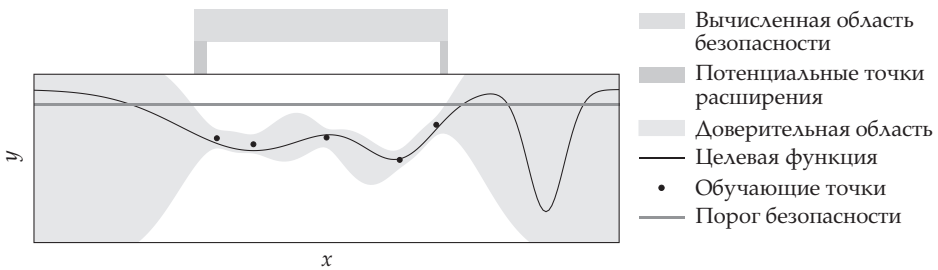


Рис. 16.11. Множество потенциальных точек расширения

Набор потенциальных точек минимума состоит из безопасных точек, нижняя доверительная граница которых ниже нижней верхней границы:

$$\mathcal{M}_i = \left\{ \mathbf{x} \in \mathcal{S}_i \middle| l_i(\mathbf{x}) \leq \min_{\mathbf{x}' \in \mathcal{S}_i} u_i(\mathbf{x}') \right\}. \tag{16.16}$$

На шаге i множество $\mathcal{E}_i$ потенциальных точек, расширяющих область, состоит из безопасных точек, которые, если их добавить к гауссовскому процессу, оптимистично принимая нижнюю границу, создают апостериорное распределение с бо́льшим набором безопасных точек. Потенциальные точки расширения естественным образом находятся вблизи границы безопасной области.

[3] Вариант этого алгоритма см. в [17].

По начальной безопасной точке[4] $\mathbf{x}^{(1)}$ SafeOpt выбирает в множествах $\mathcal{M}$ и $\mathcal{E}$ расчетную точку с наибольшей неопределенностью, которая определяется по ширине $w_i(x) = u(x) - l(x)$:

$$x^{(i)} = \underset{\mathbf{x} \in \mathcal{M}_i \cup \mathcal{E}_i}{\arg\max}\, w_i(\mathbf{x}). \tag{16.17}$$

Выполнение алгоритма SafeOpt продолжается до тех пор, пока не будет выполнено условие завершения. Обычно алгоритм запускается при фиксированном количестве итераций или до тех пор, пока максимальная ширина не станет меньше установленного порога.

Поддержание множеств в многомерных пространствах может быть сложным в вычислительном отношении. Алгоритм SafeOpt предполагает конечное пространство проектирования $\mathcal{X}$, которое может быть получено с помощью метода выбора, примененного к области непрерывного поиска. Увеличение плотности конечного пространства проектирования приводит к более точным результатам по отношению к непрерывному пространству, но занимает больше времени на итерацию.

Реализация SafeOpt приведена в алгоритме 16.3, который вызывает алгоритм 16.4 для обновления прогнозируемых доверительных интервалов, алгоритм 16.5 для вычисления безопасных, минимизирующих и расширяющих областей; и алгоритм 16.6 для выбора тестовой точки. Последовательность шагов алгоритма SafeOpt показана для одного измерения на рис. 16.12, а для двух измерений — на рис. 16.13.

Алгоритм 16.3. Алгоритм SafeOpt применяется к пустому гауссовскому процессу `GP`, конечному расчетному пространству `X`, индексу начальной безопасной точки `i`, целевой функции `f` и порогу безопасности `y_max`. Необязательные параметры — это скаляр доверительного уровня `β` и количество итераций `k_max`. Возвращается кортеж, содержащий наилучшую безопасную верхнюю границу и ее индекс в `X`

```
function safe_opt(GP, X, i, f, y_max; β = 3.0, k_max = 10)
    push!(GP, X[i], f(X[i])) # создаем первое наблюдение

    m = length(X)
    u, l = fill(Inf, m), fill(-Inf, m)
    S, M, E = falses(m), falses(m), falses(m)
```

[4] Алгоритм SafeOpt не может гарантировать безопасность, если он не инициализирован хотя бы одной априори безопасной точкой.

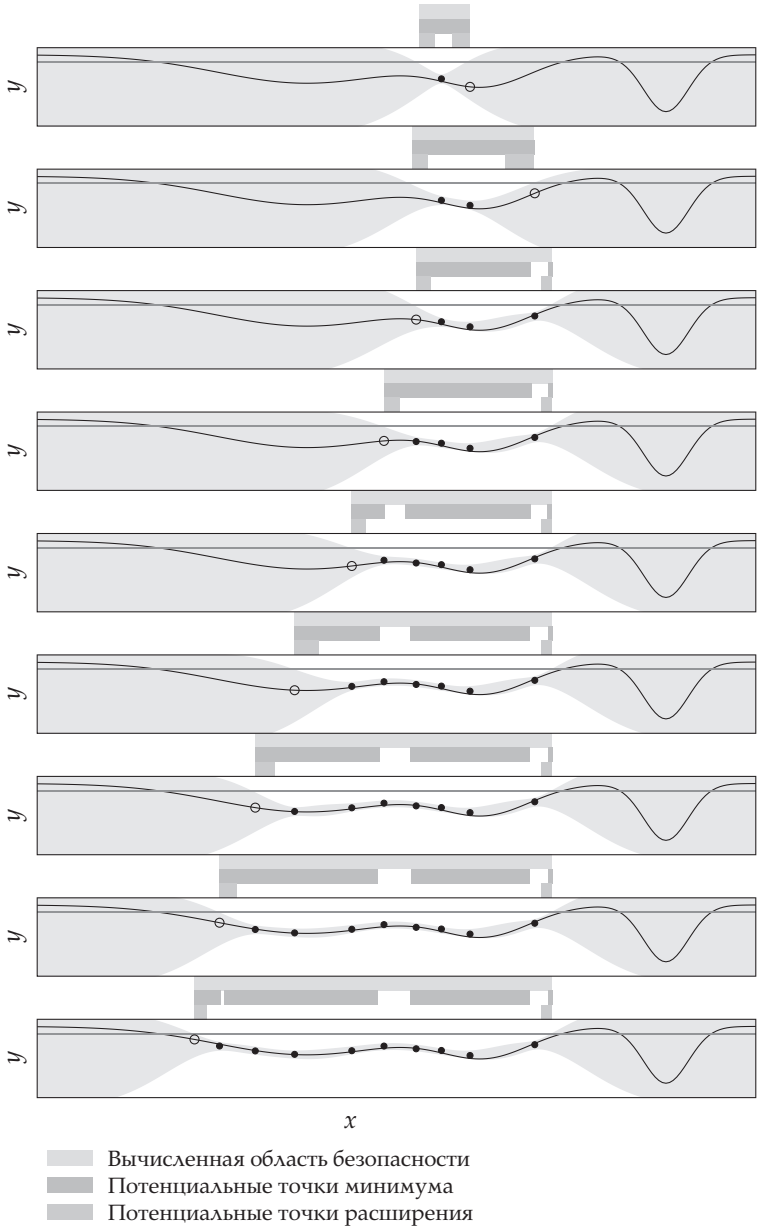


Рис. 16.12. Первые восемь итераций SafeOpt для одномерной функции. SafeOpt никогда не сможет достичь глобального оптимума с правой стороны, потому что он требует пересечения небезопасной области. Можем только надеяться найти глобальные минимумы в локально допустимой безопасной области

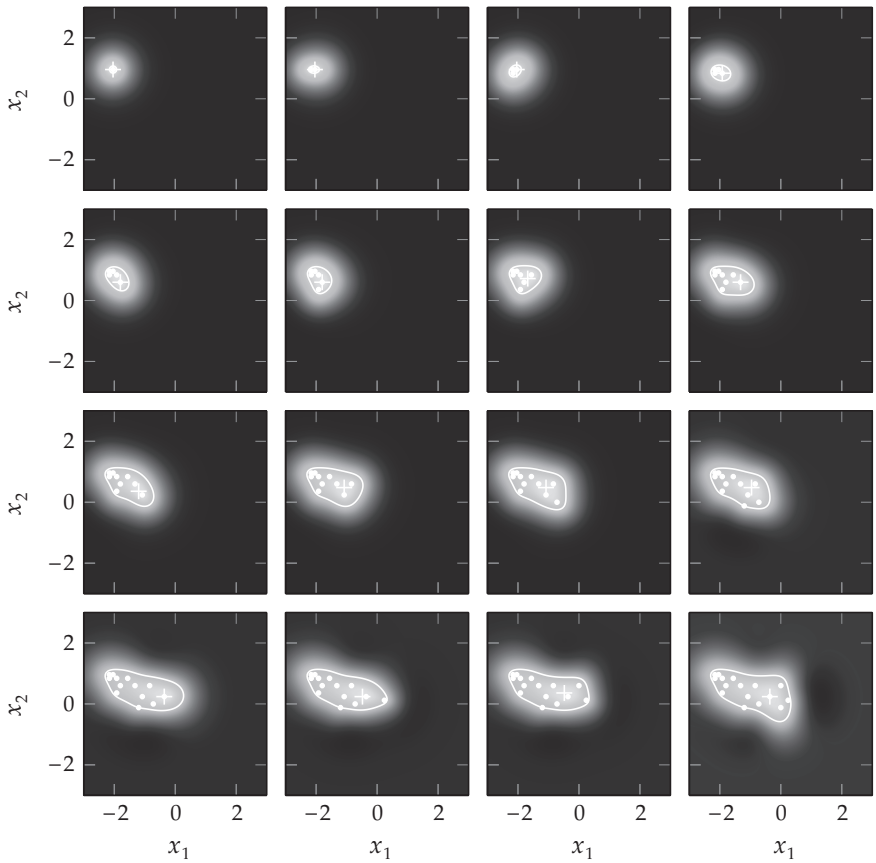
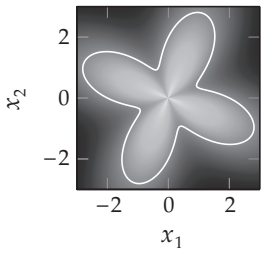


Рис. 16.13. Применение алгоритма SafeOpt к двум функциям цветка (приложение Б.4), если $y_{\max} = 2$, математическое ожидание гауссовского процесса $\mu(\mathbf{x}) = 2{,}5$; дисперсия $v = 0{,}01$; $\beta = 10$ на равномерной сетке 51×51 в пространстве поиска и две начальные точки $\mathbf{x}^{(1)} = [-2{,}04;\ 0{,}96]$. Цвет указывает значение верхней границы, крестик обозначает безопасную точку с самой низкой верхней границей, а белая контурная линия является вычисленной безопасной областью. Целевая функция с истинно безопасной областью, выделенной белым цветом



```
    for k in 1 : k_max(
        update_confidence_intervals!(GP, X, u, l, β)
        compute_sets!(GP, S, M, E, X, u, l, y_max, β)
        i = get_new_query_point(M, E, u, l)
        i != 0 || break
        push!(GP, X[i], f(X[i]))
    end

    # возвращаем наилучшую точку
    update_confidence_intervals!(GP, X, u, l, β)
    S[:] = u .≤ y_max
    if any(S)
        u_best, i_best = findmin(u[S])
        i_best = findfirst(isequal(i_best), cumsum(S))
        return (u_best, i_best)
    else
        return(NaN, 0)
    end
end
```

Алгоритм 16.4. Метод обновления нижней и верхней границ, используемый в алгоритме SafeOpt, который применяет гауссовский процесс `GP`, конечное пространство поиска `X`, векторы верхний и нижней границ `u` и `l`, а также скаляр доверительного уровня `β`

```
function update_confidence_intervals!(GP, X, u, l, β)
    μp, νp = predict(GP, X)
    u[:] = μp + sqrt.(β * νp)
    l[:] = μp - sqrt.(β * νp)
    return (u, l)
end
```

Алгоритм 16.5. Метод обновления безопасных множеств `S`, точек минимума `M` и точек расширения `E`, используемый в алгоритме SafeOpt. Все множества являются булевыми векторами, указывающими, находится ли соответствующая расчетная точка `x` в множестве. Метод также принимает гауссовский процесс `GP`, верхнюю и нижнюю границы `u` и `l` соответственно, порог безопасности `y_max` и скаляр доверительного уровня `β`

```
function compute_sets!(GP, S, M, E, X, u, l, y_max, β)
    fill!(M, false)
```

```
    fill!(E, false)

    # безопасное множество
     S[:] = u .≤ y_max

     if any(S)

         # потенциальные точки минимума
         M[S] = l[S] .< minimum(u[S])

         # максимальная ширина (в M)
         w_max = maximum(u[M] - l[M])

         # точки расширения - пропускаем значения в M
         # и те, для которых w ≤ w_max
         E[:] = S .& .~ M # пропускаем точки в M
         if any(E)
            E[E] .= maximum(u[E] - l[E]) .> w_max
            for (i, e) in enumerate(E)
                 if e && u[i] - l[i] > w_max
                    push!(GP, X[i], l[i])
                    μp, νp = predict(GP, X[.~ S])
                    pop!(GP)
                    E[i] = any(μp + sqrt.(β * νp) .≥ y_max)
                    if E[i]; w_max = u[i] - l[i]; end
                 end
            end
         end
     end

     return (S, M, E)
end
```

Алгоритм 16.6. Метод получения следующей тестовой точки в алгоритме SafeOpt. Возвращается индекс точки с наибольшей шириной в множестве **X**

```
function get_new_query_point(M, E, u, l)
   ME = M .| E
   if any(ME)
           v = argmax(u[ME] - l[ME])
```

```
        return findfirst(isequal(v), cumsum(ME))
    else
        return 0
    end
end
```

16.7. Резюме

- Гауссовские процессы могут использоваться для управления процессом оптимизации с использованием различных стратегий, которые применяют оценки величин, таких как нижняя доверительная граница, вероятность улучшения и ожидаемое улучшение.
- Некоторые задачи не позволяют оценить небезопасные конструкции, и в этом случае мы можем использовать безопасные стратегии исследования, основанные на гауссовских процессах.

16.8. Упражнения

Упражнение 16.1. Приведите пример, в котором оптимизация на основе прогнозов не удалась.

Упражнение 16.2. В чем основное различие между исследованием на основе нижнего доверительного предела и исследованием на основе ошибок в контексте оптимизации?

Упражнение 16.3. Рассмотрим функцию $f(x) = (x - 2)^2/40 - 0{,}5$ с $x \in [-5, 5]$ и расчетные точки –1 и 1. Предположим, что мы используем суррогатную модель гауссовского процесса с функцией нулевого математического ожидания и квадратом экспоненциального ядра $\exp\left(-r^2/2\right)$, где r — евклидово расстояние между двумя точками. Какое значение для $\mathbf{x}$ мы бы выбрали следующим, если бы максимизировали вероятность улучшения? Какое значение для $\mathbf{x}$ мы бы выбрали следующим, если бы максимизировали ожидаемое улучшение?

Глава 17

Оптимизация в условиях неопределенности

В предыдущих главах предполагалось, что целью оптимизации является минимизация детерминированной функции, зависящей от расчетных точек. Однако во многих инженерных задачах может существовать неопределенность в целевой функции или ограничениях. Неопределенность может возникнуть из-за ряда факторов, таких как аппроксимация модели, погрешность измерения и колебания параметров во времени. В этой главе рассматриваются различные методы учета неопределенности в методах оптимизации для повышения надежности.[1]

17.1. Неопределенность

Неопределенность в процессе оптимизации может возникнуть по ряду причин. Может существовать *неустранимая неопределенность*[2], присущая системе, такая как фоновый шум, изменяющиеся свойства материала и квантовые эффекты. Этих неопределенностей нельзя избежать, и проект должен их учитывать. Также может существовать *эпистемологическая неопределенность*[3], которая является неопределенностью, вызванной субъективным недостатком знаний конструктора. Эта неопределенность может возникать из-за приближений в модели[4], используемых при формулировании задачи проектирования, и ошибок, вносимых численными методами.

---

[1] См. также [18, 119].

[2] Эта форма неопределенности иногда называется *случайной неопределенностью* или *случайной погрешностью*.

[3] Эпистемологическая неопределенность также называется *редуцируемой неопределенностью*.

[4] Известный статистик Джордж Бокс писал: “Все модели ошибочны и лишь некоторые полезны” [21].

Учет различных форм неопределенности имеет решающее значение для обеспечения надежности проектов. В этой главе вектор случайных значений обозначается как $\mathbf{z} \in \mathcal{Z}$. Мы хотим минимизировать $f(\mathbf{x}, \mathbf{z})$, но у нас нет контроля над $\mathbf{z}$. Допустимость зависит как от расчетного вектора $\mathbf{x}$, так и от неопределенного вектора $\mathbf{z}$. Здесь вводится допустимое множество для пар $\mathbf{x}$ и $\mathbf{z}$, которое обозначается как $\mathcal{F}$. Пара векторов является допустимой тогда и только тогда, когда $(\mathbf{x}, \mathbf{z}) \in \mathcal{F}$. Обозначим символом $\mathcal{X}$ пространство проектных параметров, которое может включать потенциально недопустимые проекты в зависимости от значения $\mathbf{z}$.

Оптимизация в условиях неопределенности была кратко представлена в разделе 15.5 в контексте использования гауссовского процесса для представления целевой функции, полученной в результате зашумленных измерений. Мы имели функцию $f(\mathbf{x}, z) = f(\mathbf{x}) + z$ с дополнительным допущением, что случайная величина z имеет нормальное распределение с нулевым математическим ожиданием.[5] Неопределенность может быть включена в оценку расчетной точки другими способами. Например, если бы у нас был шум на входе в целевую функцию[6], мы бы получили $f(\mathbf{x}, \mathbf{z}) = f(\mathbf{x} + \mathbf{z})$. В общем, $f(\mathbf{x}, \mathbf{z})$ может быть сложной нелинейной функцией от $\mathbf{x}$ и $\mathbf{z}$. Кроме того, случайная величина $\mathbf{z}$ может не иметь нормального распределения; на самом деле она может иметь неизвестное распределение.

На рис. 17.1 показано, как степень неопределенности может повлиять на выбор проектных параметров. Для простоты x является скаляром, а z выбирается из нормального распределения с нулевым математическим ожиданием. Мы предполагаем, что z соответствует шуму на входе в функцию f, и поэтому $f(x, z) = f(x + z)$. На рисунке показано ожидаемое значение целевой функции для разных уровней шума. Глобальный минимум без шума обозначен символом a. Тем не менее стремление к параметрам вблизи точки a может быть рискованным, поскольку эта точка лежит на дне глубокого оврага, что делает ее довольно чувствительной к шуму. Даже при низком уровне шума лучше выбрать параметры рядом с точкой b. Конструкции вблизи точки c могут обеспечить еще большую устойчивость к большому количеству шума. Если шум очень высокий, то лучшие проектные параметры могут даже находиться между b и c, что соответствует локальному максимуму при отсутствии шума.

[5] Здесь версия f с двумя аргументами принимает в качестве входных данных расчетную точку и случайный вектор, а версия f с одним аргументом представляет детерминированную функцию расчетной точки без шума.

[6] Например, может существовать вариабельность в самом процессе проектирования.

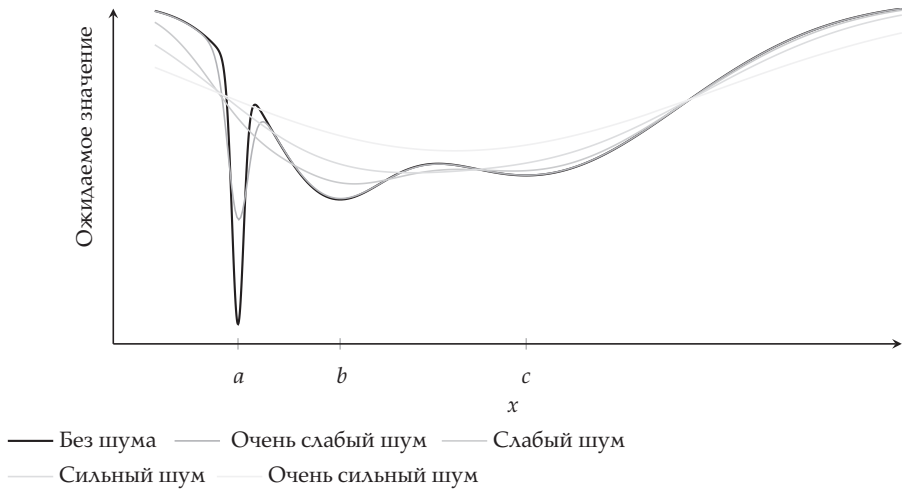


Рис. 17.1. Глобальный минимум в точке *a*, соответствующий ситуации без шума, чувствителен к возмущениям. Другие расчетные точки могут быть более устойчивыми в зависимости от ожидаемого уровня шума

Существует множество различных способов учета неопределенности в оптимизации. Мы обсудим как неопределенность, основанную на множестве, так и вероятностную неопределенность.[7]

17.2. Неопределенность на основе множеств

Методы учета *неопределенности на основе множеств* предполагают, что $\mathbf{z}$ принадлежит множеству $\mathcal{Z}$, но не делают предположений об относительной вероятности различных точек в пределах этого множества. Множество $\mathcal{Z}$ определяется по-разному. Один из способов — найти интервалы для каждого компонента $\mathcal{Z}$. Другой способ — определить $\mathcal{Z}$ с помощью набора ограничений в виде неравенств $g(\mathbf{x}, \mathbf{z}) \leq 0$.

17.2.1. Минимакс

В задачах с неопределенностью, основанной на множествах, мы часто хотим минимизировать максимально возможное значение целевой функции. Такой *минимаксный подход* решает задачу[8]

$$\min_{\mathbf{x} \in \mathcal{X}} \max_{\mathbf{x} \in \mathcal{Z}} f(\mathbf{x}, \mathbf{z}). \tag{17.1}$$

[7] Другие подходы к представлению неопределенности включают теорию Демпстера–Шефера, теорию нечетких множеств и теорию возможностей, которые выходят за рамки этой книги.

[8] Эта задача также называется *подходом надежной оптимизации* или *надежной регуляризации*.

Иначе говоря, мы хотим найти вектор $\mathbf{x}$, который минимизирует f, принимая наихудшее значение для $\mathbf{z}$.

Эта оптимизация эквивалентна определению модифицированной целевой функции

$$f_{\text{mod}}(\mathbf{x}) = \max_{\mathbf{z}\in\mathcal{Z}} f(\mathbf{x},\mathbf{z}) \tag{17.2}$$

с последующим решением задачи

$$\min_{\mathbf{x}\in\mathcal{X}} f_{\text{mod}}(\mathbf{x}). \tag{17.3}$$

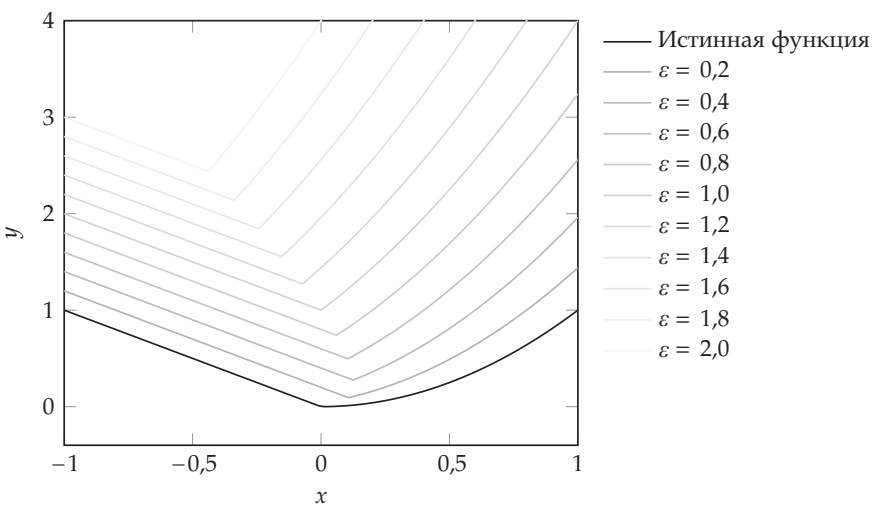
Пример 17.1 демонстрирует эту оптимизацию для одномерной задачи и иллюстрирует влияние различных уровней неопределенности.

Пример 17.1. Пример минимаксного подхода к оптимизации при наличии неопределенности, основанной на множествах

Рассмотрим целевую функцию

$$f(x,z) = f(x+z) = f(\tilde{x}) = \begin{cases} -\tilde{x}, \text{ если } \tilde{x} \le 0, \\ \tilde{x}^2 \text{ в противном случае,} \end{cases}$$

где $\tilde{x} = x + z$ с заданной областью неопределенности $z \in [-\varepsilon, \varepsilon]$. Минимаксный подход состоит в минимизации модифицированной целевой функции $f_{\text{mod}}(x) = \max_{z\in[-\varepsilon,\varepsilon]} f(x, z)$.



На рисунке, приведенном выше, показана функция $f_{\text{mod}}(x)$ для нескольких различных значений ε. Минимум для $\varepsilon = 0$ совпадает с минимумом $f(x, 0)$. При увеличении ε минимум сначала смещается вправо, пока x растет быстрее,

чем x^2, а затем смещается влево, когда x^2 растет быстрее, чем x. Надежная точка минимума обычно не совпадает с минимумом $f(x, 0)$.

В задачах с ограничениями задача оптимизации принимает вид

$$\min_{\mathbf{x}\in\mathcal{X}} \max_{\mathbf{z}\in\mathcal{Z}} f(\mathbf{x},\mathbf{z}) \quad \text{при условии, что } (\mathbf{x},\mathbf{z}) \in \mathcal{F}. \tag{17.4}$$

В примере 17.2 показано влияние применения принципа минимакса на пространство допустимых расчетных значений при наличии ограничений.

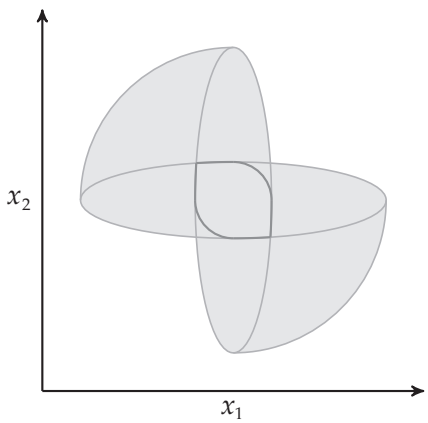
Пример 17.2. Пример минимаксного подхода к оптимизации с ограничениями

Рассмотрим неопределенное допустимое множество в виде повернутого эллипса, где $(\mathbf{x}, z) \in \mathcal{F}$, если и только если $z \in [0, \pi/2]$ и

$$\left(x_1 \cos z + x_2 \sin z\right)^2 + \left(x_1 \sin z - x_2 \cos z\right)^2 / 16 \le 1.$$

Если $z = 0$, то большая ось эллипса ориентирована вертикально. Увеличение значения z медленно поворачивает его против часовой стрелки к горизонтали при $z = \pi/2$. На рисунке, приведенном ниже, синим цветом выделены вертикальные и горизонтальные эллипсы и набор всех точек, которые возможны хотя бы для одного z.

Минимаксный подход к оптимизации должен учитывать только те расчетные точки, которые являются допустимыми при всех значениях z. Множество проектных параметров, которые всегда являются допустимыми, представляет собой пересечение всех эллипсов, образованных изменяющимся параметром z. Это множество выделено красным цветом.



17.2.2. Теория информационных пробелов при принятии решений

В *теории информационных пробелов при принятии решений* [69] область неопределенности $\mathcal{Z}$ не фиксируется, а параметризуется с помощью неотрицательного скалярного параметра, *пробела* ε. Этот параметр регулирует объем параметризованного множества $\mathcal{Z}(\varepsilon)$ с центром в некотором номинальном значении $\overline{\mathbf{z}} = \mathcal{Z}(0)$. Одним из способов определения множества $\mathcal{Z}(\varepsilon)$ является гиперсфера радиуса ε с центром в номинальной точке $\overline{\mathbf{z}}$:

$$\mathcal{Z}(\varepsilon) = \left\{\mathbf{z} \middle| \left\|\mathbf{z} - \overline{\mathbf{z}}\right\|_2 \le \varepsilon\right\}. \tag{17.5}$$

На рис. 17.2 показано это определение в двух измерениях.

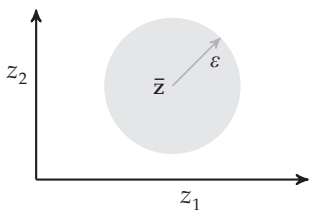


Рис. 17.2. Параметризованное множество неопределенности $\mathcal{Z}(\varepsilon)$ в форме гиперсферы

Параметризуя область неопределенности, мы избегаем фиксации конкретной области неопределенности. Области неопределенности, которые являются слишком большими, жертвуют качеством решения, а слишком маленькие области неопределенности жертвуют надежностью. Расчетные точки, которые остаются допустимыми при больших пробелах, считаются более надежными.

В теории информационных пробелов при принятии решений мы пытаемся найти точку проектирования, которая соответствует самому большому пробелу при сохранении условий допустимости. Эта расчетная точка может быть получена путем решения следующей задачи оптимизации

$$\mathbf{x}^* = \arg\max_{\mathbf{x}\in\mathcal{X}} \max_{\varepsilon\in[0,\infty)} \begin{cases} \varepsilon, \text{если } (\mathbf{x},\mathbf{z}) \in F \text{ при всех } \mathbf{z} \in \mathcal{Z}(\varepsilon), \\ 0 \text{ в противном случае.} \end{cases} \tag{17.6}$$

Эта оптимизация направлена на поиск проектных параметров, которые обеспечивают допустимость при наличии неопределенности. Фактически уравнение (17.6) явно не включает в себя целевую функцию f. Тем не менее мы можем добавить ограничение, что $f(\mathbf{x}, \mathbf{z})$ не будет больше некоторого порога $y_{\max}$. Такие ограничения позволят избежать чрезмерного неприятия риска. Применение теории информационных пробелов при принятии решений иллюстрируют рис. 17.3 и пример 17.3.

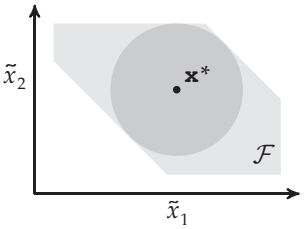
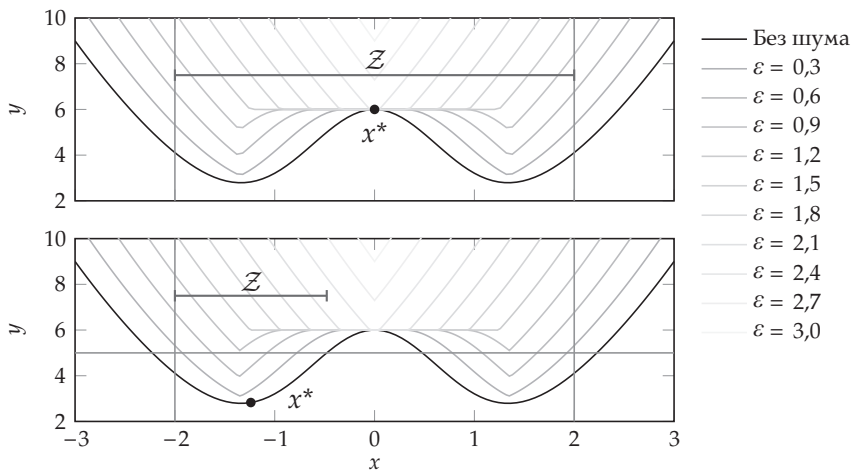


Рис. 17.3. Применение теории информационных пробелов при принятии решений к целевой функции $f(\tilde{\mathbf{x}})$ с аддитивным шумом, где $\tilde{\mathbf{x}} = \mathbf{x} + \mathbf{z}$, и круговой области неопределенности $\mathcal{Z}(\varepsilon) = \{\mathbf{z} \mid \|\mathbf{z}\|_2 \leq \varepsilon\}$. Проект $\mathbf{x}^*$ является оптимальным в теории информационных пробелов при принятии решений, поскольку допускает такое максимально возможное значение ε, что все векторы $\mathbf{x}^* + \mathbf{z}$ являются допустимыми

Пример 17.3. Чрезмерный риск можно смягчить, применяя ограничение к максимально допустимому значению целевой функции с помощью теории информационных пробелов при принятии решений

Рассмотрим устойчивую оптимизацию функции $f(x, z) = \tilde{x}^2 + 5e^{-\tilde{x}^2}$ при $\tilde{x} = x + z$, где $\tilde{x} \in [-2, 2]$ с областью неопределенности $\mathcal{Z}(\varepsilon) = [-\varepsilon, \varepsilon]$.



Применение теории информационных пробелов при принятии решений к этой задаче приводит к области неопределенности максимального размера и параметру проекта, сосредоточенному в субоптимальной области целевой функции. Применение дополнительного ограничения к максимальному значению целевой функции, $f(\mathbf{x}, \mathbf{z}) \leq 5$, позволяет использовать тот же подход, чтобы найти проект с лучшими показателями без шума. Синие линии указывают значение целевой функции в наихудшем случае для данного параметра неопределенности ε.

17.3. Вероятностная неопределенность

Вероятностные подходы моделируют неопределенность с использованием распределений по множеству $\mathcal{Z}$. Распределения вероятностей предоставляют

больше информации, чем неопределенность, основанная на множествах, что позволяет проектировщику учитывать вероятность различных результатов проекта. Эти распределения могут быть определены с использованием экспертных знаний или изучения данных. Учитывая распределение p по $\mathcal{Z}$, мы можем вывести распределение по выходным значениям f, применяя методы, которые будут обсуждаться в главе 18. В этом разделе будут описаны пять различных метрик для преобразования распределения в скалярное значение с учетом конкретного проекта $\mathbf{x}$. Затем мы можем выполнить оптимизацию по этим показателям.[9]

17.3.1. Ожидаемое значение

Один из способов преобразования распределения значений целевой функции f в скалярное значение — использовать *ожидаемое значение* или *математическое ожидание*. Ожидаемое значение — это среднее значение, которое мы можем ожидать при рассмотрении всех значений $f(x, z)$ для всех $z \in \mathcal{Z}$ и соответствующих им вероятностей. Ожидаемое значение как функция расчетной точки $\mathbf{x}$ задается формулой

$$\mathbb{E}_{\mathbf{z}\sim p}\left[f\left(\mathbf{x}, \mathbf{z}\right)\right] = \int_{\mathcal{Z}} f\left(\mathbf{x}, \mathbf{z}\right) p\left(\mathbf{z}\right) d\mathbf{z}. \tag{17.7}$$

Ожидаемое значение не обязательно соответствует целевой функции без шума, как показано в примере 17.4.

Пример 17.4. Ожидаемое значение неопределенной целевой функции зависит от того, как неопределенность включена в целевую функцию

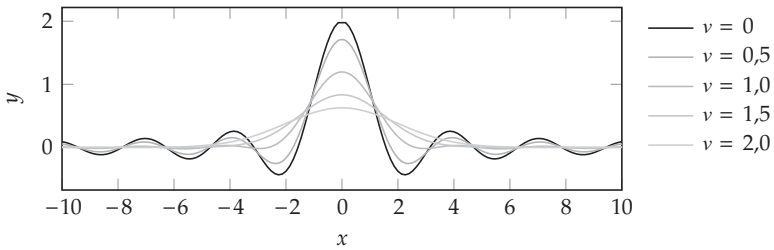
Как правило, к целевой функции добавляется гауссовский шум с нулевым математическим ожиданием, $f(\mathbf{x}, \mathbf{z}) = f(\mathbf{x}) + \mathbf{z}$, как это было в случае с гауссовскими процессами в главе 16. Ожидаемое значение эквивалентно случаю без шума:

$$\mathbb{E}_{\mathbf{z}\sim\mathcal{N}(\mathbf{0},\mathbf{\Sigma})}\left[f(\mathbf{x}) + \mathbf{z}\right] = \mathbb{E}_{\mathbf{z}\sim\mathcal{N}(\mathbf{0},\mathbf{\Sigma})}\left[f(\mathbf{x})\right] + \mathbb{E}_{\mathbf{z}\sim\mathcal{N}(\mathbf{0},\mathbf{\Sigma})}\left[\mathbf{z}\right] = f(\mathbf{x}).$$

Также обычно добавляют шум непосредственно к расчетному вектору, $f(\mathbf{x}, \mathbf{z}) = f(\mathbf{x} + \mathbf{z}) = f(\tilde{\mathbf{x}})$. В таких случаях на ожидаемое значение влияет дисперсия нормального шума с нулевым математическим ожиданием.

Рассмотрим возможность минимизации ожидаемого значения $f\left(\tilde{x}\right) = \sin 2\tilde{x}/\tilde{x}$ при $\tilde{x} = x + z$ для случайной величины z, имеющей нормальное распределение с нулевым математическим ожиданием $\mathcal{N}(0, v)$. Увеличение дисперсии усиливает эффект, который ландшафт локальной функции оказывает на проектные параметры.

[9] Дальнейшее обсуждение разных показателей см. в [140].



Приведенный выше график демонстрирует, что изменение дисперсии влияет на местоположение оптимальной точки.

Аналитически вычислить интеграл в уравнении (17.7) иногда невозможно. Можно аппроксимировать это значение, используя выборку или другие более сложные методы, рассмотренные в главе 18.

17.3.2. Дисперсия

Помимо оптимизации по ожидаемому значению функции, мы также можем быть заинтересованы в выборе расчетных точек, значения которых не слишком чувствительны к неопределенности.[10] Такие области можно количественно определить с помощью дисперсии значений f:

$$\mathrm{Var}\left[f(\mathbf{x},\mathbf{z})\right] = \mathbb{E}_{\mathbf{z}\sim p}\left[\left(f(\mathbf{x},\mathbf{z}) - \mathbb{E}_{\mathbf{z}\sim p}\left[f(\mathbf{x},\mathbf{z})\right]\right)^2\right] = \tag{17.8}$$

$$= \int_{\mathcal{Z}} f(\mathbf{x},\mathbf{z})^2\, p(\mathbf{z})\, d\mathbf{z} - \mathbb{E}_{\mathbf{z}\sim p}\left[f(\mathbf{x},\mathbf{z})\right]^2. \tag{17.9}$$

Расчетные точки с большой дисперсией называются *чувствительными*, а точки с небольшой дисперсией — *устойчивыми*. Примеры чувствительных и устойчивых точек показаны на рис. 17.4. Нас, как правило, интересуют устойчивые точки, которые соответствуют их ожидаемым значениям. Управление компромиссом между ожидаемым значением целевой функции и дисперсией является многокритериальной задачей

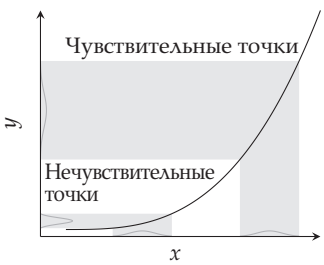


Рис. 17.4. Вероятностные подходы порождают распределения вероятностей на результатах моделей. Расчетные точки могут быть как чувствительными, так и нечувствительными к неопределенности. Синим цветом закрашены области, иллюстрирующие, как нормальное распределение влияет на целевую функцию

[10] Иногда разработчики ищут платоподобные области, где выходные данные целевой функции относительно постоянны, например для производства материалов с постоянной производительностью или составления расписания поездов таким образом, чтобы они приходили в одно и то же время.

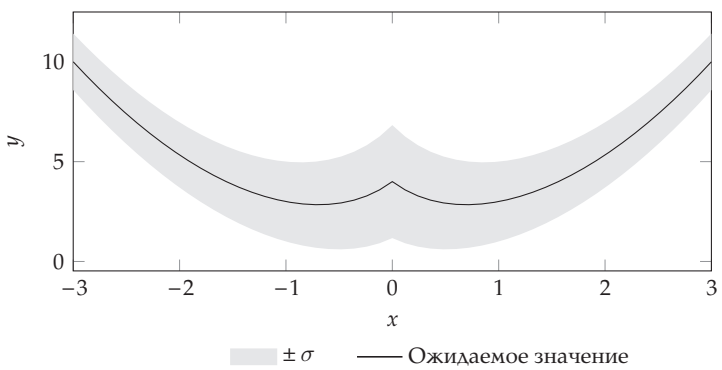
оптимизации (см. пример 17.5), и мы можем использовать методы, рассмотренные в главе 12.

Пример 17.5. Применение ожидаемого значения и дисперсии в оптимизации в условиях неопределенности

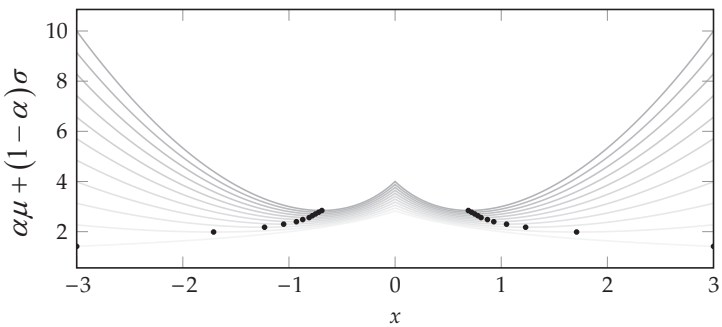
Рассмотрим целевую функцию $f(x, z) = x^2 + z$, где z — случайная величина с гамма-распределением Мы можем построить функцию `dist(x)`, которая при оптимизации в условиях неопределенности возвращает гамма-распределение из пакета **`Distributions.jl`**:

```
dist(x) = Gamma(2 / (1 + abs(x)), 2)
```

Это распределение имеет математическое ожидание $4/(1 + |x|)$ и дисперсию $8/(1 + |x|)$.



Мы можем найти устойчивую оптимальную точку, минимизирующую как ожидаемое значение, так и дисперсию. Минимизация по ожидаемому значению, игнорируя дисперсию, дает два минимума при $x \approx \pm 0{,}695$. Включение штрафа за отклонение сдвигает эти минимумы от начала координат. На рисунке, приведенном ниже, показаны целевые функции вида $\mathbb{E}[y|x] + (1-\alpha)\sqrt{\mathrm{Var}[y|x]}$ для $\alpha \in [0, 1]$ вместе с соответствующими минимумами.



17.3.3. Статистическая допустимость

Альтернативной метрикой оптимизации является *статистическая допустимость*. Зная $p(\mathbf{z})$, мы можем вычислить вероятность того, что расчетная точка $\mathbf{x}$ является допустимой:

$$P\big((\mathbf{x}, \mathbf{z}) \in \mathcal{F}\big) = \int_{\mathcal{Z}} \big((\mathbf{x}, \mathbf{z}) \in \mathcal{F}\big)\, p(\mathbf{z})\, d\mathbf{z}. \tag{17.10}$$

Эта вероятность может быть оценена с помощью выборки. Если мы также заинтересованы в том, чтобы объективное значение не превышало некоторого порогового значения, мы можем включить ограничение $f(\mathbf{x}, \mathbf{z}) \leq y_{\max}$, как это делается в теории информационных пробелов при принятии решений. В отличие от ожидаемого значения и дисперсии, этот показатель мы хотим максимизировать.

17.3.4. Стоимостная мера риска

Стоимостная мера риска (*VaR*) является наилучшим объективным значением, которое может быть гарантировано с вероятностью α. Мы можем записать это определение математически в терминах кумулятивной функции распределения $\Phi(y)$ случайных значений целевой функции. Вероятность того, что результат меньше или равен y, определяется с помощью $\Phi(y)$. Показатель VaR с доверительным уровнем α — это такое минимальное значение y, что $\Phi(y) \geq \alpha$. Данное определение эквивалентно определению *квантиля* распределения вероятностей. Если параметр α близок к единице, то показатель VaR чувствителен к неблагоприятным выбросам, а если параметр α близок к нулю, то показатель VaR слишком оптимистичен и близок к наилучшему результату.

17.3.5. Условная стоимостная мера риска

Условная стоимостная мера риска (*CVaR*) связана со стоимостной мерой риска.[11] CVaR — это ожидаемое значение первого квантиля распределения вероятности по значениям целевой функции. Эта величина проиллюстрирована на рис. 17.5.

Показатель CVaR имеет некоторые теоретические и вычислительные преимущества перед показателем VaR. Показатель CVaR менее чувствителен к ошибкам оценки при распределении по выходным данным. Например, если кумулятивная функция распределения в некоторых интервалах является плоской, то VaR при небольших изменениях может перепрыгнуть на уровень α. Кроме того, показатель VaR не учитывает стоимость, превышающую уровень α-квантиля, что нежелательно, если существуют редкие выбросы с очень плохими объективными значениями.

---

[11] Условная стоимостная мера риска также известна как *средние сверхубытки*, *ожидаемая стоимость потерь* и *хвостовая стоимостная мера риска* [130].

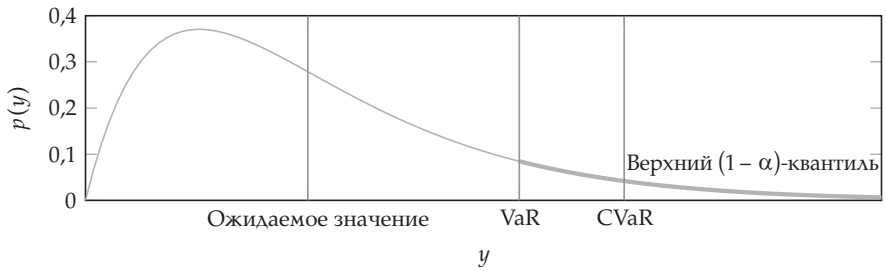


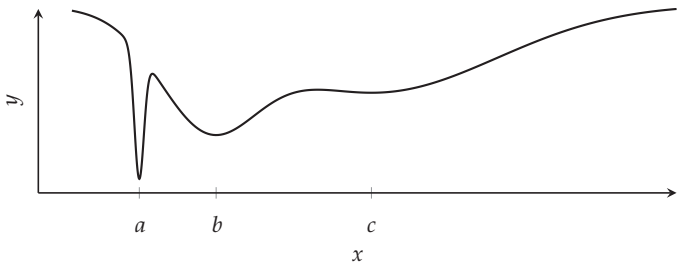
Рис. 17.5. Показатели CVaR и VaR для определенного уровня α. CVaR — это ожидаемое значение наибольшего $(1 - \alpha)$-квантиля, тогда как VaR является самым низким значением целевой функции по сравнению с тем же квантилем

17.4. Резюме

- Неопределенность в процессе оптимизации может возникнуть из-за ошибок в данных, моделях или самом методе оптимизации.
- Учет этих источников неопределенности важен для обеспечения надежных конструкций.
- Оптимизация в отношении неопределенности, основанной на множестве, включает минимаксный подход, который предполагает теорию принятия решений для наихудшего случая и теорию информационных пробелов, позволяющую найти устойчивый проект при максимально большой области неопределенности.
- Вероятностные подходы обычно сводят к минимуму ожидаемое значение, дисперсию, риск недопустимости, стоимостную меру риска, условную стоимостную меру риска или их комбинацию (см. [120, 130]).

17.5. Упражнения

Упражнение 17.1. Предположим, что у нас есть нулевой гауссовский шум на входе, такой что $f(x, z) = f(x + z)$. Рассмотрим три точки a, b и c на рисунке ниже:



Какая расчетная точка лучше, если мы минимизируем ожидаемое значение минус стандартное отклонение?

Упражнение 17.2. Оптимальные точки, изображенные на рис. 17.6, часто лежат на границе ограничений и, таким образом, чувствительны к неопределенности, которая может привести к их недопустимости. Один из подходов к преодолению неопределенности в отношении допустимости состоит в том, чтобы сделать ограничения более строгими, уменьшив размер допустимой области, как показано на рис. 17.7.

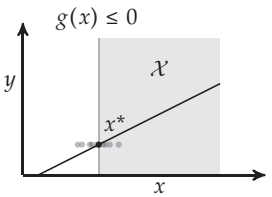


Рис. 17.6. Оптимальные точки с активными ограничениями часто чувствительны к неопределенности

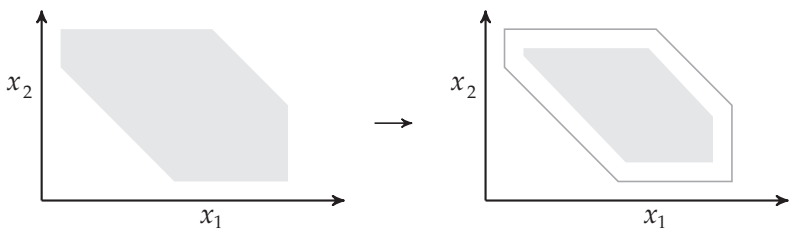


Рис. 17.7. Применение более строгих ограничений при оптимизации не позволяет проектам оказаться слишком близко к истинной границе допустимости

Обычно ограничения вида $g(\mathbf{x}) \leq g_{\max}$ переписывают в виде $\gamma g(\mathbf{x}) \leq g_{\max}$, где $\gamma > 1$ является *фактором безопасности*. Оптимизация, при которой значения ограничений остаются ниже $g_{\max}/\gamma$, обеспечивает дополнительный буфер безопасности.

Рассмотрим балку с квадратным поперечным сечением, которая выходит из строя, когда напряжение превышает $\sigma_{\max} = 1$. Мы хотим минимизировать сечение $f(x) = x^2$, где x — длина сечения. Напряжение в балке также зависит от длины сечения $g(x) = x^{-2}$. Нарисуйте график вероятности того, что оптимизированная конструкция не сломается, если коэффициент безопасности варьируется от 1 до 2.

- Неопределенность в максимальном напряжении, $g(x, z) = x^{-2} + z$.
- Неопределенность в допусках на конструкцию, $g(x, z) = (x + z)^{-2}$.
- Неопределенность в свойствах материала, $g(x, z) = (1 + z)x^{-2}$,

где z — средний шум с дисперсией 0,01.

Упражнение 17.3. *Метод шести сигм* является частным случаем статистической допустимости, при котором производственный или промышленный процесс улучшается до тех пор, пока его предполагаемый гауссовский выход не нарушает проектные требования выбросами, которые превышают шесть стандартных отклонений. Это требование довольно жесткое, как показано на рис. 17.8.

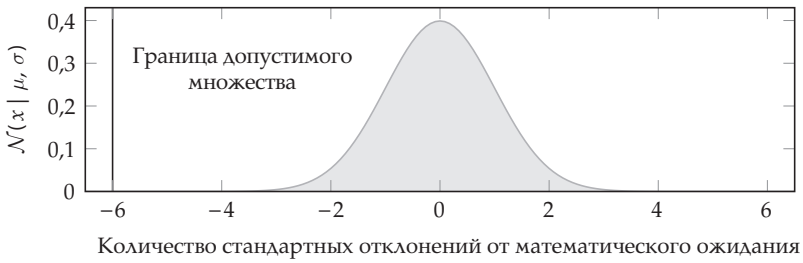


Рис. 17.8. Статистическая допустимость может быть достигнута либо путем смещения математического ожидания значений целевой функции от границы области допустимости, либо путем уменьшения дисперсии целевой функции

Рассмотрите задачу оптимизации

$$\min_{\mathbf{x}} x_1$$

при условии, что $e^{x_1} \leq x_2 + z \leq 2e^{x_1}$,

при условии, что $z \sim \mathcal{N}(0, 1)$. Найдите оптимальную точку $\mathbf{x}^*$, такую что точка (x, z) является допустимой при всех $|z| \leq 6$.

Глава 18

Распространение неопределенности

Как указано в предыдущей главе, вероятностные подходы к оптимизации в условиях неопределенности моделируют некоторые входные данные для целевой функции с помощью распределения вероятностей. В этой главе обсуждается, как распространять известные входные распределения для оценки таких величин, связанных с выходным распределением, как математическое ожидание и дисперсия целевой функции. Существует множество подходов к *распространению неопределенности*, некоторые из них основаны на таких математических понятиях, как метод Монте-Карло, аппроксимация с помощью ряда Тейлора, ортогональные полиномы и гауссовские процессы. Эти подходы отличаются по своим предположениям и качеству оценок.

18.1. Методы выбора

Математическое ожидание и дисперсию целевой функции в конкретной расчетной точке можно аппроксимировать с помощью *метода интегрирования Монте-Карло*[1], который аппроксимирует интеграл на основе m выборок $\mathbf{z}^{(1)}$, ..., $\mathbf{z}^{(m)}$, извлеченных из распределения p на $\mathcal{Z}$. Эти оценки также называются *выборочным средним* и *выборочной дисперсией*.

$$\mathbb{E}_{\mathbf{z}\sim p}\left[f(\mathbf{z})\right] \approx \hat{\mu} = \frac{1}{m}\sum_{i=1}^{m} f\left(\mathbf{z}^{(i)}\right), \tag{18.1}$$

$$\mathrm{Var}_{\mathbf{z}\sim p}\left[f(\mathbf{z})\right] \approx \hat{\nu} = \left(\frac{1}{m}\sum_{i=1}^{m} f\left(\mathbf{z}^{(i)}\right)^2\right) - \hat{\mu}^2. \tag{18.2}$$

В приведенном выше уравнении и в оставшейся части этой главы мы пропустили $\mathbf{x}$ в выражении $f(\mathbf{x}, \mathbf{z})$ для удобства обозначения, но зависимость от $\mathbf{x}$ все еще

[1] В качестве альтернативы для ускорения сходимости при получении оценок можно использовать метод интегрирования квази-Монте-Карло, который рассматривается в главе 13.

существует. Для каждой новой расчетной точки $\mathbf{x}$ в процессе оптимизации мы пересчитываем среднее значение и дисперсию.

Полезным свойством этого подхода на основе выборки является то, что распределение p не обязательно знать точно. Мы можем получить выборки непосредственно с помощью моделирования или реальных экспериментов. Потенциальным ограничением этого подхода является то, что может потребоваться много выборок, прежде чем процесс сойдется к подходящей оценке. Дисперсия среднего выборочного значения для нормально распределенной случайной величины f равна $\mathrm{Var}[\hat{\mu}] = v/m$, где v является истинной дисперсией f. Таким образом, удвоение количества выборок m приводит к уменьшению среднего выборочного значения вдвое.

18.2. Аппроксимация с помощью ряда Тейлора

Другой способ оценки $\hat{\mu}$ и $\hat{v}$ состоит в том, чтобы использовать аппроксимацию функции f в фиксированной расчетной точке $\mathbf{x}$ с помощью ряда Тейлора.[2] На данный момент мы будем предполагать, что n компонентов вектора $\mathbf{z}$ независимы и имеют конечную дисперсию. Обозначим математическое ожидание распределения по $\mathbf{z}$ как μ, а дисперсии отдельных компонентов вектора $\mathbf{z}$ как v.[3] Ниже приведена аппроксимация функции $f(\mathbf{z})$ в точке $\mathbf{z} = \boldsymbol{\mu}$:

$$\hat{f}(\mathbf{z}) = f(\boldsymbol{\mu}) + \sum_{i=1}^{n} \frac{\partial f}{\partial z_i}(z_i - \mu_i) + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial^2 f}{\partial z_i \partial z_j}(z_i - \mu_i)(z_j - \mu_j). \tag{18.3}$$

Из этого приближения мы можем аналитически вычислить оценки математического ожидания и дисперсии случайной величины f:

$$\hat{\mu} = f(\boldsymbol{\mu}) + \frac{1}{2}\sum_{i=1}^{n} \frac{\partial^2 f}{\partial z_i^2} v_i \bigg|_{\mathbf{z}=\boldsymbol{\mu}}, \tag{18.4}$$

$$\hat{v} = \sum_{i=1}^{n} \left(\frac{\partial f}{\partial z_i}\right)^2 v_i + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \left(\frac{\partial^2 f}{\partial z_i \partial z_i}\right) v_i v_j \bigg|_{\mathbf{z}=\boldsymbol{\mu}}. \tag{18.5}$$

Членами высшего порядка можно пренебречь, чтобы получить приближение первого порядка:

$$\hat{\mu} = f(\boldsymbol{\mu}), \quad \hat{v} = \sum_{i=1}^{n} \left(\frac{\partial f}{\partial z_i}\right)^2 v_i \bigg|_{\mathbf{z}=\boldsymbol{\mu}}. \tag{18.6}$$

---

[2] Вывод оценок математического ожидания и дисперсии n случайных величин см. в [16].

[3] Если компоненты вектора $\mathbf{z}$ независимы, то ковариационная матрица является диагональной и v будет вектором, состоящим из диагональных элементов.

Мы можем ослабить предположение, что компоненты $\mathbf{z}$ независимы, но это сделает математические вычисления более сложными. На практике может быть проще преобразовать случайные переменные так, чтобы они были независимыми. Мы можем преобразовать вектор из n коррелированных случайных величин с ковариационной матрицей $\mathbf{C}$ в m некоррелированных случайных величин $\mathbf{z}$ путем умножения на ортогональную матрицу $\mathbf{T}$ размером $m \times n$, содержащую собственные векторы, соответствующие m наибольшим собственным значениям матрицы $\mathbf{C}$. В итоге получим $\mathbf{z} = \mathbf{Tc}$.[4]

Метод аппроксимации Тейлора реализован в алгоритме 18.1. Аппроксимации первого и второго порядка сравниваются в примере 18.1.

Алгоритм 18.1. Метод автоматического вычисления аппроксимации по Тейлору математического ожидания и дисперсии целевой функции `f` в расчетной точке `x` с вектором математических ожиданий шума `μ` и вектором дисперсии `ν`. Булев параметр `secondorder` управляет вычислением аппроксимации первого или второго порядка

```
function taylor_approx(f, μ, ν, secondorder = false)
    μhat = f(μ)
    ∇ = (z -> ForwardDiff.gradient(f, z))(μ)
    νhat = ∇. ^ 2 ⋅ ν
    if secondorder
       H = (z -> ForwardDiff.hessian(f, z))(μ)
       μhat += (diag(H) ⋅ ν) / 2
       νhat += ν ⋅ (H. ^ 2 * ν) / 2
    end
    return (μhat, νhat)
end
```

18.3. Полиномиальный хаос

Полиномиальный хаос — это метод подгонки полинома к оценкам $f(\mathbf{z})$ и использования полученной суррогатной модели для оценки математического ожидания и дисперсии. Начнем этот раздел с обсуждения того, как полиномиальный хаос используется в одномерном случае. Затем обобщим концепцию для многомерных функций и покажем, как получить оценки математического ожидания и дисперсии путем интегрирования функции, представленной суррогатной моделью.

[4] Также распространено масштабирование выходных данных таким образом, чтобы ковариационная матрица стала единичной. Этот процесс называется *отбеливанием* [54].

18.3.1. Одномерный случай

В одномерном пространстве мы приближаем $f(z)$ суррогатной моделью, состоящей из k полиномиальных базисных функций, b_1, ..., b_k:

$$f(z) \approx \hat{f}(z) = \sum_{i=1}^{k} \theta_i b_i(z). \tag{18.7}$$

В отличие от методов Монте-Карло, рассмотренных в разделе 18.1, наши выборки z не должны быть случайным образом извлечены из распределения p. Фактически следует получить выборки, используя один из планов выбора, рассмотренных в главе 13. Как получить базисные коэффициенты, рассказывается в разделе 18.3.2.

Пример 18.1. Применение приближения Тейлора к одномерной задаче проектирования с двумерным гауссовским шумом

Рассмотрим целевую функцию $f(x, z) = \sin(x + z_1)\cos(x + z_2)$, где z_1 и z_2 — гауссовский шум с нулевым математическим ожиданием и дисперсиями 0,1 и 0,2 соответственно.

Первая и вторая частные производные функции f по z_s имеют вид

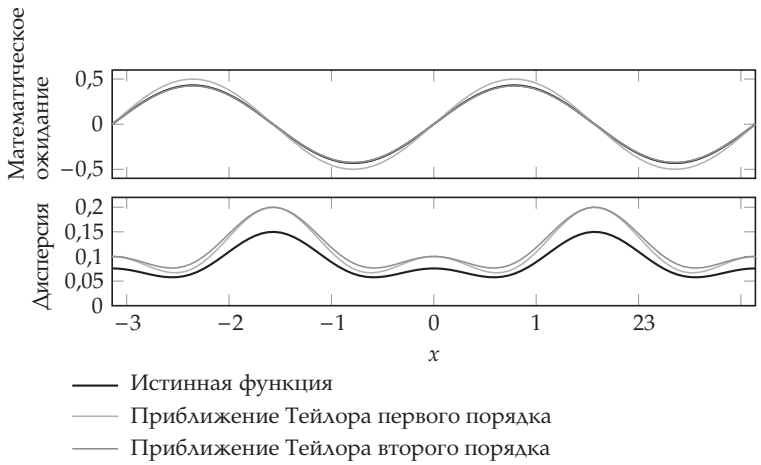
$$\frac{\partial f}{\partial z_1} = \cos(x+z_1)\cos(x+z_2),$$
$$\frac{\partial f}{\partial z_1} = -\sin(x+z_1)\sin(x+z_2),$$
$$\frac{\partial^2 f}{\partial z_1^2} = -\sin(x+z_1)\cos(x+z_2),$$
$$\frac{\partial^2 f}{\partial z_2^2} = -\sin(x+z_1)\cos(x+z_2),$$
$$\frac{\partial^2 f}{\partial z_1 \partial z_2} = -\cos(x+z_1)\sin(x+z_2),$$

что позволяет построить приближение Тейлора:

$$\hat{\mu}(x) = 0{,}85\sin x\cos x,$$
$$\hat{v}(x) = 0{,}3\sin^2 x\cos^2 x - 0{,}035\sin x\cos x.$$

Для вычисления этого приближения в заданной точке **x** используется функция `taylor_approx`.

```
taylor_approx(z -> sin(x + z[1]) * cos(x + z[2]), [0, 0], [0.1, 0.2])
```



Мы можем использовать суррогатную модель для оценки среднего:

$$\hat{\mu} = \mathbb{E}\left[\hat{f}\right] = \tag{18.8}$$

$$= \int_{\mathcal{Z}} \hat{f}(z)\, p(z)\, dz = \tag{18.9}$$

$$= \int_{\mathcal{Z}} \sum_{i=1}^{k} \theta_i b_i(z) p(z)\, dz = \tag{18.10}$$

$$= \sum_{i=1}^{k} \theta_i \int_{\mathcal{Z}} b_i(z)\, p(z)\, dz = \tag{18.11}$$

$$= \theta_1 \int_{\mathcal{Z}} b_i(z)\, p(z)\, dz + \ldots + \theta_k \int_{\mathcal{Z}} b_k(z)\, p(z)\, dz. \tag{18.12}$$

Мы также можем оценить дисперсию:

$$\hat{v} = \mathbb{E}\left[\left(\hat{f} - \mathbb{E}\left[\hat{f}\right]\right)^2\right] = \tag{18.13}$$

$$= \mathbb{E}\left[\hat{f}^2\right] - \mathbb{E}\left[\hat{f}\right]^2 = \tag{18.14}$$

$$= \int_{\mathcal{Z}} \hat{f}(z)^2\, p(z)\, dz - \mu^2 = \tag{18.15}$$

$$= \int_{\mathcal{Z}} \sum_{i=1}^{k} \sum_{j=1}^{k} \theta_i \theta_j b_i(z) b_j(z)\, p(z)\, dz - \mu^2 = \tag{18.16}$$

$$= \int_{\mathcal{Z}} \left(\sum_{i=1}^{k} \theta_i^2 b_i(z)^2 + 2 \sum_{i=2}^{k} \sum_{j=1}^{i-1} \theta_i \theta_j b_i(z) b_j(z) \right) p(z)\, dz - \mu^2 = \tag{18.17}$$

$$= \sum_{i=1}^{k} \left(\theta_i^2 \int_{\mathcal{Z}} b_i(z)^2 p(z) dz \right) + 2 \sum_{i=2}^{k} \sum_{j=1}^{i-1} \left(\theta_i \theta_j \int_{\mathcal{Z}} b_i(z) b_j(z) p(z) dz \right) - \mu^2 = \quad (18.18)$$

$$= \sum_{i=1}^{k} \theta_i^2 \int_{\mathcal{Z}} b_i(z)^2 p(z) dz + 2 \sum_{i=2}^{k} \sum_{j=1}^{i-1} \theta_i \theta_j \int_{\mathcal{Z}} b_i(z) b_j(z) p(z) dz - \mu^2. \quad (18.19)$$

Математическое ожидание и дисперсия могут быть эффективно вычислены, если базисные функции выбраны *ортогональными* относительно плотности вероятности p. Две базисные функции b_i и b_j ортогональны относительно плотности вероятности $p(z)$, если

$$\int_{\mathcal{Z}} b_i(z) b_j(z) p(z) dz = 0, \text{ если } i \neq j. \quad (18.20)$$

Если все выбранные базисные функции ортогональны друг другу и первая базисная функция $b_1(z) = 1$, то математическое ожидание равно:

$$\hat{\mu}_1 = \theta_1 \int_{\mathcal{Z}} b_1(z) p(z) dz + \theta_2 \int_{\mathcal{Z}} b_2(z) p(z) dz + \ldots + \theta_k \int_{\mathcal{Z}} b_k(z) p(z) dz = \quad (18.21)$$

$$= \theta_1 \int_{\mathcal{Z}} b_1(z)^2 p(z) dz + \theta_2 \int_{\mathcal{Z}} b_1(z) b_2(z) p(z) dz + \ldots + \theta_k \int_{\mathcal{Z}} b_1(z) b_k(z) p(z) dz = \quad (18.22)$$

$$= \theta_1 \int_{\mathcal{Z}} p(z) dz + 0 + \ldots + 0 = \quad (18.23)$$

$$= \theta_1. \quad (18.24)$$

Точно так же дисперсия равна:

$$\hat{v} = \sum_{i=1}^{k} \theta_i^2 \int_{\mathcal{Z}} b_i(z)^2 p(z) dz + 2 \sum_{i=2}^{k} \sum_{j=1}^{i-1} \theta_i \theta_j \int_{\mathcal{Z}} b_i(z) b_j(z) p(z) dz - \mu^2 = \quad (18.25)$$

$$= \sum_{i=1}^{k} \theta_i^2 \int_{\mathcal{Z}} b_i(z)^2 p(z) dz - \mu^2 = \quad (18.26)$$

$$= \theta_1^2 \int_{\mathcal{Z}} b_1(z)^2 p(z) dz + \sum_{i=2}^{k} \theta_i^2 \int_{\mathcal{Z}} b_i(z)^2 p(z) dz - \theta_1^2 = \quad (18.27)$$

$$= \sum_{i=2}^{k} \theta_i^2 \int_{\mathcal{Z}} b_i(z)^2 p(z) dz. \quad (18.28)$$

Таким образом, математическое ожидание непосредственно вычисляется в ходе подгонки суррогатной модели к наблюдаемым данным, а дисперсия может быть очень эффективно вычислена при известных значениях $\int_{\mathcal{Z}} b_i(z)^2 p(z) dz$ путем

выбора базисных функций и распределения вероятностей.[5] В примере 18.2 эти процедуры используются для оценки математического ожидания и дисперсии с различными размерами выборки.

Полиномиальный хаос аппроксимирует функцию, используя ортогональные полиномиальные базисные функции k-й степени с $i \in \{1, ..., k + 1\}$ и $b_1 = 1$. Все ортогональные полиномы удовлетворяют рекуррентному соотношению:

$$b_{i+1}(z) = \begin{cases} (z - \alpha_i) b_i(z) & \text{при } i = 1, \\ (z - \alpha_i) b_i(z) - \beta_i b_{i-1}(z) & \text{при } i > 1, \end{cases} \tag{18.29}$$

где $b_1(z) = 1$ и

$$\alpha_i = \frac{\int_{\mathcal{Z}} z b_i(z)^2 p(z) dz}{\int_{\mathcal{Z}} b_i(z)^2 p(z) dz}, \qquad \beta_i = \frac{\int_{\mathcal{Z}} b_i(z)^2 p(z) dz}{\int_{\mathcal{Z}} b_{i-1}(z)^2 p(z) dz}. \tag{18.30}$$

Отношение рекуррентности можно использовать для генерации базисных функций. Каждая базисная функция b_i является полиномом степени $i - 1$. Базисные функции для нескольких распространенных вероятностных распределений приведены в табл. 18.1. Они могут быть сгенерированы с использованием методов, указанных в алгоритме 18.2, и представлены на рис. 18.1. Пример 18.3 иллюстрирует влияние порядка полинома на оценки математического ожидания и дисперсии.

Пример 18.2. Оценка ожидаемого значения неизвестной целевой функции с использованием полиномиального хаоса

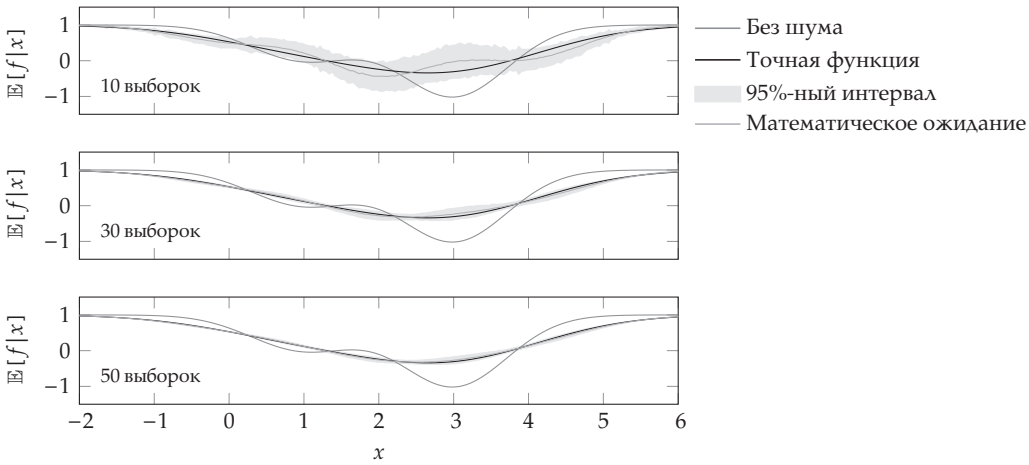
Рассмотрим возможность оптимизации (неизвестной) целевой функции

$$f(x, z) = 1 - e^{-(x+z-1)^2} - 2e^{-(x+z-3)^2},$$

если значения z извлечены из нормального распределения с нулевым математическим ожиданием.

Целевая функция, ее истинное ожидаемое значение и предполагаемые ожидаемые значения с различным количеством выборок приведены ниже. Расчетное ожидаемое значение вычисляется с помощью полиномов Эрмита третьего порядка.

[5] Интегралы этого вида могут быть эффективно вычислены с использованием квадратурной формулы Гаусса, описанной в приложении В.8.



Алгоритм 18.2. Методы построения полиномиальных ортогональных базисных функций, где `i` указывает на конструкцию b_i

```
using Polynomials
function legendre(i)
    n = i - 1
    p = Poly([-1, 0, 1]) ^ n
    for i in 1 : n
        p = polyder(p)
    end
    return p / (2 ^ n * factorial(n))
end
function laguerre(i)
    p = Poly([1])
    for j in 2 : i
        p = polyint(polyder(p) - p) + 1
    end
    return p
end
function hermite(i)
     p = Poly([1])
     x = Poly([0, 1])
     for j in 2 : i
```

Таблица 18.1. Ортогональные полиномиальные базисные функции для некоторых распространенных распределений вероятностей

Распределение	*Область определения*	*Плотность*	*Название*	*Рекурсивная форма*	*Замкнутая форма*
Равномерное	$[-1, 1]$	$\frac{1}{2}$	Лежандра	$\mathrm{Le}_k(x) = \frac{1}{2^k k!}\frac{d^k}{dx^k}\left[\left(x^2-1\right)^k\right]$	$b_i(x) = \sum_{j=0}^{i-1} C_{i-1}^j C_{-i-2}^j \left(\frac{1-x}{2}\right)^j$
Экспоненциальное	$[0, \infty)$	e^{-x}	Лагерра	$\frac{d}{dx}\mathrm{La}_k(x) = \left(\frac{d}{dx}-1\right)\mathrm{La}_{k-1}$	$b_i(x) = \sum_{j=0}^{i-1} C_{i-1}^j \frac{(-1)^j}{j!} x^j$
Стандартное нормальное	$(-\infty, \infty)$	$\frac{1}{\sqrt{2\pi}} e^{-x^2/2}$	Эрмита	$\mathrm{H}_k(x) = x\mathrm{H}_{k-1} - \frac{d}{dx}\mathrm{H}_{k-1}$	$b_i(x) = \sum_{j=0}^{\lfloor (i-1)/2 \rfloor} (i-1)! \frac{(-1)^{\frac{i-1}{2}-j}}{(2j)!\left(\frac{i-1}{2}-j\right)!}(2x)^{2j}$

```
        p = x * p - polyder(p)
    end
    return p
end
```

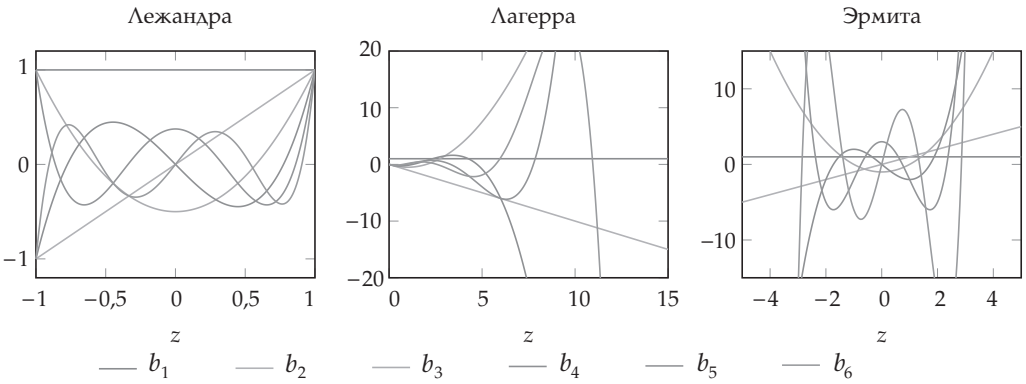


Рис. 18.1. Ортогональные базисные функции для равномерного, экспоненциального и стандартного нормального распределений

Базисные функции для произвольных функций и областей плотности вероятности могут быть построены как аналитически, так и численно.[6] Алгоритм Стилтьеса[7] (алгоритм 18.3) генерирует ортогональные полиномы с использованием рекуррентного соотношения в уравнении (18.29). Пример 18.4 показывает, как порядок полиномов влияет на оценки математического ожидания и дисперсии.

Пример 18.3. Полиномы Лежандра, используемые для оценки математического ожидания и дисперсии функции с равномерно распределенными входными данными

Рассмотрим функцию $f(z) = \sin \pi z$ с аргументом z, извлеченным из равномерного распределения по области [–1, 1]. Истинное математическое ожидание и дисперсия могут быть вычислены аналитически:

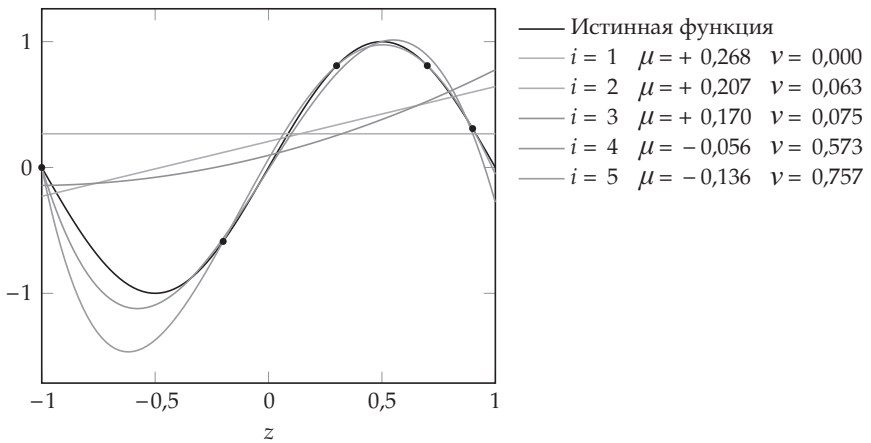
$$\mu = \int_a^b f(z)\, p(z) dz = \int_{-1}^{1} \sin \pi z \frac{1}{2} dz = 0, \tag{18.31}$$

$$\nu = \int_a^b f(z)^2\, p(z) dz - \mu^2 = \int_{-1}^{1} \sin^2 \pi z \frac{1}{2} dz - 0 = \frac{1}{2}. \tag{18.32}$$

[6] Многочлены можно масштабировать с помощью ненулевого коэффициента. Удобно установить $b_1(x) = 1$.

[7] См. [148]. Обзор на английском языке см. в [55].

Предположим, у нас есть пять выборок f при $z = \{-1; -0{,}2; 0{,}3; 0{,}7; 0{,}9\}$. Мы можем подогнать полином Лежандра к данным, чтобы получить суррогатную модель $\hat{f}$. Полиномы разной степени дают следующие графики:



Алгоритм 18.3. Алгоритм Стилтьеса для построения следующей полиномиальной базисной функции b_{i+1} в соответствии с ортогональным рекуррентным соотношением, где **bs** содержит $\{b_1, ..., b_i\}$, **p** — распределение вероятностей, а **dom** — кортеж, содержащий нижнюю и верхнюю границы для **z**. Необязательный параметр **ε** контролирует абсолютный допуск численного интегрирования. Мы используем пакет **Polynomials.jl**

```
using Polynomials
function orthogonal_recurrence(bs, p, dom, ε = 1e-6)
    i = length(bs)
    c1 = quadgk(z -> z * bs[i](z) ^ 2 * p(z), dom ..., atol = ε)[1]
    c2 = quadgk(z -> bs[i](z) ^ 2 * p(z), dom ..., atol = ε)[1]
    α = c1 / c2
    if i > 1
        c3 = quadgk(z -> bs[i - 1](z) ^ 2 * p(z), dom..., atol = ε)[1]
        β = c2 / c3
        return Poly([-α, 1]) * bs[i] - β * bs[i - 1]
    else
        return Poly([-α, 1]) * bs[i]
    end
end
```

Пример 18.4. Многочлены Лежандра, построенные с использованием метода Стилтьеса для оценки математического ожидания и дисперсии функции со входом случайной величины

Рассмотрим функцию $f(z) = \sin \pi z$ с аргументом z, извлеченным из усеченного нормального распределения с математическим ожиданием, равным 3, и дисперсией, равной 1, по области [2, 5]. Истинное математическое ожидание и дисперсия равны:

$$\mu = \int_a^b f(z)\,p(z)\,dz = \int_2^5 \sin(\pi z)p(z)\,dz \approx 0{,}104,$$

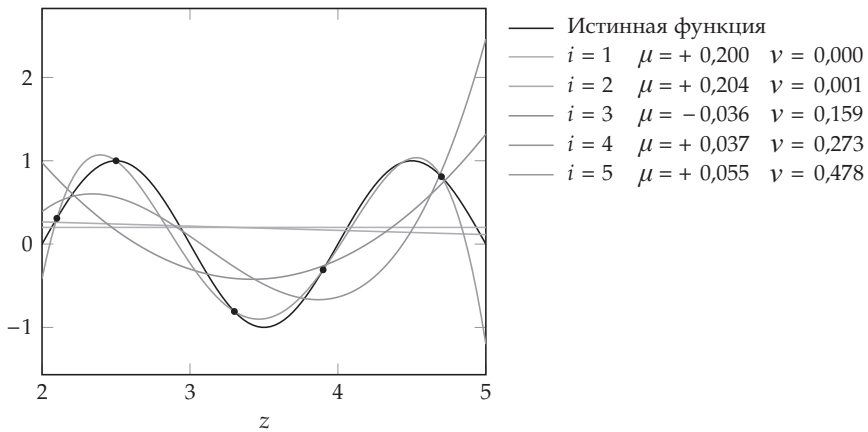
$$\nu = \int_a^b f(z)^2\,p(z)\,dz - \mu^2 = \int_2^5 \sin^2(\pi z)p(z)\,dz - 0{,}104^2 \approx 0{,}495,$$

где плотность вероятности усеченного нормального распределения равна:

$$p(z) = \begin{cases} \dfrac{N(z|3,\ 1)}{\int_2^5 N(\tau|3,\ 1)\,d\tau}, \text{ если } z \in [2,\ 5], \\ 0 \text{ — в противном случае.} \end{cases}$$

Предположим, у нас есть пять выборок f при $z = \{2{,}1;\ 2{,}5;\ 3{,}3;\ 3{,}9;\ 4{,}7\}$. Мы можем подогнать ортогональные полиномы к данным, чтобы получить суррогатную модель $\hat{f}$.

Полиномы разных степеней дают следующие графики:



18.3.2. Коэффициенты

Коэффициенты θ_1, ..., θ_k в уравнении (18.7) могут быть выведены двумя различными способами. Первый способ заключается в подборе значений выборок из $\mathcal{Z}$ с использованием метода линейной регрессии, описанного в разделе 14.3. Второй способ — использовать ортогональность базисных функций, вычисляя интеграл в числителе с помощью квадратурной формулы Гаусса.

Умножим каждую часть уравнения (18.7) на j-й базис, функцию плотности вероятности и интегрируем:

$$f(z) \approx \sum_{i=1}^{k} \theta_i b_i(z), \tag{18.33}$$

$$\int_{\mathcal{Z}} f(z) b_j(z) p(z) dz \approx \int_{\mathcal{Z}} \left(\sum_{i=1}^{k} \theta_i b_i(z) \right) b_j(z) p(z) dz = \tag{18.34}$$

$$= \sum_{i=1}^{k} \theta_i \int_{\mathcal{Z}} b_i(z) b_j(z) p(z) dz = \tag{18.35}$$

$$= \theta_j \int_{\mathcal{Z}} b_j(z)^2 p(z) dz, \tag{18.36}$$

используя свойство ортогональности из уравнения (18.20).

Отсюда следует, что j-й коэффициент имеет вид

$$\theta_j = \frac{\int_{\mathcal{Z}} f(z) b_j(z) p(z) dz}{\int_{\mathcal{Z}} b_j(z)^2 p(z) dz}. \tag{18.37}$$

Знаменатель уравнения (18.37) обычно имеет известное аналитическое решение или может быть вычислен заранее. Вычисление коэффициента, таким образом, требует численного интегрирования в числителе, которое можно выполнить с помощью квадратурных формул.[8]

18.3.3. Многомерный случай

Полиномиальный хаос может применяться к функциям с несколькими случайными аргументами. Многомерные базисные функции по m переменным строятся как произведение по одномерным ортогональным полиномам:

---

[8] Квадратура Гаусса реализована в пакете `QuadGK.jl` через функцию `quadgk` и описана в приложении В.8. Квадратурные правила (18.37) можно также получить, используя собственные значения и собственные векторы трехдиагональной матрицы, сформированной с использованием коэффициентов a_i и b_i из уравнения (18.30) [59].

$$b_i(\mathbf{z}) = \prod_{j=1}^{m} b_{a_j}(z_j), \tag{18.38}$$

где **a** — вектор назначения, который присваивает a_j-ую базисную функцию j-му случайному компоненту. Эта конструкция базисной функции продемонстрирована в примере 18.5.

Пример 18.5. Построение многомерного базисного полиномиального хаоса на основе уравнения (18.38)

Рассмотрим трехмерную модель полиномиального хаоса, в которой одна из многомерных базисных функций имеет вид $b(\mathbf{z}) = b_3(z_1)b_1(z_2)b_3(z_3)$. Соответствующий вектор назначения равен **a** = [3, 1, 3].

Распространенным методом построения многомерных базисных функций является генерация одномерных ортогональных полиномов для каждой случайной величины, а затем построение многомерной базисной функции для каждой возможной комбинации.[9] Эта процедура реализована в алгоритме 18.4. Построение базисных функций таким образом предполагает, что переменные независимы. Взаимозависимость может быть решена с использованием того же преобразования, что обсуждалось в разделе 18.2.

Алгоритм 18.4. Метод построения многомерных базисных функций, в котором **base1d** содержит списки одномерных ортогональных базисных функций для каждой случайной величины

```
function polynomial_chaos_bases(bases1d)
    bases = []
    for a in product(bases1d...)
        push!(bases,
            z -> prod(b(z[i]) for (i, b) in enumerate(a))
        )
    end
    return bases
end
```

Многомерное приближение полиномиального хаоса с k базисными функциями все еще является линейной комбинацией членов

[9] Здесь количество многомерных экспоненциальных базисных функций растет экспоненциально в зависимости от количества переменных.

$$f(\mathbf{z}) \approx \hat{f}(\mathbf{z}) = \sum_{i=1}^{k} \theta_i b_i(\mathbf{z}), \tag{18.39}$$

где математическое ожидание и дисперсия могут быть вычислены с использованием уравнений из раздела 18.3.1 при условии, что $b_1(\mathbf{z}) = 1$.

18.4. Байесовский метод Монте-Карло

Гауссовские процессы, описанные в главе 16, представляют собой распределения вероятностей по функциям. Их можно использовать как суррогаты для стохастических целевых функций. Мы можем включить априорную информацию, такую как ожидаемая гладкость целевой функции, в процесс, известный как *байесовский метод Монте-Карло* или *квадратура Байеса–Эрмита*.

Рассмотрим гауссовский процесс, аппроксимирующий несколько точек с одинаковым значением для расчетной точки $\mathbf{x}$, но разными значениями для неопределенной точки $\mathbf{z}$. Полученный гауссовский процесс представляет собой распределение по функциям на основе наблюдаемых данных. При получении ожидаемого значения посредством интегрирования мы должны учитывать ожидаемое значение функций в распределении вероятностей, представленных гауссовским процессом $p\left(\hat{f}\right)$:

$$\mathbb{E}_{\mathbf{z}\sim p}[f] \approx \mathbb{E}_{\hat{f}\sim p(\hat{f})}\left[\hat{f}\right] = \tag{18.40}$$

$$= \int_{\hat{\mathcal{F}}} \left(\int_{\mathcal{Z}} \left(\hat{f}(\mathbf{z}) p(\mathbf{z}) d\mathbf{z} \right) \right) p\left(\hat{f}\right) d\hat{f} = \tag{18.41}$$

$$= \int_{\mathcal{Z}} \left(\int_{\hat{\mathcal{F}}} \left(\hat{f}(\mathbf{z}) p\left(\hat{f}\right) d\hat{f} \right) \right) p(\mathbf{z}) d\mathbf{z} = \tag{18.42}$$

$$= \int_{\mathcal{Z}} \hat{\mu}(\mathbf{z}) p(\mathbf{z}) d\mathbf{z}. \tag{18.43}$$

Здесь $\hat{\mu}(\mathbf{z})$ — это предсказанное математическое ожидание гауссовского процесса, а $\mathcal{F}$ — пространство функций. Дисперсия оценки равна

$$\mathrm{Var}_{\mathbf{z}\sim p}[f] \approx \mathrm{Var}_{\hat{f}\sim p\left(\hat{f}\right)}\left[\hat{f}\right] = \tag{18.44}$$

$$= \int_{\hat{\mathcal{F}}} \left(\int_{\mathcal{Z}} \hat{f}(\mathbf{z}) p(\mathbf{z}) d\mathbf{z} - \int_{\mathcal{Z}} \mathbb{E}\left[\hat{f}(\mathbf{z}')\right] p(\mathbf{z}') d\mathbf{z}' \right)^2 p\left(\hat{f}\right) d\hat{f} = \tag{18.45}$$

$$= \int_{\mathcal{Z}} \int_{\mathcal{Z}} \int_{\hat{\mathcal{F}}} \left[\hat{f}(\mathbf{z}) - \mathbb{E}\left[\hat{f}(\mathbf{z})\right]\right] \left[\hat{f}(\mathbf{z}') - \mathbb{E}\left[\hat{f}(\mathbf{z}')\right]\right] p\left(\hat{f}\right) d\hat{f} p(\mathbf{z}) p(\mathbf{z}') d\mathbf{z} d\mathbf{z}' = \tag{18.46}$$

$$= \int_{\mathcal{Z}} \int_{\mathcal{Z}} \mathrm{Cov}\left(\hat{f}(\mathbf{z}), \hat{f}(\mathbf{z}')\right) p(\mathbf{z}) p(\mathbf{z}') d\mathbf{z} d\mathbf{z}', \tag{18.47}$$

где Cov — апостериорная ковариация гауссовского процесса:

$$\mathrm{Cov}\left(\hat{f}(\mathbf{z}), \hat{f}(\mathbf{z}')\right) = k(\mathbf{z}, \mathbf{z}') - k(\mathbf{z}, Z)\mathbf{K}(Z, Z)^{-1} k(Z, \mathbf{z}'). \tag{18.48}$$

Здесь Z содержит наблюдаемые входные данные.

Существуют аналитические выражения для математического ожидания и дисперсии в частном случае, когда z является нормально распределенной случайной величиной.[10] Для гауссового ядра

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}\sum_{i=1}^{n} \frac{(x_i - x_i')^2}{w_i^2}\right) \tag{18.49}$$

математическое ожидание нормально распределенной случайной величины $\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}_\mathbf{z}, \boldsymbol{\Sigma}_\mathbf{z})$ равно

$$\mathbb{E}_{\mathbf{z}\sim p}[f] = \mathbf{q}^{\mathrm{T}} \mathbf{K}^{-1} \mathbf{y}, \tag{18.50}$$

где

$$q_i = \left|\mathbf{W}^{-1}\boldsymbol{\Sigma}_\mathbf{z} + \mathbf{I}\right|^{-1/2} \exp\left(-\frac{1}{2}\left(\boldsymbol{\mu}_\mathbf{z} - \hat{\mu}\left(\mathbf{z}^{(i)}\right)\right)^{\mathrm{T}} \left(\boldsymbol{\Sigma}_\mathbf{z} + \mathbf{W}\right)^{-1} \left(\boldsymbol{\mu}_\mathbf{z} - \mathbf{z}^{(i)}\right)\right). \tag{18.51}$$

Здесь $\mathbf{W} = diag\left[w_1^2, ..., w_n^2\right]$, и мы построили наш гауссовский процесс, используя выборки $(\mathbf{z}^{(i)}, y_i)$ для $i \in \{1, ..., m\}$ [56].

Дисперсия равна

$$\mathrm{Var}_{\mathbf{z}\sim p}\left[f\right] = \left|2\mathbf{W}^{-1}\boldsymbol{\Sigma}_\mathbf{z} + \mathbf{I}\right|^{-1/2} - \mathbf{q}^{\mathrm{T}}\mathbf{K}^{-1}\mathbf{q}. \tag{18.52}$$

Даже когда аналитические выражения недоступны, существует много проблем, для которых численная оценка ожидания является достаточно недорогой, чтобы подход с использованием гауссовского процесса был лучше, чем оценка по методу Монте-Карло.

Байесовский метод Монте-Карло реализован в алгоритме 18.5 и проиллюстрирован в примере 18.6.

---

[10] Требуется также, чтобы ковариационная функция подчинялась правилу корреляции произведений, т.е. ее можно записать как произведение значений одномерной положительно-определенной функции r:

$$k(\mathbf{x}, \mathbf{x}') = \prod_{i=1}^{n} r(x_i - x_i').$$

Аналитические результаты существуют также для полиномиальных ядер и смесей нормальных распределений [125].

Алгоритм 18.5. Метод получения байесовской оценки Монте-Карло для ожидаемого значения функции при гауссовском процессе `GP` с гауссовым ядром и весами `w`, где переменные взяты из нормального распределения со математическим ожиданием `μz` и ковариацией `Σz`

```
function bayesian_monte_carlo(GP, w, μz, Σz)
    W = Matrix(Diagonal(w .^ 2))
    invK = inv(K(GP.X, GP.X, GP.k))
    q = [exp(-((z - μz) ⋅ (inv(W + Σz) * (z - Σz))) / 2) for z in GP.X]
    q .*= (det(W \ Σz + I)) ^ (-0.5)
    μ = q' * invK * GP.y
    ν = (det(2W \ Σz + I)) ^ (-0.5) - (q' * invK * q)[1]
    return (μ, ν)
end
```

Пример 18.6. Пример использования байесовского Монте-Карло для оценки ожидаемого значения и дисперсии функции. На рисунке сравнивается байесовский метод Монте-Карло с выборочным средним для оценки ожидаемого значения функции. В каждый метод передаются одни и те же случайные значения `z`, сгенерированные для каждого оцененного значения `x`

Рассмотрим еще раз оценку ожидаемого значения и дисперсии значений $f(x, z) = \sin(x + z_1)\cos(x + z_2)$, где z_1 и z_2 — гауссовский шум с нулевым математическим ожиданием и дисперсиями 1 и 1/2 соответственно: $\boldsymbol{\mu}_z = [0, 0]$ и $\boldsymbol{\Sigma}_z = \text{diag}([1, 1/2])$.

Мы используем байесовский метод Монте-Карло с гауссовым ядром с единичными весами $\mathbf{w} = [1, 1]$ для $\mathbf{x} = 0$ с выборками $Z = \{[0, 0], [1, 0], [-1, 0], [0, 1], [0, -1]\}$.

Мы вычисляем:

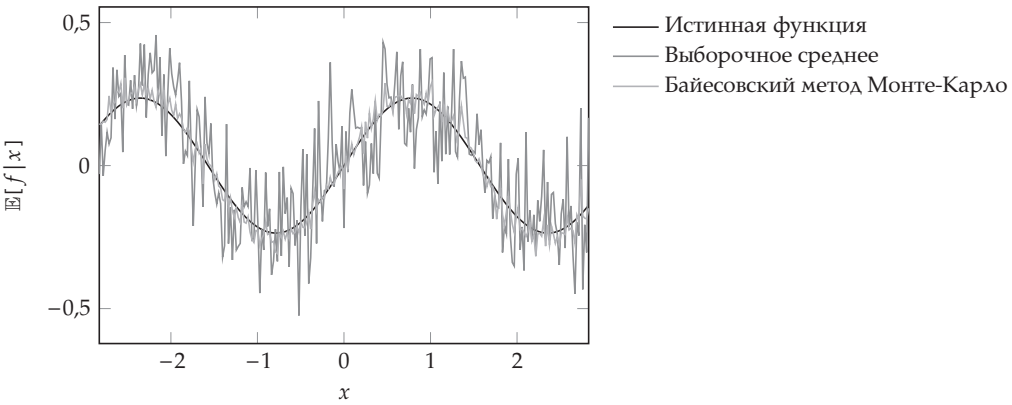
$$\mathbf{W} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

$$\mathbf{K} = \begin{bmatrix} 1 & 0{,}607 & 0{,}607 & 0{,}607 & 0{,}607 \\ 0{,}607 & 1 & 0{,}135 & 0{,}368 & 0{,}368 \\ 0{,}607 & 0{,}135 & 1 & 0{,}368 & 0{,}368 \\ 0{,}607 & 0{,}368 & 0{,}368 & 1 & 0{,}135 \\ 0{,}607 & 0{,}368 & 0{,}368 & 0{,}135 & 1 \end{bmatrix},$$

$$\mathbf{q} = [0{,}577;\ 0{,}450;\ 0{,}450;\ 0{,}417;\ 0{,}417]$$

$$\mathbb{E}_{\mathbf{z}\sim p}[f] = 0{,}0;$$
$$\mathrm{Var}_{\mathbf{z}\sim p}[f] = 0{,}327.$$

Ниже мы строим ожидаемое значение как функцию от x, используя тот же подход с десятью случайными выборками $\mathbf{z}$ в каждой точке.



18.5. Резюме

- Ожидаемое значение и дисперсия целевой функции полезны при оптимизации проблем, связанных с неопределенностью, но надежное вычисление этих величин может быть сложной задачей.
- Один из самых простых подходов состоит в оценке моментов с использованием выборки в процессе, известном как интегрирование методом Монте-Карло.
- Другие подходы, такие как аппроксимация с помощью ряда Тейлора, используют знания о частных производных целевой функции.
- Полиномиальный хаос — это мощный метод распространения неопределенности, основанный на ортогональных полиномах.
- Байесовский метод Монте-Карло использует гауссовские процессы для эффективного достижения моментов с аналитическими результатами для гауссовых ядер.

18.6. Упражнения

Упражнение 18.1. Предположим, мы извлекаем выборку из одномерного нормального распределения. Какова вероятность того, что наша выборка попадает в интервал одного стандартного отклонения от математического ожидания $x \in [\mu - \sigma, \mu + \sigma]$? Какова вероятность того, что наша выборка попадет в интервал менее одного стандартного отклонения выше среднего ($x < \mu + \sigma$)?

Упражнение 18.2. Пусть $x^{(1)}, x^{(2)}, ..., x^{(m)}$ — случайная выборка независимых одинаково распределенных значений размера m из распределения с математическим ожиданием μ и дисперсией v. Покажите, что дисперсия математического ожидания выборки $\text{Var}(\hat{\mu})$ равна v/m.

Упражнение 18.3. Выведите уравнение рекуррентного соотношения (18.29), которому удовлетворяют все ортогональные полиномы.

Упражнение 18.4. Предположим, что мы подобрали модель полиномиального хаоса для целевой функции $f(\mathbf{x}, \mathbf{z})$ в конкретной расчетной точке $\mathbf{x}$, используя m оценок с помощью $\mathbf{z}^{(1)}, ..., \mathbf{z}^{(m)}$. Выведите выражение для оценки частной производной коэффициентов полиномиального хаоса по компоненту x_i.

Упражнение 18.5. Рассмотрим целевую функцию $f(\mathbf{x}, \mathbf{z})$ с расчетными переменными $\mathbf{x}$ и случайными переменными $\mathbf{z}$. Как обсуждалось в главе 17, оптимизация в условиях неопределенности часто подразумевает минимизацию линейной комбинации оцененного математического ожидания и дисперсии:

$$f_{\text{mod}}(\mathbf{x}, \mathbf{z}) = \alpha\hat{\mu}(\mathbf{x}) + (1-\alpha)\hat{v}(\mathbf{x}).$$

Как можно использовать полиномиальный хаос для оценки градиента функции f_{mod} относительно проектной переменной $\mathbf{x}$?

Глава 19

Дискретная оптимизация

Предыдущие главы были посвящены задачам оптимизации, связанным с непрерывными переменными. Однако многие задачи имеют переменные, которые являются естественно дискретными, например производственные задачи, связанные с механическими компонентами, имеющим фиксированные размеры, или задачи навигации, связанные с выбором дискретных траекторий. Особенность задач *дискретной оптимизации* заключается в том, что переменные должны выбираться из дискретного множества. Одни дискретные задачи оптимизации имеют бесконечные пространства проектных параметров, а другие — конечные.[1] Даже для конечных задач, где, в принципе, мы могли бы перебрать все возможные решения, это практически невозможно с вычислительной точки зрения. В этой главе рассматриваются как точные, так и приближенные подходы к решению дискретных задач оптимизации, которые избегают перебора. Многие из описанных ранее методов, таких как имитация отжига и генетическое программирование, могут быть легко адаптированы для задач дискретной оптимизации, но мы сосредоточим эту главу на группе методов, которые еще не обсуждали.

19.1. Задачи целочисленного программирования

Задача целочисленного программирования — это задача линейного программирования[2] с целочисленными ограничениями. Под целочисленными ограничениями мы подразумеваем, что переменные должны принадлежать множеству целых чисел.[3] Задачи целочисленного программирования иногда называют зада-

[1] Дискретная оптимизация с конечным пространством проектирования иногда называется *комбинаторной оптимизацией*. Обзор см. в [88].

[2] См. главу 11.

[3] Целочисленное программирование — это очень зрелая область, в которой используются приложения для исследования операций, сетей связи, планирования задач и других дисциплин. Современное программное обеспечение для решения таких задач, например Gurobi и CPLEX, может без труда решать задачи с миллионами переменных. Есть пакеты для языка Julia, которые предоставляют доступ к Gurobi, CPLEX и многим другим пакетам.

чами целочисленного линейного программирования, чтобы подчеркнуть, что целевая функция и ограничения являются линейными.

Задача целочисленного программирования в стандартной форме выражается как:

$$\begin{aligned} &\min_{\mathbf{x}} \mathbf{c}^{\mathsf{T}}\mathbf{x} \\ &\text{при условии, что } \mathbf{A}\mathbf{x} \leq \mathbf{b}, \\ &\qquad\qquad\qquad\quad \mathbf{x} \geq 0, \\ &\qquad\qquad\qquad\quad \mathbf{x} \in \mathbb{Z}^n, \end{aligned} \tag{19.1}$$

где $\mathbb{Z}^n$ — множество n-мерных целочисленных векторов.

Как и задачи линейного программирования, задачи целочисленного программирования часто имеют ограничения в форме равенства. Преобразование целочисленной задачи в форму с ограничениями в виде равенства часто требует добавления дополнительных переменных, которые не обязаны быть целочисленными. Таким образом, задача целочисленного программирования с ограничениями в виде равенства имеет вид:

$$\begin{aligned} &\min_{\mathbf{x}} \mathbf{c}^{\mathsf{T}}\mathbf{x} \\ &\text{при условии, что } \mathbf{A}\mathbf{x} + \mathbf{s} = \mathbf{b}, \\ &\qquad\qquad\qquad\quad \mathbf{x} \geq 0, \\ &\qquad\qquad\qquad\quad \mathbf{s} \geq 0, \\ &\qquad\qquad\qquad\quad \mathbf{x} \in \mathbb{Z}^n. \end{aligned} \tag{19.2}$$

В более общем смысле *смешанная задача целочисленного программирования* (алгоритм 19.1) включает как непрерывные, так и дискретные компоненты проектирования. Такая задача с ограничениями в виде равенства выражается как:

$$\begin{aligned} &\min_{\mathbf{x}} \mathbf{c}^{\mathsf{T}}\mathbf{x} \\ &\text{при условии, что } \mathbf{A}\mathbf{x} = \mathbf{b}, \\ &\qquad\qquad\qquad\quad \mathbf{x} \geq 0, \\ &\qquad\qquad\qquad\quad \mathbf{x} \in \mathbb{Z}^{\|\mathcal{D}\|}, \end{aligned} \tag{19.3}$$

где $\mathcal{D}$ — это подмножество индексов в расчетных переменных, которые должны быть дискретными. Здесь $\mathbf{x}_{\mathcal{D}}$ представляет собой вектор дискретных переменных и $\mathbf{x}_{\mathcal{C}}$ — вектор непрерывных переменных.

Алгоритм 19.1. Смешанная задача целочисленного программирования в виде (19.3). Здесь `D` — это дискретное множество переменных

```
mutable struct MixedIntegerProgram
    A
    b
```

```
    c
    D
end
```

19.2. Округление

Общий подход к дискретной оптимизации заключается в ослаблении ограничения на то, что расчетные точки должны принадлежать дискретному множеству. Преимущество этого ослабления состоит в том, что мы можем применить методы, такие как градиентный спуск или линейное программирование, использующие непрерывный характер целевой функции для выбора направления поиска. Найдя непрерывное решение, переменные *округляют* до ближайшего возможного дискретного значения.

Есть потенциальные проблемы, связанные с округлением. Округление может привести к недопустимой точке расчета, как показано на рис. 19.1. Даже если округление приводит к допустимой точке, она может быть далека от оптимальной (рис. 19.2). Добавление дискретного ограничения обычно ухудшает объективное значение (пример 19.1). Однако для некоторых задач мы можем показать, что ослабленное решение близко к оптимальному дискретному решению.

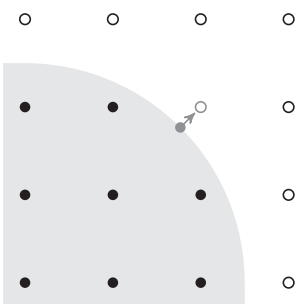


Рис. 19.1. Округление может привести к недопустимой расчетной точке

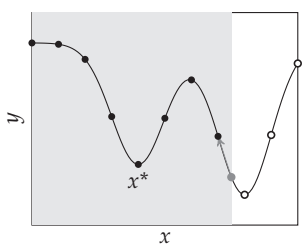


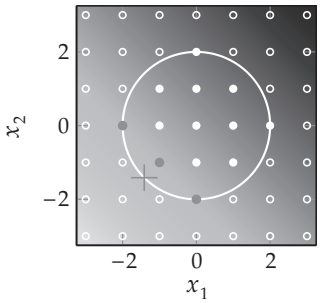
Рис. 19.2. Ближайшая возможная дискретная конструкция может быть значительно хуже, чем лучшая возможная дискретная конструкция

Пример 19.1. Дискретные варианты задач ограничивают решение, часто приводя к худшим решениям, чем их непрерывные аналоги

Дискретная оптимизация накладывает на решения целочисленные условия. Рассмотрим задачу

$$\begin{aligned} &\max_{\mathbf{x}} && x_1 + x_2 \\ &\text{при условии, что} && \|\mathbf{x}\| \le 2, \\ & && \mathbf{x} - \text{целочисленная}. \end{aligned}$$

В непрерывном случае оптимальной является точка $\mathbf{x}^* = \left[-\sqrt{2},\ -\sqrt{2}\right]$ со значением $y = -2\sqrt{2} \approx -2{,}828$. Если x_1 и x_2 ограничены целочисленными значениями, то лучшее, что мы можем сделать, это получить значение $y = -2$ в точке $\mathbf{x}^* \in \{[-2, 0], [-1, -1], [0, -2]\}$.



Мы можем решать задачи целочисленного программирования, используя округление, удаляя целочисленное ограничение, решая соответствующую задачу линейного программирования, или LP, а затем округляя решение до ближайшего целого числа.

Алгоритм 19.2. Методы преобразования смешанной задачи целочисленного линейного программирования в задачу линейного программирования и решения смешанной задачи целочисленного линейного программирования с помощью округления. Оба метода принимают смешанную задачу целочисленного линейного программирования `MIP`. Решение, полученное округлением, может быть неоптимальным или недопустимым

```
relax(MIP) = LinearProgram(MIP.A, MIP.b, MIP.c)
function round_ip(MIP)
    x = minimize_lp(relax(MIP))
    for i in MIP.D
        x[i] = round(Int, x[i])
    end
    return x
end
```

Мы можем показать, что округление непрерывного решения для ограничения $\mathbf{Ax} \le \mathbf{b}$, когда $\mathbf{A}$ — целочисленная матрица, никогда не бывает слишком далеко от

оптимального целочисленного решения [28]. Если $\mathbf{x}_C^*$ — оптимальное решение задачи LP с матрицей **A** размером $m \times n$, то существует такое оптимальное дискретное решение $\mathbf{x}_d^*$, что $\left\|\mathbf{x}_C^* - \mathbf{x}_d^*\right\|_\infty$ меньше максимального абсолютного значения определителей подматриц матрицы **A**, умноженного на n, или равно ему.

Вектор $\mathbf{c}$ не обязательно должен быть целым, чтобы задача LP имела оптимальное целочисленное решение, потому что допустимая область определяется исключительно матрицей **A** и вектором $\mathbf{b}$. Некоторые подходы используют двойственную формулировку для задачи LP, которая имеет допустимую область, зависящую от $\mathbf{c}$, и в этом случае также требуется, чтобы точка $\mathbf{c}$ была целочисленной.

В особом случае *вполне унимодулярных* задач целочисленного программирования, в которой **A**, $\mathbf{b}$ и $\mathbf{c}$ являются целочисленными, причем матрица **A** — вполне унимодулярная, симплекс-метод гарантированно возвращает целочисленное решение. Матрица называется вполне унимодулярной, если определитель каждой *подматрицы*[4] равен 0, 1 или –1, а обратная матрица полностью унимодулярной матрицы также является целочисленной. Фактически каждое вершинное решение вполне унимодулярной задачи целочисленного программирования является целочисленным. Таким образом, каждый система уравнений $\mathbf{Ax} = \mathbf{b}$ для унимодулярной матрицы **A** и целочисленного вектора $\mathbf{b}$ имеет целочисленное решение.

Несколько матриц и их полная унимодулярность рассматриваются в примере 19.2. Методы определения того, являются ли матрица или задача целочисленного линейного программирования вполне унимодулярными, приведены в алгоритме 19.3.

Пример 19.2. Примеры вполне унимодулярных матриц

Рассмотрим следующие матрицы:

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & -1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & 0 & 0 & 0 \\ 1 & 0 & -1 & -1 & 0 \\ 0 & 1 & 1 & 0 & -1 \end{bmatrix}$$

Левая матрица не вполне унимодулярная, поскольку

$$\begin{vmatrix} 1 & 1 \\ 1 & -1 \end{vmatrix} = -2.$$

Две другие матрицы вполне унимодулярны.

[4] Подматрица — это матрица, полученная путем удаления строк и/или столбцов другой матрицы.

Алгоритм 19.3. Методы определения того, являются ли матрица `A` и смешанная задача целочисленного программирования **`MIP`** вполне унимодулярными. Метод **`isint`** возвращает **`true`**, если заданное значение является целым

```
isint(x, ε = 1e-10) = abs(round(x) - x) ≤ ε
function is_totally_unimodular(A :: Matrix)
      # все элементы должны принимать значения {0, 1, -1}
      if any(a ∉ (0, -1, 1) for a in A)
         return false
      end
      # полный перебор определителей всех подматриц
      r,c = size(A)
      for i in 1 : min(r, c)
          for a in subsets(1 : r, i)
              for b in subsets(1 : c, i)
                  B = A[a, b]
                  if det(B) ∉ (0, -1, 1)
                     return false
                  end
              end
          end
      end
      return true
end
function is_totally_unimodular(MIP)
      return is_totally_unimodular(MIP.A) &&
              all(isint, MIP.b) && all(isint, MIP.c)
end
```

19.3. Метод секущих плоскостей

Метод секущих плоскостей является точным методом решения смешанных задач целочисленного программирования, в которых матрица **A** не является вполне унимодулярной [60]. Современные практические методы решения задач целочисленного программирования используют *алгоритмы ветвлений и отсечений* [117], которые объединяют метод секущих плоскостей с методом ветвей и границ, обсуждаемым в следующем разделе. Метод секущих плоскостей решает ослабленную задачу LP и добавляет линейные ограничения, которые приводят к оптимальному решению.

Метод секущих плоскостей начинается с решения $\mathbf{x}_{\mathcal{C}}^*$ ослабленной задачи LP, которое должно быть вершиной, удовлетворяющей системе уравнений $\mathbf{Ax} = \mathbf{b}$. Если $\mathcal{D}$ компонентов вектора $\mathbf{x}$ являются целочисленными, то он также является оптимальным решением исходной смешанной задачи целочисленного программирования, и все готово. Пока $\mathcal{D}$ компонентов вектора $\mathbf{x}$ не являются целыми, мы находим гиперплоскость, относительно которой вектор $\mathbf{x}$ лежит на одной стороне, а все возможные дискретные решения — на другой. Эта *секущая плоскость* является дополнительным линейным ограничением для исключения $\mathbf{x}_{\mathcal{C}}^*$. Затем расширенная задача LP решается для новой точки $\mathbf{x}_{\mathcal{C}}^*$.

На каждой итерации алгоритма 19.4 вводятся секущие плоскости, которые делают нецелые компоненты вектора $\mathbf{x}_{\mathcal{C}}^*$ недопустимыми при сохранении условий допустимости ближайших целочисленных решений и остальной части допустимого множества. Затем решается целочисленная задача, дополненная ограничениями, заданными секущими плоскостями, и находится новое ослабленное решение. Этот процесс иллюстрирует рис. 19.3.

Алгоритм 19.4. Метод секущих плоскостей решает заданную смешанную задачу целочисленного программирования **MIP** и возвращает оптимальный вектор параметров. Если допустимого решения не существует, то выдается ошибка. Вспомогательная функция **frac** возвращает дробную часть числа. Реализация алгоритма 11.5 **minimal_lp** была скорректирована, чтобы возвращать базисные и небазисные индексы **b_inds** и **v_inds** вместе с оптимальным вектором **x**

```
frac(x) = modf(x)[1]
function cutting_plane(MIP)
    LP = relax(MIP)
    x, b_inds, v_inds = minimize_lp(LP)
    n_orig = length(x)
    D = copy(MIP.D)
    while !all(isint(x[i])for i in D)
       AB, AV = LP.A[:, b_inds], LP.A[:, v_inds]
       Abar = AB \ AV
       b = 0
       for i in D
           if !isint(x[i])
               b += 1
               A2 = [LP.Azeros(size(LP.A, 1));
                     zeros(1, size(LP.A,2) + 1)]
               A2[end, end] = 1
```

```
            A2[end, v_inds] = (x -> floor(x) - x).(Abar[b, :])
            b2 = vcat(LP.b, -frac(x[i]))
            c2 = vcat(LP.c, 0)
            LP = LinearProgram(A2, b2, c2)
        end
    end
    x, b_inds, v_inds = minimize_lp(LP)
  end
  return x[1 : n_orig]
end
```

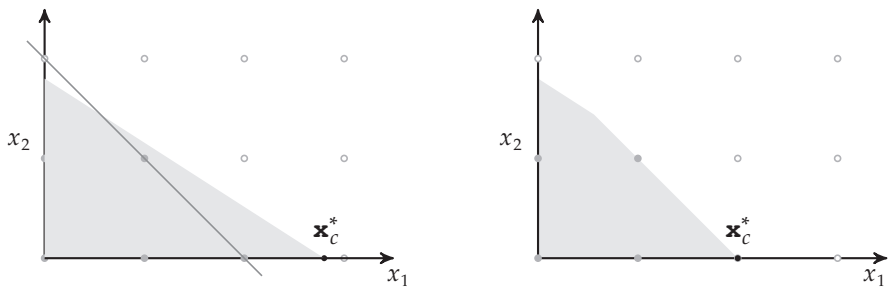


Рис. 19.3. Метод секущих плоскостей вводит ограничения до тех пор, пока решение задачи LP не станет целочисленным. Секущая плоскость показана красной линией слева. Допустимая область расширенной задачи LP находится справа

Мы хотим добавить ограничения, которые исключают нецелые компоненты $\mathbf{x}_{\mathcal{C}}^*$. В задаче LP в форме равенства с ограничением $\mathbf{Ax} = \mathbf{b}$ можно разделить вершинное решение $\mathbf{x}_{\mathcal{C}}^*$ (см. раздел 11.2.1) и получить следующее разложение:

$$\mathbf{A}_{\mathcal{B}}\mathbf{x}_{\mathcal{B}}^* + \mathbf{A}_{\mathcal{V}}\mathbf{x}_{\mathcal{V}}^* = \mathbf{b}, \tag{19.4}$$

где $\mathbf{x}_{\mathcal{V}}^* = 0$. Таким образом, нецелые компоненты $\mathbf{x}_{\mathcal{C}}^*$ будут встречаться только в $\mathbf{x}_{\mathcal{B}}^*$. Мы можем ввести дополнительное ограничение в виде неравенств для каждого $b \in \mathcal{B}$, чтобы x_b^* был нецелочисленным вектором[5]:

$$x_b^* - \lfloor x_b^* \rfloor - \sum_{v \in \mathcal{V}} \left(\overline{A}_{bv} - \lfloor \overline{A}_{bv} \rfloor \right) x_v \le 0, \tag{19.5}$$

где $\overline{\mathbf{A}} = \mathbf{A}_{\mathcal{B}}^{-1}\mathbf{A}_{\mathcal{V}}$. Эти секущие плоскости используют только $\mathcal{V}$-компоненты, чтобы отсечь нецелочисленные компоненты вектора $\mathbf{x}_{\mathcal{C}}^*$.

Введение ограничения в виде секущей плоскости отсекает ослабленное $\mathbf{x}_{\mathcal{C}}^*$, потому что все $x_{\mathcal{V}}$ равны нулю:

[5] Здесь $\lfloor x \rfloor$ — операция округления x с недостатком до ближайшего целого числа.

$$\underbrace{x_b^* - \lfloor x_b^* \rfloor}_{>0} - \underbrace{\sum_{v \in \mathcal{V}} \left(\bar{A}_{bv} - \lfloor \bar{A}_{bv} \rfloor \right) x_v}_{0} > 0. \tag{19.6}$$

Секущая плоскость записывается в форме равенства с использованием дополнительной целочисленной переменной x_k:

$$x_k + \sum_{v \in \mathcal{V}} \left(\lfloor \bar{A}_{bv} \rfloor - \bar{A}_{bv} \right) x_v = \lfloor x_b^* \rfloor - x_b^*. \tag{19.7}$$

Таким образом, каждая итерация алгоритма 19.4 увеличивает количество ограничений и переменных до тех пор, пока решение задачи LP не приведет к целочисленному решению. Возвращаются только те компоненты, которые соответствуют исходным переменным.

В примере 19.3 метод секущих плоскостей используется для решения простой задачи целочисленного линейного программирования.

Пример 19.3. Метод секущих плоскостей, используемый для решения задачи целочисленного программирования

Рассмотрим задачу целочисленного программирования

$$\min_{\mathbf{x}} 2x_1 + x_2 + 3x_3$$

$$\text{при условии, что } \begin{bmatrix} 0{,}5 & -0{,}5 & 1{,}0 \\ 2{,}0 & 0{,}5 & -1{,}5 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 2{,}5 \\ -1{,}5 \end{bmatrix}.$$

$$\mathbf{x} \geq 0 \qquad \mathbf{x} \in \mathbb{Z}^3$$

Ослабленное решение равно $\mathbf{x}^* \approx [0{,}818;\ 2{,}091]$, что приводит к следующему разложению:

$$\mathbf{A}_{\mathcal{B}} = \begin{bmatrix} 0{,}5 & 1 \\ 2 & -1{,}5 \end{bmatrix} \quad \mathbf{A}_{\mathcal{V}} = \begin{bmatrix} -0{,}5 \\ 0{,}5 \end{bmatrix} \quad \bar{\mathbf{A}} = \begin{bmatrix} -0{,}091 \\ -0{,}455 \end{bmatrix}.$$

Из уравнения (19.7) следует, что ограничение переменной x_1, заданное с помощью дополнительной переменной x_4, имеет вид

$$x_4 + \left(\lfloor -0{,}091 \rfloor - -0{,}091 \right) x_2 = \lfloor 0{,}818 \rfloor - 0{,}818;$$

$$x_4 - 0{,}909 x_2 = -0{,}818.$$

Ограничение для x_3, заданное с помощью дополнительной переменной x_5, имеет вид:

$$x_5 + \left([-0{,}455] - -0{,}455 \right) x_2 = \lfloor 2{,}091 \rfloor - 2{,}091;$$

$$x_5 - 0{,}545 x_2 = -0{,}091.$$

Получаем следующую модифицированную задачу целочисленного программирования:

$$\mathbf{A} = \begin{bmatrix} 0{,}5 & -0{,}5 & 1 & 0 & 0 \\ 2 & 0{,}5 & -1{,}5 & 0 & 0 \\ 0 & -0{,}909 & 0 & 1 & 0 \\ 0 & -0{,}545 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 2{,}5 \\ -1{,}5 \\ -0{,}818 \\ -0{,}091 \end{bmatrix} \quad \mathbf{c} = \begin{bmatrix} 2 \\ 1 \\ 3 \\ 0 \\ 0 \end{bmatrix}.$$

Решая модифицированную задачу LP, получаем $\mathbf{x}^* \approx [0{,}9;\ 0{,}9;\ 2{,}5;\ 0{,}0;\ 0{,}4]$. Поскольку координаты этой точки не являются целочисленными, повторим эту процедуру с ограничениями:

$$x_6 - 0{,}9x_4 = -0{,}9; \qquad x_7 - 0{,}9x_4 = -0{,}9;$$
$$x_8 - 0{,}5x_4 = -0{,}5; \qquad x_9 - 0{,}4x_4 = -0{,}4.$$

После решения третьей задачи LP, получаем $\mathbf{x}^* \approx [1, 2, 3, 1, 1, 0, 0, 0, 0]$ и окончательное решение $\mathbf{x}_i^* = [1, 2, 3]$.

19.4. Метод ветвей и границ

Один из методов нахождения глобального минимума дискретной задачи, такой как целочисленная задача, состоит в перечислении всех возможных решений. *Метод ветвей и границ* [94] гарантирует, что оптимальное решение будет найдено без перебора всех возможных решений. Многие коммерческие программы для решения задач целочисленного программирования используют идеи как метода секущих плоскостей, так и ветвей и границ. Метод получил свое имя от *операции ветвления*, которая разбивает пространство решений[6], и операции *ограничения*, которая вычисляет нижнюю границу для разбиения.

Метод ветвей и границ — это универсальный метод, который можно применять ко многим видам задач дискретной оптимизации, но мы сосредоточимся на том, как его можно использовать для целочисленного программирования. Алгоритм 19.5 описывает реализацию, использующую очередь с приоритетами, которая представляет собой структуру данных, связывающую приоритеты с элементами в коллекции. Мы можем добавить элемент и его значение приоритета в очередь с приоритетами, используя операцию `enqueue!`. Мы можем удалить элемент с минимальным значением приоритета, используя операцию `dequeue!`.

Алгоритм 19.5. Алгоритм ветвей и границ для решения смешанной задачи целочисленного программирования `MIP`. Вспомогательный метод `minimal_lp_and_y` решает задачу `LP`, возвращая и решение, и его значение. Если задача

[6] Подмножества обычно не пересекаются, но это не обязательно. Для того чтобы метод ветвей и границ работал, необходимо, чтобы по крайней мере одно подмножество имело оптимальное решение [9].

LP не имеет решения, метод возвращает значения **NaN** и **Inf**. Более сложные реализации для ускорения вычислений будут отбрасывать переменные, решения которых известны

```
import DataStructures: PriorityQueue
function minimize_lp_and_y(LP)
   try
      x = minimize_lp(LP)
      return (x, x ⋅ LP.c)
   catch
      return (fill(NaN, length(LP.c)), Inf)
   end
end
function branch_and_bound(MIP)
   LP = relax(MIP)
   x, y = minimize_lp_and_y(LP)
   n = length(x)
   x_best, y_best = deepcopy(x), Inf
   Q = PriorityQueue()
   enqueue!(Q, (LP, x, y), y)
   while !isempty(Q)
      LP, x, y = dequeue!(Q)
      if any(isnan.(x)) || all(isint(x[i]) for i in MIP.D)
          if y < y_best
               x_best, y_best = x[1 : n], y
          end
      else
          i = argmax([abs(x[i] - round(x[i])) for i in MIP.D])
          # x_i ≤ floor(x_i)
          A, b, c = LP.A, LP.b, LP.c
          A2 = [A zeros(size(A,1)); [j == i for j in 1 : size(A, 2)]' 1]
          b2, c2 = vcat(b, floor(x[i])), vcat(c, 0)
          LP2 = LinearProgram(A2, b2, c2)
          x2, y2 = minimize_lp_and_y(LP2)
          if y2 ≤ y_best
             enqueue!(Q, (LP2, x2, y2), y2)
          end
          # x_i ≥ ceil(x_i)
          A2 = [A zeros(size(A, 1)); [j == i for j in 1:size(A, 2)]' -1]
          b2, c2 = vcat(b, ceil(x[i])), vcat(c, 0)
```

```
            LP2 = LinearProgram(A2, b2, c2)
            x2, y2 = minimize_lp_and_y(LP2)
            if y2 ≤ y_best
               enqueue!(Q, (LP2, x2, y2), y2)
            end
        end
    end
    return x_best
end
```

Алгоритм начинается с объявления очереди с приоритетами, содержащей одиночную ослабленную LP исходной смешанной задачи целочисленного программирования. С этой задачей LP связано решение $\mathbf{x}_{\mathcal{C}}^*$ и значение целевой функции $y_{\mathcal{C}} = \mathbf{c}^{\mathrm{T}}\mathbf{x}_{\mathcal{C}}^*$. Значение целевой функции служит нижней границей решения и, таким образом, является приоритетом задачи LP в очереди с приоритетами. На каждой итерации алгоритма мы проверяем, пуста ли очередь с приоритетами. Если она не пустая, то мы удаляем из очереди задачу LP с наименьшим приоритетом. Если решение, связанное с этим элементом, имеет необходимые целочисленные компоненты, мы отслеживаем, является ли оно наилучшим целочисленным решением, найденным до сих пор.

Если у решения, удаленного из очереди, есть один или несколько компонентов в $\mathcal{D}$, которые являются нецелыми, мы выбираем из $\mathbf{x}_{\mathcal{C}}^*$ компонент, находящийся дальше всего от целочисленного значения. Предположим, что этот компонент соответствует i-й переменной. Выполним *ветвление*, рассматривая две новые задачи LP, каждая из которых создается путем добавления одного из следующих ограничений к удаленной задаче LP:[7]

$$x_i \le \left\lfloor x_{i,\mathcal{C}}^* \right\rfloor \quad \text{или} \quad x_i \ge \left\lceil x_{i,\mathcal{C}}^* \right\rceil, \tag{19.8}$$

как показано на рис. 19.4. Этот процесс продемонстрирован в примере 19.4.

Вычислим решение, связанное с этими двумя задачами LP, которые дают нижние границы значения исходной смешанной задачи целочисленного программирования. Если какое-либо решение снижает значение целевой функции по сравнению с наилучшим целочисленным решением, которое когда-либо наблюдалось, оно помещается в очередь с приоритетами. Если не существует решений, уступающих лучшему целочисленному решению, которое мы видели до сих пор, метод ветвей и границ сокращает пространство поиска. Процесс продолжается до тех пор, пока очередь с приоритетами не станет пустой, и мы вернем наилучшее из возможных целочисленных решений. Пример 19.5 демонстрирует, как метод вет-

[7] Здесь $\lceil x \rceil$ — операция округления x с избытком до ближайшего целого числа.

вей и границ можно применить к небольшой задаче целочисленного программирования.

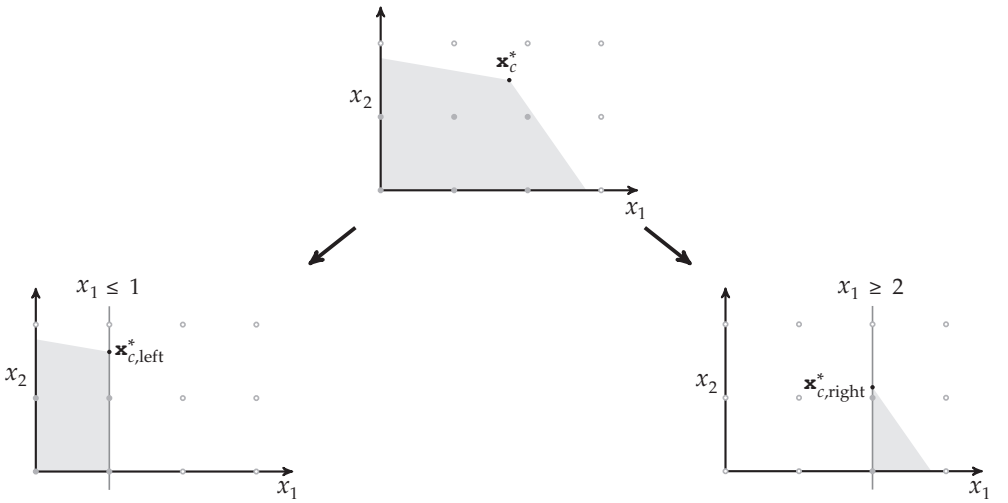


Рис. 19.4. Ветвление разбивает допустимое множество на подмножества с дополнительным ограничением в виде целочисленного неравенства

Пример 19.4. Пример отдельного применения шага ветвления в методе ветвей и границ

Рассмотрим ослабленное решение $\mathbf{x}_C^* = [3; 2{,}4; 1{,}2; 5{,}8]$ для задачи целочисленного программирования, в которой $\mathbf{c} = [-1, -2, -3, -4]$. Нижняя граница равна

$$y \ge \mathbf{c}^{\mathrm{T}}\mathbf{x}_C^* = -34{,}6.$$

Мы выполняем разветвление по нецелой координате $\mathbf{x}_C^*$, которая, как правило, является самой далекой от целочисленного значения. В данном случае выбираем первую нецелую координату, расстояние от которой до ближайшего целочисленного значения равно 0,4. Затем рассматриваем две новые задачи LP, одну с неравенством $x_2 \le 2$ в виде дополнительного ограничения, а другую — с неравенством $x_2 \ge 3$ в качестве дополнительного ограничения.

Пример 19.5. Применение метода ветвей и границ для решения задачи целочисленного программирования

Применим метод ветвей и границ для решения задачи целочисленного программирования, поставленной в примере 19.3. Как и ранее, ослабленное решение равно $\mathbf{x}_C^* = [0{,}818; 2{,}09]$, а соответствующее значение целевой функции — 7,909. Выполним ветвление по первой компоненте. В результате получим две

задачи целочисленного программирования: одну — с неравенством $x_1 \leq 0$, а другую — с неравенством $x_1 \geq 1$.

$$\mathbf{A}_{\text{left}} = \begin{bmatrix} 0,5 & -0,5 & 1 & 0 \\ 2 & 0,5 & -1,5 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{b}_{\text{left}} = \begin{bmatrix} 2,5 \\ -1,5 \\ 0 \end{bmatrix} \quad \mathbf{c}_{\text{left}} = \begin{bmatrix} 2 \\ 1 \\ 3 \\ 0 \end{bmatrix},$$

$$\mathbf{A}_{\text{right}} = \begin{bmatrix} 0,5 & -0,5 & 1 & 0 \\ 2 & 0,5 & -1,5 & 0 \\ 1 & 0 & 0 & -1 \end{bmatrix} \quad \mathbf{b}_{\text{right}} = \begin{bmatrix} 2,5 \\ -1,5 \\ 0 \end{bmatrix} \quad \mathbf{c}_{\text{right}} = \begin{bmatrix} 2 \\ 1 \\ 3 \\ 0 \end{bmatrix}.$$

Задача линейного программирования с матрицей $\mathbf{A}_{\text{left}}$ и неравенством $x_1 \leq 1$ не имеет допустимого решения. Задача линейного программирования с матрицей $\mathbf{A}_{\text{right}}$ и неравенством $x_1 \geq 1$ имеет ослабленное решение $\mathbf{x}_C^* = [1, 2, 3, 0]$ с соответствующим значением целевой функции, равным 13. Таким образом, мы получаем целочисленное решение $\mathbf{x}_i^* = [1, 2, 3]$.

19.5. Динамическое программирование

Динамическое программирование[8] — это метод, который можно применять к задачам с *оптимальной подструктурой* и *перекрывающимися подзадачами*. Задача имеет оптимальную подструктуру, если оптимальное решение может быть построено из оптимальных решений ее подзадач. Пример показан на рис. 19.5.

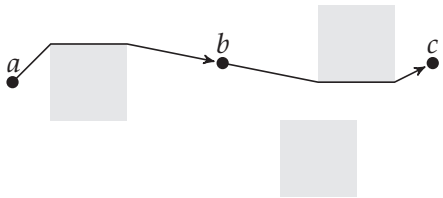


Рис. 19.5. Задачи о поиске кратчайшего пути имеют оптимальную подструктуру, потому что, если кратчайший путь от любой точки *a* до любой точки *c* проходит через точку *b*, то подпути $a \rightarrow b$ и $b \rightarrow c$ являются кратчайшими путями

[8] Термин *динамическое программирование* был выбран Беллманом для отражения изменяющегося во времени аспекта задач, к которым он его применял, и для того, чтобы избежать иногда негативных коннотаций в словах, таких как *исследование* и *математика*. Он писал: “Я думал, что динамическое программирование — это хорошее название. Это было то, на что даже конгрессмен не мог возразить. Поэтому я использовал его в качестве зонтика для своей деятельности” [15].

Задача с перекрывающимися подзадачами, решаемая рекурсивно, требует многократного решения одной и той же подзадачи. Вместо того чтобы экспоненциально много раз перебирать множество потенциальных решений, динамическое программирование либо сохраняет решения подзадач и, таким образом, избегает необходимости заново их решать, либо рекурсивно создает оптимальное решение за один проход. Задачи с рекуррентными отношениями часто имеют перекрывающиеся подзадачи. Пример показан на рис. 19.6.

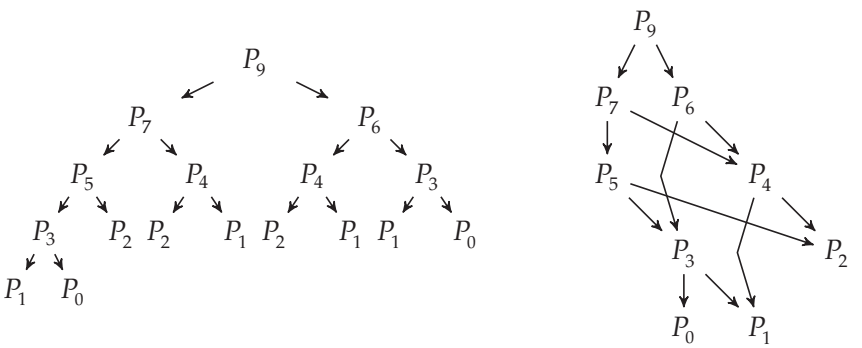


Рис. 19.6. Вычислить n-й член *последовательности Падована*, $P_n = P_{n-2} + P_{n-3}$, с $P_0 = P_1 = P_2 = 1$, можно, вычислив все подвыражения (*слева*). Более эффективный подход состоит в том, чтобы вычислять подвыражения один раз и повторно использовать их значения в последующих вычислениях, используя перекрывающуюся подструктуру задачи (*справа*)

Динамическое программирование может быть реализовано как сверху вниз, так и снизу вверх, как показано в алгоритме 19.6. Нисходящий подход начинается с желаемой задачи и рекурсивно сводится ко все меньшим и меньшим подзадачам. Решения подзадач хранятся таким образом, что, когда нам дается новая подзадача, мы можем либо получить вычисленное решение, либо решить и сохранить его для будущего использования.[9] Восходящий подход начинается с решения меньших подзадач и использует их решения для получения решений бо́льших задач.

Алгоритм 19.6. Вычисление последовательности Падована с использованием динамического программирования с подходами как сверху вниз, так и снизу вверх

```
function padovan_topdown(n, P = Dict())
   if !haskey(P, n)
      P[n] = n < 3 ? 1
          padovan_topdown(n - 2, P) + padovan_topdown(n - 3, P)
   end
```

[9] Хранение решений подзадач таким способом называется *запоминанием*.

```
    return P[n]
end
function padovan_bottomup(n)
    P = Dict(0 => 1,1 => 1,2 => 1)
    for i in 3 : n
        P[i] = P[i - 2] + P[i - 3]
    end
    return P[n]
end
```

Задача о рюкзаке — это хорошо известная задача комбинаторной оптимизации, которая часто возникает при распределении ресурсов.[10] Предположим, мы упаковываем походный рюкзак, но у нас мало места и мы хотим упаковать только наиболее ценные вещи. Существует несколько вариантов задачи о рюкзаке. Задача о рюкзаке $0-1$ представляет собой следующую задачу оптимизации:

$$\begin{aligned} &\min_{\mathbf{x}} -\sum_{i=1}^{n} v_i x_i \\ &\text{при условии, что } \sum_{i=1}^{n} w_i x_i \le w_{\max}, \\ &x_i \in \{0,\ 1\} \text{ для всех } i \in \{1,\ \ldots,\ n\}. \end{aligned} \tag{19.9}$$

У нас есть n предметов, i-й предмет имеет целочисленный вес $w_i > 0$ и значение v_i. Вектор $\mathbf{x}$ состоит из двоичных значений, которые указывают, упакован ли предмет. Общий вес не может превышать нашу общую емкость $w_{\max}$, и мы стремимся максимально увеличить общую стоимость упакованных предметов.

Существует 2^n возможных вариантов, что делает непосредственный перебор для больших n неразрешимой задачей. Тем не менее мы можем использовать динамическое программирование. Задача о рюкзаке $0-1$ имеет оптимальную подструктуру и перекрывающиеся подзадачи. Рассмотрим решение задачи с рюкзаком для n предметов и нескольких рюкзаков вплоть до $w_{\max}$. Решение более крупной задачи с рюкзаком с одним дополнительным предметом весом w_{n+1} и вместимостью $w_{\max}$ будет включать или не включать новый предмет.

- Если не стоит включать новый предмет, то значение целевой функции будет совпадать со значением целевой функции для решения задачи с $n-1$ предметом и вместимостью $w_{\max}$.

[10] Задача о рюкзаке — это задача целочисленного программирования с одним ограничением, но ее можно эффективно решить с помощью динамического программирования.

- Если стоит добавить новый предмет, то значение целевой функции будет представлять собой сумму значения целевой функции для решения задачи о рюкзаке с $n-1$ предметом и вместимостью $w_{\max}-w_{n+1}$ и стоимости нового предмета.

Рекуррентное отношение имеет вид:

$$\text{knapsack}\left(i, w_{\max}\right)=\begin{cases}0,\\ \text{knapsack}\left(i-1, w_{\max}\right),\\ \max\begin{cases}\text{knapsack}\left(i-1, w_{\max}\right), & \text{(не кладем новый предмет)}\\ \text{knapsack}\left(i-1, w_{\max}-w_i\right)+v_i. & \text{(кладем новый предмет)}\end{cases}\end{cases} \tag{19.10}$$

Задачу о рюкзаке $0-1$ можно решить с помощью алгоритма 19.7.

Алгоритм 19.7. Метод решения задачи о рюкзаке $0-1$ со стоимостью предметов **v**, целыми весами предметов **w** и целочисленной емкостью **w_max**. Восстановление вектора из сохраненных решений требует дополнительной итерации

```
function knapsack(v, w, w_max)
   n = length(v)
   y = Dict((0, j) => 0.0 for j in 0 : w_max
   for i in 1 : n
       for j in 0 : w_max
           y[i, j] = w[i] > j ? y[i - 1,j] :
                     max(y[i - 1, j],
                         y[i - 1, j - w[i]] + v[i])
       end
   end

   # восстанавливаем решение
   x, j = falses(n), w_max
   for i in n: -1 : 1
       if w[i] ≤ j && y[i, j] - y[i - 1, j - w[i]] == v[i]
          # i-й предмет в рюкзаке
          x[i] = true
          j -= w[i]
       end
   end
   return x
end
```

19.6. Муравьиный алгоритм

Муравьиный алгоритм [40] — это стохастический метод оптимизации маршрутов через графы. Этот метод был вдохновлен некоторыми видами муравьев, которые случайно бродят в поисках пищи, оставляя следы *феромонов* на ходу. Другие муравьи, которые натыкаются на след феромона, вероятно, начнут следовать за ним, таким образом усиливая аромат следа. Феромоны медленно испаряются с течением времени, в результате неиспользованные следы исчезают. Короткие тропы с более сильными феромонами муравьи проходят чаще и в результате эти тропы привлекают еще больше муравьев. Таким образом, короткие пути создают положительную обратную связь, которая заставляет других муравьев следовать по ним и еще более усиливать запах более короткого пути.

Основные задачи кратчайшего пути, такие как кратчайшие пути, найденные муравьями между муравейником и источниками пищи, могут быть эффективно решены с помощью динамического программирования. Муравьиный алгоритм использовался для поиска почти оптимальных решений *задачи о коммивояжере*, гораздо более сложной задачи, в которой мы хотим найти кратчайший путь, проходящий через каждый узел ровно один раз. Муравьиный алгоритм использовался для маршрутизации нескольких транспортных средств, поиска оптимальных мест для размещения фабрик и оптимального сворачивания белков.[11] Алгоритм является стохастическим по своей природе и, следовательно, устойчив к изменениям задачи во времени, таким как задержки движения, влияющие на эффективную длину ребер в графе, а также к проблемам, связанным с сетями без ребер.

Муравьи случайным образом двигаются в зависимости от привлекательности доступных им ребер. Привлекательность перехода зависит от уровня феромона и необязательного априорного фактора:

$$A(i \to j) = \tau(i \to j)^{\alpha} \eta(i \to j)^{\beta}, \tag{19.11}$$

где α и β также являются показателями степени, задающими уровень феромонов τ и априорный фактор η соответственно.[12] Для задач, связанных с поиском кратчайших путей, мы можем установить априорный фактор равным величине, обратной к длине ребра $l(i \to j)$, чтобы стимулировать обход более коротких путей: $\eta(i \to j) = 1/l(i \to j)$. Метод вычисления привлекательности ребер приведен в алгоритме 19.8.

---

[11] Эти и другие приложения обсуждаются в работах [99, 141, 151].

[12] В работе [40] рекомендуются значения $\alpha = 1$и $\beta = 5$.

Алгоритм 19.8. Метод вычисления таблицы привлекательности ребер для заданного графа `graph`, уровней феромонов `τ`, априорных весов ребер `η`, показателя феромонов `α` и априорного показателя `β`

```
function edge_attractiveness(graph, τ, η; α = 1,β = 5)
    A = Dict()
    for i in 1 : nv(graph)
        neighbors = outneighbors(graph, i)
        for j in neighbors
            v = τ[(i, j)] ^ α * η[(i, j)] ^ β
            A[(i, j)] = v
        end
    end
    return A
end
```

Предположим, что муравей находится в узле i и может перейти в любой из узлов $j \in \mathcal{J}$. Набор узлов-преемников $\mathcal{J}$ содержит всех допустимых исходящих соседей.[13] Иногда ребра исключаются, например в задаче о коммивояжере, когда муравьям не разрешается посещать один и тот же узел дважды. Следовательно, множество $\mathcal{J}$ зависит как от i, так и от истории блужданий муравья.

Вероятность граничного перехода $i \to j$ равна

$$P(i \to j) = \frac{A(i \to j)}{\sum_{j' \in \mathcal{J}} A(i \to j')}. \tag{19.12}$$

Муравьи влияют на последующие поколения муравьев, оставляя феромоны. Существует несколько методов моделирования феромонов. Обычный подход заключается в размещении феромонов после завершения полного пути [41]. Муравьи, которые не находят пути, не оставляют феромоны. Для задач о кратчайшем пути, успешный муравей, который прошел путь длиной l, оставляет феромоны с интенсивностью $1/l$ на каждом ребре пройденного пути.

Муравьиный алгоритм также моделирует испарение феромонов, которое естественным образом происходит в реальном мире. Моделирование испарения помогает предотвратить преждевременную сходимость алгоритма к одному, потенциально субоптимальному решению. В конце каждой итерации после завер-

[13] *Исходящими соседями* узла i являются все такие узлы j, что путь $i \to j$ находится на графе. В неориентированном графе соседи и исходящие соседи идентичны.

шения всех действий муравья происходит испарение феромонов, снижающее уровень феромонов каждого перехода с коэффициентом $1-\rho$, где $\rho \in [0, 1]$.[14]

Для муравьев на итерации k эффективное обновление феромона осуществляется по формуле

$$\tau(i \to j)^{(k+1)} = (1-\rho)\tau(i \to j)^{(k)} + \sum_{\alpha=1}^{m} \frac{1}{l^{(\alpha)}}\left((i \to j) \in \mathcal{P}^{(\alpha)}\right), \qquad (19.13)$$

где $l^{(\alpha)}$ — длина пути, а $\mathcal{P}^{(\alpha)}$ — множество ребер, пройденных муравьем α.

Муравьиный алгоритм реализован в алгоритме 19.10, при этом моделирование отдельных муравьев осуществляется с помощью алгоритма 19.9. На рис. 19.7 представлен муравьиный алгоритм для решения задачи о коммивояжере.

Алгоритм 19.9. Метод для моделирования одного муравья в задаче коммивояжера, при котором муравей запускается с первого узла и пытается посетить каждый узел ровно один раз. Уровни феромонов повышаются в конце успешного тура. Параметры — это граф графика, длина ребер, привлекательность уровней феромона **A**, лучшее решение, найденное до сих пор **x_best**, и его значение **y_best**

```
import StatsBase: Weights, sample
function run_ant(graph, lengths, τ, A, x_best, y_best)
    x = [1]
    while length(x) < nv(graph)
        i = x[end]
        neighbors = setdiff(outneighbors(graph, i), x)
        if isempty(neighbors) # муравей зашел в тупик
           return (x_best, y_best)
        end

        as = [A[(i, j)] for j in neighbors]
        push!(x, neighbors[sample(Weights(as))])
    end

    l = sum(lengths[(x[i - 1], x[i])] for i in 2 : length(x))
    for i in 2 : length(x)
        τ[(x[i - 1], x[i])] += 1/l
    end
    if l < y_best
        return (x, l)
```

[14] Обычно используется значение ρ=1/2.

```
    else
        return (x_best, y_best)
    end
end
```

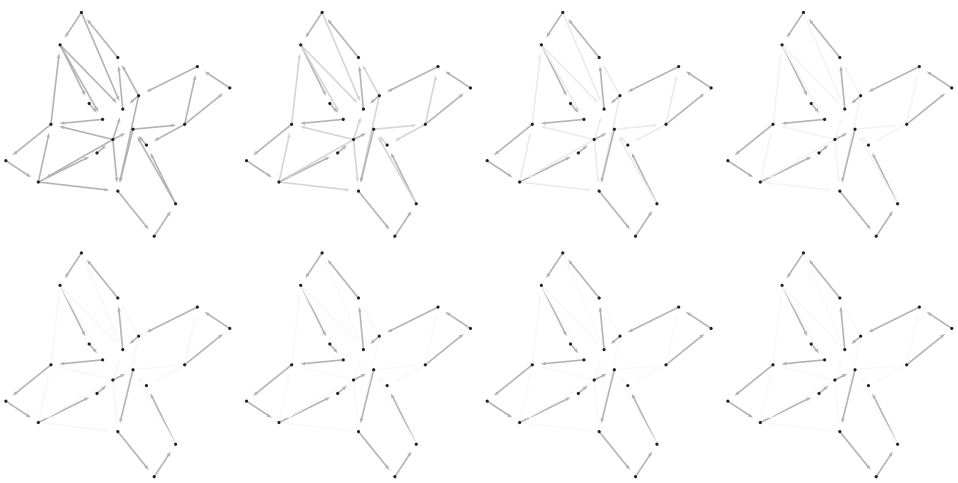


Рис. 19.7. Применение муравьиного алгоритма для решения задачи коммивояжера на ориентированном графе с использованием 50 муравьев на итерацию. Длина пути — это евклидовы расстояния. Непрозрачность цвета соответствует уровню феромона

Алгоритм 19.10. Муравьиный алгоритм, получающий ориентированный или неориентированный граф **graph** из пакета **LightGraphs.jl** и словарь кортежей ребер с длинами путей **lengths**. Муравьи начинают путь с первого узла на графе. Необязательные параметры включают количество муравьев на итерацию **m**, количество итераций **k_max**, показатель феромона **α**, априорный показатель **β**, скалярный показатель испарения **ρ** и словарь априорных весов ребер **η**

```
function ant_colony_optimization(graph, lengths;
   m = 1000, k_max = 100, α = 1.0, β = 5.0, ρ = 0.5,
   η = Dict((e.src, e.dst) => 1 / lengths[(e.src, e.dst)]
            for e in edges(graph)))
   τ = Dict((e.src, e.dst) => 1.0 for e in edges(graph))
   x_best, y_best = [], Inf
   for k in 1 : k_max
       A = edge_attractiveness(graph, τ, η, α = α, β = β)
       for (e, v) in τ
           τ[e] = (1 - ρ) * v
       end
```

```
        for ant in 1 : m
            x_best, y_best = run_ant(graph, lengths, τ, A, x_best, y_best)
        end
    end
    return x_best
end
```

19.7. Резюме

- Дискретные задачи оптимизации требуют, чтобы проектные переменные выбирались из дискретных множеств.
- Релаксация, с помощью которой дискретная задача сводится к непрерывной, сама по себе является ненадежным методом поиска оптимального дискретного решения, но служит основой для более сложных алгоритмов.
- Многие задачи комбинаторной оптимизации можно представить в виде задачи целочисленного программирования, которая представляет собой задачу линейного программирования с целочисленными ограничениями.
- Для эффективного и точного решения задач целочисленного программирования могут использоваться как метод секущих плоскостей, так и методы ветвей и границ. Метод ветвей и границ является довольно универсальным и может применяться к широкому спектру задач дискретной оптимизации.
- Динамическое программирование — это мощный метод, который использует оптимальную перекрывающуюся подструктуру в некоторых задачах.
- Муравьиный алгоритм — это алгоритм, навеянный наблюдениями за природой, который можно использовать для оптимизации путей в графах.

19.8. Упражнения

Упражнение 19.1. *Задача о выполнимости булевых формул* (Boolean satisfiability problem — SAT) заключается в следующем: существует ли логическая схема, которая заставляет булевозначную целевую функцию выводить значение `true`. Задачи SAT были первыми, для которых было доказано, что они принадлежат к классу сложности NP-полных задач [28]. Это означает, что задача SAT, по крайней мере, так же сложна, как и все другие задачи, решения которых могут быть проверены за полиномиальное время.

Рассмотрим булеву целевую функцию:

$$f(\mathbf{x}) = x_1 \wedge (x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2)$$

Найдите оптимальное решение, используя перебор. Сколько вариантов необходимо перебрать для n-мерного вектора в худшем случае?

Упражнение 19.2. Сформулируйте задачу из упражнения 19.1 как задачу целочисленного линейного программирования. Можно ли сформулировать задачу булевой выполнимости в виде задачи целочисленного линейного программирования?

Упражнение 19.3. Почему нас интересуют вполне унимодулярные матрицы? Кроме того, почему каждая вполне унимодулярная матрица содержит только те элементы, которые равны 0, 1 или –1?

Упражнение 19.4. В этой главе описано решение задачи о рюкзаке 0–1 с помощью динамического программирования. Покажите, как применить метод ветвей и границ к задаче о рюкзаке 0–1, и предложите свой подход для решения задачи о рюкзаке со значениями $\mathbf{v}$ = [9, 4, 2, 3, 5, 3] и весами $\mathbf{w}$ = [7, 8, 4, 5, 9, 4] с емкостью $w_{\max} = 20$.

Глава 20

Оптимизация выражений

В предыдущих главах обсуждалась оптимизация по фиксированному набору проектных переменных. Для многих задач число переменных неизвестно, например при оптимизации графических структур или компьютерных программ. Проекты в этих контекстах могут быть представлены выражениями, которые принадлежат грамматике. В этой главе обсуждаются способы повышения эффективности поиска оптимальных конструкций с учетом грамматической структуры пространства проектирования.

20.1. Грамматика

Выражение может быть представлено деревом *символов*. Например, математическое выражение $x + \ln 2$ может быть представлено с помощью дерева, показанного на рис. 20.1, состоящего из символов +, x, ln и 2. *Грамматики* задают ограничения на пространстве возможных выражений.

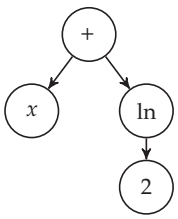


Рис. 20.1. Выражение $x + \ln 2$ представляется в виде дерева

Грамматика представляется набором *правил вывода*. Эти правила включают символы, а также *типы*. Тип можно интерпретировать как набор деревьев выражений. Правило вывода представляет собой возможное расширение типа до выражения, включающего символы или типы. Если правило распространяется только на символы, то оно называется *терминальным*, поскольку его нельзя расширить далее. Примером нетерминального правила является $\mathbb{R} \mapsto \mathbb{R} + \mathbb{R}$, это означает, что тип $\mathbb{R}$ может состоять из элементов набора $\mathbb{R}$, добавленных к элементам в наборе $\mathbb{R}$.[1]

Можно сгенерировать выражение из грамматики, используя *начальный тип*, а затем рекурсивно применяя различные правила вывода. Мы останавливаемся,

[1] В этой главе основное внимание уделяется контекстно-свободным грамматикам, но существуют и другие формы (см. [77]).

когда дерево содержит только символы. На рис. 20.2 иллюстрируется этот процесс для выражения $x + \ln 2$. Применение этого процесса к выражениям на естественном языке показано в примере 20.1.

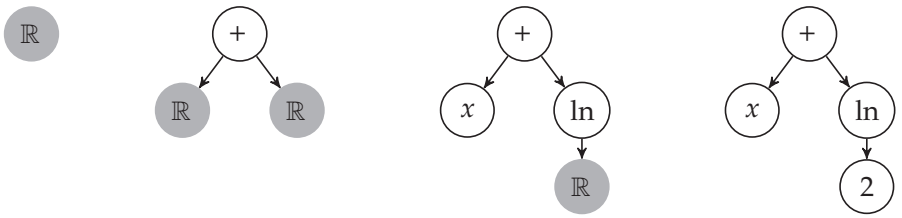


Рис. 20.2. Использование правил вывода $\mathbb{R} \mapsto \mathbb{R} + \mathbb{R}$, $\mathbb{R} \mapsto x$, $\mathbb{R} \mapsto \ln(\mathbb{R})$, $\mathbb{R} \mapsto 2$ для генерации выражения $x + \ln 2$. Синие узлы — это нерасширенные типы

Пример 20.1. Грамматика для вывода простых английских высказываний. Использование символа | в правой части выражения является обозначением операции *или*. Таким образом, правило $\mathbb{A} \mapsto$ **быстро | эффективно эквивалентно наличию двух правил: $\mathbb{A} \mapsto$ быстро и $\mathbb{A} \mapsto$ эффективно**

Рассмотрим грамматику, которая позволяет генерировать простые выражения:

$$\mathbb{S} \mapsto \mathbb{N}\ \mathbb{V}$$

$$\mathbb{V} \mapsto \mathbb{V}\ \mathbb{A}$$

$$\mathbb{A} \mapsto \text{быстро} \mid \text{эффективно}$$

$$\mathbb{N} \mapsto \text{Алиса} \mid \text{Боб} \mid \text{Майкл} \mid \text{Тим}$$

$$\mathbb{V} \mapsto \text{бежит} \mid \text{читает} \mid \text{пишет}$$

Символы $\mathbb{S}$, $\mathbb{N}$, $\mathbb{V}$ и $\mathbb{A}$ соответствуют предложениям, существительным, глаголам и наречиям соответственно. Выражение генерируется, начиная с типа $\mathbb{S}$, с помощью итеративной подстановки типов:

$\mathbb{S}$

$\mathbb{N}\ \mathbb{V}$

Майкл $\mathbb{V}\ \mathbb{A}$

Майкл пишет быстро

Не все категории терминальных правил должны быть использованы. Например, в данном примере можно было также сгенерировать предложение “Алиса бежит”.

Количество возможных выражений, допускаемых грамматикой, может быть бесконечным. Пример 20.2 демонстрирует грамматику, которая допускает бесконечное количество допустимых выражений.

Пример 20.2. Некоторые проблемы, связанные с грамматикой, на примере грамматики для калькулятора с четырьмя арифметическими функциями

Рассмотрим грамматику калькулятора с четырьмя функциями, которая применяет сложение, вычитание, умножение и деление к десяти цифрам:

$$\mathbb{R} \mapsto \mathbb{R} + \mathbb{R}$$
$$\mathbb{R} \mapsto \mathbb{R} - \mathbb{R}$$
$$\mathbb{R} \mapsto \mathbb{R} \times \mathbb{R}$$
$$\mathbb{R} \mapsto \mathbb{R} / \mathbb{R}$$
$$\mathbb{R} \mapsto 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

Можно генерировать бесконечное количество выражений, потому что нетерминальный тип $\mathbb{R}$ всегда можно развернуть в одну из операций калькулятора.

Многие выражения приведут к одному и тому же значению. Операторы сложения и умножения являются коммутативными, т.е. порядок не имеет значения. Например, $a + b$ равно $b + a$. Эти операции также являются ассоциативными, т.е. порядок, в котором выполняются несколько операций одного типа, не имеет значения. Например, $a \times b \times c$ равно $c \times b \times a$. Другие операции сохраняют значения при добавлении нуля или умножении на единицу.

Не все выражения в этой грамматике математически допустимы. Например, деление на нуль не определено. Удаление нуля как терминального символа недостаточно для предотвращения этой ошибки, потому что нуль может быть создан с помощью других операций, таких как 1 – 1. Такие исключения часто обрабатываются целевой функцией, которая может перехватывать исключения и штрафовать их.

Оптимизация выражений часто ограничивает выражения максимальной глубиной или штрафует выражения на основе их глубины или количества узлов. Даже если грамматика допускает конечное число выражений, пространство часто слишком велико для исчерпывающего поиска. Следовательно, существует потребность в алгоритмах, которые эффективно ищут пространство возможных выражений для алгоритма, оптимизирующего целевую функцию.

Процедуры оптимизации выражений, описанные в этой главе, используют пакет **`ExprRules.jl`**. Грамматики можно определить с помощью макроса **`grammar`**, перечислив правила вывода, как показано в примере 20.3.

Пример 20.3. Пример определения грамматики с помощью пакета **ExprRules.jl**

Мы можем определить грамматику, используя макрос **grammar**. Нетерминальные символы находятся слева от знака равенства, а выражения с терминальными и нетерминальными символами — справа. Пакет включает некоторый синтаксис для более компактного представления грамматики.

```
using ExprRules
grammar = @grammar begin
    R = x              # ссылка на переменную
    R = R * A          # несколько дочерних элементов
    R = f(R)           # вызов функции
    R = _(randn())     # случайная переменная, сгенерированная
                       # при создании узла
    R = 1 | 2 | 3      # эквивалент R = 1, R = 2 и R = 3
    R = |(4 : 6)       # эквивалент R = 4, R = 5 и R = 6
    A = 7              # правила для разных возвращаемых типов
end;
```

Многие из алгоритмов оптимизации выражений предполагают манипулирование компонентами дерева выражений таким образом, чтобы сохранить способ расширения типов. Объект типа **RuleNode** отслеживает, какие правила вывода были применены при выполнении расширения. Вызов функции **rand** с указанным начальным типом сгенерирует случайное выражение, представленное объектом типа **RuleNode**. Вызов функции **sample** приведет к выбору случайного объекта типа **RuleNode** из существующего дерева типа **RuleNode**. Узлы вычисляются с помощью функции **eval**.

Метод **return_type** возвращает признак типа возвращаемого значения, метод **isterminal** проверяет, является ли символ терминальным, функция **child_types** возвращает список нетерминальных символов, связанных с правилом вывода в узле, а функция **nchildren** возвращает количество дочерних элементов. Каждый из этих четырех методов принимает в качестве входных данных грамматику и узел. Количество узлов в дереве выражений получается с помощью вызова `length(node)`, а глубина — с помощью вызова `depth(node)`.

Третий тип, **NodeLoc**, используется для ссылки на местоположение узла в дереве выражений. Манипулирование поддеревьями часто связано с типом **NodeLocs**.

```
loc = sample (NodeLoc, node);              # равномерная выборка узлов loc
loc = sample (NodeLoc, node, :R, grammar); # выбор узла loc типа R
subtree = get(node, loc);
```

20.2. Генетическое программирование

Генетические алгоритмы (см. главу 9) используют хромосомы, которые кодируют расчетные точки в последовательном формате. *Генетическое программирование* [89] вместо особей использует деревья (рис. 20.3), которые лучше представляют математические функции, программы, решающие деревья и другие иерархические структуры.

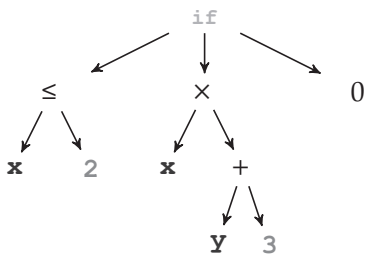


Рис. 20.3. Представление дерева с помощью метода на языке Julia `x ≤ 2 ? x * (y + 3) : 0`

Подобно генетическим алгоритмам, генетические программы инициализируются случайным образом и поддерживают кроссовер и мутацию. При *кроссовере деревьев* (рис. 20.4) два родительских дерева смешиваются, образуя дочернее дерево. В каждом родительском узле выбирается случайный узел, и поддерево в таком узле в первом родительском дереве заменяется поддеревом в выбранном узле второго. Кроссовер деревьев применяется к родительским деревьям с различными размерами и формами, позволяя смешивать произвольные деревья. В некоторых случаях необходимо убедиться, что замещающие узлы имеют определенные типы, такие как логические значения, вводимые в условие оператора `if`.[2] Кроссовер дерева реализован в алгоритме 20.1.

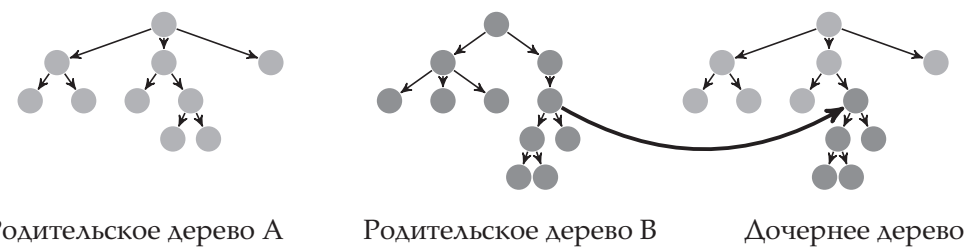


Рис. 20.4. Кроссовер деревьев используется для объединения двух родительских деревьев и создания дочернего дерева

[2] Эта книга посвящена только генетическим операциям, которые придерживаются ограничений грамматики. Иногда генетическое программирование с этим ограничением называют *строго типизированным генетическим программированием* [105].

Алгоритм 20.1. Кроссовер дерева реализован для объектов **a** и **b** типа **RuleNode** из пакета **ExprRules.jl**. Структура **TreeCrossover** содержит грамматику набора правил и максимальную глубину **max_depth**

```
struct TreeCrossover <: CrossoverMethod
    grammar
    max_depth
end
function crossover(C::TreeCrossover, a, b)
    child = deepcopy(a)
    crosspoint = sample(b)
    typ = return_type(C.grammar, crosspoint.ind)
    d_subtree = depth(crosspoint)
    d_max = C.max_depth + 1 - d_subtree
    if d_max > 0 && contains_returntype(child, C.grammar, typ, d_max)
        loc = sample(NodeLoc, child, typ, C.grammar, d_max)
        insert!(child, loc, deepcopy(crosspoint))
    end
    child
end
```

Кроссовер деревьев имеет тенденцию производить деревья с большей глубиной, чем родительские деревья. Каждое поколение имеет тенденцию к увеличению сложности, что часто приводит к чрезмерно сложным решениям и замедлению времени выполнения. Мы поощряем *лаконичность*, или простоту, решения путем введения небольшого смещения в значение целевой функции на основе глубины дерева или числа узлов.

Применение *мутации дерева* (рис. 20.5) начинается с выбора случайного узла в дереве. Поддерево с корнем в этом узле удаляется, и генерируется новое случайное поддерево для замены старого. В отличие от мутации в бинарных хромосомах, древесная мутация обычно может происходить не более одного раза, часто с низкой вероятностью около 1%. Мутация деревьев реализована в алгоритме 20.2.

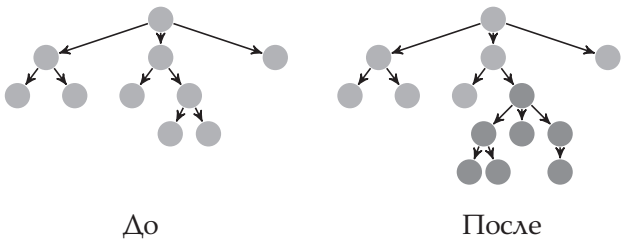


Рис. 20.5. Мутация дерева удаляет случайное поддерево и генерирует новое, чтобы заменить его

Алгоритм 20.2. Мутация дерева реализована для отдельной особи типа `RuleNode` из пакета `ExprRules.jl`. Структура `TreeMutation` содержит набор правил `grammar` и вероятность мутации `p`

```
struct TreeMutation <: MutationMethod
   grammar
   p
end
function mutate(M::TreeMutation, a)
   child = deepcopy(a)
   if rand() < M.p
      loc = sample(NodeLoc, child)
      typ = return_type(M.grammar, get(child, loc).ind)
      subtree = rand(RuleNode, M.grammar, typ)
      insert!(child, loc, subtree)
   end
   return child
end
```

Перестановка деревьев (рис. 20.6) является второй формой генетической мутации. Здесь дочерние элементы случайно выбранного узла переставляются случайным образом. Одной перестановки деревьев, как правило, недостаточно для внедрения нового генетического материала в популяцию, и она часто сочетается с мутацией деревьев. Перестановка деревьев реализована в алгоритме 20.3.

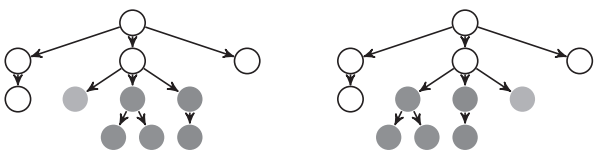


Рис. 20.6. Перестановка деревьев переставляет дочерние элементы случайно выбранного узла

Алгоритм 20.3. Перестановка деревьев реализована для отдельной особи типа `RuleNode` из пакета `ExprRules.jl`, где `p` — вероятность мутации

```
struct TreePermutation <: MutationMethod
   grammar
   p
end
```

```
function mutate(M::TreePermutation, a)
    child = deepcopy(a)
    if rand() < M.p
        node = sample(child)
        n = length(node.children)
        types = child_types(M.grammar, node)
        for i in 1 : n - 1
            c = 1
            for k in i + 1 : n
                if types[k] == types[i] &&
                    rand() < 1 / (c += 1)
                    node.children[i], node.children[k] =
                        node.children[k], node.children[i]
                end
            end
        end
    end
    return child
end
```

В остальном реализация генетического программирования идентична реализации генетических алгоритмов. Следует проявлять больше внимания при реализации подпрограмм кроссовера и мутации, особенно при определении того, какие типы узлов могут быть сгенерированы, и утверждении, что создаются только синтаксически правильные деревья. Генетическое программирование используется для генерации приближенного выражения в примере 20.4.

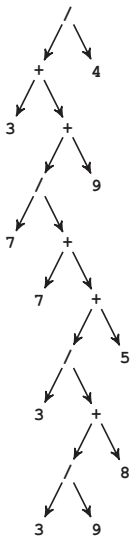
Пример 20.4. Использование генетического программирования для вычисления числа π с помощью только цифр и четырех основных арифметических действий.

Рассмотрим аппроксимацию числа π с помощью операций на калькуляторе с четырьмя функциями. Мы можем решить эту задачу, используя генетическое программирование, где узлами могут быть любые элементарные операции: сложение, вычитание, умножение и деление, а также цифры 1 ... 9.

Используем пакет **`ExprRules.jl`**, чтобы задать грамматику:

```
grammar = @grammar begin
    R = | (1 : 9)
    R = R + R
    R = R - R
```

```
    R = R / R
    R = R * R
end
```



Мы строим целевую функцию и штрафуем большие деревья:

```
function f(node)
    value = eval(node, grammar)
  if isinf(value) || isnan(value)
    return Inf
  end
  Δ = abs(value - π)
  return log(Δ) + length(node) / 1e3
end
```

Наконец, запускаем генетическую программу с помощью вызова функции **genetic_algorithm** из раздела 9.2:

```
srand(0)
population = [rand(RuleNode, grammar, :R) for i in 1 : 1000]
best_tree = genetic_algorithm(f, population, 30,
                TruncationSelection(50),
                TreeCrossover(grammar, 10),
                TreeMutation(grammar, 0.25))
```

Лучшее дерево показано справа. Оно вычисляет число 3,141586, что соответствует числу π до четырех знаков после запятой.

20.3. Грамматическая эволюция

Грамматическая эволюция [135] работает с целочисленным массивом вместо дерева, что позволяет применять те же методы, что и в генетических алгоритмах. В отличие от генетических алгоритмов, хромосомы в грамматической эволюции кодируют выражения, основанные на грамматике. Грамматическая эволюция была вдохновлена генетическим материалом, который по своей природе является последовательным, как хромосомы, используемые в генетических алгоритмах.[3] В грамматической эволюции дизайны представляют собой целочисленные масси-

[3] Наша последовательная ДНК считывается и используется для создания сложных белковых структур. ДНК часто называют генотипом — объектом, на котором выполняются генетические операции. Структура белка — это фенотип, т.е. объект, закодированный генотипом, эффективность которого оценивается. Литература по грамматической эволюции часто упоминает целочисленный вектор как генотип, а полученное выражение — как фенотип.

вы, очень похожие на хромосомы, используемые в генетических алгоритмах. Каждое целое число не ограничено, потому что индексация выполняется с помощью модульной арифметики. Целочисленный массив можно преобразовать в дерево выражений, анализируя его слева направо.

Мы начинаем с начального символа и грамматики. Предположим, что к начальному символу можно применить n правил грамматики. Применяется j-е правило, где $j = i \bmod_1 n$, а i — первое целое число в массиве целых чисел.[4]

Затем мы рассмотрим правила, применимые к полученному выражению, и используем подобную модульную арифметику на основе второго целого числа в массиве, чтобы выбрать, какое правило применить. Этот процесс повторяется до тех пор, пока никакие правила не могут быть применены и фенотип не будет завершен.[5] Процесс декодирования реализован в алгоритме 20.4 и проиллюстрирован в примере 20.5.

Алгоритм 20.4. Метод декодирования целочисленного вектора для создания выражения, где **`x`** — вектор целых чисел, **`grammar`** — грамматика, а **`sym`** — корневой символ. Счетчик **`c`** используется во время процесса рекурсии, а параметр **`c_max`** является верхним пределом для максимального числа применений правил, чтобы предотвратить бесконечный цикл. Метод возвращает объект типа **`DecodedExpression`**, который содержит дерево выражений и количество правил, примененных во время процесса декодирования

```
struct DecodedExpression
    node
    n_rules_applied
end
function decode(x, grammar, sym, c_max = 1000, c = 0)
    node, c = _decode(x, grammar, sym, c_max, c)
    DecodedExpression(node, c)
end
function _decode(x, grammar, typ, c_max, c)
    types = grammar[typ]
    if length(types) > 1
        g = x[mod1(c += 1, length(x))]
        rule = types[mod1(g, length(types))]
    else
```

[4] Мы используем выражение $x \bmod_1 n$ для ссылки на операцию $((x - 1) \bmod n) + 1$. Этот тип модуля полезен при индексировании с основанием единица. В языке Julia этому соответствует функция $\bmod_1$.

[5] В примере 20.5 никакая генетическая информация не читается и не обрабатывается, если есть только одно применимое правило.

```
        rule = types[1]
    end
    node = RuleNode(rule)
    childtypes = child_types(grammar, node)
    if !isempty(childtypes) && c < c_max
        for ctyp in childtypes
            cnode, c = _decode(x, grammar, ctyp, c_max, c)
            push!(node.children, cnode)
        end
    end
    return (node, c)
end
```

Пример 20.5. Процесс, посредством которого целочисленный вектор в грамматической эволюции декодируется в выражение. Используем реализацию "сначала в глубину". Если бы вместо значения `51` стояло значение `52`, было бы применено правило $\mathbb{D}' \mapsto \mathbb{D}\mathbb{D}'$ с последующим выбором правила для нового $\mathbb{D}$, что в итоге приведет к числу *43950,950E+8*

Рассмотрим грамматику для строк, состоящих из действительных чисел:

$$\mathbb{R} \mapsto \mathbb{D}\mathbb{D}'\mathbb{P}\mathbb{E}$$
$$\mathbb{D}' \mapsto \mathbb{D}\mathbb{D}'|\varepsilon$$
$$\mathbb{P} \mapsto .\mathbb{D}\mathbb{D}'|\varepsilon$$
$$\mathbb{E} \mapsto \mathbb{E}\mathbb{S}\mathbb{D}\mathbb{D}'|\varepsilon$$
$$\mathbb{S} \mapsto + \mid - \mid \varepsilon$$
$$\mathbb{D} \mapsto 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9,$$

где $\mathbb{R}$ — действительное значение, $\mathbb{D}$ — терминальное десятичное число, $\mathbb{D}'$ — нетерминальное десятичное число, $\mathbb{P}$ — десятичная часть, $\mathbb{E}$ — показатель степени, а $\mathbb{S}$ — знак. Любые значения ε создают пустые строки.

Предположим, что наш вектор [205, 52, 4, 27, 10, 59, 6] и у нас есть начальный символ $\mathbb{R}$. Существует только одно применимое правило, поэтому мы не используем генетическую информацию и меняем $\mathbb{R}$ на $\mathbb{D}\mathbb{D}'\mathbb{P}\mathbb{E}$.

Далее мы должны заменить $\mathbb{D}$. Есть 10 вариантов. Мы выбираем $205 \bmod_1 10 = 5$ и, следовательно, получаем $4\mathbb{D}'\mathbb{P}\mathbb{E}$.

Для замены $\mathbb{D}'$ существует два варианта. Выбираем индекс $52 \bmod_1 2 = 2$, что соответствует ε.

Продолжая таким образом, получаем строку *4E+8*.

Вышеупомянутая грамматика может быть реализована в структуре **ExprRules**:

```
grammar = @grammar begin
  R = D * De * P * E
  De = D * De | ""
  P = "." * D * De | ""
  E = "E" * S * D * De | ""
  S = "+"|"-"|""
  D = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"
end
```

и привести к результату

```
x = [205, 52, 4, 27, 10, 59, 6]
str = eval(decode(x, grammar, :R).node, grammar)
```

Возможно, целочисленный массив будет слишком коротким, что приведет к тому, что процесс трансляции будет превышать длину массива. Вместо того чтобы создавать недопустимую особь и штрафовать ее в целевой функции, процесс переходит к началу массива. Этот эффект перехода означает, что в процессе транскрипции одно и то же решение может быть прочитано несколько раз. Транскрипция может привести к бесконечным циклам, которые могут быть предотвращены максимальной глубиной.

Генетические операции работают непосредственно с целочисленным массивом. Мы можем перенести операции, используемые на действительные хромосомы, и применить их к целочисленным хромосомам. Единственное изменение заключается в том, что мутация должна сохранять действительные числа. Метод мутации для целочисленных хромосом, использующий возмущения Гаусса с нулевым средним, реализован в алгоритме 20.5.

Алгоритм 20.5. Метод гауссовской мутации, модифицированный для сохранения целочисленных значений, представляющих целочисленные хромосомы. Каждое значение возмущается случайной величиной с нормальным распределением, имеющим нулевое математическое ожидание и стандартное отклонение σ, и затем округляется до ближайшего целого числа

```
struct IntegerGaussianMutation <: MutationMethod
   σ
end
function mutate(M::IntegerGaussianMutation, child)
   return child + round.(Int, randn(length(child)) .* M.σ)
end
```

Грамматическая эволюция использует два дополнительных генетических оператора. Первый, *дупликация генов*, возникает естественным образом как ошибка в репликации и репарации ДНК. Дупликация генов может позволить генерировать новый генетический материал и хранить вторую копию полезного гена, чтобы уменьшить вероятность летальной мутации, удаляющей ген из генофонда. Дупликация генов выбирает случайный интервал генов в хромосоме для дублирования. Копия выбранного интервала добавляется к задней части хромосомы. Дублирование реализовано в алгоритме 20.6.

Алгоритм 20.6. Метод дубликации генов, используемый в грамматической эволюции

```
struct GeneDuplication <: MutationMethod
end
function mutate(M::GeneDuplication, child)
    n = length(child)
    i, j = rand(1 : n), rand(1 : n)
    interval = min(i, j) : max(i, j)
    return vcat(child, deepcopy(child[interval]))
end
```

Вторая генетическая операция — *отсечение* — решает проблему, возникшую во время кроссовера. Как показано на рис. 20.7, операция кроссовера случайным образом выбирает точку в каждой хромосоме и строит новую хромосому, используя левую часть первой и правую часть второй хромосомы. В отличие от генетических алгоритмов, последние записи в хромосомах грамматической эволюции могут не использоваться; во время синтаксического анализа, когда дерево завершено, оставшиеся записи игнорируются. Чем больше неиспользованных записей, тем больше вероятность того, что точка кроссовера находится в неактивной области, что не дает нового полезного материала. Особь отсекается с определенной вероятностью, и, если она отсекается, ее хромосома обрезается так, что сохраняются только активные гены. Отсечение реализовано в алгоритме 20.7 и показано на рис. 20.8.

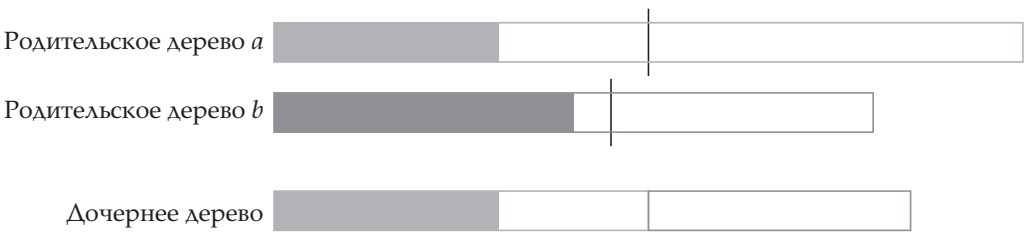


Рис. 20.7. Кроссовер, примененный к хромосомам в грамматической эволюции, может не влиять на активные гены в начальной части хромосомы

Показанный здесь потомок наследует все активные затененные гены от родительского элемента *a*, поэтому он будет эффективно действовать как идентичное выражение. Операция отсечения была разработана, чтобы преодолеть эту проблему.

Алгоритм 20.7. Метод отсечения генов, используемый в грамматической эволюции

```
struct GenePruning <: MutationMethod
   p
   grammar
   typ
end
function mutate(M::GenePruning, child)
   if rand() < M.p
      c = decode(child, M.grammar, M.typ).n_rules_applied
      if c < length(child)
         child = child[1 : c]
       end
    end
    return child
end
```

Рис. 20.8. Отсечение обрезает хромосому до и после, так что остаются только активные гены

Как и генетическое программирование, грамматическая эволюция может использовать метод генетического алгоритма.[6] Мы можем создать метод мутации, который применяет множественные мутации в тандеме. Такой метод реализован в алгоритме 20.8.

Алгоритм 20.8. Метод `MutationMethod` для применения всех методов мутации, хранящихся в векторе `Ms`

```
struct MultiMutate <: MutationMethod
   Ms
end
```

[6] Методы превращения генотипа в фенотип с целью использования методов отсечения, дубликации и стандартных мутаций могут присутствовать в целевой функции.

```
function mutate(M::MultiMutate, child)
    for m in M.Ms
      child = mutate(m, child)
    end
    return child
end
```

Грамматическая эволюция страдает от двух основных недостатков. Во-первых, трудно сказать, допустима ли хромосома, не расшифровав ее в выражение. Во-вторых, небольшое изменение в хромосоме может привести к значительному изменению в соответствующей экспрессии.

20.4. Вероятностные грамматики

Вероятностная грамматика [19] добавляет вес каждому правилу в грамматике генетической программы. Из всех применимых правил для данного узла мы выбираем правило случайным образом в соответствии с относительными весами. Вероятность выражения является произведением вероятностей выбора каждого правила. Алгоритм 20.9 реализует вычисление вероятности. Пример 20.6 демонстрирует выбор выражения из вероятностной грамматики и вычисляет его вероятность.

Алгоритм 20.9. Метод вычисления вероятности выражения на основе вероятностной грамматики, где **probgram** — это вероятностная грамматика, состоящая из грамматики **grammar** и отображения типов в веса для всех применимых правил **ws**, а **node** — выражение типа **RuleNode**

```
struct ProbabilisticGrammar
    grammar
    ws
end
function probability(probgram, node)
    typ = return_type(probgram.grammar, node)
    i = findfirst(isequal(node.ind), probgram.grammar[typ])
    prob = probgram.ws[typ][i] / sum(probgram.ws[typ])
    for (i, c) in enumerate(node.children)
         prob *= probability(probgram, c)
     end
     return prob
end
```

Пример 20.6. Выборка выражения из вероятностной грамматики и вычисление вероятности выражения

Рассмотрим вероятностную грамматику для строк, состоящих полностью из буквы “a”:

$$\begin{aligned}
\mathbb{A} &\mapsto \text{a}\ \mathbb{A} && w_1^{\mathbb{A}} = 1\\
&\mapsto \text{a}\ \mathbb{B}\ \text{a}\ \mathbb{A} && w_2^{\mathbb{A}} = 3\\
&\mapsto \varepsilon && w_3^{\mathbb{A}} = 2\\
\mathbb{B} &\mapsto \text{a}\ \mathbb{B} && w_1^{\mathbb{B}} = 4\\
&\mapsto \varepsilon && w_3^{\mathbb{B}} = 1,
\end{aligned}$$

где **w** — набор весов для каждого родительского типа, а ε — пустая строка.

Предположим, мы генерируем выражение, начинающееся с типа $\mathbb{A}$. Распределение вероятностей по трем возможным правилам имеет следующий вид:

$$P(\mathbb{A} \mapsto \text{a}\ \mathbb{A}) = 1 / (1 + 3 + 2) = 1/6$$
$$P(\mathbb{A} \mapsto \text{a}\ \mathbb{B}\ \text{a}\ \mathbb{A}) = 3 / (1 + 3 + 2) = 1/2$$
$$P(\mathbb{A} \mapsto \varepsilon) = 2 / (1 + 3 + 2) = 1/3$$

Предположим, мы отобрали второе правило и получили строку “a $\mathbb{B}$ a $\mathbb{A}$”.

Далее мы выбираем правило, которое нужно применить к $\mathbb{B}$. Распределение вероятностей по двум возможным правилам таково:

$$P(\mathbb{B} \mapsto \text{a}\ \mathbb{B}) = 4 / (4 + 1) = 4/5$$
$$P(\mathbb{B} \mapsto \varepsilon) = 1 / (4 + 1) = 1/5$$

Предположим, мы выбрали второе правило и получили строку “a ε a $\mathbb{A}$”.

Далее задействуем правило, которое нужно применить к $\mathbb{A}$. Предположим, мы пробуем $\mathbb{A} \mapsto \varepsilon$. чтобы получить строку “a ε a ε”, которая производит строку “aa”. Вероятность последовательности правил, применяемых для получения “aa” в рамках вероятностной грамматики, равна:

$$P\left(\mathbb{A} \mapsto \text{a}\ \mathbb{B}\ \text{a}\ \mathbb{A}\right)P\left(\mathbb{B} \mapsto \varepsilon\right)P\left(\mathbb{A} \mapsto \varepsilon\right) = \frac{1}{2}\frac{1}{5}\frac{1}{3} = \frac{1}{30}.$$

Обратите внимание: это не то же самое, что вероятность получения строки “aa”, так как другие последовательности правил вывода также могли бы ее создать.

Оптимизация с помощью вероятностного грамматического анализа улучшает вес с каждой итерацией с помощью извлечения элитных выборок из совокупности. На каждой итерации отбирается совокупность выражений и вычисляются значения их целевых функций. Некоторые из лучших выражений считаются элитными выборками и могут использоваться для обновления весов. Для вероятност-

ной грамматики генерируется новый набор весов, где вес $w_i^{\mathbb{T}}$ для i-го правила вывода, применимого к возвращаемому типу $\mathbb{T}$, устанавливается равным количеству применений правила вывода при создании элитных выборок. Эта процедура обновления реализована в алгоритме 20.10.

Алгоритм 20.10. Метод применения обучающего обновления к вероятностной грамматике **probgram** на основе элитной выборки выражений **Xs**

```
function _update!(probgram, x)
    grammar = probgram.grammar
    typ = return_type(grammar, x)
    i = findfirst(isequal(x.ind), grammar[typ])
    probgram.ws[typ][i] += 1
    for c in x.children
        _update!(probgram, c)
    end
    return probgram
end
function update!(probgram, Xs)
    for w in values(probgram.ws)
        fill!(w, 0)
    end
    for x in Xs
        _update!(probgram, x)
    end
    return probgram
end
```

Вышеприведенные вероятностные грамматики можно распространить на более сложные распределения вероятностей, учитывающие другие факторы, такие как глубина выражения или локальные зависимости между одноуровневыми узлами в поддеревьях. Один из подходов заключается в использовании байесовских сетей [161].

20.5. Вероятностные деревья прототипов

Вероятностные деревья прототипов [137] — это другой подход, который выводит распределение для каждого узла в дереве выражений. Каждый узел в вероятностном дереве прототипов содержит вектор вероятности, представляющий категориальное распределение по правилам вывода грамматики. Векторы вероятности обновляются, отражая знания, полученные от последовательных

поколений выражений. Максимальное количество дочерних узлов для каждого узла — это максимальное количество нетерминальных правил среди правил в грамматике.[7]

При создании узла векторы вероятности инициализируются случайным образом. Случайные векторы вероятности могут быть извлечены из распределения Дирихле.[8] Первоначальная реализация инициализирует терминальные правила скалярным значением 0,6, и нетерминальные — 0,4. Для обработки строго типизированных грамматик мы поддерживаем вектор вероятности для правил, применимых к каждому родительскому типу. Алгоритм 20.11 определяет тип узла и реализует этот метод инициализации.

Алгоритм 20.11. Вероятностный тип узла дерева прототипов и связанная с ним функция инициализации, где **ps** — это словарь, отображающий символ, соответствующий возвращаемому типу в вектор вероятности по применимым правилам, а **children** — список объектов типа **PPTNodes**. Метод **get_child** автоматически раскроет дерево при попытке доступа к несуществующему дочернему элементу

```
struct PPTNode
   ps
   children
end
function PPTNode(grammar;
   w_terminal = 0.6,
   w_nonterm = 1 - w_terminal,
)

   ps = Dict(typ => normalize!([isterminal(grammar, i) ?
                    w_terminal : w_nonterm
                    for i in grammar[typ]], 1)
         for typ in nonterminals(grammar))
   PPTNode(ps, PPTNode[])
end
function get_child(ppt::PPTNode, grammar, i)
   if i > length(ppt.children)
```

[7] *Арность* функции — это количество ее аргументов. Арность грамматического правила, которое можно рассматривать как функцию, — это количество нетерминальных типов в правиле.

[8] Распределение Дирихле часто используется для представления распределения по дискретным распределениям [11].

```
        push!(ppt.children, PPTNode(grammar))
    end
    return ppt.children[i]
end
```

Выражения выбираются с помощью векторов вероятности в вероятностном дереве прототипов. Правило в узле извлекается из категориального распределения, определенного вектором вероятности соответствующего узла для требуемого возвращаемого типа с последующей нормировкой соответствующих значений векторов вероятности для получения корректного распределения вероятности. Дерево обходится в глубину. Эта процедура выбора реализована в алгоритме 20.12 и представлена на рис. 20.9.

Алгоритм 20.12. Метод выбора выражений из вероятностного дерева прототипов. Дерево расширяется по мере необходимости

```
function rand(ppt, grammar, typ)
    rules = grammar[typ]
    rule_index = sample(rules, Weights(ppt.ps[typ]))
    ctypes = child_types(grammar, rule_index)

    arr = Vector{RuleNode}(undef, length(ctypes))
    node = iseval(grammar, rule_index) ?
           RuleNode(rule_index, eval(grammar, rule_index), arr) :
           RuleNode(rule_index, arr)

    for (i, typ) in enumerate(ctypes)
        node.children[i] =
            rand(get_child(ppt, grammar, i), grammar, typ)
    end
    return node
end
```

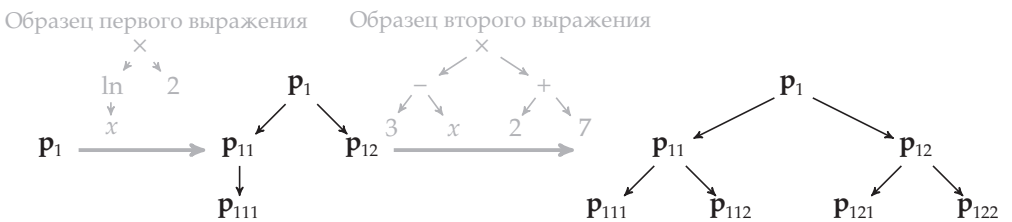


Рис. 20.9. Вероятностное дерево прототипов изначально содержит только корневой узел, но оно расширяется, так как для генерации выражений необходимы дополнительные узлы

Процесс обучения может использовать информацию либо из всей выборочной совокупности, либо из элитных выборок. Пусть лучшим выражением в текущем поколении будет x_{best}, а лучшим найденным выражением — x_{elite}. Вероятности узлов обновляются так, чтобы увеличить вероятность генерации x_{best}.[9]

Вероятность генерации выражения x_{best} является произведением вероятностей выбора каждого правила в x_{best} при обходе вероятностного дерева прототипов. Мы вычисляем целевую вероятность для $P(x_{\text{best}})$:

$$P_{\text{target}} = P(x_{\text{best}}) + (1 - P(x_{\text{best}}))\alpha \frac{\varepsilon - y_{\text{elite}}}{\varepsilon - y_{\text{best}}}, \tag{20.1}$$

где α и ε — положительные константы. Дробь с правой стороны стимулирует бо́льшие шаги к выражениям с лучшими значениями целевой функции. Целевая вероятность может быть вычислена с помощью алгоритма 20.13.

Алгоритм 20.13. Методы для вычисления вероятности выражений и целевой вероятности, где **ppt** — корневой узел вероятностного дерева прототипов, **grammar** — грамматика, **expr** и **x_best** — выражения типа **RuleNode**, **y_best** и **y_elite** — значения скалярной целевой функции, а **α** и **ε** являются скалярными параметрами

```
function probability(ppt, grammar, expr)
    typ = return_type(grammar, expr)
    i = findfirst(grammar[typ], expr.ind)
    prob = ppt.ps[typ][i]
    for (i, c) in enumerate(expr.children)
        prob *= probability(get_child(ppt, grammar, i), grammar, c)
    end
    return prob
end
function p_target(ppt, grammar, x_best, y_best, y_elite, α, ε)
    p_best = probability(ppt, grammar, x_best)
    return p_best + (1-p_best) * α * (ε - y_elite) / (ε - y_best)
end
```

Целевая вероятность используется для корректировки вероятностных векторов в вероятностном дереве прототипов. Вероятности, связанные с выбранными узлами, итеративно увеличиваются, пока не будет превышена целевая вероятность:

$$P\left(x_{\text{best}}^{(i)}\right) \leftarrow P\left(x_{\text{best}}^{(i)}\right) + c\alpha\left(1 - P\left(x_{\text{best}}^{(i)}\right)\right) \text{ для всех } i \in \{1, 2, \ldots\}, \tag{20.2}$$

где x_{best} — это i-е правило, применяемое в выражении x_{best}, а c — скаляр.[10]

[9] Первоначальная реализация дерева вероятностных прототипов будет периодически увеличивать вероятность генерации x_{elite}.

[10] Рекомендуемое значение: $c = 0{,}1$.

Затем адаптированные векторы вероятности нормируются к единице путем уменьшения значений всех нерасширенных компонентов вектора пропорционально их текущему значению. Вектор вероятности p, где i-й компонент был увеличен, корректируется в соответствии с формулой

$$p_j \leftarrow p_j \frac{1 - p_i}{\|\mathbf{p}\|_1 - p_i} \quad \text{для } j \neq i. \tag{20.3}$$

Процесс обучения реализован в алгоритме 20.14.

Алгоритм 20.14. Метод обучения вероятностного дерева прототипов с корнем **ppt**, грамматикой **grammar**, наилучшим выражением **x_best** со значением целевой функции **y_best**, значением элитной целевой функции **y_elite**, скоростью обучения **α**, скоростью обучения **c** и параметром **ε**

```
function _update!(ppt, grammar, x, c, α)
    typ = return_type(grammar, x)
    i = findfirst(isequal(x.ind), grammar[typ])
    p = ppt.ps[typ]
    p[i] += c * α * (1 - p[i])
    psum = sum(p)
    for j in 1 : length(p)
        if j != i
            p[j] *= (1 - p[i]) / (psum - p[i])
        end
    end
    for (pptchild, xchild) in zip(ppt.children, x.children)
        _update!(pptchild, grammar, xchild, c, α)
    end
    return ppt
end
function update!(ppt, grammar, x_best, y_best, y_elite, α, c, ε)
    p_targ = p_target(ppt, grammar, x_best, y_best, y_elite, α, ε)
    while probability(ppt, grammar, x_best) < p_targ
        _update!(ppt, grammar, x_best, c, α)
    end
    return ppt
end
```

В дополнение к популяционному обучению вероятностные деревья прототипов также могут исследовать пространство проектирования посредством мутаций. Дерево мутировало, чтобы исследовать область вокруг узла $\mathbf{x}_{\text{best}}$. Пусть $\mathbf{p}$ — вектор вероятности в узле, к которому мы обращались при генерации выражения $\mathbf{x}_{\text{best}}$. Каждый компонент в векторе $\mathbf{p}$ мутировал с вероятностью, пропорциональной размеру задачи:

$$\frac{p_{\text{mutation}}}{\#\mathbf{p}\sqrt{\#\mathbf{x}_{\text{best}}}}, \tag{20.4}$$

где p_{mutation} — параметр мутации, $\#\mathbf{p}$ — количество компонентов в векторе $\mathbf{p}$, а $\#\mathbf{x}_{\text{best}}$ — количество правил, применяемых в узле $\mathbf{x}_{\text{best}}$. Компонент i, выбранный для мутации, корректируется в соответствии с формулой:

$$p_i \leftarrow p_i + \beta(1 - p_i), \tag{20.5}$$

где коэффициент β контролирует количество мутаций. Малые вероятности подвергаются более сильным изменениям, чем бо́льшие. Все измененные векторы вероятности должны быть перенормированы. Мутация реализована в алгоритме 20.15 и представлена на рис. 20.10.

Алгоритм 20.15. Метод мутации вероятностного дерева прототипов с корнем `ppt`, грамматикой `grammar`, наилучшим выражением `x_best`, параметром мутации `p_mutation` и частотой мутаций `β`

```
function mutate!(ppt, grammar, x_best, p_mutation, β;
    sqrtlen = sqrt(length(x_best)),
    )
    typ = return_type(grammar, x_best)
    p = ppt.ps[typ]
    prob = p_mutation / (length(p) * sqrtlen)
    for i in 1 : length(p)
        if rand() < prob
            p[i] += β * (1 - p[i])
        end
    end
    normalize!(p, 1)
    for (pptchild, xchild) in zip(ppt.children, x_best.children)
        mutate!(pptchild, grammar, xchild, p_mutation, β,
                sqrtlen = sqrtlen)
    end
    return ppt
end
```

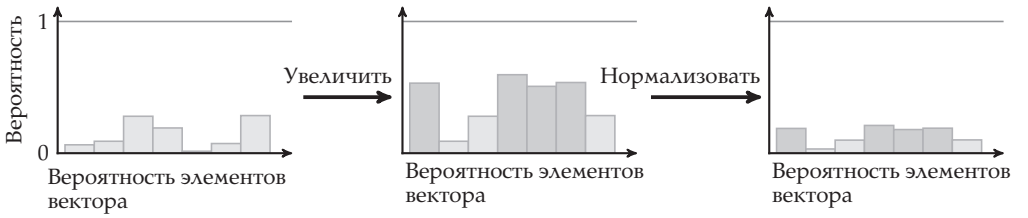


Рис. 20.10. Изменение вектора вероятности в вероятностном дереве прототипов с $\beta = 0{,}5$. Измененные компоненты были увеличены в соответствии с уравнением (20.5), а результирующий вектор вероятности был перенормирован. Обратите внимание на то, как меньшие вероятности получают большее увеличение

Наконец, поддеревья в вероятностном дереве прототипов отсекаются, чтобы удалить устаревшие части дерева. Дочерний узел удаляется, если его родительский узел содержит компонент вероятности выше заданного порогового значения, так что при выборе этот дочерний узел становится нерелевантным. Это всегда относится к терминальному правилу, но может относиться также и к нетерминальному правилу. Отсечение реализовано в алгоритме 20.16 и продемонстрировано в примере 20.7.

Алгоритм 20.16. Метод сокращения вероятностного дерева прототипов с корневым `ppt`, грамматикой `grammar` и порогом вероятности сокращения `p_treshold`

```
function prune!(ppt, grammar; p_threshold = 0.99)
    kmax, pmax = :None, 0.0
    for (k, p) in ppt.ps
        pmax' = maximum(p)
        if pmax' > pmax
          kmax, pmax = k, pmax'
        end
    end
    if pmax > p_threshold
       i = argmax(ppt.ps[kmax])
       if isterminal(grammar, i)
          empty!(ppt.children[kmax])
       else
          max_arity_for_rule = maximum(nchildren(grammar, r) for
                                       r in grammar[kmax])
          while length(ppt.children) > max_arity_for_rule
             pop!(ppt.children)
          end
```

```
      end
    end
    return ppt
end
```

Пример 20.7. Пример отсечения в вероятностном дереве прототипов

Рассмотрим узел с вектором вероятностей, заданном на множестве правил:

$$\mathbb{R} \mapsto \mathbb{R} + \mathbb{R}$$
$$\mathbb{R} \mapsto \ln(\mathbb{R})$$
$$\mathbb{R} \mapsto 2 \mid x$$
$$\mathbb{R} \mapsto \mathbb{S}$$

Если вероятность выбора двойки или x возрастает, то любой их дочерний узел в вероятностном дереве прототипов становится маловероятным и может быть отсечен. Аналогично, если вероятность выбора типа $\mathbb{S}$ возрастает, то его дочерние узлы с возвращаемым типом $\mathbb{R}$ становятся ненужными и могут быть отсечены.

20.6. Резюме

- Оптимизация выражений позволяет оптимизировать древовидные структуры, которые с помощью грамматики могут выражать сложные программы, структуры и другие конструкции, не имеющие фиксированного размера.
- Грамматики определяют правила, используемые для построения выражений.
- Генетическое программирование адаптирует генетические алгоритмы для выполнения мутаций и кроссоверов на деревьях выражений.
- Грамматическая эволюция работает с целочисленным массивом, который можно декодировать в дерево выражений.
- Вероятностные грамматики узнают, какие правила лучше всего генерировать, а вероятностные деревья прототипов изучают вероятности для каждой итерации правила генерации выражения.

20.7. Упражнения

Упражнение 20.1. Сколько деревьев выражений можно сгенерировать с помощью следующей грамматики и начального множества $\{\mathbb{R}, \mathbb{I}, \mathbb{F}\}$?

$$\mathbb{R} \mapsto \mathbb{I} \mid \mathbb{F}$$
$$\mathbb{I} \mapsto 1 \mid 2$$
$$\mathbb{F} \mapsto \pi$$

Упражнение 20.2. Количество деревьев выражений, высотой не больше h, которые могут быть сгенерированы в грамматике, растет сверхэкспоненциально. Определите, сколько выражений высоты h можно сгенерировать с помощью грамматики:[11]

$$\mathbb{N} \mapsto \{\mathbb{N}, \mathbb{N}\} \mid \{\mathbb{N},\} \mid \{, \mathbb{N}\} \mid \{\} \tag{20.6}$$

Упражнение 20.3. Определите грамматику, которая может генерировать любое неотрицательное целое число.

Упражнение 20.4. Как методы оптимизации выражений обрабатывают деление на нуль или другие исключения, встречающиеся при генерации случайных поддеревьев?

Упражнение 20.5. Рассмотрим арифметическую грамматику

$$\mathbb{R} \mapsto x \mid y \mid z \mid \mathbb{R} + \mathbb{R} \mid \mathbb{R} - \mathbb{R} \mid \mathbb{R} \times \mathbb{R} \mid \mathbb{R} / \mathbb{R} \mid \ln \mathbb{R} \mid \sin \mathbb{R} \mid \cos \mathbb{R}$$

Предположим, что переменные x, y и z имеют единицы измерения, и ожидается, что выходные данные будут выражены в конкретных единицах измерения. Как такую грамматику можно изменить для учета единиц измерения?

Упражнение 20.6. Рассмотрим грамматику

$$\mathbb{S} \mapsto \mathbb{N}\ \mathbb{P}\ \mathbb{V}\ \mathbb{P}$$
$$\mathbb{N}\ \mathbb{P} \mapsto \mathbb{A}\ \mathbb{D}\ \mathbb{J}\ \mathbb{N}\ \mathbb{P} \mid \mathbb{A}\ \mathbb{D}\ \mathbb{J}\ \mathbb{N}$$
$$\mathbb{V}\ \mathbb{P} \mapsto \mathbb{V}\ \mathbb{A}\ \mathbb{D}\ \mathbb{V}$$
$$\mathbb{A}\ \mathbb{D}\ \mathbb{J} \mapsto \text{a} \mid \text{the} \mid \text{big} \mid \text{little} \mid \text{blue} \mid \text{red}$$
$$\mathbb{N} \mapsto \text{mouse} \mid \text{cat} \mid \text{dog} \mid \text{pony}$$
$$\mathbb{V} \mapsto \text{ran} \mid \text{ sat} \mid \text{slept} \mid \text{ate}$$
$$\mathbb{A}\ \mathbb{D}\ \mathbb{V} \mapsto \text{quietly} \mid \text{quickly} \mid \text{soundly} \mid \text{happily}$$

Какой фенотип соответствует генотипу [2, 10, 19, 0, 6] и исходному символу $\mathbb{S}$?

Упражнение 20.7. Используйте генетическое программирование, чтобы вывести передаточные коэффициенты для часов. Предположим, что все шестерни имеют радиусы, выбранные из множества $\mathcal{R} = \{10, 25, 30, 50, 60, 100\}$. Каждая шестерня может быть либо прикреплена к оси своей родительской шестерни, тем самым разделяя с ней один и тот же период вращения, либо блокироваться на ободе своей родительской шестерни, тем самым имея период вращения, зависящий от периода вращения родительской шестерни и передаточного коэффициента. Часы также могут содержать стрелки, которые установлены на оси родительской ше-

[11] Пусть пустое выражение имеет высоту 0, выражение {} имеет высоту 1 и т.д.

стерни. Предположим, что корневая шестерня вращается с периодом $t_{\text{root}} = 0{,}1$ и имеет радиус, равный 25. Цель — создать часы с секундной, минутной и часовой стрелками.

Оценка каждой особи вычисляется по формуле

$$\left(\min_{\text{стрелки}}\left(1 - t_{\text{стрелки}}\right)^2\right) + \left(\min_{\text{стрелки}}\left(60 - t_{\text{стрелки}}\right)^2\right) + \left(\min_{\text{стрелки}}\left(3600 - t_{\text{стрелки}}\right)^2\right) + \#\text{nodes} \cdot 10^{-3},$$

где $t_{\text{стрелки}}$ — период вращения конкретной стрелки в секундах, а $\#\text{nodes}$ — количество узлов в дереве выражений. Направление вращения можно игнорировать.

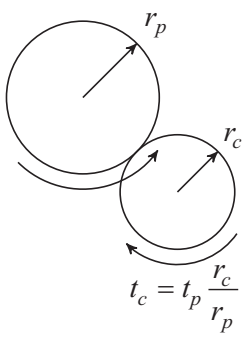


Рис. 20.11. Период вращения t_c дочерней шестерни, прикрепленной к ободу основной шестерни, зависит от периода вращения основной шестерни, t_p и отношения радиусов шестерен f

Упражнение 20.8. "Четыре четверки" [10] — это математическая задача, в которой с помощью четырех четверок и арифметических действий необходимо выразить все целые числа от 0 до 100. Например, первые два целых числа могут быть получены с помощью выражения 4 + 4 – 4 – 4 и 44/44 соответственно. Завершите головоломку "четыре четверки".

Упражнение 20.9. Рассмотрим вероятностную грамматику

$$\begin{array}{ll} \mathbb{R} \mapsto \mathbb{R} + \mathbb{R} \mid \mathbb{R} \times \mathbb{R} \mid \mathbb{F} \mid \mathbb{I} & w_{\mathbb{R}} = [1, 1, 5, 5] \\ \mathbb{F} \mapsto 1{,}5 \mid \infty & p_{\mathbb{F}} = [4, 3] \\ \mathbb{I} \mapsto 1 \mid 2 \mid 3 & p_{\mathbb{I}} = [1, 1, 1] \end{array}$$

Какова вероятность генерации выражения 1,5 + 2?

Упражнение 20.10. Как выглядит вероятностная грамматика из предыдущего упражнения после сброса счетчиков и применения правил обучения к выражению 1,5 + 2?

Глава 21

Междисциплинарная оптимизация

Междисциплинарная оптимизация проектов (Multidisciplinary Design Optimization — MDO) включает в себя решение задач оптимизации, охватывающих разные дисциплины. Многие реальные проблемы связаны со сложным взаимодействием между несколькими дисциплинами, и индивидуальная оптимизация дисциплин может не привести к оптимальному решению. В этой главе обсуждаются различные методы, позволяющие воспользоваться структурой задач MDO для сокращения усилий, необходимых для создания хороших проектов.[1]

21.1. Дисциплинарный анализ

Существует множество различных методов *дисциплинарного анализа*, которые могут повлиять на проект. Например, при конструировании ракеты могут использоваться методы таких дисциплин, как структурный анализ, аэродинамика и системы управления. Различные дисциплины имеют свои собственные аналитические инструменты, например метод конечных элементов. Эти методы бывают довольно сложными и вычислительно дорогими. Кроме того, методы разных дисциплин зачастую тесно связаны друг с другом, когда одна дисциплина может потребовать результатов применения методов другой дисциплины. Разрешение этих взаимозависимостей может быть нетривиальным.

В контексте MDO у нас все еще есть набор проектных переменных, как и раньше, но мы также отслеживаем выходные данные, или *отклики*, отдельно для каждого метода разных дисциплин.[2] Мы записываем отклики i-го дисциплинарного анализа как $\mathbf{y}^{(i)}$. В общем случае i-й дисциплинарный анализ F_i может зависеть от проектных переменных или отклика любой другой дисциплины:

---

[1] Обширный обзор предоставлен в работе [101]. Дальнейшее обсуждение можно найти в [1, 144].

[2] Дисциплинарный анализ может предоставить входные данные для других дисциплин, целевой функции или ограничений. Кроме того, он дает информацию о градиенте для поиска оптимальной точки.

$$\mathbf{y}^{(i)} \leftarrow F_i(\mathbf{x}, \mathbf{y}^{(1)}, ..., \mathbf{y}^{(i-1)}, \mathbf{y}^{(i+1)}, ..., \mathbf{y}^{(m)}), \tag{21.1}$$

где m — общее количество дисциплин. Входные данные для вычислительного анализа гидродинамики летательного аппарата могут включать, например, прогибы крыла, которые получены в результате структурного анализа, требующего применения методов вычислительной гидродинамики. Важной частью постановки задач MDO является учет таких зависимостей между разными дисциплинами.

Чтобы облегчить рассуждения о дисциплинарном анализе, мы вводим понятие *задания* (assignment). Задание $\mathcal{A}$ — это множество имен переменных и их соответствующих значений, относящихся к задаче междисциплинарной оптимизации проекта. Чтобы получить доступ к переменной v, мы пишем $\mathcal{A}[v]$.

Дисциплинарный анализ — это функция, которая принимает задание и использует расчетную точку и отклики из других анализов, чтобы перезаписать отклик для своей дисциплины:

$$\mathcal{A}' \leftarrow \mathcal{F}_i(\mathcal{A}), \tag{21.2}$$

где $\mathcal{F}_i(\mathcal{A})$ обновляет $\mathbf{y}^{(i)}$ в $\mathcal{A}$, чтобы произвести $\mathcal{A}'$.

Задания могут быть представлены в коде с помощью словарей.[3] Каждой переменной присваивается имя типа `String`. Переменные не ограничиваются числами с плавающей запятой, но могут включать в себя другие объекты, например векторы. Реализацию с помощью словаря демонстрирует пример 21.1.

Пример 21.1. Синтаксис кода для описания задач оптимизации междисциплинарного проектирования с помощью заданий

Рассмотрим оптимизацию с одной переменной x и двумя дисциплинами. Предположим, что первый дисциплинарный анализ F_1 вычисляет отклик $y^{(1)} = f_1(x, y^{(2)})$, а второй дисциплинарный анализ F_2 вычисляет отклик $y^{(2)} = f_2(x, y^{(1)})$.

Эта задача может быть реализована так:

```
function F1(A)
    A["y1"] = f1(A["x"], A["y2"])
    return A
end
function F2(A)
    A["y2"] = f2(A["x"], A["y1"])
    return A
end
```

[3] *Словарь*, или *ассоциативный массив*, — это структура данных, позволяющих выполнять индексацию по ключам, а не целым числам (см. приложение A.1.7).

Задание может быть инициализировано с догадками для $y^{(1)}$ и $y^{(2)}$ и известным значением для x. Например:

```
A = Dict("x" => 1, "y1" => 2, "y2" => 3)
```

21.2. Междисциплинарная совместимость

Вычисление значения целевой функции и допустимости расчетной точки **x** требует получения значений отклика, которые удовлетворяют условию *междисциплинарной совместимости*, а это означает, что отклики должны соответствовать дисциплинарному анализу. Междисциплинарная совместимость имеет место для конкретного задания, если задание остается неизменным при всех дисциплинарных анализах:

$$\mathcal{F}_i(\mathcal{A}) = \mathcal{A} \text{ для } i \in \{1, ..., m\}. \tag{21.3}$$

Выполнение любого анализа даст одни и те же значения. Поиск задания, удовлетворяющего условию междисциплинарной совместимости, называется *междисциплинарным анализом*.

Оптимизация системы в рамках отдельной дисциплины требует выбора проектных переменных и запроса на дисциплинарный анализ для оценки ограничений и целевой функции, как показано на рис. 21.1. Оптимизация в рамках одной дисциплины не требует рассмотрения дисциплинарной взаимосвязи.

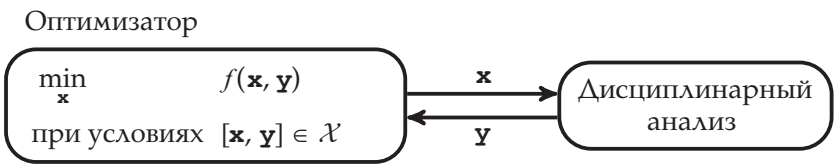


Рис. 21.1. Диаграмма оптимизации в рамках отдельной дисциплины. Градиенты могут как вычисляться, так и не вычисляться

Оптимизация системы в рамках нескольких дисциплин может вводить зависимости, и в этом случае возникает проблема взаимосвязи. Диаграмма для двух связанных дисциплин приведена на рис. 21.2. Применение традиционной оптимизации к этой задаче сложнее, потому что должна быть установлена междисциплинарная совместимость.

Если междисциплинарный анализ не имеет *циклической зависимости*[4], то обеспечение междисциплинарной совместимости является простым делом. Мы говорим, что дисциплина i всегда зависит от дисциплины j, если дисциплине i

[4] Циклическая зависимость возникает, когда дисциплины зависят друг от друга.

нужны какие-либо из выводов дисциплины *j*. Это отношение зависимости может использоваться для формирования графа зависимости, где каждый узел соответствует дисциплине, а ребро $j \rightarrow i$ включается, если дисциплина *i* зависит от дисциплины *j*. На рис. 21.3 показаны примеры графов зависимостей, включающих две дисциплины с циклами и без них.

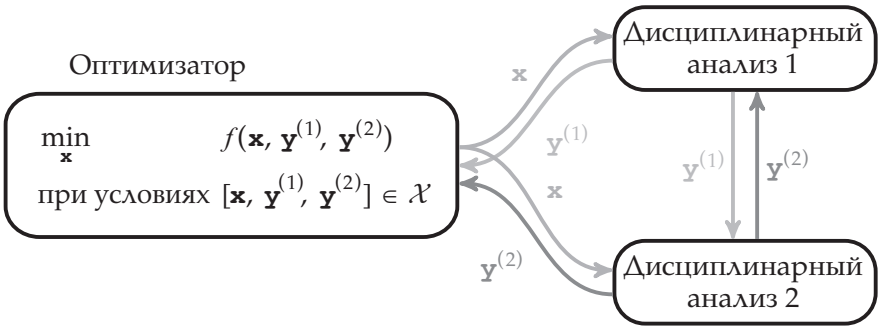


Рис. 21.2. Диаграмма оптимизации для двухдисциплинарного анализа с междисциплинарным взаимодействием

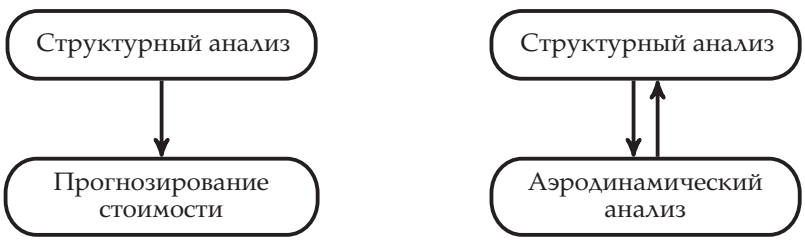


Рис. 21.3. Графы циклических и ациклических зависимостей

Если в графе зависимостей нет циклов, то всегда существует порядок вычислений, который, если его придерживаться, гарантирует, что необходимые дисциплинарные анализы будут выполнены до дисциплинарных анализов, которые от них зависят. Такой порядок называется *топологическим* и может быть обеспечен с помощью метода топологической сортировки, такого как *алгоритм Кана* [76]. Переупорядочение анализов показано на рис. 21.4.

Если граф зависимостей имеет циклы, то топологического порядка не существует. Для обращения к циклам используется *метод Гаусса–Зейделя* (алгоритм 21.1), который пытается выполнить междисциплинарный анализ итеративно, пока не будет обеспечена сходимость.[5] Алгоритм Гаусса–Зайделя чувствителен к упорядочению дисциплин, как показано в примере 21.2. Плохой порядок

[5] Алгоритм Гаусса–Зейделя можно сделать параллельным.

может исключить или замедлить сходимость. Лучший порядок — тот, при котором существует минимальная обратная связь.[6]

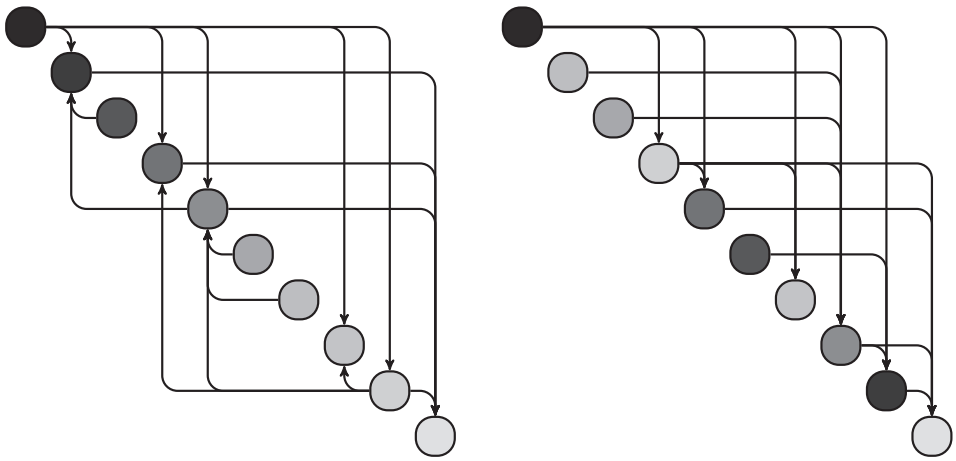


Рис. 21.4. Для изменения порядка дисциплинарных анализов с целью устранения связей с обратной связью может использоваться топологическая сортировка

Алгоритм 21.1. Алгоритм Гаусса–Зейделя для проведения междисциплинарного анализа. Здесь **Fs** — это вектор функций дисциплинарного анализа, которые принимают и модифицируют задачу **A**. Есть два необязательных аргумента: максимальное количество итераций **k_max** и относительная погрешность **ε**. Метод возвращает измененное задание и признак сходимости

```
function gauss_seidel!(Fs, A; k_max = 100, ε = 1e-4)
    k, converged = 0, false
    while !converged && k ≤ k_max
        k += 1
        A_old = deepcopy(A)
        for F in Fs
            F(A)
        end
        converged = all(isapprox(A[v], A_old[v], rtol=.)
                        for v in keys(A))
        end
        return (A, converged)
end
```

[6] В некоторых случаях дисциплины могут быть разделены на разные группы, которые не зависят друг от друга. Каждый связанный кластер может быть изучен с использованием своего собственного, более детального междисциплинарного анализа.

Пример 21.2. Пример, который иллюстрирует важность выбора соответствующего порядка при проведении междисциплинарного анализа

Рассмотрим междисциплинарную задачу оптимизации дизайна с одной переменной дизайна x и тремя дисциплинами, каждая с одной переменной ответа:

$$y^{(1)} \leftarrow F_1\left(x, y^{(2)}, y^{(3)}\right) = y^{(2)} - x,$$
$$y^{(2)} \leftarrow F_2\left(x, y^{(1)}, y^{(3)}\right) = \sin\left(y^{(1)} + y^{(3)}\right),$$
$$y^{(3)} \leftarrow F_3\left(x, y^{(1)}, y^{(2)}\right) = \cos\left(x + y^{(1)} + y^{(2)}\right).$$

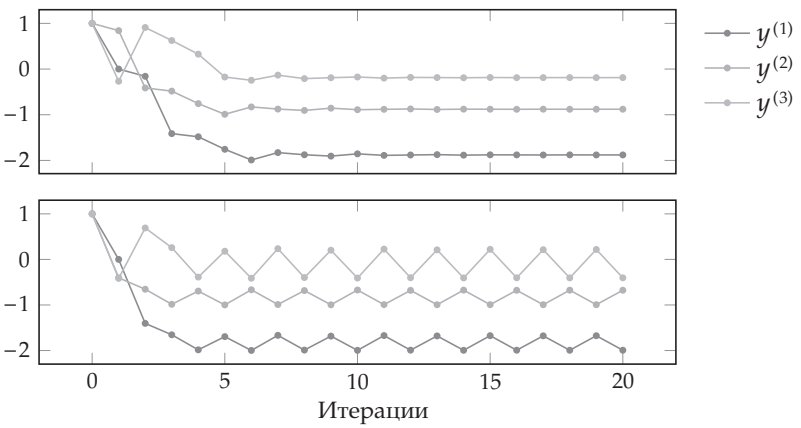
Дисциплинарный анализ может быть реализован следующим образом:

```
function F1(A)
    A["y1"] = A["y2"] - A["x"]
    return A
end
function F2(A)
    A["y2"] = sin(A["y1"] + A["y3"])
    return A
end
function F3(A)
    A["y3"] = cos(A["x"] + A["y2"] + A["y1"])
    return A
end
```

Рассмотрим междисциплинарный анализ для $x = 1$, инициализировав наше задание единицами:

```
A = Dict("x" => 1, "y1" => 1, "y2" => 1, "y3" => 1)
```

Выполнение алгоритма Гаусса–Зейделя в порядке F_1, F_2, F_3 сходится, а в порядке F_1, F_3, F_2 — нет.



Может оказаться полезным объединить дисциплины в новый дисциплинарный анализ — сгруппировать концептуально связанные методы, одновременно выполнить тесно связанные методы или более эффективно применить архитектуры, обсуждаемые в этой главе. Объединение разных дисциплин может сформировать новую дисциплину, чьи отклики состоят из откликов дисциплин, входящих в объединение. Форма новой дисциплины зависит от дисциплинарных взаимозависимостей. Если объединенные дисциплины являются ациклическими, то существует порядок, в котором методы разных дисциплин могут выполняться последовательно. Если объединенные дисциплины являются циклическими, то новая дисциплина должна обеспечить междисциплинарный анализ для достижения совместимости.

21.3. Архитектура

Задачи междисциплинарной оптимизации проектов можно записать в следующем виде:

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathcal{A}) \\ &\text{при условии, что } \mathcal{A} \in \mathcal{X} \\ &\mathcal{F}_i(\mathcal{A}) = \mathcal{A} \text{ для любой дисциплины } i \in \{1, ..., m\}, \end{aligned} \tag{21.4}$$

где целевая функция f и допустимое множество $\mathcal{X}$ зависят как от проектных переменных, так и отклика. Проектные переменные в задании определяются конструктором. Условие $\mathcal{F}_i(\mathcal{A}) = \mathcal{A}$ гарантирует, что i-я дисциплина соответствует значениям в задании $\mathcal{A}$. Это последнее условие обеспечивает междисциплинарную совместимость.

Существует несколько проблем, связанных с оптимизацией междисциплинарных задач. Взаимозависимость дисциплинарных анализов приводит к тому, что порядок их выполнения имеет большое значение и часто делает их параллельное выполнение трудным или невозможным. Существует компромисс между оптимизатором, который напрямую контролирует все переменные, с субоптимизатором[7], который использует специальные знания дисциплины для проведения дисциплинарного анализа, и затратами на глобальную оптимизацию слишком большого количества переменных. Наконец, каждая архитектура должна обеспечивать междисциплинарную совместимость в конечном решении.

В оставшейся части этой главы обсуждается множество различных архитектур оптимизации для решения этих задач. Эти архитектуры демонстрируются с использованием гипотетической задачи райдшеринга, представленной в примере 21.3.

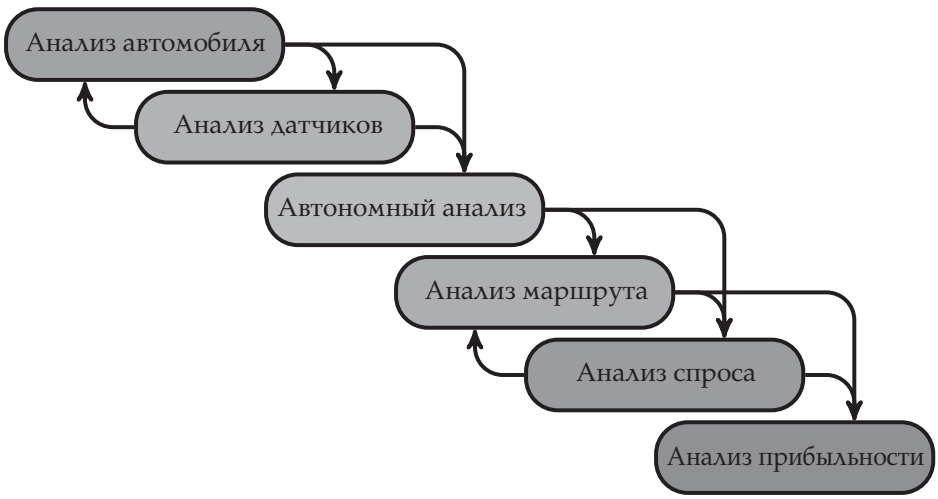
---

[7] *Субоптимизатор* — это программа оптимизации, вызываемая другой программой оптимизации.

Пример 21.3. Задача райдшеринга, использованная в этой главе для демонстрации архитектур оптимизации

Рассмотрим компанию по райдшерингу, которая организовывает автопарк беспилотных автомобилей. Эта гипотетическая компания одновременно разрабатывает транспортное средство, его комплект датчиков, стратегию маршрутизации и схему ценообразования. Данным частям проекта соответствуют векторы **v**, **s**, **r** и **p** соответственно, каждый из которых содержит множество проектных переменных. Транспортное средство, например, может включать в себя параметры, управляющие структурной геометрией, двигателем и трансмиссией, емкостью аккумулятора и пассажировместимостью.

Цель компании райдшеринга — максимизировать прибыль. Прибыль зависит от масштабного моделирования алгоритма маршрутизации и спроса пассажиров, что, в свою очередь, зависит от отклика, полученного в результате анализа автономного транспортного средства и его комплекта датчиков. Несколько анализов извлекают дополнительную информацию. Производительность алгоритма маршрутизации зависит от спроса, создаваемого схемой ценообразования, а спрос, создаваемый схемой ценообразования, зависит от производительности алгоритма маршрутизации. Дальность поездки и эффективность использования топлива зависят от веса, сопротивления воздуха и потребляемой мощности пакета датчиков. Пакету датчиков нужна информация о геометрии и характеристиках автомобиля, чтобы соответствовать необходимым требованиям безопасности. Диаграмма зависимостей представлена ниже.



21.4. Допустимая архитектура междисциплинарного проекта

Допустимая архитектура междисциплинарного проекта структурирует задачу MDO так, что стандартные алгоритмы оптимизации могут быть непосредственно применены для оптимизации проектных переменных. Мультидисциплинарный проектный анализ выполняется для любой заданной расчетной точки, чтобы получить совместимые значения отклика.

Диаграмма архитектуры приведена на рис. 21.5. Она состоит из двух блоков: оптимизатора и междисциплинарного анализа. Оптимизатор — это метод, используемый для выбора расчетных точек с целью минимизации целевой функции. Оптимизатор вызывает блок междисциплинарного анализа, передавая ему расчетную точку $\mathbf{x}$ и получая совместимое задание $\mathcal{A}$. Если междисциплинарная совместимость невозможна, блок междисциплинарного анализа информирует оптимизатор, и такие расчетные точки рассматриваются как недопустимые. На рис. 21.6 показано, как преобразовать задачу MDO в стандартную задачу оптимизации с помощью междисциплинарного анализа проектов.

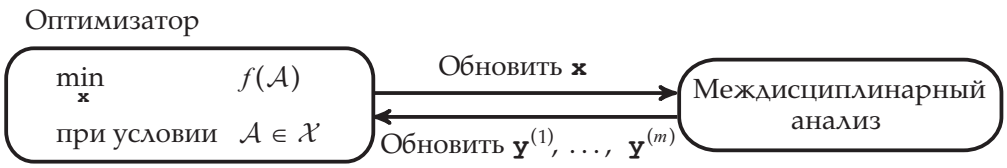


Рис. 21.5. Допустимая архитектура междисциплинарного проекта. Оптимизатор выбирает расчетные точки x, а междисциплинарный анализ вычисляет согласованное задание $\mathcal{A}$. Эта структура аналогична структуре оптимизации по одной дисциплине

$$\begin{aligned} &\min_{\mathbf{x}} && f\left(\mathbf{x}, \mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(m)}\right) \\ &\text{при условии} && \left[\mathbf{x}, \mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(m)}\right] \in \mathcal{X} \end{aligned} \quad \longrightarrow \quad \begin{aligned} &\min_{\mathbf{x}} && f(\mathrm{MDA}(\mathbf{x})) \\ &\text{при условии} && \mathrm{MDA}(\mathbf{x}) \in \mathcal{X} \end{aligned}$$

Рис. 21.6. Формулирование задачи MDO как типичной задачи оптимизации с использованием междисциплинарного анализа проектов, где функция $\mathrm{MDA}(\mathbf{x})$ возвращает междисциплинарное совместимое задание

Основными преимуществами возможной архитектуры междисциплинарного проектирования являются ее концептуальная простота и то, что она гарантированно поддерживает междисциплинарную совместимость на каждом этапе оптимизации. Его название отражает тот факт, что междисциплинарный анализ проекта выполняется при каждой оценке проекта, гарантируя, что оптимизатор системного уровня рассматривает только допустимые проекты.

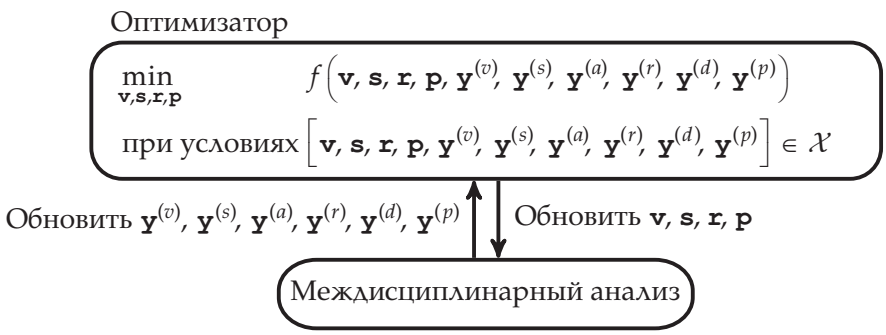
Основным недостатком является то, что анализ междисциплинарного проектирования является дорогостоящим и обычно требует нескольких итераций для всех анализов. Итерационные методы Гаусса–Зейделя могут медленно сходиться или не сходиться вообще, в зависимости от инициализации откликов и порядка дисциплинарного анализа.

Объединение анализа воедино делает необходимым, чтобы все локальные переменные — как правило, относящиеся только к конкретной дисциплине, — были рассмотрены при анализе в целом. Многие практические задачи имеют очень большое количество локальных проектных переменных, таких как контрольные точки сетки в аэродинамике, размеры элементов в конструкциях, размещение компонентов в электротехнике и веса нейронных сетей в машинном обучении. Возможная оптимизация междисциплинарного проектирования требует, чтобы системный оптимизатор задавал все эти значения во всех дисциплинах, удовлетворяя при этом всем ограничениям.

В примере 21.4 возможная архитектура междисциплинарного проектирования применяется к задаче райдшеринга.

Пример 21.4. Возможная архитектура междисциплинарного проекта, применяемая для решения задачи райдшеринга. Междисциплинарный анализ всех откликов должен быть завершен для каждой возможной расчетной точки. Это, как правило, слишком дорого с вычислительной точки зрения

Допустимую архитектуру междисциплинарного проекта можно применить для решения задачи райдшеринга. Диаграмма архитектуры приведена ниже.



21.5. Последовательная оптимизация

Архитектура последовательной оптимизации (рис. 21.7) использует специальные инструменты и опыт дисциплины для оптимизации подзадач, но может привести к неоптимальным решениям. Эта архитектура включена для демонстра-

ции ограничений наивного подхода и служит базой, с которой можно сравнивать другие архитектуры.

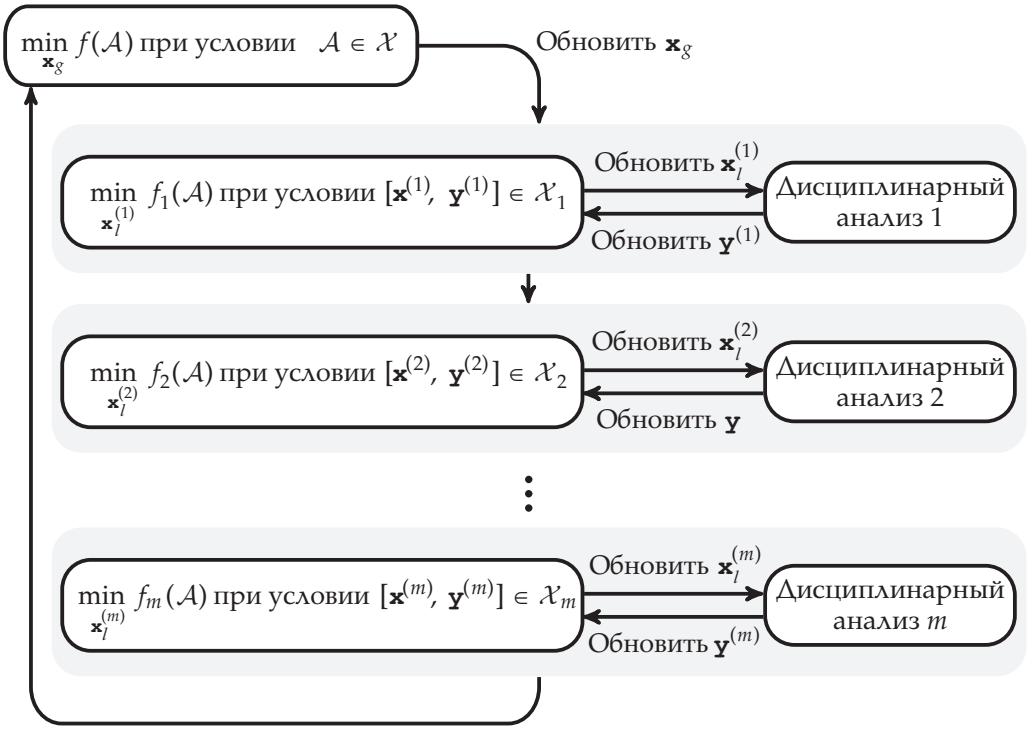


Рис. 21.7. Архитектура последовательной оптимизации. Каждая подзадача представлена синим блоком и оптимизирует локальную целевую функцию для конкретной дисциплины

Подзадача — это процедура оптимизации, выполняемая на каждой итерации всеобъемлющего процесса оптимизации. Иногда проектные переменные могут быть удалены из процедуры внешней оптимизации, *оптимизатора системного уровня*, и более эффективно оптимизированы в подзадачах.

Проектные переменные для i-й дисциплины могут быть разделены в соответствии: $\mathbf{x}^{(i)} = \left[\mathbf{x}_g^{(i)}, \mathbf{x}_l^{(i)}\right]$, где $\mathbf{x}_g$ — *глобальные проектные переменные*, используемые совместно с другими дисциплинами, а $\mathbf{x}_l^{(i)}$ — *локальные проектные переменные*, применяемые только соответствующей дисциплинарной подзадачей.[8] Отклики могут быть аналогичным образом разделены на глобальные $\mathbf{y}_g^{(i)}$ и ло-

[8] Подзадача о транспортных средствах в задаче рейдшеринга может включать в себя такие глобальные проектные переменные, как вместимость и размеры автомобиля, влияющие на другие дисциплины, но также могут содержать локальные проектные переменные, например конфигурацию сидений, которые не влияют на другие дисциплины.

кальные — $\mathbf{y}_l^{(i)}$. Дисциплинарная автономия достигается путем оптимизации локальных переменных в собственных дисциплинарных оптимизаторах. Локальная целевая функция f_i должна быть выбрана так, чтобы ее оптимизация также приносила пользу глобальной цели. Оптимизатор верхнего уровня отвечает за оптимизацию глобальных проектных переменных $\mathbf{x}_g$ относительно исходной целевой функции. Реализация $\mathbf{x}_g$ вычисляется с помощью последовательной оптимизации; каждая подзадача оптимизируется одна за другой, передавая свои результаты следующей подзадаче, пока все не будут решены.

Последовательная оптимизация использует преимущества дисциплины, которые заключаются в том, что многие переменные являются уникальными для конкретной дисциплины и не требуют совместного применения путем пересечения границы дисциплины. Последовательная оптимизация использует умение каждой дисциплины решать ее специфические задачи. Оптимизаторы подзадач имеют полный контроль над специфическими для дисциплины проектными переменными, чтобы соответствовать целям и ограничениям локального проекта.

За исключением особых случаев, последовательная оптимизация не приводит к оптимальному решению исходной задачи по той же причине, по которой метод Гаусса–Зейделя сходится не всегда. Решение чувствительно к локальным целевым функциям, и поиск подходящих локальных целевых функций часто является сложной задачей. Последовательная оптимизация не поддерживает параллельное выполнение, а междисциплинарная совместимость обеспечивается путем итерации и не всегда сходится.

Пример 21.5 применяет последовательную оптимизацию для решения задачи рейдшеринга.

Пример 21.5. Последовательная оптимизация для решения задачи рейдшеринга

Архитектура последовательной оптимизации может оптимизировать некоторые переменные локально. На рис. 21.8 показан результат применения последовательной оптимизации к задаче совместного использования.

Конструктивные переменные для транспортного средства, сенсорной системы, алгоритма маршрутизации и схемы ценообразования разделены на локальные переменные, специфичные для дисциплины, и глобальные переменные верхнего уровня. Например, подзадача о транспортном средстве может оптимизировать локальные параметры транспортного средства $\mathbf{v}_l$, в частности компоненты трансмиссии. Однако такие параметры, как вместимость транспортного средства, используемые в других анализах, контролируются глобально в $\mathbf{v}_g$.

Тесная связь между транспортным средством и сенсорными системами плохо обрабатывается архитектурой последовательной оптимизации. В то время как изменения, внесенные подзадачей о транспортном средстве, немедленно рас-

сматриваются подзадачей о датчике, влияние подзадачи о датчике на подзадачу о транспортном средстве не рассматривается до следующей итерации.

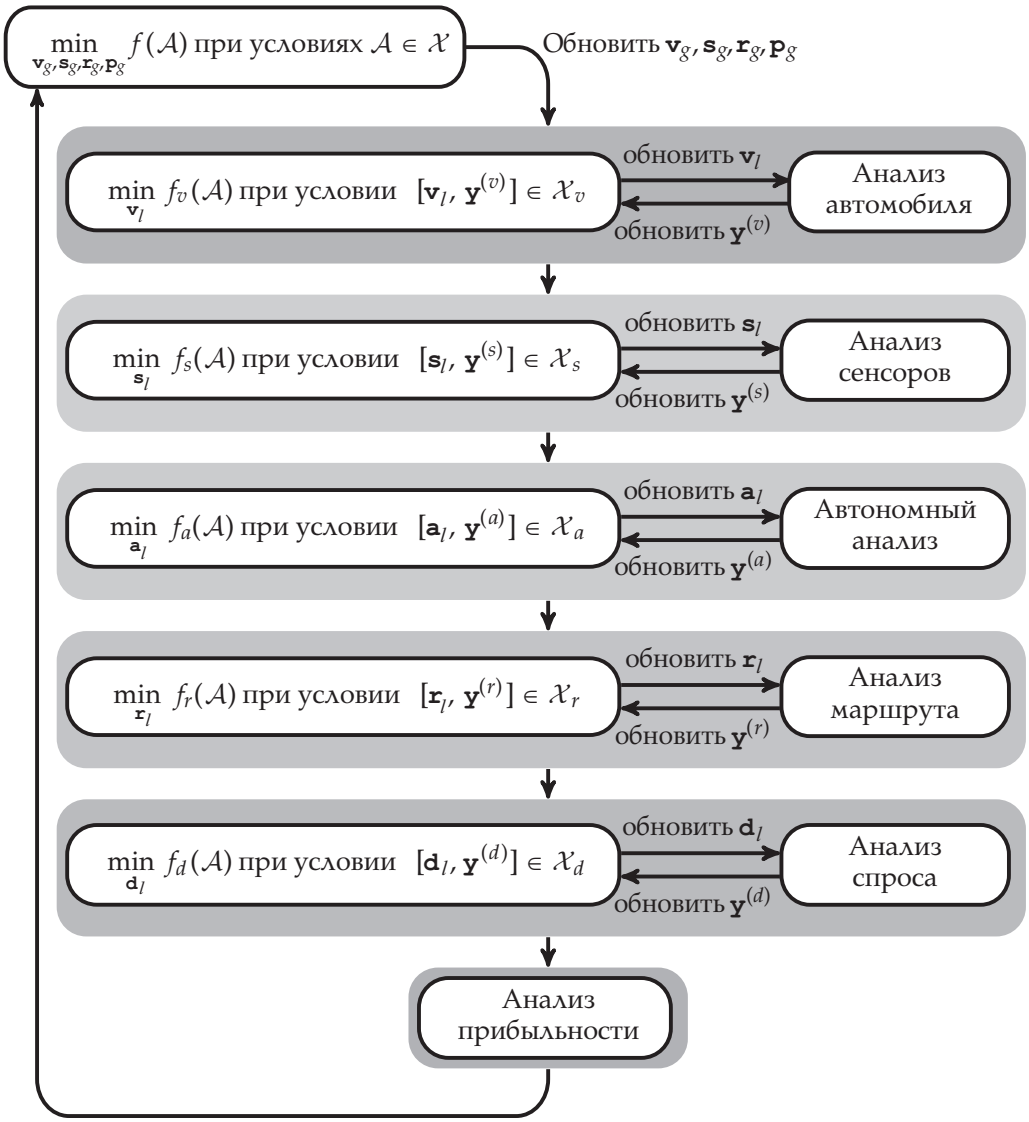


Рис. 21.8. Применение архитектуры последовательной оптимизации к задаче рейдшеринга

Не все анализы требуют своих собственных подзадач. Предполагается, что анализ прибыли не имеет каких-либо локальных проектных переменных и, следовательно, может быть выполнен без использования блока подзадач.

21.6. Допустимая архитектура отдельной дисциплины

Допустимая архитектура отдельной дисциплины (individual discipline feasible architecture — IDF) устраняет необходимость в проведении дорогостоящих междисциплинарных анализов проектов и позволяет выполнять дисциплинарные анализы параллельно. Это не гарантирует, что междисциплинарная совместимость поддерживается на протяжении всего ее исполнения, а возможная совместимость обеспечивается через ограничения в виде равенства в оптимизаторе. Совместимость обеспечивается не в междисциплинарном анализе, а самим оптимизатором.

Архитектура IDF вводит в область проектирования *переменные связи*. Для каждой дисциплины дополнительный вектор $\mathbf{c}^{(i)}$ добавляется к задаче оптимизации, чтобы действовать как псевдоним для отклика $\mathbf{y}^{(i)}$. Отклики неизвестны до тех пор, пока они не будут вычислены с помощью анализа соответствующих предметных областей; включение переменных связи позволяет оптимизатору предоставлять эти оценки одновременно нескольким дисциплинам при параллельном анализе. Равенство между переменными связи и отклика обычно достигается путем итерации. Равенство — это ограничение оптимизации, $\mathbf{c}^{(i)} = \mathbf{y}^{(i)}$, для каждой дисциплины.

Общая архитектура IDF показана на рис. 21.9. Оптимизатор системного уровня работает с переменными связи и использует их для заполнения задания, которое копируется в дисциплинарный анализ на каждой итерации:

$$\mathcal{A}[\mathbf{x}, \mathbf{y}^{(1)}, ..., \mathbf{y}^{(m)}] \leftarrow [\mathbf{x}, \mathbf{c}^{(1)}, ..., \mathbf{c}^{(m)}] \tag{21.5}$$

Несмотря на дополнительную свободу параллельного выполнения анализа, архитектура IDF страдает от недостатка, заключающегося в том, что она не может использовать процедуры оптимизации для конкретной предметной области так же, как последовательная оптимизация, поскольку оптимизация является только верхним уровнем. Кроме того, оптимизатор должен удовлетворять дополнительным ограничениям равенства и иметь больше переменных для оптимизации. У архитектуры IDF могут возникнуть трудности с оптимизацией на основе градиента, поскольку выбранное направление поиска должно учитывать ограничения, как показано на рис. 21.10. Изменения в переменных проектирования не должны приводить к тому, что переменные связи становятся недопустимыми в отношении дисциплинарного анализа. Оценка градиентов функции цели и ограничения очень затратна, когда дисциплинарный анализ дорог.

На рис. 20.11. допустимая архитектура отдельной дисциплины применяется к задаче рейдшеринга.

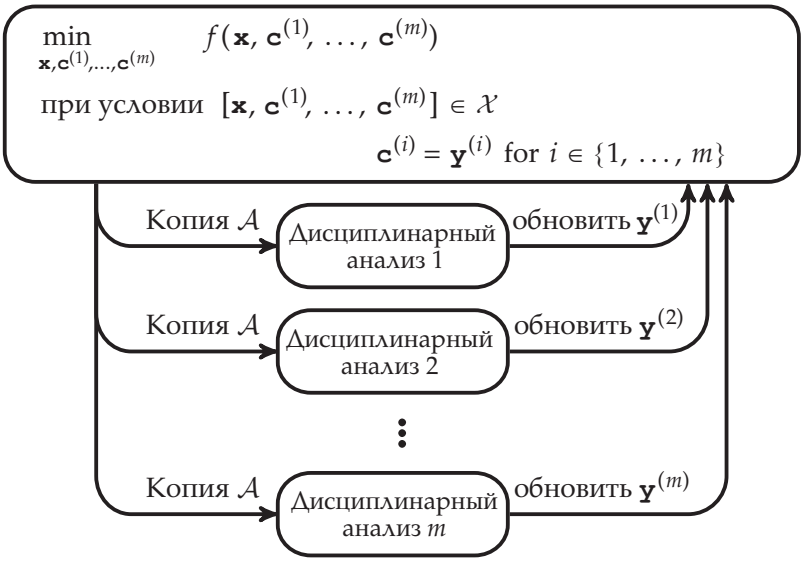


Рис. 21.9. Допустимая архитектура отдельной дисциплины позволяет проводить дисциплинарный анализ параллельно. В этой главе предполагается, что дисциплинарный анализ изменяет свои входные данные, поэтому копии назначений оптимизатора системного уровня передаются каждому дисциплинарному анализу

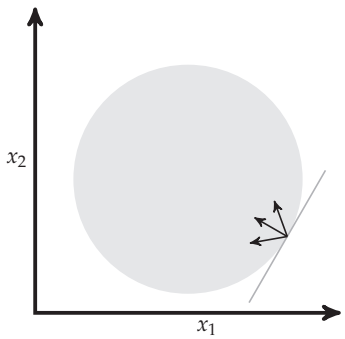


Рис. 21.10. Направление поиска для точки на границе ограничения должно привести к допустимому множеству

21.7. Совместная оптимизация

Архитектура совместной оптимизации (рис. 21.12) разбивает задачу на дисциплинарные подзадачи, которые имеют полный контроль над своими локальными переменными проектирования и специфическими для дисциплины ограничениями. Подзадачи могут быть решены с помощью специальных инструментов и оптимизированы параллельно.

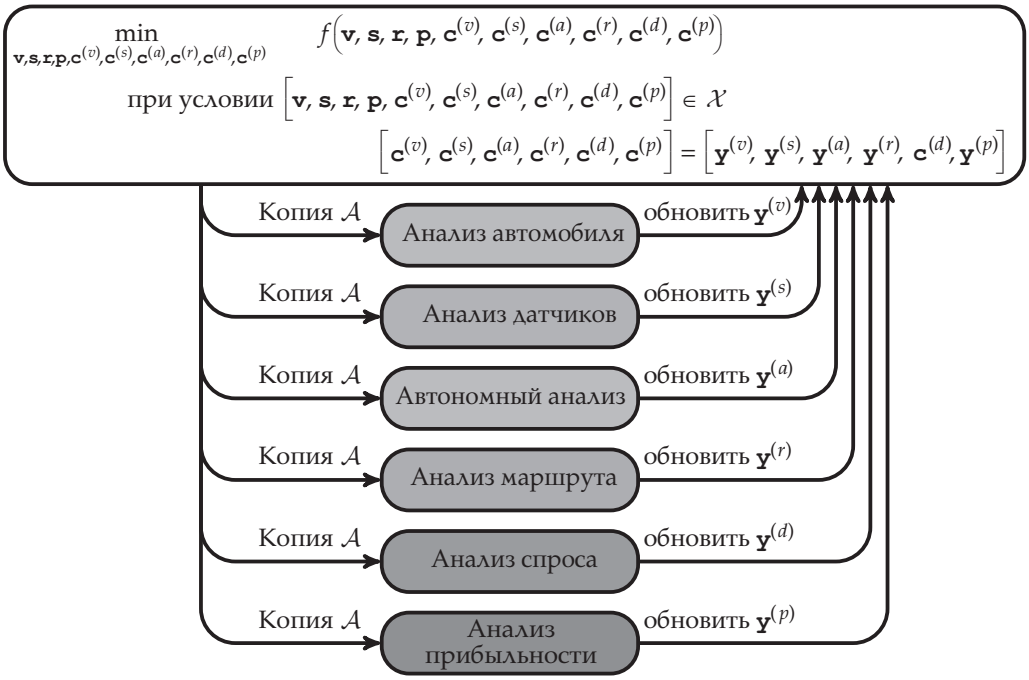


Рис. 21.11. Применение допустимой архитектуры отдельной дисциплины к задаче рейдшеринга. Архитектура IDF допускает параллельное выполнение анализов, но оптимизатор системного уровня должен оптимизировать большое количество переменных

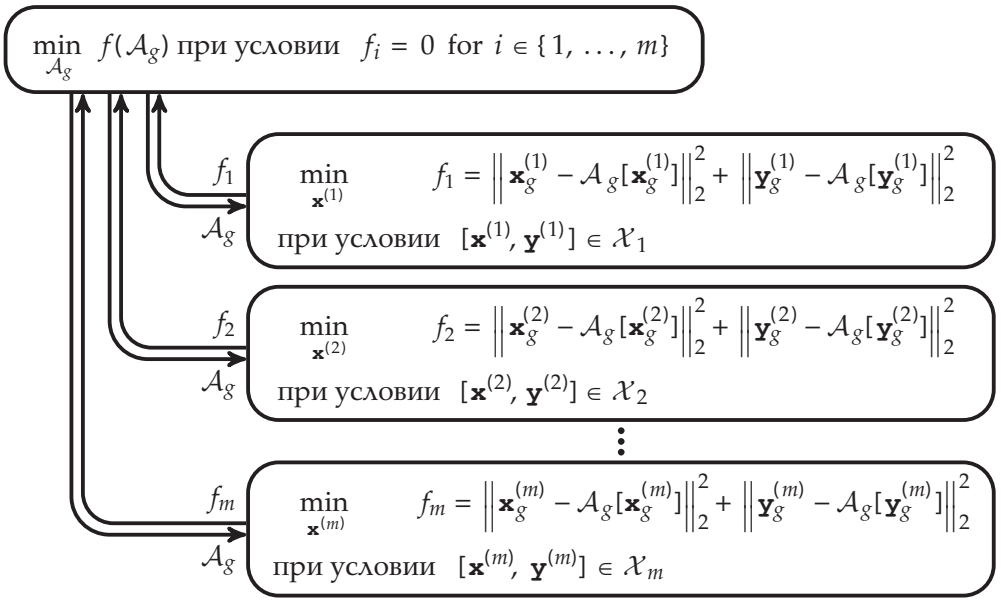


Рис. 21.12. Архитектура совместной оптимизации проекта

Подзадача *i* имеет вид:

$$\begin{aligned} &\min_{\mathbf{x}^{(i)}} && f_i\left(\mathbf{x}^{(i)},\ \mathbf{y}^{(i)}\right) \\ &\text{при условии, что} && \left[\mathbf{x}^{(i)},\ \mathbf{y}^{(i)}\right] \in \mathcal{X}_i, \end{aligned} \tag{21.6}$$

где вектор $\mathbf{x}^{(i)}$ содержит подмножество проектных переменных $\mathbf{x}$ и откликов $\mathbf{y}^{(i)}$. Ограничение гарантирует, что решение удовлетворяет дисциплинарным ограничениям.

Междисциплинарная совместимость требует, чтобы глобальные переменные $\mathbf{x}_g^{(i)}$ и $\mathbf{y}_g^{(i)}$ согласовывались между всеми дисциплинами. Мы определяем набор переменных связи $\mathcal{A}_g$, который включает в себя переменные, соответствующие всем проектным переменным и откликам, которые являются глобальными по крайней мере в одной подзадаче. Соглашение обеспечивается путем ограничения каждого вектора $\mathbf{x}_g^{(i)}$ и $\mathbf{y}_g^{(i)}$, чтобы обеспечить соответствие между переменными связи:

$$\mathbf{x}_g^{(i)} = \mathcal{A}_g\left[\mathbf{x}_g^{(i)}\right] \text{ и } \mathbf{y}_g^{(i)} = \mathcal{A}_g\left[\mathbf{y}_g^{(i)}\right], \tag{21.7}$$

где $\mathcal{A}_g\left[\mathbf{x}_g^{(i)}\right]$ и $\mathcal{A}_g\left[\mathbf{y}_g^{(i)}\right]$ — переменные связи, соответствующие глобальному проекту и отклику в *i*-й дисциплине. Это ограничение применяется с помощью целевой функции подзадачи:

$$f_i = \left\|\mathbf{x}_g^{(i)} - \mathcal{A}_g\left[\mathbf{x}_g^{(i)}\right]\right\|_2^2 + \left\|\mathbf{y}_g^{(i)} - \mathcal{A}_g\left[\mathbf{y}_g^{(i)}\right]\right\|_2^2. \tag{21.8}$$

Таким образом, каждая подзадача ищет возможные решения, которые минимально отклоняются от переменных связи.

Подзадачи управляются оптимизатором системного уровня, который отвечает за оптимизацию переменных связи $\mathcal{A}_g$ для минимизации целевой функции. Оценка экземпляра связующих переменных требует выполнения каждой дисциплинарной подзадачи, как правило, параллельно.

Дисциплинарные подзадачи могут отличаться от переменных связи в процессе оптимизации. Это несоответствие возникает, когда две или более дисциплин расходятся в отношении переменной или когда ограничения подзадачи препятствуют сопоставлению целевых значений, установленных оптимизатором системного уровня. Ограничение верхнего уровня, что $f_i = 0$ для каждой дисциплины, гарантирует, что в конечном итоге связь будет достигнута.

Основные преимущества совместной оптимизации обусловлены ее способностью выделять некоторые проектные переменные в дисциплинарных подзадачах. Совместная оптимизация легко применима к решению реальных междисциплинарных задач, так как каждая дисциплина, как правило, хорошо разделена и, сле-

довательно, в значительной степени не зависит от небольших решений, принимаемых в других дисциплинах. Децентрализованная формулировка позволяет применять традиционные методы оптимизации дисциплины, позволяя разработчикам задач использовать существующие инструменты и методологии.

Совместная оптимизация требует оптимизации по переменным связи, которые включают как проектные переменные, так и отклики. Совместная оптимизация неэффективна в задачах с сильной связью, потому что дополнительные переменные связи могут перевесить преимущества локальной оптимизации.

Совместная оптимизация — это *распределенная архитектура*, которая разбивает одну задачу на меньший набор задач оптимизации, которые имеют одинаковое решение при объединении их решений. Преимущество распределенных архитектур заключается в сокращении времени решения, поскольку подзадачи можно оптимизировать параллельно.

Совместная оптимизация применяется к проблеме разделения поездок в примере 21.6.

Пример 21.6. Применение совместной оптимизации к проблеме совместного использования

Архитектура совместной оптимизации может быть применена к задаче о маршрутизации транспортных средств путем создания шести различных дисциплинарных подзадач. К сожалению, наличие шести различных подзадач требует, чтобы любые переменные, общие для разных дисциплин, были оптимизированы на глобальном уровне.

На рис. 21.13 показаны две дисциплинарные подзадачи, полученные путем группировки дисциплины “транспортное средство”, “датчик” и “автономия” в подзадачу транспорта, а дисциплины маршрутизации, спроса и прибыли — в подзадачу сети. Дисциплины, сгруппированные в каждой подзадаче, тесно связаны между собой. Наличие только двух подзадач значительно сокращает число глобальных переменных, рассматриваемых оптимизатором системного уровня, поскольку предположительно очень мало проектных переменных непосредственно используются как транспортными, так и сетевыми подзадачами.

Каждая из подзадач — это междисциплинарные проблемы оптимизации, которые сами поддаются оптимизации с использованием методов, описанных в этой главе. Мы можем, например, использовать последовательную оптимизацию в транспортной подзадаче. Мы также можем добавить еще один пример совместной оптимизации в сетевой подзадаче.

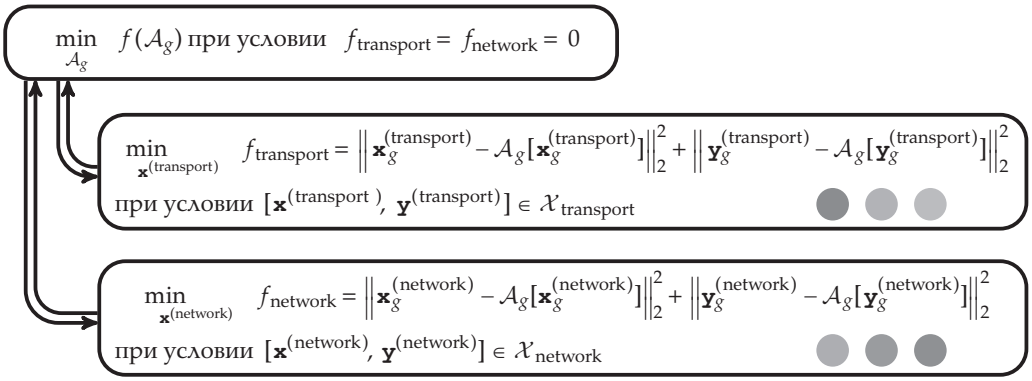


Рис. 21.13. Применение архитектуры совместной оптимизации к задаче рейдшеринга. Цветные круги соответствуют дисциплинарным анализам, содержащимся в каждой подзадаче

21.8. Архитектура одновременного анализа и проектирования

Архитектура одновременного анализа и проектирования (Simultaneous Analysis And Design — SAND) позволяет избежать главной проблемы координации между несколькими дисциплинарными анализами, поскольку анализ проводит оптимизатор. Вместо запуска анализа $F_i(\mathcal{A})$ для получения невязки архитектура SAND оптимизирует как проектные переменные, так и отклики с учетом ограничения $F_i(\mathcal{A}) = \mathcal{A}$. Оптимизатор отвечает за одновременную оптимизацию проектных переменных и поиск соответствующих откликов.

Любой дисциплинарный анализ может быть преобразован в форму невязок. Невязка $r_i(\mathcal{A})$ используется для указания того, является ли задание $\mathcal{A}$ совместимым с i-й дисциплиной. Если $F_i(\mathcal{A}) = \mathcal{A}$, то $r_i(\mathcal{A}) = 0$; в противном случае $r_i(\mathcal{A}) = 0$. Мы можем получить форму невязки, используя дисциплинарный анализ:

$$r_i(\mathcal{A}) = \left\| F_i(\mathcal{A}) - \mathcal{A}\left[\mathbf{y}^{(i)}\right] \right\|, \tag{21.9}$$

хотя это, как правило, неэффективно, как показано в примере 21.7.

Пример 21.7. Оценка дисциплинарного анализа в виде невязки, как правило, контрпродуктивна. Анализ обычно должен выполнять дополнительную работу для решения задачи, тогда как более строгая форма невязки может более эффективно проверять, совместимы ли входные данные

Рассмотрим дисциплинарный анализ, который решает уравнение $\mathbf{A}\mathbf{y} = \mathbf{x}$. Анализ $F(\mathbf{x}) = \mathbf{A}^{-1}\mathbf{x}$ требует дорогостоящего обращения матрицы. Мы можем построить остаточную форму, используя уравнение (21.9):

$$r_1(\mathbf{x},\mathbf{y}) = \left\| F(\mathbf{x}) - \mathbf{y} \right\| = \left\| \mathbf{A}^{-1}\mathbf{x} - \mathbf{y} \right\|.$$

В качестве альтернативы используем исходное ограничение для создания более эффективной формы невязки:

$$r_2(\mathbf{x},\mathbf{y}) = \left\| \mathbf{A}\mathbf{y} - \mathbf{x} \right\|.$$

Форма невязки для дисциплины состоит из набора дисциплинарных уравнений, которые решаются с помощью дисциплинарного анализа.[9] Зачастую намного проще оценить невязку, чем выполнить дисциплинарный анализ. В архитектуре SAND (рис. 21.14) анализ может выполнять оптимизатор.

$\min_{\mathcal{A}} f(\mathcal{A})$ при условии $\mathcal{A} \in \mathcal{X}$, $r_i(\mathcal{A}) = 0$ для каждой дисциплины

Рис. 21.14. Одновременный анализ и проектирование возлагает основную нагрузку на оптимизатор. Он использует дисциплинарную невязку, а не дисциплинарный анализ

Архитектура SAND может исследовать области проектного пространства, которые недопустимы по отношению к уравнениям невязок, как показано на рис. 21.15. Изучение недопустимых областей может позволить нам легче обходить пространство проектирования и находить решения в областях, не связанных с допустимой областью начальной точки проектирования. Архитектура SAND страдает от необходимости одновременно оптимизировать очень большое количество переменных, для которых недоступны производные и другие специальные знания по дисциплинам. Кроме того, преимущества архитектуры SAND обеспечиваются в основном за счет невязок, которые могут быть вычислены более эффективно, чем выполнение дисциплинарного анализа. Использование архитектуры SAND в реальных приложениях часто ограничивается неспособностью изменить существующий код дисциплинарного анализа для создания эффективной формы невязок.

Применение архитектуры SAND к решению задачи рейдшеринга показано в примере 21.8.

[9] В аэродинамике они могут включать уравнения Навье–Стокса, в структурной инженерии — уравнения упругости, а в электротехнике — дифференциальные уравнения, описывающие движение тока.

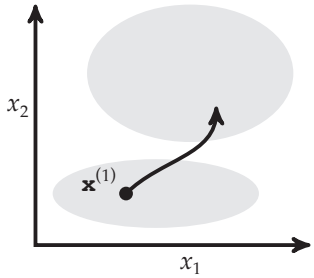


Рис. 21.15. Архитектура SAND может исследовать области пространства проектирования, которые недопустимы и потенциально могут устранить разрыв между возможными подмножествами

Пример 21.8. Применение одновременного анализа и архитектуры к задаче рейдшеринга

Применение одновременного анализа и проектирования к задаче рейдшеринга требует использования дисциплинарных невязок. Они могут потенциально зависеть от всех переменных проекта и откликов. Архитектура требует оптимизировать все переменные проекта и все отклики.

$$
\begin{aligned}
&\min_{\mathbf{v},\mathbf{s},\mathbf{r},\mathbf{p},\mathbf{y}^{(v)},\mathbf{y}^{(s)},\mathbf{y}^{(a)},\mathbf{y}^{(r)},\mathbf{y}^{(d)},\mathbf{y}^{(p)}} && f\left(\mathbf{v},\mathbf{s},\mathbf{r},\mathbf{p},\mathbf{y}^{(v)},\mathbf{y}^{(s)},\mathbf{y}^{(a)},\mathbf{y}^{(r)},\mathbf{y}^{(d)},\mathbf{y}^{(p)}\right)\\
&\text{при условиях} && \left[\mathbf{v},\mathbf{s},\mathbf{r},\mathbf{p},\mathbf{y}^{(v)},\mathbf{y}^{(s)},\mathbf{y}^{(a)},\mathbf{y}^{(r)},\mathbf{y}^{(d)},\mathbf{y}^{(p)}\right] \in \mathcal{X},\\
& && r_v(\mathbf{v},\mathbf{s},\mathbf{r},\mathbf{p},\mathbf{y}^{(v)},\mathbf{y}^{(s)},\mathbf{y}^{(a)},\mathbf{y}^{(r)},\mathbf{y}^{(d)},\mathbf{y}^{(p)}) = 0,\\
& && r_s(\mathbf{v},\mathbf{s},\mathbf{r},\mathbf{p},\mathbf{y}^{(v)},\mathbf{y}^{(s)},\mathbf{y}^{(a)},\mathbf{y}^{(r)},\mathbf{y}^{(d)},\mathbf{y}^{(p)}) = 0,\\
& && r_a(\mathbf{v},\mathbf{s},\mathbf{r},\mathbf{p},\mathbf{y}^{(v)},\mathbf{y}^{(s)},\mathbf{y}^{(a)},\mathbf{y}^{(r)},\mathbf{y}^{(d)},\mathbf{y}^{(p)}) = 0,\\
& && r_r(\mathbf{v},\mathbf{s},\mathbf{r},\mathbf{p},\mathbf{y}^{(v)},\mathbf{y}^{(s)},\mathbf{y}^{(a)},\mathbf{y}^{(r)},\mathbf{y}^{(d)},\mathbf{y}^{(p)}) = 0,\\
& && r_d(\mathbf{v},\mathbf{s},\mathbf{r},\mathbf{p},\mathbf{y}^{(v)},\mathbf{y}^{(s)},\mathbf{y}^{(a)},\mathbf{y}^{(r)},\mathbf{y}^{(d)},\mathbf{y}^{(p)}) = 0,\\
& && r_p(\mathbf{v},\mathbf{s},\mathbf{r},\mathbf{p},\mathbf{y}^{(v)},\mathbf{y}^{(s)},\mathbf{y}^{(a)},\mathbf{y}^{(r)},\mathbf{y}^{(d)},\mathbf{y}^{(p)}) = 0.
\end{aligned}
$$

21.9. Резюме

- Для оптимизации междисциплинарного проектирования требуется рассуждать о нескольких дисциплинах и достигать соглашения между связанными переменными.

- Дисциплинарный анализ можно упорядочить, чтобы минимизировать циклы зависимости.
- Проблемы междисциплинарного проектирования могут быть структурированы в разных архитектурах, которые используют преимущества особенностей задач для улучшения процесса оптимизации.
- Возможная архитектура междисциплинарного проектирования поддерживает допустимость и совместимость благодаря использованию медленного и потенциально расходящегося анализа междисциплинарного проектирования.
- Последовательная оптимизация позволяет каждой дисциплине оптимизировать свои специфичные для дисциплины переменные, но не всегда дает оптимальный проект.
- Возможная архитектура отдельной дисциплины позволяет выполнять параллельное выполнение анализа за счет добавления переменных связи в глобальный оптимизатор.
- Совместная оптимизация включает субоптимизаторы, которые могут использовать специализацию предметной области для локальной оптимизации некоторых переменных.
- Одновременный анализ и проектирование заменяют анализ проекта невязками, что позволяет оптимизатору находить совместимые решения, но не позволяет напрямую использовать дисциплинарные методы.

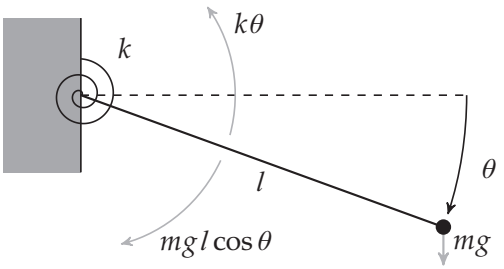
21.10. Упражнения

Упражнение 21.1. Приведите пример практической инженерной проблемы, которая является междисциплинарной.

Упражнение 21.2. Приведите абстрактный пример междисциплинарной проблемы, в которой важен порядок анализа.

Упражнение 21.3. Что является одним из преимуществ возможной архитектуры отдельной дисциплины по сравнению с междисциплинарным проектированием и последовательной архитектурой оптимизации?

Упражнение 21.4. Рассмотрите возможность применения междисциплинарного конструкторского анализа для минимизации веса крыла, деформация и нагрузка которого рассчитываются по отдельным дисциплинам. Мы будем использовать упрощенную версию задачи, представляя крыло как горизонтально установленный маятник, поддерживаемый торсионной пружиной.



Задача — минимизировать жесткость пружины k, чтобы смещение маятника не превышало целевой порог. Длина маятника l, точечная масса маятника m и гравитационная постоянная g фиксированы.

Мы используем два упрощенных анализа вместо более сложных анализов, используемых для расчета деформаций и нагрузок на крылья самолета. Предполагая, что маятник жесткий, момент нагрузки M равен $mgl\cos\theta$. Пружина кручения сопротивляется деформации, так что угловое смещение маятника θ равно M/k.

Сформулируйте задачу о пружинном маятнике в рамках допустимой архитектуры междисциплинарного проектирования, а затем решите ее в соответствии с этой архитектурой для следующих показателей:

$$m = 1 \text{ кг}, l = 1 \text{ м}, g = 9{,}81 \text{ мс}^{-2}, \theta_{\max} = 10 \text{ градусов}.$$

Упражнение 21.5. Сформулируйте задачу о пружинном маятнике в рамках допустимой архитектуры отдельной дисциплины.

Упражнение 21.6. Сформулируйте задачу о пружинном маятнике в рамках архитектуры совместной оптимизации. Поставьте две дисциплинарные задачи оптимизации и задачу оптимизации на уровне системы.

Приложение А

Язык Julia

Julia — научный язык программирования с открытым исходным кодом.[1] Это относительно новый язык, который черпает вдохновение в таких языках, как Python, MATLAB и R. Он был выбран для использования в данной книге, потому что имеет достаточно высокий уровень[2] и позволяет писать компактные, читабельные и быстрые алгоритмы. Коды, приведенные в книге, совместимы с версией Julia 1.0. В этом приложении вводятся необходимые понятия для их понимания.

A.1. Типы

В языке Julia есть множество основных типов, которые могут представлять данные, такие как значения истинности, числа, строки, массивы, кортежи и словари. Пользователи также могут определять свои собственные типы. В этом разделе объясняется, как использовать некоторые из основных типов и определять новые типы.

A.1.1. Логические типы

Логический тип в языке Julia называется `Bool` и включает значения `true` и `false`. Эти значения можно присваивать переменным. Имена переменных могут быть любой строкой символов, включая строки в кодировке Unicode, с некоторыми ограничениями.

```
done = false
α = false
```

Левая часть знака равенства — это имя переменной, а правая часть — значение.

Можно выполнять присваивания на консоли языка Julia. Консоль возвращает значение вычисляемого выражения.

```
julia> x = true
true
julia> y = false
```

[1] Компилятор языка Julia можно найти по адресу **http://julialang.org.**

[2] В отличие от таких языков, как C++, Julia не требует от программистов заботы об управлении памятью и других деталях более низкого уровня.

```
false
julia> typeof(x)
Bool
```

В языке поддерживаются стандартные логические операторы.

```
julia> !x      # нет
false
julia> x && y # и
false
julia> x || y # или
true
```

Символ # указывает, что остальная часть строки является комментарием и не должна вычисляться.

A.1.2. Числа

Язык Julia поддерживает целые числа и числа с плавающей точкой, как показано ниже.

```
julia> typeof(42)
Int64
julia> typeof(42.0)
Float64
```

Здесь `Int64` обозначает 64-разрядное целое число, а `Float64` — 64-разрядное значение с плавающей точкой.[3] Мы также можем выполнять стандартные математические операции:

```
julia> x = 4
4
julia> y = 2
2
julia> x + y
6
julia> x - y
2
julia> x * y
8
julia> x / y
2.0
```

[3] На 32-битных машинах целочисленный литерал, такой как 42, интерпретируется как `Int32`.

```
julia> x ^ y
16
julia> x % y # остаток от деления x на y
0
```

Обратите внимание: результатом выражения `x / y` становится тип `Float64`, даже если `x` и `y` являются целыми числами. Мы также можем выполнять эти операции одновременно с присваиванием. Например, `x += 1` является сокращенным вариантом выражения `x = x + 1`.

Можно выполнять такие сравнения:

```
julia> 3 > 4
false
julia> 3 >= 4
false
julia> 3 ≥ 4     # кодировка unicode допускается
false
julia> 3 < 4
true
julia> 3 <= 4
true
julia> 3 ≤ 4     # кодировка unicode допускается
true
julia> 3 == 4
false
julia> 3 < 4 < 5
true
```

A.1.3. Строки

Строка — это массив символов. Строки не очень часто используются в этом учебнике, за исключением сообщения об определенных ошибках. Объект типа `String` может быть создан с помощью символов `"`. Например:

```
julia> x = "optimal"
"optimal"
julia> typeof(x)
String
```

A.1.4. Векторы

Вектор — это одномерный массив, в котором хранится последовательность значений. Можно создать вектор, используя квадратные скобки, разделяя элементы запятыми. Точки с запятой в этих примерах подавляют вывод.

```
julia> x = [];                # пустой вектор
julia> x = trues(3);          # булев вектор, содержащий три значения true
julia> x = ones(3);           # вектор из трех единиц
julia> x = zeros(3);          # вектор из трех нулей
julia> x = rand(3);           # вектор из трех случайных  чисел от 0 до 1
julia> x = [3, 1, 4];         # вектор целых чисел
julia> x = [3.1415, 1.618, 2.7182];  # вектор действительных чисел
```

Для создания векторов можно использовать *генератор массивов*. Ниже мы используем функцию печати, чтобы результаты выводились горизонтально.

```
julia> print([sin(x) for x = 1 : 5])
[0.841471, 0.909297, 0.14112, -0.756802, -0.958924]
```

Можно проверять тип векторов:

```
julia> typeof([3, 1, 4])                # одномерный массив типа Int64
Array{Int64, 1}
julia> typeof([3.1415, 1.618, 2.7182])  # одномерный массив типа Float64
Array{Float64, 1}
```

Векторы индексируются с помощью квадратных скобок.

```
julia> x[1]       # индекс первого элемента равен 1
3.1415
julia> x[3]       # третий элемент
2.7182
julia> x[end]     # индекс последнего элемента массива равен end
2.7182
julia> x[end -1]  # возвращает предпоследний элемент
1.618
```

Можно вывести на печать ряд элементов из массива. Диапазоны указываются с использованием двоеточия.

```
julia> x = [1, 1, 2, 3, 5, 8, 13];
julia> print(x[1 : 3])          # выводим первые три элемента
[1, 1, 2]
julia> print(x[1 : 2 : end])    # выводим каждый второй элемент
[1, 2, 5, 13]
julia> print(x[end : -1 : 1])   # выводим все элементы в обратном порядке
[13, 8, 5, 3, 2, 1, 1]
```

Можно выполнять различные операции с массивами. Восклицательный знак в конце имен функций часто используется для указания того, что функция изменяет входные данные.

```
julia> print([x, x])                # конкатенация
Array{Int64,1}[[1, 1, 2, 3, 5, 8, 13], [1, 1, 2, 3, 5, 8, 13]]
julia> length(x)
7
julia> print(push!(x, -1))          # добавляем элемент в конец массива
[1, 1, 2, 3, 5, 8, 13, -1]
julia> pop!(x)                      # удаляем элемент из конца массива
-1
julia> print(append!(x, [2, 3]))    # добавляем массив в конец массива x
[1, 1, 2, 3, 5, 8, 13, 2, 3]
julia> print(sort!(x))              # сортируем элементы вектора
[1, 1, 2, 2, 3, 3, 5, 8, 13]
julia> x[1] = 2; print(x)           # изменяем первый элемент на 2
[2, 1, 2, 2, 3, 3, 5, 8, 13]
julia> x = [1, 2];
julia> y = [3, 4];
julia> print(x + y)                 # добавляем векторы
[4, 6]
julia> print(3x - [1, 2])           # умножаем на скаляр и вычитаем
[2, 4]
julia> print(dot(x, y))             # скалярное произведение
11
julia> print(x ⋅ y)                 # скалярное произведение в Unicode
11
```

Часто удобно применить функцию ко всем элементам массива одновременно

```
julia> print(x .* y)        # поэлементное умножение
[3, 8]
julia> print(x .^ 2)        # поэлементное возведение в квадрат
[1, 4]
julia> print(sin.(x))       # поэлементное вычисление синуса
[0.841471, 0.909297]
julia> print(sqrt.(x))      # поэлементное извлечение квадратного корня
[1.0, 1.41421]
```

А.1.5. Матрицы

Матрица — это двумерный массив. Как и вектор, она создается с помощью квадратных скобок. Мы используем пробелы для разделения элементов в одной строке и точки с запятой для разделения строк. Мы также можем индексировать матрицу и выводить подматрицы, используя диапазоны.

```
julia> X = [1 2 3; 4 5 6; 7 8 9; 10 11 12];
julia> typeof(X)  # двумерный массив типа Int64
Array{Int64,2}
julia> X[2]       # доступ ко второму элементу в порядке обхода по столбцам
4
julia> X[3,2]     # элемент в третьей строке и втором столбце
8
julia> print(X[1, :])         # извлекаем первую строку
[1, 2, 3]
julia> print(X[:, 2])         # извлекаем второй столбец
[2, 5, 8, 11]
julia> print(X[:, 1 : 2])     # извлекаем два первых столбца
[1 2; 4 5; 7 8; 10 11]
julia> print(X[1 : 2, 1 : 2]) # извлекаем матрицу размером 2x2
[1 2; 4 5]                    # из левого верхнего угла матрицы X
```

Мы также можем создать множество специальных матриц и использовать генератор массивов.

```
julia> print(eye(3))              # единичная матрица 3x3
[1.0 0.0 0.0; 0.0 1.0 0.0; 0.0 0.0 1.0]
julia> print(diagm([3, 2, 1]))    # диагональная матрица 3x3
[3 0 0; 0 2 0; 0 0 1]             # числами 3, 2, 1 на диагонали

julia> print(rand(3,2))           # случайная матрица 3x2
[0.921459 0.307439; 0.255633 0.386419; 0.0664965 0.495675]
julia> print(zeros(3,2))          # матрица нулей 3x2
[0.0 0.0; 0.0 0.0; 0.0 0.0]
julia> print([sin(x + y) for x = 1 : 3, y = 1 : 2])  # заполнение массива
[0.909297 0.14112; 0.14112 -0.756802; -0.756802 -0.958924]
```

В языке Julia есть операторы для работы с матрицами.

```
julia> print(X')                 # комплексное сопряженное транспонирование
[1 4 7 10; 2 5 8 11; 3 6 9 12]
julia> print(3X + 2)             # умножение на скаляр и добавление скаляра
[5 8 11; 14 17 20; 23 26 29; 32 35 38]
julia> X = [1 3; 3 1];           # создание обратимой матрицы
julia> print(inv(X))             # обращение
[-0.125 0.375; 0.375 -0.125]
julia> det(X)                    # определитель
-8.0
```

```
julia> print([X X])              # горизонтальная конкатенация
[1 3 1 3; 3 1 3 1]
julia> print([X; X])             # вертикальная конкатенация
[1 3; 3 1; 1 3; 3 1]
julia> print(sin.(X))            # поэлементное применение функции sin
[0.841471 0.14112; 0.14112 0.841471]
```

A.1.6. Кортежи

Кортеж — это упорядоченный список значений, потенциально разных типов. Они создаются с помощью круглых скобок. Кортежи похожи на массивы, но не могут быть изменены.

```
julia> x = (1,) # кортеж из одного элемента, отмеченного замыкающей запятой
(1,)
julia> x = (1, 0, [1, 2], 2.5029, 4.6692) # третий элемент — вектор
(1, 0, [1, 2], 2.5029, 4.6692)
julia> x[2]
0
julia> x[end]
4.6692
julia> x[4 : end]
(2.5029, 4.6692)
julia> length(x)
5
```

A.1.7. Словари

Словарь представляет собой набор пар ключ-значение. Пары ключ-значение указываются с помощью оператора двойной стрелки. Можно индексировать словари, используя квадратные скобки как массивы и кортежи.

```
julia> x = Dict();               # пустой словарь
julia> x[3] = 4                  # ассоциация между значением 4 и ключом 3
4
julia> x = Dict(3 => 4, 5 => 1) # словарь с двумя парами ключ-значение
Dict{Int64,Int64} with 2 entries:
  3 => 4
  5 => 1
julia> x[5]          # возвращаем значение, ассоциированное с ключом 5
1
julia> haskey(x, 3) # проверяем, содержит ли словарь ключ 3
true
```

```
julia> haskey(x, 4) # проверяем, содержит ли словарь ключ 4
false
```

А.1.8. Составные типы

Составной тип — это коллекция именованных полей. По умолчанию экземпляр составного типа является неизменным. Мы используем ключевое слово `struct`, а затем присваиваем новому типу имя и перечисляем имена полей.

```
struct A
    a
    b
end
```

Добавление ключевого слова `mutable` позволяет экземпляру измениться.

```
mutable struct B
    a
    b
end
```

Составные типы создаются с помощью скобок, между которыми мы передаем значения для различных полей. Например,

```
x = A(1.414, 1.732)
```

Для аннотирования типов полей можно использовать оператор с двойным двоеточием.

```
struct A
   a :: Int64
   b :: Float64
end
```

Эта аннотация требует, чтобы мы передали числа типа `Int64` для первого поля и `Float64` для второго. Для компактности мы не использовали аннотации типов, но это снижает производительность кода. Аннотации типов позволяют языку Julia повысить производительность во время выполнения, поскольку компилятор может оптимизировать базовый код для конкретных типов.

А.1.9. Абстрактные типы

До сих пор мы обсуждали конкретные типы, которые можно построить. Однако конкретные типы являются только частью иерархии типов. Существуют также абстрактные типы, которые являются супертипами конкретных типов и других абстрактных типов.

Можно исследовать иерархию типов типа `Float64`, показанную на рис. A.1, используя функции супертипа и подтипа.

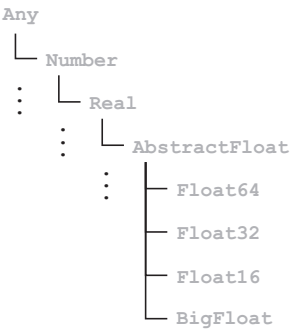


Рис. A.1. Иерархия типов для типа `Float64`

```
julia> supertype(Float64)
AbstractFloat
julia> supertype(AbstractFloat)
Real
julia> supertype(Real)
Number
julia> supertype(Number)
Any
julia> supertype(Any)          # Any — вершина иерархии
Any
julia> subtypes(AbstractFloat) # Разные типы AbstractFloats
4-element Array{Union{DataType, UnionAll}, 1}:
 BigFloat
 Float16
 Float32
 Float64
julia> subtypes(Float64)       # Тип Float64 не имеет подтипов
0-element Array{Union{DataType, UnionAll}, 1}
```

Можно определять свои собственные абстрактные типы.

```
abstract type C end
abstract type D <: C end       # D — абстрактный подтип типа C
struct E <: D                  # E — составной тип — подтип типа D
    a
end
```

А.1.10. Параметрические типы

Язык Julia поддерживает параметрические типы, которые принимают параметры. Мы уже видели параметрический тип в нашем примере словаря.

```
julia> x = Dict(3 => 4, 5 => 1)
Dict{Int64,Int64} with 2 entries:
  3 => 4
  5 => 1
```

Этот оператор создает словарь `Dict{Int64, Int64}`. Параметры параметрического типа перечисляются в фигурных скобках и разделяются запятыми. Для типа словаря первый параметр указывает тип ключа, а второй — тип значения. В языке Julia можно делать вывод, основываясь на вводе, но мы могли бы указать это явно.

```
julia> x = Dict{Int64,Int64}(3 => 4, 5 => 1)
Dict{Int64,Int64} with 2 entries:
   3 => 4
   5 => 1
```

Можно определить и свои собственные параметрические типы, но в книге мы этого не делаем.

А.2. Функции

Функция — это объект, который отображает кортеж значений аргументов в возвращаемое значение. В этом разделе обсуждается, как определить и работать с функциями.

А.2.1. Именованные функции

Одним из способов определения именованной функции является использование ключевого слова `function`, за которым следует имя функции и кортеж имен аргументов.

```
function f(x, y)
   return x + y
end
```

Функции также можно объявлять компактно, используя оператор присваивания.

```
julia> f(x, y) = x + y;
julia> f(3, 0.1415)
3.1415
```

А.2.2. Анонимные функции

Анонимной функции не присваивается имя, хотя она может быть назначена именованной переменной. Одним из способов определения анонимной функции является использование оператора стрелки.

```
julia> h = x -> x ^ 2 + 1 # присваиваем анонимную функцию  переменной
(:: #1) (generic function with 1 method)
julia> g(f, a, b) = [f(a), f(b)];  # применяем функцию f к параметрам
                                   # a и b и возвращаем массив array
julia> g(h, 5, 10)
2-element Array{Int64, 1}:
   26
  101
julia> g(x -> sin(x) + 1, 10, 20)
2-element Array{Float64, 1}:
  0.455979
  1.91295
```

А.2.3. Необязательные аргументы

Можно задавать необязательные аргументы, указывая их значения по умолчанию.

```
julia> f(x = 10) = x ^ 2;
julia> f()
100
julia> f(3)
9
julia> f(x, y, z = 1) = x * y + z;
julia> f(1, 2, 3)
5
julia> f(1, 2)
3
```

А.2.4. Именованные аргументы

Функции с именованными аргументами определяются с помощью точки с запятой.

```
julia> f(; x = 0) = x + 1;
julia> f()
1
julia> f(x = 10)
11
```

```
julia> f(x, y = 10; z = 2) = (x + y) * z;
julia> f(1)
22
julia> f(2, z = 3)
36
julia> f(2, 3)
10
julia> f(2, 3, z = 1)
5
```

А.2.5. Перегрузка функций

Типы аргументов, передаваемых в функцию, можно указывать с помощью оператора двойного двоеточия. Если предусмотрено несколько функций с одним и тем же именем, компилятор языка Julia выполнит соответствующую функцию.

```
julia> f(x :: Int64) = x + 10;
julia> f(x :: Float64) = x + 3.1415;
julia> f(1)
11
julia> f(1.0)
4.141500000000001
julia> f(1.3)
4.4415000000000004
```

Будет использована реализация наиболее точно заданной функции.

```
julia> f(x) = 5;
julia> f(x :: Float64) = 3.1415;
julia> f([3, 2, 1])
5
julia> f(0.00787499699)
3.1415
```

А.3. Управление потоком выполнения

Потоком выполнения программ можно управлять, применяя условные операторы и циклы. В этом разделе представлен синтаксис, использованный в книге.

А.3.1. Условная оценка

Условный оператор проверит значение логического выражения, а затем вычислит соответствующий блок кода. Один из наиболее распространенных способов сделать это — применить оператор `if`.

```
if x < y
   # выполняется, если x < y
elseif x > y
   # выполняется, если x > y
else
   # выполняется, если x == y
end
```

Мы также можем использовать *тернарный оператор* с вопросительным знаком и двоеточием. Он проверяет логическое выражение, стоящее перед знаком вопроса. Если значение выражение истинно, то оператор возвращает то, что стоит перед двоеточием; в противном случае возвращается то, что стоит после двоеточия.

```
julia> f(x) = x > 0 ? x : 0;
julia> f(-10)
0
julia> f(10)
10
```

А.3.2. Циклы

Цикл допускает повторную оценку выражений. Один тип цикла — цикл `while`. Он повторно оценивает блок выражений до тех пор, пока не будет выполнено указанное условие. В следующем примере будут суммироваться значения в массиве **x**.

```
x = [1, 2, 3, 4, 6, 8, 11, 13, 16, 18]
s = 0
while x != []
   s += pop!(x)
end
```

Другой тип цикла — это цикл `for`. Следующий пример также суммирует значения в массиве **x**, но не изменяет **x**.

```
x = [1, 2, 3, 4, 6, 8, 11, 13, 16, 18]
s = 0
```

```
for i = 1 : length(x)
   s += x[i]
end
```

Знак = можно заменить ключевым словом in или символом ∈. Следующий код эквивалентен предыдущему.

```
x = [1, 2, 3, 4, 6, 8, 11, 13, 16, 18]
s = 0
for y in x
   s += y
end
```

А.4. Пакеты

В языке Julia существует встроенный менеджер пакетов. Список зарегистрированных пакетов можно найти по адресу https://pkg.julialang.org. Для того чтобы добавить зарегистрированный пакет, например **Distributions.jl**, можно выполнить команду

```
Pkg.add ("Distributions")
```

Для обновления пакетов применяется команда

```
Pkg.update()
```

Для использования пакета указывается ключевое слово using:

```
using Distributions
```

Некоторые блоки кода в этой книге определяют импорт пакета с помощью ключевого слова using. Другие блоки кода используют функции, которые не импортированы явно. Например, функция **var** предоставляется пакетом **Statistics.jl**, а отношение золотого сечения φ определено в пакете **Base.MathConstants.jl**. Такими пакетами, не указанными явно, являются **Base.MathConstants.jl**, **Iterators.jl**, **LinearAlgebra.jl**, **QuadGK.jl**, **Random.jl**, **Statistics.jl** и **StatsBase.jl**.

Приложение Б

Тестовые функции

Исследователи в области оптимизации применяют несколько тестовых функций для оценки алгоритмов оптимизации. Ниже описаны тестовые функции, используемые в этой книге.

Б.1. Функция Экли

Функция Экли (рис. Б.1) используется для проверки склонности метода к застреванию в локальных минимумах. Она состоит из двух основных частей — синусоидального компонента, который создает множество локальных минимумов, и экспоненциальной кривой в виде колокола с центром в начале координат, определяющей глобальный минимум функции.

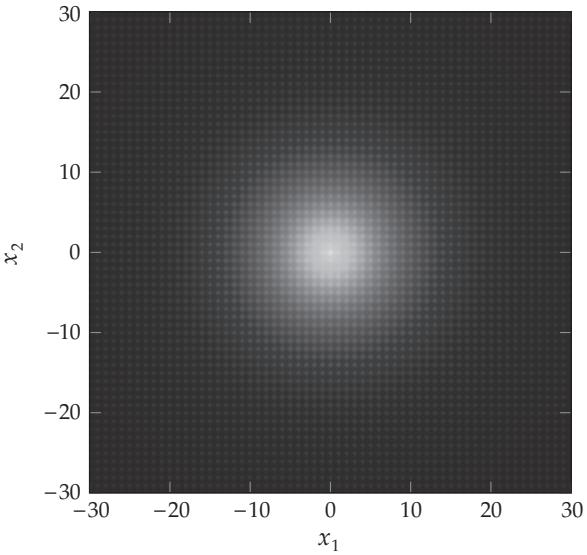


Рис. Б.1. Двумерная версия функции Экли.
Глобальный минимум находится в начале координат

Функция Экли определена для любого числа измерений d:

$$f(\mathbf{x}) = -a\exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d}x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos cx_i\right) + a + \exp(1) \qquad \text{(Б.1)}$$

с глобальным минимумом в начале координат и оптимальным значением, равным нулю. Как правило, $a = 20$, $b = 0{,}2$ и $c = 2\pi$. Функция Экли реализована в алгоритме Б.1.

Алгоритм Б.1. Функция Экли с d-мерным входным вектором **x** и тремя необязательными параметрами

```
function ackley(x, a = 20, b = 0.2, c = 2π)
   d = length(x)
   return -a * exp(-b * sqrt(sum(xi ^ 2) / d)) -
          exp(sum(cos.(c * xi) for xi in x) / d) + a + exp(1)
end
```

Б.2. Функция Бута

Функция Бута (Booth's function) (рис. Б.2) — это двумерная квадратичная функция.

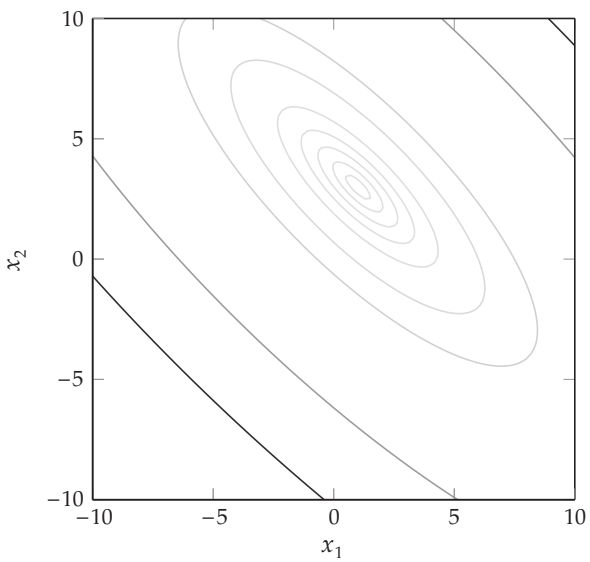


Рис. Б.2. Функция Бута с глобальным минимумом в [1, 3]

Ее вид задается формулой

$$f(\mathbf{x}) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2 \qquad (Б.2)$$

с глобальным минимумом в точке [1, 3] и оптимальным значением, равным нулю. Эта функция реализована в алгоритме Б.2.

Алгоритм Б.2. Функция Бута с двумерным входным вектором **x**

```
booth(x) = (x[1] + 2x[2] - 7) ^ 2 + (2x[1] + x[2] - 5) ^ 2
```

Б.3. Функция Бранина

Функция Бранина (Branin function) (рис. Б.3) является двумерной функцией:

$$f(\mathbf{x}) = a\left(x_2 - bx_1^2 + cx_1 - r\right)^2 + s(1-t)\cos x_1 + s \qquad \text{(Б.3)}$$

с рекомендуемыми значениями $a = 1$, $b = 5{,}1/(4\pi^2)$, $c = 5/\pi$, $r = 6$, $s = 10$ и $t = 1/(8\pi)$.

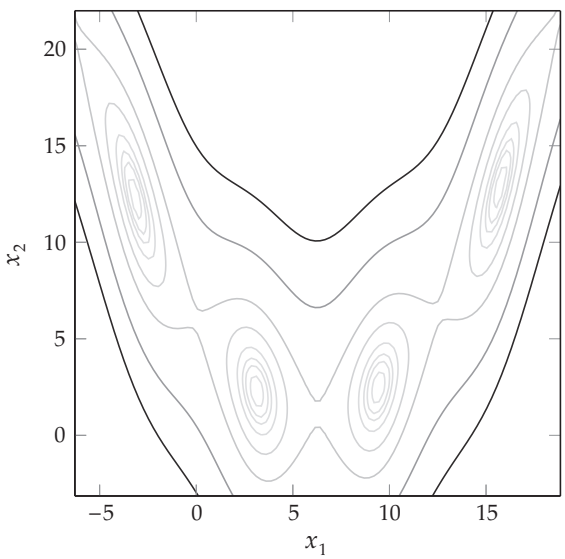


Рис. Б.3. Функция Бранина с четырьмя глобальными минимумами

Эта функция не имеет локальных минимумов, кроме глобальных минимумов в точках с координатами $x_1 = \pi + 2\pi m$, где m — целое число. Четыре из этих минимумов имеют вид:

$$\left\{\begin{bmatrix} -\pi \\ 12{,}275 \end{bmatrix}, \begin{bmatrix} \pi \\ 2{,}275 \end{bmatrix}, \begin{bmatrix} 3\pi \\ 2{,}475 \end{bmatrix}, \begin{bmatrix} 5\pi \\ 12{,}875 \end{bmatrix}\right\} \qquad \text{(Б.4)}$$

с $f(\mathbf{x}^*) \approx 0{,}397887$. Это реализовано в алгоритме Б.3.

Алгоритм Б.3. Функция Бранина с двумерным аргументом **x** и шестью необязательными параметрами

```
function branin(x; a = 1, b = 5.1 / (4π ^ 2), c = 5 /π, r = 6, s = 10,
                t = 1 /(8π))
```

```
    return a * (x[2] - b * x[1] ^ 2 + c * x[1] - r) ^ 2 + s *
                (1 - t) * cos(x[1]) + s
end
```

Б.4. Функция цветка

Функция цветка (рис. Б.4) — это двумерная тестовая функция, изолинии которой выглядят как лепестки цветка с центром в начале координат

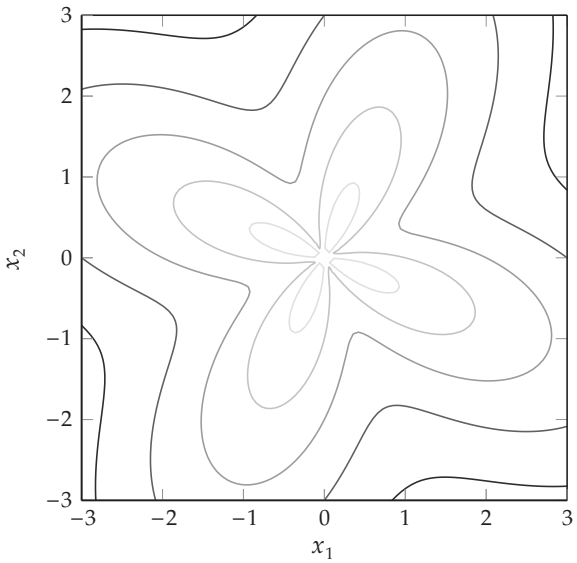


Рис. Б.4. Функция цветка

Уравнение функции цветка имеет вид

$$f(\mathbf{x}) = a\|\mathbf{x}\| + b\sin\left(c\,\mathrm{tg}^{-1}(x_2, x_1)\right) \tag{Б.5}$$

с параметрами, как правило, равными $a = 1$, $b = 1$ и $c = 4$.

Функция цветка достигает минимума к окрестности начала координат, но не имеет глобального минимума из-за неопределенности тангенса в точке [0, 0]. Эта функция реализована в алгоритме Б.4.

Алгоритм Б.4. Функция цветка с двумерным аргументом **x** и тремя необязательными параметрами

```
function flower(x; a = 1, b = 1, c = 4)
    return a * norm(x) + b * sin(c * atan(x[2], x[1]))
end
```

Б.5. Функция Михалевича

Функция Михалевича (Michalewicz function) (рис. Б.5) является d-мерной функцией с несколькими крутыми оврагами.

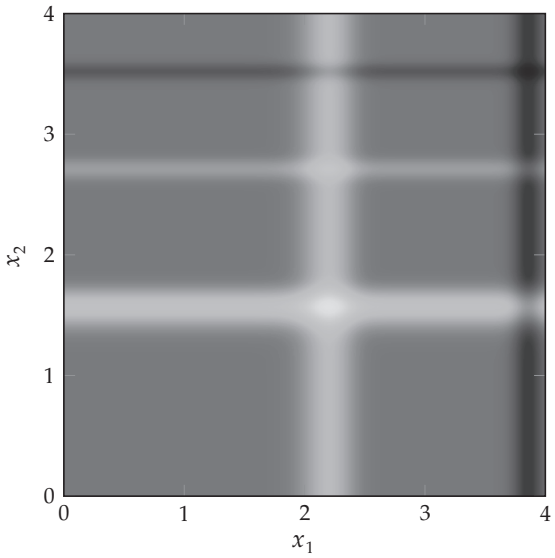


Рис. Б.5. Функция Михалевича

Функция Михалевича имеет вид

$$f(\mathbf{x}) = -\sum_{i=1}^{d} \sin x_i \sin^{2m} \frac{i x_i^2}{\pi}, \tag{Б.6}$$

где параметр m, обычно равный 10, контролирует крутизну функции. Глобальный минимум зависит от количества измерений. В двумерном пространстве точка минимума приближенно равна [2,20, 1,57] с $f(\mathbf{x}^*) = -1{,}8011$. Эта функция реализована в алгоритме Б.5.

Алгоритм Б.5. Функция Михалевича с входным вектором **x** и тремя необязательными параметрами крутизны **m**

```
function michalewicz(x; m = 10)
   return -sum(sin(v) * sin(i * v ^ 2 / π) ^ (2m) for
           (i, v) in enumerate(x))
end
```

Б.6. Банановая функция Розенброка

Функция Розенброка (Rosenbrock's function) (рис. Б.6), также называемая долиной Розенброка или банановой функцией Розенброка, является хорошо известной тестовой функцией без ограничений, изобретенной Розенброком в 1960 г. [132]. Она имеет глобальный минимум внутри длинной изогнутой долины. Большинство алгоритмов оптимизации не испытывают проблем при поиске долины. Трудности возникают с поиском наибольшего или наименьшего значения функции, проходящим вдоль долины к глобальному минимуму.

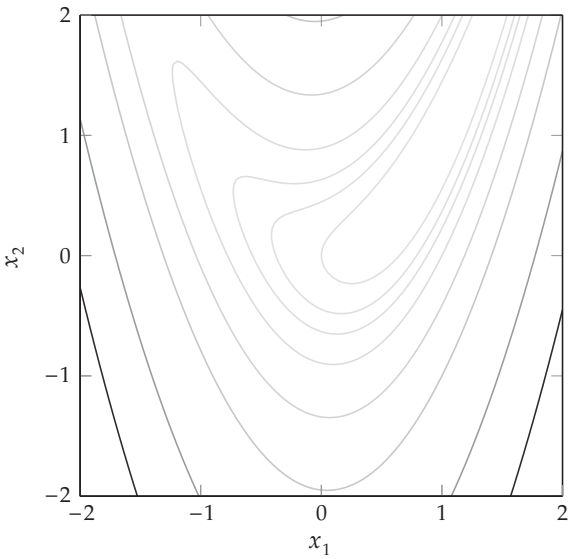


Рис. Б.6. Функция Розенброка с $a = 1$ и $b = 5$. Глобальный минимум равен [1, 1]

Функция Розенброка имеет вид

$$f(\mathbf{x}) = (a - x_1)^2 + b\left(x_2 - x_1^2\right)^2 \tag{Б.7}$$

с глобальным минимумом в точке $[a, a^2]$, при которой $f(\mathbf{x}^*) = 0$. В этой книге мы используем параметры $a = 1$ и $b = 5$. Функция Розенброка реализована в алгоритме Б.6.

Алгоритм Б.6. Функция Розенброка с двумерным входным вектором **x** и двумя необязательными параметрами

```
rosenbrock(x; a = 1, b = 5) = (a - x[1]) ^ 2 + b * (x[2] - x[1] ^ 2) ^ 2
```

Б.7. Гребень Уилера

Гребень Уилера (Wheeler's ridge) (рис. Б.7) — это двумерная функция с единственным глобальным минимумом на сильно изогнутом пике. Функция имеет два гребня, один вдоль положительной и один вдоль отрицательной первой координатной оси. Метод градиентного спуска будет расходиться по гребню отрицательной оси. Вдали от оптимальной точки и гребня функция становится плоской.

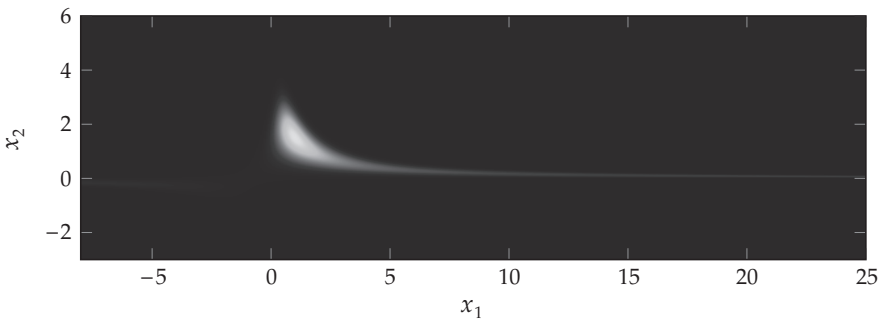


Рис. Б.7. Гребень Уилера с двумя гребнями и пиком, содержащим глобальный минимум

Функция Уилера имеет вид

$$f(\mathbf{x}) = -\exp\left((-(x_1x_2 - a)^2 - (x_2 - a)^2\right) \tag{Б.8}$$

с параметром a, как правило, равным 1,5, при котором точка глобального минимума равна $[1, 3/2]$.

Гребень Уилера имеет гладкие изолинии (рис. Б.8) в области $x_1 \in [0, 3]$, $x_2 \in [0, 3]$. Ее реализация приведена в алгоритме Б.7.

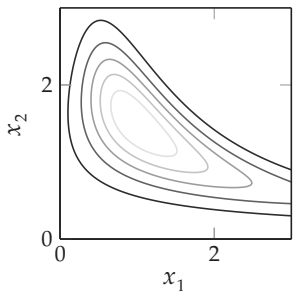


Рис. Б.8. Изолинии гребня Уилера в окрестности минимума

Алгоритм Б.7. Функция Уилера, которая получает двумерный вектор **x** и необязательный скалярный параметр **a**

```
wheeler(x, a = 1.5) = -exp(-(x[1] * a[2] - a) ^ 2 - (x[2] - a) ^ 2)
```

Б.8. Функция круга

Функция круга (circle function) (алгоритм Б.8) является простой многоцелевой тестовой функцией, заданной формулой

$$f(\mathbf{x}) = \begin{bmatrix} 1 - r\cos\theta \\ 1 - r\sin\theta \end{bmatrix}, \tag{Б.9}$$

где $\theta = x_1$ и r вычисляется по координате x_2 по формуле

$$r = \frac{1}{2} + \frac{1}{2}\left(\frac{2x_2}{1+x_2^2}\right). \tag{Б.10}$$

Граница Парето имеет параметры $r = 1$ и $\mathrm{mod}(\theta, 2\pi) \in [0, \pi/2]$ или $r = -1$ и $\mathrm{mod}(\theta, 2\pi) \in [\pi, 3\pi/2]$.

Алгоритм Б.8. Функция круга, которая получает двумерный вектор **x** и вычисляет двумерное значение целевой функции

```
function circle(x)
   θ = x[1]
   r = 0.5 + 0.5 * (2x[2] / (1 + x[2] ^ 2))
   y1 = 1 - r * cos(θ)
   y2 = 1 - r * sin(θ)
   return [y1, y2]
end
```

Приложение B

Математические понятия

В этом приложении рассматриваются математические понятия, используемые при выводе и анализе методов оптимизации. Эти понятия применяются во всей книге.

B.1. Асимптотические обозначения

Асимптотические обозначения часто используются для характеристики роста функции. Это обозначение иногда называют обозначением "большого O", поскольку буква O используется для описания скорости роста функции. Эти обозначения могут применяться для описания ошибок, связанных с численными методами, а также временной или пространственной сложности алгоритма. Эта система обозначений позволяет указать верхнюю границу для функции, когда ее аргумент стремится к определенному значению.

Формально, если $f(x) = O\big(g(x)\big)$ при $x \to a$, то абсолютное значение $f(x)$ ограничено абсолютным значением $g(x)$, умноженным на некоторое положительное и конечное число c для значений x, достаточно близких к a:

$$|f(x)| \leq c|g(x)| \text{ при } x \to a. \tag{B.1}$$

Часто запись $f(x) = O(g(x))$ интерпретируют неправильно. Например, $x^2 = O\big(x^2\big)$ и $2x^2 = O\big(x^2\big)$, но, конечно, $x^2 \neq 2x^2$. В некоторых математических текстах $O\big(g(x)\big)$ представляет собой множество всех функций, которые не растут быстрее, чем $g(x)$. Можно написать, например, $5x^2 \in O\big(x^2\big)$.

Если $f(x)$ является линейной комбинацией[1] слагаемых, то $O\big(f\big)$ соответствует порядку наиболее быстро растущего слагаемого. В примере B.2 сравниваются порядки нескольких слагаемых.

---

[1] Линейная комбинация — это взвешенная сумма слагаемых. Если слагаемые являются членами вектора $\mathbf{x}$, то линейная комбинация имеет вид $w_1x_1 + w_2x_2 + ... + w_nx_n = \mathbf{w}^T\mathbf{x}$.

Пример В.1. Асимптотические обозначения для функций, умноженных на константу

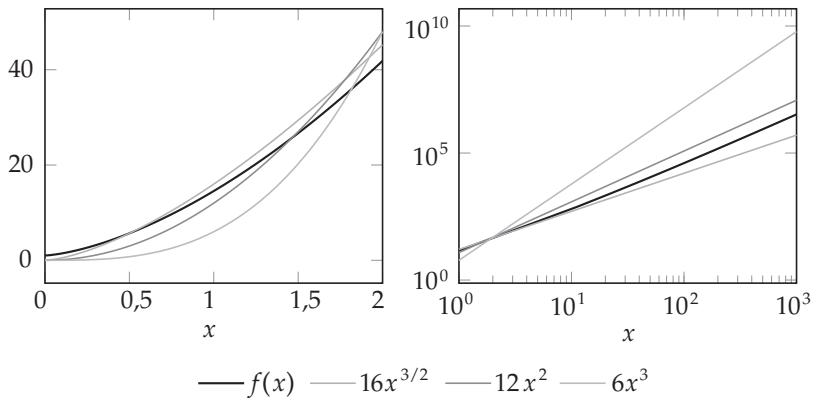
Рассмотрим функцию $f(x) = 10^6 e^x$ при $x \to \infty$. Здесь функция f является произведением константы 10^6 и функции e^x. Константа 10^6 просто поглощается константой c:

$$|f(x)| \le c|g(x)|,$$
$$10^6|e^x| \le c|g(x)|,$$
$$|e^x| \le c|g(x)|.$$

Следовательно, $f = O(e^x)$ при $x \to \infty$.

Пример В.2. Иллюстрация нахождения порядка линейной комбинации слагаемых

Рассмотрим функцию $f(x) = \cos x + x + 10x^{3/2} + 3x^2$. Здесь f — линейная комбинация слагаемых. Члены $\cos x$, x, $x^{3/2}$ и x^2 указаны в порядке возрастания значений, когда x стремится к бесконечности. Построим графики функций $f(x)$ и $c|g(x)|$, где константа c для каждого слагаемого выбрана так, чтобы значение $c|g(x = 2)|$ превосходило $f(x = 2)$.



Не существует такой константы c, что $f(x)$ всегда меньше, чем $c|x^{3/2}|$, для достаточно больших значений x. То же самое верно для функций $\cos x$ и x.

Итак, $f(x) = O(x^3)$ и вообще $f(x) = O(x^m)$ для $m \ge 2$, наряду с другими классами функций, такими как $f(x) = e^x$. Обычно мы указываем порядок, который обеспечивает наименьшую среди верхних границ. Таким образом, $f = O(x^2)$ при $x \to \infty$.

B.2. Разложение Тейлора

Разложение Тейлора, или *ряд Тейлора*, для функции имеет решающее значение при анализе многих методов оптимизации, описанных в этой книге, поэтому мы выведем ее здесь.

Из *основной формулы интегрального исчисления*[2] следует, что

$$f(x+h) = f(x) + \int_0^h f'(x+a)\,da. \tag{B.2}$$

Использование этого определения приводит к разложению Тейлора для функции f относительно x:

$$f(x+h) = f(x) + \int_0^h \left(f'(x) + \int_0^a f''(x+b)\,db \right) da = \tag{B.3}$$

$$= f(x) + f'(x)h + \int_0^h \int_0^a f''(x+b)\,dbda = \tag{B.4}$$

$$= f(x) + f'(x)h + \int_0^h \int_0^a \left(f'' + \int_0^b f'''(x+c)\,dc \right) dbda = \tag{B.5}$$

$$= f(x) + f'(x)h + \frac{f''(x)}{2!}h^2 + \int_0^h \int_0^a \int_0^b f'''(x+c)\,dcdbda = \tag{B.6}$$

$$\ldots \tag{B.7}$$

$$= f(x) + \frac{f'(x)}{1!}h + \frac{f''(x)}{2!}h^2 + \frac{f'''(x)}{3!}h^3 + \ \ldots \ = \tag{B.8}$$

$$= \sum_{n=0}^{\infty} \frac{f^{(n)}(x)}{n!}h^n. \tag{B.9}$$

В приведенной выше формулировке значение x обычно является фиксированным, и функция вычисляется через h. Часто удобнее записать разложение Тейлора функции $f(x)$ в точке a так, чтобы она оставалась функцией от x:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n. \tag{B.10}$$

Разложение Тейлора представляет собой функцию в виде бесконечной суммы полиномиальных членов, основанных на повторных производных в одной точке.

---

[2] Основная формула интегрального исчисления связывает функцию с интегралом ее производной.

Любая аналитическая функция может быть представлена своим разложением Тейлора в локальной окрестности точки.

Функция может быть аппроксимирована локально с помощью первых нескольких членов разложения Тейлора. На рис. В.1 показаны все более точные приближения функции $\cos x$ в точке $x = 1$. Включение большего числа слагаемых увеличивает точность локального приближения, но ошибка все равно накапливается при удалении от точки разложения.

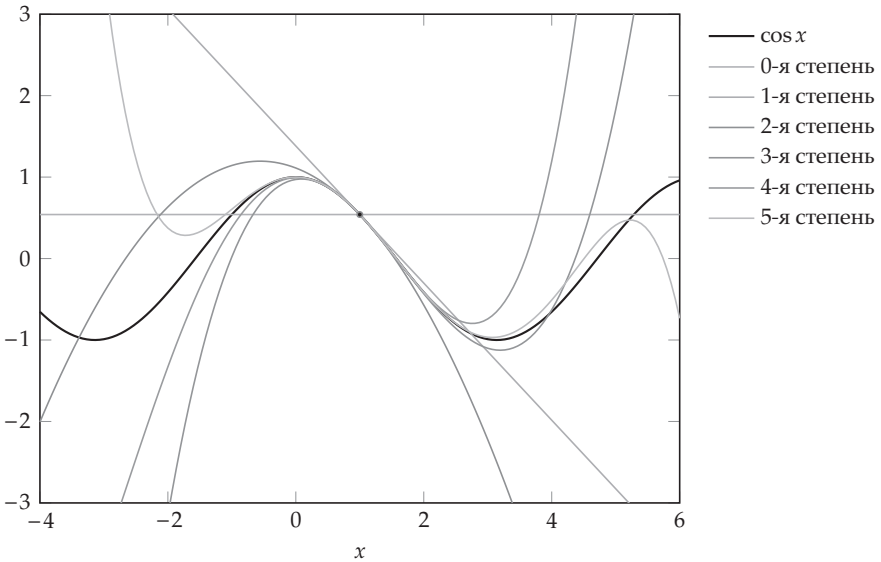


Рис. В.1. Последовательные приближения $\cos x$ в окрестности точки 1, основанные на первых n членах разложения Тейлора

В *линейном приближении* Тейлора используются первые два члена разложения Тейлора:

$$f(x) \approx f(a) + f'(a)(x-a). \tag{В.11}$$

В *квадратичном приближении Тейлора* используются первые три члена:

$$f(x) \approx f(a) + f'(a)(x-a) + \frac{1}{2} f''(x-a)^2 \tag{В.12}$$

и так далее.

В многомерном пространстве разложение Тейлора в точке $\mathbf{a}$ имеет вид

$$f(x) = f(\mathbf{a}) + \nabla f(\mathbf{a})^{\mathrm{T}} (\mathbf{x}-\mathbf{a}) + \frac{1}{2}(\mathbf{x}-\mathbf{a})^{\mathrm{T}} \nabla^2 f(\mathbf{a})(\mathbf{x}-\mathbf{a}) + \ldots \tag{В.13}$$

Первые два члена образуют касательную плоскость в точке $\mathbf{a}$. Третий член включает в себя локальную кривизну. В этой книге используются только первые три члена, показанные здесь.

B.3. Выпуклость

Выпуклая комбинация двух векторов $\mathbf{x}$ и $\mathbf{y}$ имеет вид

$$\alpha\mathbf{x} + (1 - \alpha)\mathbf{y} \tag{B.14}$$

для некоторого числа $\alpha \in [0, 1]$. Выпуклые комбинации могут состоять из m векторов:

$$w_1\mathbf{v}^{(1)} + w_2\mathbf{v}^{(2)} + ... + w_m\mathbf{v}^{(m)} \tag{B.15}$$

с неотрицательными весами $\mathbf{w}$, сумма которых равна единице.

Выпуклое множество — это множество, для которого отрезок, соединяющий любые две точки этого множества, полностью лежит внутри этого множества. Множество $\mathcal{S}$ выпукло, если

$$\alpha\mathbf{x} + (1 - \alpha)\mathbf{y} \in \mathcal{S} \tag{B.16}$$

для всех $\mathbf{x}$ в $\mathcal{S}$ и всех α в [0, 1]. Выпуклое и невыпуклое множество показаны на рис. B.2.

Рис. B.2. Выпуклые и невыпуклые множества

Выпуклая функция представляет собой чашеобразную функцию, область которой является выпуклым множеством. Под чашеобразной мы подразумеваем такую функцию, что любой отрезок, соединяющий любые две точки в ее области, не лежит ниже этой функции. Функция f является выпуклой над выпуклым множеством $\mathcal{S}$, если для всех x в $\mathcal{S}$ и всех α в [0, 1] выполняется неравенство

$$f\big(\alpha\mathbf{x} + (1-\alpha)\mathbf{y}\big) \le \alpha f(\mathbf{x}) + (1-\alpha) f(\mathbf{y}). \tag{B.17}$$

Пример областей, в которых функция является выпуклой или вогнутой, показан на рис. B.3.

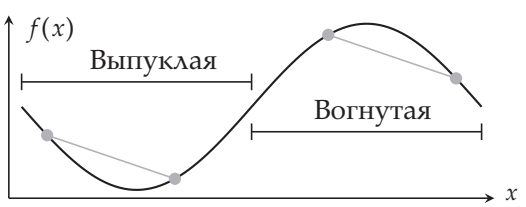


Рис. B.3. Области, в которых функция является выпуклой или вогнутой

Функция f называется *строго выпуклой* над выпуклым множеством $\mathcal{S}$, если для всех $\mathbf{x}$, $\mathbf{y}$ в $\mathcal{S}$ и α в (0, 1) выполняется неравенство:

$$f(\alpha\mathbf{x} + (1-\alpha)\mathbf{y}) < \alpha f(\mathbf{x}) + (1-\alpha)f(\mathbf{y}). \tag{B.18}$$

Строго выпуклые функции имеют не более одного минимума, тогда как выпуклая функция может иметь плоские области.[3] Примеры строгой и нестрогой выпуклости показаны на рис. В.4.

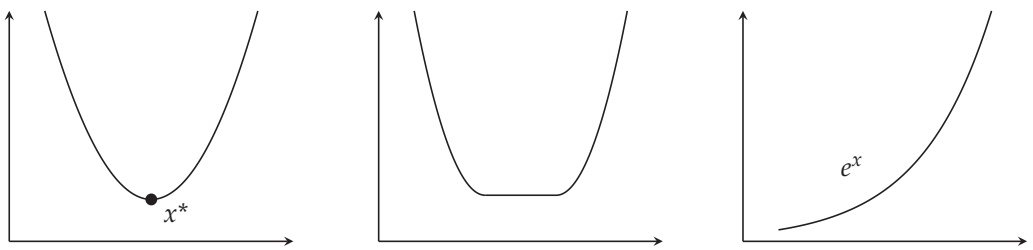


Рис. В.4. Не все выпуклые функции имеют один глобальный минимум

Функция *f вогнутая*, если $-f$ выпуклая. Кроме того, функция *f строго вогнутая*, если $-f$ строго выпуклая.

Не все выпуклые функции являются одномодальными, и не все одномодальные функции являются выпуклыми, как показано на рис. В.5.

Рис. В.5. Выпуклость и одномодальность — это не одно и то же

В.4. Нормы

Норма — это функция, которая присваивает длину вектору. Чтобы вычислить расстояние между двумя векторами, мы оцениваем норму разности между ними. Например, расстояние между точками *a* и евклидовой нормой равно

[3] Оптимизация выпуклых функций является предметом учебника [22].

$$\|\mathbf{a}-\mathbf{b}\|_2=\sqrt{(a_1-b_1)^2+(a_2-b_2)^2+\ldots+(a_n-b_n)^2}. \tag{B.19}$$

Функция f является нормой[4], если

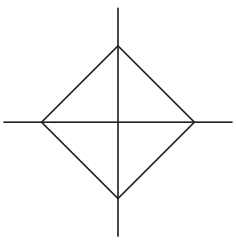
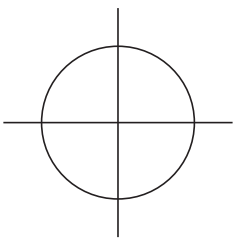
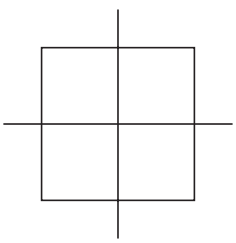
1. $f(\mathbf{x}) = 0$, если и только если $\mathbf{a}$ является нулевым вектором.

2. $f(\alpha\mathbf{x}) = |\alpha| f(\mathbf{x})$, так что длина масштабируется.

3. $f(\mathbf{a} + \mathbf{b}) \le f(\mathbf{a}) + f(\mathbf{b})$, *неравенство треугольника*.

Нормы L_p — это обычно используемый набор норм, параметризованных скаляром $p \ge 1$. Евклидова норма в уравнении (B.19) является нормой L_2. Несколько норм L_p приведены в табл. B.1.

Таблица B.1. Общие нормы L_p. Иллюстрация демонстрирует форму контуров нормы в двумерном пространстве. Все точки на контуре расположены на одинаковом расстоянии от начала координат по соответствующей норме

L_1: $\|\mathbf{x}\|_1 = \|x_1\| + \|x_2\| + \ldots + \|x_n\|$	
L_2: $\|\mathbf{x}\|_2 = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}$	
L_∞: $\|\mathbf{x}\|_\infty = \max\{\|x_1\|, \|x_2\|, \ldots, \|x_n\|\}$	

[4] Из этих аксиом вытекают некоторые важные свойства, в частности $f(-\mathbf{x}) = f(\mathbf{x})$ и $f(\mathbf{x}) \ge 0$.

Нормы L_p определяются в соответствии с:

$$\|\mathbf{x}\|_p = \lim_{\rho \to p} \left(|x_1|^\rho + |x_2|^\rho + \ldots + |x_n|^\rho \right)^{\frac{1}{\rho}}, \tag{B.20}$$

где предел необходим для определения нормы L_∞.[5]

B.5. Матричное исчисление

В этом разделе выводятся два распространенных градиента: $\nabla_{\mathbf{x}} \mathbf{b}^{\mathrm{T}} \mathbf{x}$ и $\nabla_{\mathbf{x}} \mathbf{x}^{\mathrm{T}} \mathbf{A} \mathbf{x}$. Чтобы получить $\nabla_{\mathbf{x}} \mathbf{b}^{\mathrm{T}} \mathbf{x}$, мы сначала раскроем скалярное произведение:

$$\mathbf{b}^{\mathrm{T}} \mathbf{x} = [b_1 x_1 + b_2 x_2 + \ldots + b_n x_n]. \tag{B.21}$$

Частная производная по единственной координате имеет вид:

$$\frac{\partial}{\partial x_i} \mathbf{b}^{\mathrm{T}} \mathbf{x} = b_i. \tag{B.22}$$

Таким образом, градиент равен

$$\nabla_{\mathbf{x}} \mathbf{b}^{\mathrm{T}} \mathbf{x} = \nabla_{\mathbf{x}} \mathbf{x}^{\mathrm{T}} \mathbf{b} = \mathbf{b}. \tag{B.23}$$

Для того чтобы получить $\nabla_{\mathbf{x}} \mathbf{x}^{\mathrm{T}} \mathbf{A} \mathbf{x}$ для квадратной матрицы $\mathbf{A}$, сначала раскроем выражение $\mathbf{x}^{\mathrm{T}} \mathbf{A} \mathbf{x}$:

$$\mathbf{x}^{\mathrm{T}} \mathbf{A} \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{bmatrix} = \tag{B.24}$$

$$= \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \begin{bmatrix} x_1 a_{11} + x_2 a_{12} + \ldots + x_n a_{1n} \\ x_1 a_{21} + x_2 a_{22} + \ldots + x_n a_{2n} \\ \vdots \\ x_1 a_{n1} + x_2 a_{n2} + \ldots + x_n a_{nn} \end{bmatrix} = \tag{B.25}$$

$$= \begin{matrix} x_1^2 a_{11} + x_1 x_2 a_{12} + \ldots + x_1 x_n a_{1n} + \\ + x_1 x_2 a_{21} + x_2^2 a_{22} + \ldots + x_2 x_n a_{2n} + \\ \vdots \\ + x_1 x_n a_{n1} + x_2 x_n a_{n2} + \ldots + x_n^2 a_{nn} \end{matrix}. \tag{B.26}$$

[5] Норму L_∞ также называют *равномерной нормой*, *нормой Чебышева* или *нормой шахматной доски.* Последнее название происходит от минимального количества ходов, которые должен сделать шахматный король, чтобы перейти с одного поля на другое.

Частная производная по i-му компоненту

$$\frac{\partial}{\partial x_i}\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} = \sum_{j=1}^{n} x_j\left(a_{ij} + a_{ji}\right). \tag{B.27}$$

Следовательно, градиент равен

$$\nabla_{\mathbf{x}}\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} = \begin{bmatrix} \sum_{j=1}^{n} x_j\left(a_{1j} + a_{j1}\right) \\ \sum_{j=1}^{n} x_j\left(a_{2j} + a_{j2}\right) \\ \vdots \\ \sum_{j=1}^{n} x_j\left(a_{nj} + a_{jn}\right) \end{bmatrix} = \tag{B.28}$$

$$= \begin{bmatrix} a_{11} + a_{11} & a_{21} + a_{12} & \ldots & a_{n1} + a_{1n} \\ a_{12} + a_{21} & a_{22} + a_{22} & \ldots & a_{n2} + a_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} + a_{n1} & a_{2n} + a_{n2} & \ldots & a_{nn} + a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \tag{B.29}$$

$$= \left(\mathbf{A} + \mathbf{A}^{\mathrm{T}}\right)\mathbf{x}. \tag{B.30}$$

B.6. Положительная определенность

Понятие *положительно определенной* или *положительно полуопределенной* матрицы часто возникает в линейной алгебре и оптимизации по разным причинам. Например, если матрица $\mathbf{A}$ является положительно определенной в функции $f(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}$, то f имеет единственный глобальный минимум.

Напомним, что квадратичное приближение дважды дифференцируемой функции f в точке $\mathbf{x}_0$ равно

$$f\left(\mathbf{x}\right) \approx f\left(\mathbf{x}_0\right) + \nabla f\left(\mathbf{x}_0\right)^{\mathrm{T}}\left(\mathbf{x} - \mathbf{x}_0\right) + \frac{1}{2}\left(\mathbf{x} - \mathbf{x}_0\right)^{\mathrm{T}}\mathbf{H}_0\left(\mathbf{x} - \mathbf{x}_0\right), \tag{B.31}$$

где $\mathbf{H}_0$ — гессиан функции f, вычисленный в точке $\mathbf{x}_0$. Знание того, что $(\mathbf{x} - \mathbf{x}_0)^{\mathrm{T}} \cdot \mathbf{H}_0(\mathbf{x} - \mathbf{x}_0)$ имеет единственный глобальный минимум, является достаточным для определения того, имеет ли все квадратичное приближение единственный глобальный минимум.[6]

[6] Компонент $\mathbf{f}(\mathbf{x}_0)$ просто смещает функцию по вертикали. Компонент $\nabla\mathbf{f}(\mathbf{x}_0)^{\mathrm{T}}(\mathbf{x} - \mathbf{x}_0)$ — это линейное слагаемое, над которым доминирует квадратичное слагаемое.

Симметричная матрица $\mathbf{A}$ является положительно определенной, если число $\mathbf{x}^T\mathbf{Ax}$ положительно для всех точек, кроме начала координат: $\mathbf{x}^T\mathbf{Ax} > 0$ для всех $\mathbf{x} \neq 0$.

Симметричная матрица $\mathbf{A}$ является положительно полуопределенной, если число $\mathbf{x}^T\mathbf{Ax}$ всегда неотрицательно: $\mathbf{x}^T\mathbf{Ax} \geq 0$ для всех $\mathbf{x}$.

B.7. Нормальное распределение

Функция плотности вероятности для одномерного *нормального распределения*[7], которое также называют *гауссовским распределением*, имеет вид:

$$\mathcal{N}(x|\mu, \nu) = \frac{1}{\sqrt{2\pi\nu}} e^{-\frac{(x-\mu)^2}{2\nu}}, \tag{B.32}$$

где μ означает математическое ожидание, а ν — дисперсию.[8] Это распределение показано на рис. B.6.

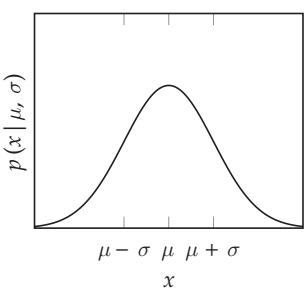


Рис. B.6. Одномерное нормальное распределение $\mathcal{N}(\mu, \nu)$

Кумулятивная функция распределения, зависящая от x, задает вероятность того, что значение, извлеченное из этого распределения, окажется меньше или равно x. Для одномерного нормального распределения кумулятивная функция распределения определяется как

$$\Phi(x) \equiv \frac{1}{2} + \frac{1}{2}\text{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right), \tag{B.33}$$

где erf — это *функция ошибки*:

$$\text{erf}(x) \equiv \frac{2}{\sqrt{\pi}} \int_0^x e^{-\tau^2} d\tau. \tag{B.34}$$

B.8. Метод Гаусса

Метод Гаусса — это метод аппроксимации интегралов с использованием взвешенной суммы значений функций.[9] Общий вид аппроксимации:

$$\int_a^b p(x) f(x) dx \approx \sum_{i=1}^m w_i f(x_i), \tag{B.35}$$

---

[7] Многомерное нормальное распределение обсуждается в главе 16.

[8] Дисперсия — это квадрат стандартного отклонения.

[9] Подробный обзор метода Гаусса см. в [149].

где $p(x)$ — известная неотрицательная весовая функция[10] на конечном или бесконечном интервале $[a, b]$.

Квадратурное правило m точек — это однозначный выбор точек $x_i \in (a, b)$ и весов $w_i > 0$ для $i \in \{1, ..., m\}$, которые определяют гауссово квадратурное приближение так, что любой многочлен степени $2m - 1$ или меньше интегрируется точно на $[a, b]$ с заданной весовой функцией.

Учитывая область и весовую функцию, мы можем вычислить класс ортогональных полиномов. Мы будем использовать $b_i(x)$ для обозначения ортогонального полинома[11] степени i. Любой полином степени m может быть представлен как линейная комбинация ортогональных полиномов, степень которых не превышает m. Мы формируем квадратурное правило, выбрав m точек x_i в качестве нулей ортогонального полинома p_m и получив веса путем решения системы уравнений:

$$\sum_{i=1}^{m} b_k(x_i) w_i = \begin{cases} \int_a^b p(x) b_0(x)^2 \, dx & \text{при } k = 0, \\ 0, & \text{при } k = 1, ..., m-1. \end{cases} \tag{B.36}$$

Гаусс решил уравнение (B.36) для интервала [1, 1] и весовой функции $p(x) = 1$. Ортогональные многочлены для этого случая являются многочленами Лежандра. Алгоритм B.1 реализует метод Гаусса для полиномов Лежандра, а пример B.3 описывает применение этого метода для интегрирования по [1,1].

Алгоритм B.1. Метод построения `m`-точечных квадратурных правил Лежандра на отрезке [–1, 1]. Результирующий тип содержит как узлы **`xs`**, так и веса **`ws`**

```
struct Quadrule
   ws
   xs
end
function quadrule_legendre(m)
   bs = [legendre(i) for i in 1 : m + 1]
   xs = roots(bs[end])
   A = [bs[k](xs[i]) for k in 1 : m, i in 1 : m]
   b = zeros(m)
   b[1] = 2
   ws = A \ b
   return Quadrule(ws, xs)
end
```

[10] Весовая функция на практике часто является функцией плотности вероятности.

[11] Ортогональные полиномы рассматриваются в главе 18.

Пример В.3. Получение трехчленного квадратурного правила для точного интегрирования полиномов, степень которых не превышает 5

Рассмотрим применение полиномов Лежандра для интегрирования функций на отрезке [–1, 1]. Допустим, что интересующая нас функция хорошо аппроксимируется полиномом пятой степени. Мы сформулируем трехточечное квадратурное правило, которое даст точный результат для полиномов, степень которых не превышает 5.

Полином Лежандра третьей степени имеет вид $\mathrm{Le}_3(x) = \frac{5}{2}x^3 - \frac{3}{2}x$. Его корни равны $x_1 = -\sqrt{3/5}$, $x_2 = 0$ и $x_3 = \sqrt{3/5}$. Полиномы Лежандра, степень которых не превышает 3, имеют вид $\mathrm{Le}_0(x) = 1$, $\mathrm{Le}_1(x) = x$ и $\mathrm{Le}_2(x) = \frac{3}{2}x^2 - \frac{1}{2}$. Веса являются решениями следующей системы уравнений.

$$\begin{bmatrix} \mathrm{Le}_0\left(-\sqrt{3/5}\right) & \mathrm{Le}_0(0) & \mathrm{Le}_0\left(\sqrt{3/5}\right) \\ \mathrm{Le}_1\left(-\sqrt{3/5}\right) & \mathrm{Le}_1(0) & \mathrm{Le}_1\left(\sqrt{3/5}\right) \\ \mathrm{Le}_2\left(-\sqrt{3/5}\right) & \mathrm{Le}_2(0) & \mathrm{Le}_2\left(\sqrt{3/5}\right) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} \int_{-1}^{1} \mathrm{Le}_0(x)^2\,dx \\ 0 \\ 0 \end{bmatrix},$$

$$\begin{bmatrix} 1 & 1 & 1 \\ -\sqrt{3/5} & 0 & \sqrt{3/5} \\ 4/10 & -1/2 & 4/10 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix}.$$

В итоге получаем, что $w_1 = w_3 = 5/9$ и $w_2 = 8/9$.

Рассмотрим интегрирование полинома пятой степени $f(x) = x^5 - 2x^4 + 3x^3 + 5x^2 - x + 4$. Точное значение равно $\int_{-1}^{1} p(x) f(x)\,dx = 158/15 \approx 10{,}533$. Квадратурная формула дает точно такой же результат:

$$\sum_{i=1}^{3} w_i f(x_i) = \frac{5}{9} f\left(-\sqrt{\frac{3}{5}}\right) + \frac{8}{9} f(0) + \frac{5}{9} f\left(\sqrt{\frac{3}{5}}\right) \approx 10{,}533.$$

Мы можем преобразовать любой интеграл по ограниченному отрезку $[a, b]$ в интеграл по отрезку [–1, 1], используя преобразование

$$\int_a^b f(x)\,dx = \frac{b-a}{2} \int_{-1}^{1} f\left(\frac{b-a}{2}x + \frac{a+b}{2}\right) dx. \qquad \text{(B.37)}$$

Таким образом, квадратурные правила могут быть предварительно вычислены для полиномов Лежандра, а затем применены к интегрированию по любому конечному интервалу.[12] Пример B.4 иллюстрирует применение такого преобразования, а алгоритм B.2 реализует интегральные преобразования в методе Гаусса для конечных областей.

Пример B.4. Интегралы по конечным отрезкам могут быть преобразованы в интегралы по отрезку [–1, 1] и решены с помощью квадратурных правил для многочленов Лежандра

Рассмотрим интегрирование функции $f(x) = x^5 - 2x^4 + 3x^3 + 5x^2 - x + 4$ больше [-3, 5]. Мы можем преобразовать это в интегрирование по [1, 1], используя уравнение (B.37):

$$\int_{-3}^{5} f(x)\,dx = \frac{5+3}{2}\int_{-1}^{1} f\left(\frac{5+3}{2}x + \frac{5-3}{2}\right)dx = 4\int_{-1}^{1} f(4x+1)\,dx.$$

Мы используем трехчленное квадратурное правило, полученное в примере B.3 для интегрирования функции $g(x) = 4f(4x + 1) = 4096x^5 + 3072x^4 + 1280x^3 + 768x^2 + 240x + 40$ на отрезке [–1, 1]:

$$\int_{-1}^{1} p(x)g(x)\,dx = \frac{5}{9}g\left(-\sqrt{3/5}\right) + \frac{8}{9}g(0) + \frac{5}{9}g\left(\sqrt{3/5}\right) = 1820{,}8.$$

Алгоритм B.2. Функция **quadint** для интегрирования одномерной функции **f** с заданным квадратурным правилом **quadrule** на конечном отрезке [**a**, **b**]

```
quadint(f, quadrule) =
    sum(w * f(x) for (w,x) in zip(quadrule.ws, quadrule.xs))
function quadint(f, quadrule, a, b)
    α = (b - a) / 2
    β = (a + b) / 2
    g = x -> α * f(α * x + β)
    return quadint(g, quadrule)
end
```

[12] Подобные методы могут быть применены к интегрированию по бесконечным интервалам, таким как [0, ∞] с использованием полиномов Лагерра и (–∞, ∞) с использованием полиномов Эрмита.

Приложение Г

Решения

Упражнение 1.1. $f(x) = x^3/3x - x$ при $x = 1$.

Упражнение 1.2. У нее нет минимума, т.е. функция является неограниченной снизу.

Упражнение 1.3. Нет. Рассмотрите минимизацию $f(x) = x$ при условии $x \geq 1$.

Упражнение 1.4. Функция f может быть разделена на две отдельные функции, которые зависят только от их конкретной координаты:

$$f(x, y) = g(x) + h(y),$$

где $g(x) = x^2$ и $h(y) = y$. Функции g и h строго возрастают при $x, y \geq 1$. Хотя функция h достигает минимума при $y = 1$, можно лишь приблизиться к единице из-за строгого неравенства $x > y$. Таким образом, функция f не имеет минимумов.

Упражнение 1.5. Точка перегиба — это точка на кривой, где меняется знак кривизны. Необходимым условием для точки перегиба является равенство нулю второй производной. Вторая производная равна $f''(x) = 6x$, т.е. она обращается в нуль при $x = 0$.

Достаточным условием для точки перегиба является изменение знака второй производной в окрестности точки x. Иначе говоря, $f''(x + \varepsilon)$ и $f''(x - \varepsilon)$ при $\varepsilon \ll 1$ имеют противоположные знаки. Это верно для $x = 0$, т.е. это — точка перегиба.

Таким образом, у функции x^3—10 есть только одна точка перегиба.

Упражнение 2.1. Гессиан может быть вычислен с помощью правой разности:

$$H_{ij} = \frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} \approx \frac{\nabla f\left(\mathbf{x} + h\mathbf{e}_j\right)_i - \nabla f(\mathbf{x})_i}{h},$$

где $\mathbf{e}_i$ — это i-й базисный вектор с $\mathbf{e}_i = 1$, а все остальные элементы равны нулю.

Таким образом, можно приблизить j-й столбец гессиана, используя формулу

$$H_j \approx \frac{\nabla f\left(\mathbf{x} + h\mathbf{e}_j\right) - \nabla f(\mathbf{x})}{h}.$$

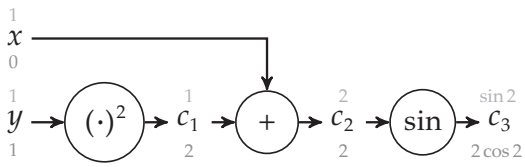
Эту процедуру можно повторить для каждого столбца гессиана.

Упражнение 2.2. Требуется два вычисления целевой функции.

Упражнение 2.3. $f'(x) = \frac{1}{x} + e^x - \frac{1}{x^2}$. Когда x близко к нулю, мы находим, что $x < 1$. Следовательно, $\frac{1}{x} > 1$ и, наконец, $\frac{1}{x^2} > \frac{1}{x} > 0$, поэтому $-\frac{1}{x^2}$ доминирует.

Упражнение 2.4. По методу комплексного шага мы имеем $f'(x) \approx \mathrm{Im}(2 + 4ih)/h = 4h/h = 4$.

Упражнение 2.5. См. рисунок ниже:



Упражнение 2.6. Производная второго порядка может быть аппроксимирована с использованием формулы центральной разности для производной первого порядка

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

при малых значениях h.

Подставляя в формулу оценки $f'(x + h/2)$ и $f'(x - h/2)$, получаем

$$f''(x) \approx \frac{\frac{f(x+h/2+h/2) - f(x+h/2-h/2)}{h} - \frac{f(x-h/2+h/2) - f(x-h/2-h/2)}{h}}{h} =$$

$$= \frac{f(x+h) - f(x)}{h^2} - \frac{f(x) - f(x-h)}{h^2} =$$

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

Упражнение 3.1. Поиск Фибоначчи предпочтителен, когда производные недоступны.

Упражнение 3.2. Для применения метода Шуберта–Пиявского необходимо знать константу Липшица, а она может быть неизвестна.

Упражнение 3.3. $f(x) = x^2$. Поскольку функция является квадратичной, после трех оценок квадратичная модель будет точно представлять эту функцию.

Упражнение 3.4. Чтобы найти корни функции $f'(x) = x - 1$ можно использовать метод деления пополам. После первой итерации мы имеем [0, 500]. Затем [0, 250]. И наконец [0, 125].

Упражнение 3.5. Нет, постоянная Липшица должна ограничивать производную всюду на интервале, в то время как $f'(1) = 2(1 + 2) = 6$.

Упражнение 3.6. Нет. Лучшее, что можно сделать — использовать поиск Фибоначчи и уменьшить неопределенность в три раза, т.е. до $(32-1)/3 = 10\frac{1}{3}$.

Упражнение 4.1. Функция может, например, быть неограниченной снизу, в этом случае условия, поставленные для величины градиента, никогда не могут быть выполнены, если алгоритм расходится. С другой стороны, мы хотим остановить процедуру оптимизации, как только приблизимся к локальному минимуму.

Упражнение 4.2. Нам нужно, чтобы выполнялось неравенство $6 + (-1 + \alpha)^2 \leq \leq 7 - 2\alpha \cdot 10^{-4}$, что дает $\alpha^2 - 2\alpha + 2 \cdot 10^{-4}\alpha \leq 0$, и наконец, $\alpha \leq 2(1 - 10^{-4})$.

Упражнение 5.1. $\nabla f(\mathbf{x}) = 2\mathbf{A}\mathbf{x} + \mathbf{b}$.

Упражнение 5.2. Производная $f'(x) = 4x^3$. Начиная с $x^{(1)} = 1$:

$$f'(1) = 4 \quad \rightarrow \quad x^{(2)} = 1 - 4 = -3, \tag{Г.1}$$

$$f'(-3) = 4 * (-27) = 108 \quad \rightarrow \quad x^{(3)} = -3 + 108 = 105. \tag{Г.2}$$

Упражнение 5.3. При больших x получаем $f'(x) = e^x - e^{-x} \approx e^x$ для большого x. Таким образом, $f'(x^{(1)}) \approx e^{10}$ и $x^{(2)} \approx -e^{10}$. Если мы применим точный линейный поиск, то получим $x^{(2)} = 0$. Таким образом, без линейного поиска невозможно гарантировать уменьшение значения целевой функции.

Упражнение 5.4. Гессиан равен $2\mathbf{H}$, и

$$\nabla q(\mathbf{d}) = \mathbf{d}^{\mathrm{T}}\left(\mathbf{H} + \mathbf{H}^{\mathrm{T}}\right) + \mathbf{b} = \mathbf{d}^{\mathrm{T}}(2\mathbf{H}) + \mathbf{b}.$$

Градиент равен $\mathbf{b}$, если $\mathbf{d} = 0$. Метод сопряженных градиентов может расходиться, потому что нет гарантии, что матрица $\mathbf{H}$ будет положительно определенной.

Упражнение 5.5. Метод Нестерова ищет точку, в которой вы будете находиться после выполнения итерационного шага, чтобы вычислить новое значение функции.

Упражнение 5.6. Метод сопряженных градиентов неявным образом повторно использует априорную информацию о функции и, таким образом, может на практике обеспечить лучшую сходимость.

Упражнение 5.7. Метод сопряженных градиентов изначально следует наиболее крутому направлению спуска. Градиент

$$\nabla f(x, y) = [2x + y,\ 2y + x], \tag{Г.3}$$

в точке $(x, y) = (1, 1)$ равен $\left[1/\sqrt{2},\ 1/\sqrt{2}\right]$. Направление самого крутого спуска противоположно градиенту, $\mathbf{d}^{(1)} = \left[-1/\sqrt{2},\ -1/\sqrt{2}\right]$.

Гессиан равен

$$\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}. \tag{Г.4}$$

Поскольку функция является квадратичной, а гессиан положительно определен, метод сопряженных градиентов сходится не более чем за два шага. Таким образом, результирующая точка $(x, y) = (0, 0)$ после двух шагов является оптимальной, и градиент в ней равен нулю.

Упражнение 5.8. Нет. Если выполняется точная минимизация, то направления спуска на смежных шага являются ортогональными , но $[1, 2, 3]^{\mathrm{T}}[0, 0, -3] \neq 0$.

Упражнение 6.1. Информация второго порядка гарантирует существование локального минимума, тогда как равенство градиента нулю — необходимое, но недостаточное условие для обеспечения локальной оптимальности.

Упражнение 6.2. Мы бы предпочли метод Ньютона, если бы начальное значение было достаточно близким к корню и существовала возможность вычислять производные аналитически. Метод Ньютона имеет лучшую скорость сходимости.

Упражнение 6.3. $f'(x) = 2x$, $f''(x) = 2$. Таким образом, $x^{(2)} = x^{(1)} - 2x^{(1)}/2 = 0$; иначе говоря, метод сходится за один шаг из любой начальной точки.

Упражнение 6.4. Поскольку $\nabla f(\mathbf{x}) = \mathbf{H}\mathbf{x}$, $\nabla^2 f(x) = \mathbf{H}$ и $\mathbf{H}$ — невырожденная матрица, то $\mathbf{x}^{(2)} = \mathbf{x}^{(1)} - \mathbf{H}^{-1}\mathbf{H}\mathbf{x}^{(1)} = \mathbf{0}$. Иначе говоря, метод Ньютона сходится за один шаг.

Метод градиентного спуска расходится.

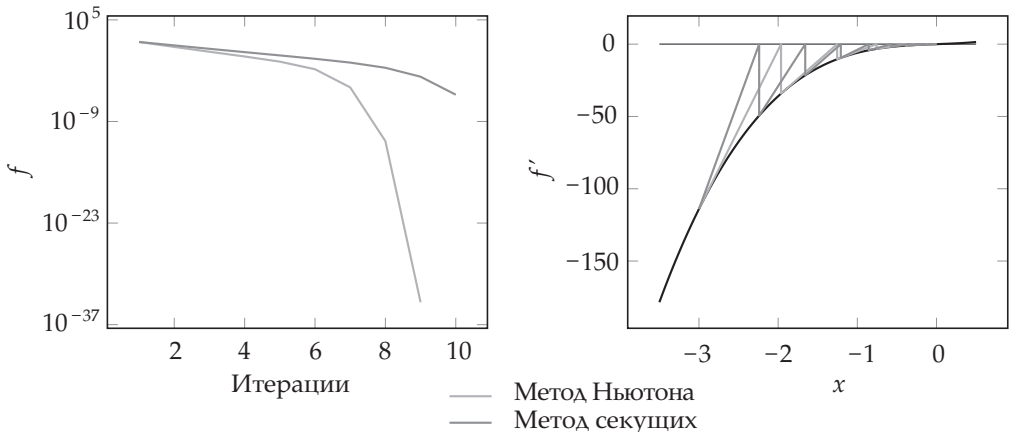
$$\mathbf{x}^{(2)} = [1, 1] - [1, 1000] = [0, -999], \tag{Г.5}$$

$$\mathbf{x}^{(3)} = [0, -999] - [0, 1000 \cdot 999] = [0, 998001]. \tag{Г.6}$$

Сопряженный градиентный спуск использует то же начальное направление поиска, что и градиентный спуск, и сходится к минимуму на втором шаге, поскольку целевая функция является квадратичной.

Упражнение 6.5. На левом графике показана сходимость метода Ньютона, приближающегося к порогу представления малых чисел с плавающей запятой за девять итераций. Секущий метод сходится медленнее, потому что он может лишь аппроксимировать производную.

Правый график показывает проекцию точных и приближенных касательных линий по отношению к f' для каждого метода. Касательные линии метода секущих имеют более высокий наклон и, таким образом, преждевременно пересекают ось x.



Упражнение 6.6. Рассмотрим последовательность $x^{(k+1)} = x^{(k)}/2$, начиная с $x^{(1)} = -1$ на примере функции $f(x) = x^2 - x$. Очевидно, что последовательность сходится к $x = 0$, значения $f(x)$ уменьшаются, и тем не менее последовательность не сходится к точке минимума.

Упражнение 6.7. Квазиньютоновский метод не требует вычислений или знаний об элементах гессиана и, следовательно, не требует решения линейной системы уравнений на каждой итерации.

Упражнение 6.8. Итерационное приближение BFGS не существует, если $\delta^{\mathrm{T}}\gamma \approx 0$. В этом случае просто пропустите шаг модификации.

Упражнение 6.9. Целевая функция является квадратичной и поэтому может быть сведена к минимуму за один шаг. Градиент равен $\nabla f = [2(x_1 + 1), 2(x_2 + 3)]$ и обращается в нуль в точке $\mathbf{x}^* = [-1, -3]$. Гессиан положительно определен, поэтому $\mathbf{x}^*$ — это минимум.

Упражнение 6.10. Новое приближение имеет вид

$$f^{(k+1)}(\mathbf{x}) = y^{(k+1)} + \left(\mathbf{g}^{(k+1)}\right)^{\mathrm{T}}\left(\mathbf{x} - \mathbf{x}^{(k+1)}\right) + \frac{1}{2}\left(\mathbf{x} - \mathbf{x}^{(k+1)}\right)^{\mathrm{T}}\mathbf{H}^{(k+1)}\left(\mathbf{x} - \mathbf{x}^{(k+1)}\right),$$

используя истинное значение функции и градиент в точке $\mathbf{x}^{(\mathrm{k+1})}$, но для него необходим обновленный гессиан $\mathbf{H}^{(k+1)}$. Эта форма записи автоматически удовлетворяет условию $f^{(k+1)}(\mathbf{x}^{(k+1)}) = y^{(k+1)}$ и $\nabla f^{(k+1)}(\mathbf{x}^{(k+1)}) = \mathbf{g}^{(k+1)}$. Мы должны выбрать новый гессиан, чтобы удовлетворить третье условие:

$$\begin{aligned}\nabla f^{(k+1)}(\mathbf{x}^{(k)}) &= \mathbf{g}^{(k+1)} + \mathbf{H}^{(k+1)}(\mathbf{x}^{(k)} - \mathbf{x}^{(k+1)}) = \\ &= \mathbf{g}^{(k+1)} - \mathbf{H}^{(k+1)}(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) = \\ &= \mathbf{g}^{(k+1)} - \mathbf{H}^{(k+1)}\boldsymbol{\delta}^{(k+1)} = \\ &= \mathbf{g}^{(k)}.\end{aligned}$$

Можно поменять порядок и выполнить подстановку:

$$\mathbf{H}^{(k+1)}\boldsymbol{\delta}^{(k+1)} = \boldsymbol{\gamma}^{(k+1)}.$$

Напомним, что матрица $\mathbf{A}$ является положительно определенной, если для любого ненулевого вектора $\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} > 0$. Умножив секущее уравнение на $\boldsymbol{\delta}^{(k+1)}$, получим условие кривизны:

$$\left(\boldsymbol{\delta}^{(k+1)}\right)^{\mathrm{T}}\mathbf{H}^{(k+1)}\boldsymbol{\delta}^{(k+1)} = \left(\boldsymbol{\delta}^{(k+1)}\right)^{\mathrm{T}}\boldsymbol{\gamma}^{(k+1)} > 0.$$

Мы ищем новую положительно определенную матрицу $\mathbf{H}^{(k+1)}$. Все положительно определенные матрицы являются симметричными, поэтому для вычисления новой положительно определенной матрицы необходимо задать $n(n+1)/2$ переменных. Уравнение секущей налагает лишь n условий на эти переменные, что приводит к бесконечному числу решений. Чтобы получить единственное решение, мы выбираем положительно определенную матрицу, наиболее близкую к $\mathbf{H}^{(k)}$. Эта цель приводит к искомой задаче оптимизации.

Упражнение 7.1. Производная имеет n элементов, а гессиан — n^2. Каждый элемент производной требует двух значений при использовании методов конечных разностей: $f(\mathbf{x})$ и $f(\mathbf{x} + h\mathbf{e}^{(i)})$. Каждый элемент гессиана требует трех значений при использовании методов конечных разностей. Таким образом, для аппроксимации градиента вам необходимы $n + 1$ значение, а для аппроксимации гессиана — примерно n^2 значений.

При больших n аппроксимация гессиана непомерно затратна. Прямые методы могут сделать сравнительно больше шагов, используя n^2 значений функций, поскольку прямые методы не должны вычислять производную или гессиан на каждом шаге.

Упражнение 7.2. Рассмотрим минимизацию функции $f(x) = xy$ и $x_0 = [0, 0]$. Продолжение в любом каноническом направлении не приведет к снижению целевой функции, но x_0 явно не является точкой минимума.

Упражнение 7.3. На каждой итерации метод Хука–Дживса выбирает $2n$ точек вдоль координатных направлений с размером шага a. Он останавливается, когда ни одна из точек не обеспечивает улучшения, а размер шага не превышает заданной погрешности ε. Это часто приводит к остановке метода Хука–Дживса, когда он сходится к ε-окрестности локального минимума, но не всегда. Например, функция может убывать между двумя направлениями координат за пределами ε-окрестности, прежде чем достигнет своего локального минимума и метод Хука–Дживса не обнаружит его.

Упражнение 7.4. Минимизация сопротивления аэродинамического профиля минимальной толщины (для сохранения структурной целостности). Оценка характеристик аэродинамического профиля с использованием вычислительной гидро-

динамики включает решение уравнений в частных производных. Поскольку функция не известна аналитически, вряд ли у нас будет аналитическое выражение для производной.

Упражнение 7.5. Алгоритм разделенных прямоугольников выполняет выбор в центре интервалов, а не там, где граница, полученная на основе известной константы Липшица, является наименьшей.

Упражнение 7.6. Алгоритм не может быть циклическим координатным поиском, так как меняется более одного компонента. Он может быть методом Пауэлла.

Упражнение 8.1. Метод перекрестной энтропии должен соответствовать параметрам распределения для каждой итерации. К сожалению, не существует известных аналитических решений для аппроксимации многомерных нормальных распределений. Вместо этого обычно используют итерационный EM-алгоритм.

Упражнение 8.2. Если количество элитных выборок близко к общему количеству выборок, то полученное распределение будет точно соответствовать популяции. Не будет существенного смещения в сторону точки минимума, и поэтому сходимость будет медленной.

Упражнение 8.3. Производная логарифмического правдоподобия по v равна

$$\frac{\partial}{\partial v} l\left(x \middle| \mu,\ v\right) = \frac{\partial}{\partial v}\left(-\frac{1}{2}\ln 2\pi - \frac{1}{2}\ln v - \frac{\left(x-\mu\right)^2}{2v} \right) =$$

$$= -\frac{1}{2v} + \frac{\left(x-\mu\right)^2}{2v^2}.$$

Если математическое ожидание уже оптимально, то второе слагаемое будет нулевым. Таким образом, производная равна $-1/2v$ и уменьшение v увеличит вероятность извлечения элитных выборок. К сожалению, v становится оптимальным, только асимптотически приближаясь к нулю. Асимптота около нуля при вычислении градиента приведет к большим размерам шага, которые не могут быть допустимыми, потому что величина v должна оставаться положительной.

Упражнение 8.4. Плотность вероятности расчетной точки $\mathbf{x}$ при многомерном нормальном распределении с математическим ожиданием $\boldsymbol{\mu}$ и ковариацией Σ равна

$$p\left(\mathbf{x} \middle| \boldsymbol{\sigma},\ \boldsymbol{\Sigma}\right) = \frac{1}{\left(2\pi\left|\boldsymbol{\Sigma}\right|\right)^{1/2}} \exp\left(-\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}\right)^{\mathrm{T}} \boldsymbol{\Sigma}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}\right) \right).$$

Можно упростить задачу, максимизируя логарифмическое правдоподобие:[1]

[1] Логарифмическая функция является вогнутой при положительных аргументах, поэтому максимизация $\log f(x)$ также максимизирует строго положительную функцию $f(x)$.

$$\ln p\left(\mathbf{x}\middle|\boldsymbol{\sigma},\boldsymbol{\Sigma}\right)=-\frac{1}{2}\ln\left(2\pi\left|\boldsymbol{\Sigma}\right|\right)-\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}\right)^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}\right)=$$
$$=-\frac{1}{2}\ln\left(2\pi\left|\boldsymbol{\Sigma}\right|\right)-\frac{1}{2}\left(\mathbf{x}^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\mathbf{x}-2\mathbf{x}^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}+\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\right).$$

Начнем с максимизации логарифмической вероятности m особей относительно математического ожидания:

$$l\left(\boldsymbol{\mu}\middle|\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(m)}\right)=\sum_{i=1}^{m}\ln p\left(\mathbf{x}^{(i)}\middle|\boldsymbol{\mu},\boldsymbol{\Sigma}\right)=$$
$$=\sum_{i=1}^{m}-\frac{1}{2}\ln\left(2\pi\left|\boldsymbol{\Sigma}\right.\right)-\frac{1}{2}\left(\left(\mathbf{x}^{(i)}\right)^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\mathbf{x}^{(i)}-2\left(\mathbf{x}^{(i)}\right)^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}+\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\right).$$

Затем вычислим градиент, используя тот факт, что $\nabla_{\mathbf{z}}\mathbf{z}^{\mathrm{T}}\mathbf{A}\mathbf{z}=(\mathbf{A}+\mathbf{A}^{\mathrm{T}})\mathbf{z}$, $\nabla_{\mathbf{z}}\mathbf{a}^{\mathrm{T}}\mathbf{z}=\mathbf{a}$, а матрица $\boldsymbol{\Sigma}$ является симметричной и положительно определенной. Таким образом, матрица $\boldsymbol{\Sigma}^{-1}$ является симметричной.

$$\nabla_{\mu}l\left(\boldsymbol{\mu}\middle|\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(m)}\right)=\sum_{i=1}^{m}-\frac{1}{2}\left(\nabla_{\boldsymbol{\mu}}\left(-2\left(\mathbf{x}^{(i)}\right)^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\right)+\nabla_{\boldsymbol{\mu}}\left(\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\right)\right)=$$
$$=\sum_{i=1}^{m}\left(\nabla_{\boldsymbol{\mu}}\left(\left(\mathbf{x}^{(i)}\right)^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\right)\right)-\frac{1}{2}\nabla_{\boldsymbol{\mu}}\left(\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\right)=$$
$$=\sum_{i=1}^{m}\boldsymbol{\Sigma}^{-1}\mathbf{x}^{(i)}-\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}.$$

Приравняем градиент к нулю.

$$\mathbf{0}=\sum_{i=1}^{m}\boldsymbol{\Sigma}^{-1}\mathbf{x}^{(i)}-\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu},$$
$$\sum_{i=1}^{m}\boldsymbol{\mu}=\sum_{i=1}^{m}\mathbf{x}^{(i)},$$
$$m\boldsymbol{\mu}=\sum_{i=1}^{m}\mathbf{x}^{(i)},$$
$$\boldsymbol{\mu}=\frac{1}{m}\sum_{i=1}^{m}\mathbf{x}^{(i)}.$$

Максимизируем логарифмическое правдоподобие по обратной ковариационой матрице, $\boldsymbol{\Lambda}=\Sigma^{-1}$, используя тот факт, что $|\mathbf{A}^{-1}|=1/|\mathbf{A}|$ с $\mathbf{b}^{(i)}=\mathbf{x}^{(i)}-\boldsymbol{\mu}$:

$$l\left(\boldsymbol{\Lambda}\middle|\boldsymbol{\mu},\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(m)}\right)=\sum_{i=1}^{m}-\frac{1}{2}\ln\left(2\pi\left|\boldsymbol{\Lambda}\right|^{-1}\right)-\frac{1}{2}\left(\left(\mathbf{x}^{(i)}-\boldsymbol{\mu}\right)^{\mathrm{T}}\Lambda\left(\mathbf{x}^{(i)}-\boldsymbol{\mu}\right)\right)=$$
$$=\sum_{i=1}^{m}\frac{1}{2}\ln\left|\boldsymbol{\Lambda}\right|-\frac{1}{2}\left(\mathbf{b}^{(i)}\right)^{\mathrm{T}}\boldsymbol{\Lambda}\mathbf{b}^{(i)}.$$

Вычислим градиент, используя тот факт, что $\nabla_{\mathbf{A}}|\mathbf{A}| = |\mathbf{A}|\mathbf{A}^{-\mathrm{T}}$ и $\nabla_{\mathbf{A}}\mathbf{z}\mathbf{A}^{\mathrm{T}}\mathbf{z} = \mathbf{z}\mathbf{z}^{\mathrm{T}}$:

$$\begin{aligned}
\nabla_{\boldsymbol{\Lambda}} l\left(\boldsymbol{\Lambda}\middle|\boldsymbol{\mu}, \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\right) &= \sum_{i=1}^{m}\nabla_{\boldsymbol{\Lambda}}\left(\frac{1}{2}\ln\left|\boldsymbol{\Lambda}\right| - \frac{1}{2}\left(\mathbf{b}^{(i)}\right)^{\mathrm{T}}\boldsymbol{\Lambda}\mathbf{b}^{(i)}\right) = \\
&= \sum_{i=1}^{m}\frac{1}{2\left|\boldsymbol{\Lambda}\right|}\nabla_{\boldsymbol{\Lambda}}\left|\boldsymbol{\Lambda}\right| - \frac{1}{2}\mathbf{b}^{(i)}\left(\mathbf{b}^{(i)}\right)^{\mathrm{T}} = \\
&= \sum_{i=1}^{m}\frac{1}{2\left|\boldsymbol{\Lambda}\right|}\left|\boldsymbol{\Lambda}\right|\boldsymbol{\Lambda}^{-\mathrm{T}} - \frac{1}{2}\mathbf{b}^{(i)}\left(\mathbf{b}^{(i)}\right)^{\mathrm{T}} = \\
&= \frac{1}{2}\sum_{i=1}^{m}\boldsymbol{\Lambda}^{-\mathrm{T}} - \mathbf{b}^{(i)}\left(\mathbf{b}^{(i)}\right)^{\mathrm{T}} = \\
&= \frac{1}{2}\sum_{i=1}^{m}\boldsymbol{\Sigma} - \mathbf{b}^{(i)}\left(\mathbf{b}^{(i)}\right)^{\mathrm{T}}.
\end{aligned}$$

и приравняем градиент к нулю:

$$\begin{aligned}
0 &= \frac{1}{2}\sum_{i=1}^{m}\boldsymbol{\Sigma} - \mathbf{b}^{(i)}\left(\mathbf{b}^{(i)}\right)^{\mathrm{T}}, \\
\sum_{i=1}^{m}\boldsymbol{\Sigma} &= \sum_{i=1}^{m}\mathbf{b}^{(i)}\left(\mathbf{b}^{(i)}\right)^{\mathrm{T}}, \\
\boldsymbol{\Sigma} &= \frac{1}{m}\sum_{i=1}^{m}\left(\mathbf{x}^{(i)} - \boldsymbol{\mu}\right)\left(\mathbf{x}^{(i)} - \boldsymbol{\mu}\right)^{\mathrm{T}}.
\end{aligned}$$

Упражнение 9.1. Выживание наиболее приспособленных особей за счет предпочтения особей с лучшими значениями целевой функции.

Упражнение 9.2. Мутация стимулирует исследование с использованием случайности, поэтому важно избегать локальных минимумов. Если мы подозреваем, что есть лучшее решение, нам нужно увеличить частоту мутаций и дать алгоритму время на его обнаружение.

Упражнение 9.3. Увеличьте размер популяции или коэффициент, который смещает поиск к отдельным минимумам.

Упражнение 10.1. Сначала переформулируйте задачу для функции $f(x) = x + \rho\max(-x, 0)^2$, производная которой имеет вид

$$f'(x) = \begin{cases} 1 + 2\rho x, \text{ если } x < 0, \\ 1 \text{ в противном случае}. \end{cases} \tag{Г.7}$$

Эта неограниченная целевая функция может быть решена, если положить $f'(x) = 0$, что дает решение $x^* = -\dfrac{1}{2\rho}$. Таким образом, при $\rho \to \infty$ получаем $x^* \to 0$.

Упражнение 10.2. Переформулируем задачу так: $f(x) = x + \rho(x < 0)$. Целевая функция неограничена снизу до тех пор, пока значение ρ конечно, а x стремится к отрицательной бесконечности. Правильное решение не найдено, тогда как метод квадратичного штрафа смог приблизиться к правильному решению.

Упражнение 10.3. Попытайтесь увеличить штрафной параметр ρ. Возможно, он слишком мал и величина штрафа неэффективна. В таких случаях итерации могут достигать недопустимой области, где функция уменьшается быстрее, чем штраф, в результате чего метод сходится к недопустимому решению.

Упражнение 10.4. Пусть x_p^* — решение неограниченной задачи. Обратите внимание на то, что $x_p^* \not> 0$. В противном случае штраф имел бы вид $\left(\min\left(x_p^*,\ 0\right)\right)^2 = 0$, т.е. x_p^* было бы решением исходной задачи. Теперь предположим, что $x_p^* = 0$. Условия оптимальности первого порядка утверждают, что $f'\left(x_p^*\right) + \mu x_p^* = 0$, откуда следует, что $f'\left(x_p^*\right) = 0$, т.е. возникает противоречие. Таким образом, если точка минимума существует, она должна быть недопустимой.

Упражнение 10.5. Для того чтобы получить точное решение, большой штраф ρ не нужен.

Упражнение 10.6. Когда итерации должны оставаться допустимыми.

Упражнение 10.7. Рассмотрим следующую задачу

$$\begin{aligned} &\min_x x^3 \\ &\text{при условии } x \geq 0, \end{aligned} \tag{Г.8}$$

в которой минимум достигается в точке $x^* = 0$. Используя метод штрафа, можно свести ее к минимизации функции

$$\min_x x^3 + \rho(\min(x, 0))^2. \tag{Г.9}$$

Для любого конечного ρ функция остается неограниченной снизу, так как переменная x становится бесконечно отрицательной. Кроме того, когда x становится бесконечно отрицательной, функция становится бесконечно крутой. Другими словами, если мы запустим метод наискорейшего спуска слишком далеко влево, мы получим $x^3 + \rho x^2 \approx x^3$, штраф станет неэффективным, а метод наискорейшего спуска будет расходиться.

Упражнение 10.8. Поиск допустимой точки можно сформулировать в виде задачи оптимизации с постоянной целевой функцией и ограничением, которое задает условия допустимости:

$$\min_{\mathbf{x}} 0$$
$$\text{при условиях } \mathbf{h}(\mathbf{x}) = \mathbf{0}, \quad \mathbf{g}(\mathbf{x}) \le \mathbf{0}. \tag{Г.10}$$

Такую задачу часто можно решить с использованием методов штрафа. Квадратичные штрафы являются самым распространенным выбором, потому что они уменьшаются в направлении допустимой области.

Упражнение 10.9. Задача минимизируется в точке $x^* = 1$, находящейся на границе ограничения. Решение с помощью t-преобразования дает неограниченную целевую функцию

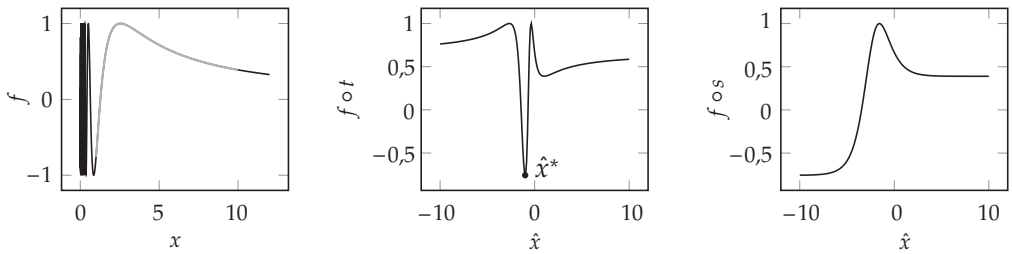
$$f_t(\hat{x}) = \sin\frac{4}{5{,}5 + 4{,}5\dfrac{2\hat{x}}{1+\hat{x}}}, \tag{Г.11}$$

которая имеет единственный глобальный минимум при $\hat{x} = -1$, точно соответствующий x^*.

Сигмоидальное преобразование имеет неограниченную целевую функцию

$$f_s(\hat{x}) = \sin\frac{4}{1 + \dfrac{9}{1+e^{-\hat{x}}}}. \tag{Г.12}$$

К сожалению, нижняя граница a, которая в данном случае соответствует условию $x = 1$, достигается только при $\hat{x}$, стремящемся к минус бесконечности. Задача оптимизации без ограничений, полученная с использованием сигмоидального преобразования, не имеет решения, и метод не может правильно определить решение исходной задачи.



Упражнение 10.10. Минимизируйте функцию $x_1^2 + x_2^2$ при $x_1 \ge 1$.

Упражнение 10.11. Можно выразить ограничение через x_1

$$x_1 = 6 - 2x_2 - 3x_3 \tag{Г.13}$$

и подставить это выражение в целевую функцию:

$$\min_{x_2,x_3} x_2^2 + x_3 - \left(2x_2 + 3x_3 - 6\right)^3. \tag{Г.14}$$

Упражнение 10.12. Ограничение должно быть согласовано с целевой функцией. Ориентация целевой функции — [–1, –2]. Ориентация ограничения — [a, 1]. Единственное значение для a, при котором они согласованы, a = 0,5.

Упражнение 10.13. Преобразованная целевая функция имеет вид $f(x) = 1 - x^2 + \rho p(x)$, где p — либо подсчет нарушений, или квадратичный штраф:

$$p_{\text{count}} = \left(|x| > 2\right) \quad p_{\text{quadratic}}(x) = \max\left(|x| - 2, 0\right)^2. \tag{Г.15}$$

Метод подсчета нарушений не предоставляет никакой информации о градиенте для процесса оптимизации. Алгоритм оптимизации, инициализированный вне допустимого множества, выйдет из допустимой области, потому что функция $1 - x^2$ минимизируется путем перемещения на бесконечное расстояние влево или вправо от начала координат. Большая величина штрафов в этом случае не является основной проблемой; небольшие штрафы могут привести к аналогичным трудностям.

Метод квадратичного штрафа предоставляет информацию о градиенте для процесса оптимизации, направляя поиски в допустимую область. Для очень больших штрафов метод квадратичных штрафов даст большие значения градиента в недопустимой области. В этой задаче частная производная имеет вид.

$$\frac{\partial f}{\partial x} = -2x + \rho \begin{cases} 2\left(x - 2\right), \text{если } x > 2, \\ 2\left(x + 2\right), \text{если } x < -2, \\ 0 \text{ в противном случае}. \end{cases} \tag{Г.16}$$

При очень больших значениях ρ частная производная в недопустимой области также велика, что может создавать проблемы для методов оптимизации. Если оно невелико, то невыполнимые пункты могут быть недостаточно оштрафованы, что приведет к невозможным решениям.

Упражнение 11.1. Каждое ограничение в виде неравенства может быть либо активным, либо нет. Есть n ограничений в виде неравенств. Таким образом, нам не нужно изучать более 2^n комбинаций ограничений.

Упражнение 11.2. Симплекс-метод гарантированно либо улучшает целевую функцию с каждым шагом, либо сохраняет ее текущее значение. Любая задача линейного программирования имеет конечное число вершин. До тех пор, пока используется эвристика, наподобие правила Бланда, предотвращающее зацикливание, симплекс-метод должен сходиться к решению.

Упражнение 11.3. Можно добавить дополнительную переменную x_3 и минимизировать $6x_1 + 5x_2 + x_3$ при ограничениях $-3x_1 + 2x_2 + x_3 = -5$ и $x_3 \geq 0$.

Упражнение 11.4. Если текущая итерация x допустима, то $\mathbf{w}^{\mathrm{T}}\mathbf{x} = b \geq 0$. Мы хотим, чтобы следующая точка была допустимой, и поэтому требуем, чтобы выполнялось условие $\mathbf{w}^{\mathrm{T}}(\mathbf{x} + \alpha\mathbf{d}) \geq 0$. Если полученное значение α положительно, то α — верхняя граница длины шага. Если полученное значение α отрицательное, его можно игнорировать.

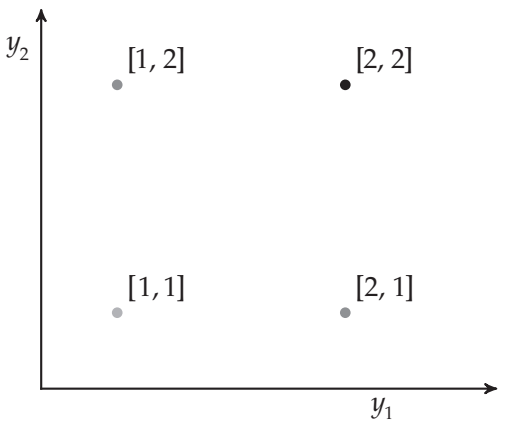
Упражнение 11.5. Можно переписать задачу в виде

$$\min_{\mathbf{x}} \mathbf{c}^{\mathrm{T}}\mathbf{x} - \mu\sum_{i} \ln\left(\mathbf{A}_{\{i\}}^{\mathrm{T}}\mathbf{x}\right). \tag{Г.17}$$

Упражнение 12.1. Метод взвешенной суммы не может найти оптимальные по Парето точки в невыпуклых областях границы Парето.

Упражнение 12.2. Методы, не относящиеся к популяционным, будут определять только одну точку на границе Парето. Граница Парето очень важна для поиска компромисса между набором очень хороших решений. Популяционные методы могут распределять точки по границе Парето и использоваться в качестве ее приближения.

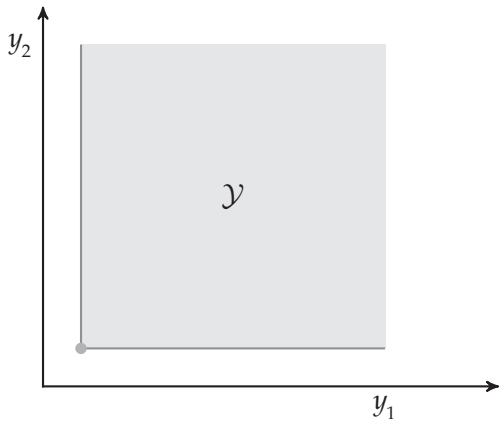
Упражнение 12.3.



Единственная оптимальная по Парето точка — [1, 1]. Точки [1, 2] и [2, 1] являются слабо оптимальными по Парето.

Упражнение 12.4. Градиент вектора представляет собой матрицу. Производные второго порядка требуют использования тензоров, а решение тензорного уравнения для выбора направления поиска часто обременительно в вычислительном отношении.

Упражнение 12.5. Единственная оптимальная по Парето точка — $\mathbf{y}$ = [0, 0]. Остальные точки на нижней левой границе слабо оптимальными по Парето.



Упражнение 12.6. Рассмотрим квадратное критериальное пространство из предыдущего упражнения. Если $\mathbf{w} = [0, 1]$, то первая целевая функция равна нулю, в результате чего весь нижний край пространства критерия имеет одинаковое значение. Как обсуждалось выше, только точка $\mathbf{y} = [0, 0]$ является оптимальной по Парето, остальные являются слабо оптимальными по Парето.

Упражнение 12.7. Например, если $\mathbf{y}^{\text{goal}}$ находится в наборе критериев, целевая функция задачи будет достигать минимума в точке $\mathbf{y}^{\text{goal}}$. Если $\mathbf{y}^{\text{goal}}$ также не является оптимальной по Парето, то решение также не будет оптимальным по Парето.

Упражнение 12.8. Метод ограничений устанавливает ограничения на все, кроме одной цели. Кривая Парето может быть сгенерирована путем изменения ограничений. Если мы ограничим первую цель, то каждая задача оптимизации примет следующий вид:

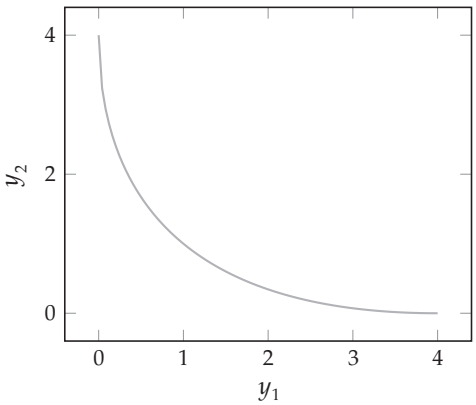
$$\min_x \left(x-2\right)^2 \tag{Г.18}$$

$$\text{при условии } x^2 \le c. \tag{Г.19}$$

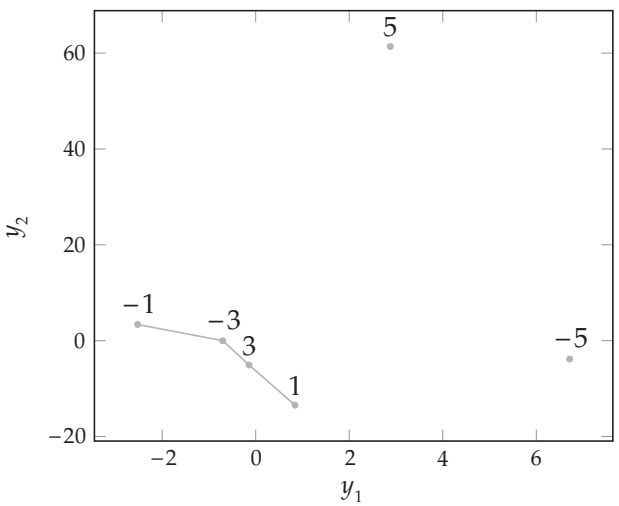
Ограничение может быть выполнено только для $c \ge 0$. Это позволяет переменной x изменяться между $\pm\sqrt{c}$. Первая цель оптимизируется путем минимизации отклонения x от 2. Таким образом, для заданного значения c мы получаем

$$x^* = \begin{cases} 2 \text{ при } c \ge 4, \\ \sqrt{c} \text{ при } c \in \left[0,\ 4\right], \\ \text{не определено в противном случае.} \end{cases} \tag{Г.20}$$

Итоговая кривая Парето имеет вид:



Упражнение 12.9. Критериальное пространство — это пространство значений целевой функции. В результате получается следующий рисунок.



Четыре точки находятся на приблизительной границе Парето, соответствующей нашим шести точкам выборки. Соответствующие расчетные точки: $x = \{-1, -3, 3, 1\}$.

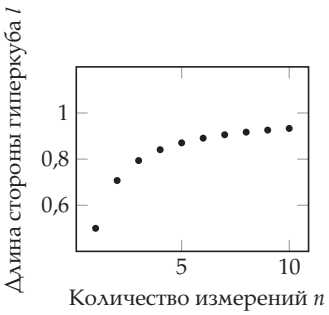
Упражнение 13.1. Одномерный единичный гиперкуб — это отрезок $x \in [0, 1]$, а его объем равен 1. В этом случае искомая длина стороны l равна 0,5. Двумерный единичный гиперкуб представляет собой декартово произведение единичных квадратов $x_i \in [0, 1]$, $i \in \{1, 2\}$ и имеет объем, равный единице. Площадь квадрата с длиной стороны l равна l^2, поэтому

$$l^2 = \frac{1}{2} \Rightarrow l = \frac{\sqrt{2}}{2} \approx 0,707. \tag{Г.21}$$

Объем гиперкуба в n-мерном пространстве равен l^n. Таким образом,

$$l^n = \frac{1}{2} \Rightarrow l = 2^{-1/n}. \tag{Г.22}$$

Длина стороны стремится к единице.



Упражнение 13.2. Вероятность того, что точка, выбранная случайным образом, находится в пределах расстояния, равного ε от поверхности, равна отношению объемов. Таким образом,

$$P\left(\|x\|_2 > 1 - \varepsilon\right) = 1 - P\left(\|x\|_2 < 1 - \varepsilon\right) = 1 - \left(1 - \varepsilon\right)^n \to 1. \tag{Г.23}$$

Упражнение 13.3.

```
function pairwise_distances(X, p = 2)
    m = length(X)
    [norm(X[i] - X[j], p) for i in 1 : (m - 1) for j in (i + 1) : m]
end
function phiq(X, q = 1, p = 2)
    dists = pairwise_distances(X, p)
    return sum(dists .^ (-q)) ^ (1/q)
end
X = [[cos(2π * i / 10), sin(2π * i / 10)] for i in 1 : 10]
@show phiq(X, 2)

phiq(X, 2) = 6.422616289332565
```

Нет. Критерий Морриса–Митчелла полностью основан на попарных расстояниях. Сдвиг всех точек на одну и ту же величину не меняет попарных расстояний и, следовательно, не изменит $\Phi_2(X)$.

Упражнение 13.4. Рациональное число можно записать в виде дроби из двух целых чисел a/b. Отсюда следует, что последовательность повторяется каждые b итераций.

$$x^{(k+1)} = x^{(k)} + \frac{a}{b} \left(\mathrm{mod}\, 1\right),$$

$$x^{(k)} = x^{(0)} + k\frac{a}{b}\left(\text{mod } 1\right) =$$

$$= x^{(0)} + k\frac{a}{b} + a \ (\text{mod } 1) =$$

$$= x^{(0)} + \left(k + b\right)\frac{a}{b}\left(\text{mod } 1\right) =$$

$$= x^{(k+b)}.$$

Упражнение 14.1. Целевая функция линейной регрессии имеет вид

$$\left\| \mathbf{y} - \mathbf{X\theta} \right\|_2^2.$$

Вычислим градиент и приравняем его к нулю:

$$\nabla(\mathbf{y} - \mathbf{X\theta})^{\mathrm{T}}(\mathbf{y} - \mathbf{X\theta}) = -2\mathbf{X}^{\mathrm{T}}(\mathbf{y} - \mathbf{X\theta}) = \mathbf{0}.$$

В результате получим уравнение нормали:

$$\mathbf{X}^{\mathrm{T}}\mathbf{X\theta} = \mathbf{X}^{\mathrm{T}}\mathbf{y}.$$

Упражнение 14.2. Как правило, при наличии большего количества данных следует использовать более описательные модели. Если доступно только несколько выборок, то такие модели подвержены переобучению, и следует использовать более простую модель (с меньшими степенями свободы).

Упражнение 14.3. У модели может быть очень большое количество параметров. В таком случае полученная линейная система будет слишком большой и потребует памяти, которая квадратично растет в зависимости от размерности пространства параметров. Итерационные процедуры, такие как стохастический градиентный спуск, требуют памяти, размер которой растет линейно в зависимости от размерности пространства параметров, и иногда они являются единственным жизнеспособным решением.

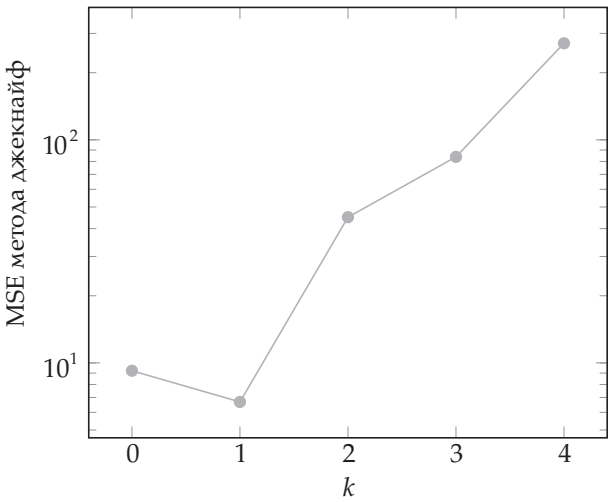
Упражнение 14.4. Оценка по принципу поэлементной перекрестной проверки получается путем выполнения *k*-блочной перекрестной проверки, где *k* равно количеству выборок в *X*. Это означает, что мы должны выполнить 4-блочную перекрестную проверку для каждой степени полинома.

Наименьшее среднеквадратичное отклонение получено для линейной модели, $k = 1$. Мы подбираем новую линейную модель для полного множества данных, чтобы получить наши параметры.

```
X = [[1], [2], [3] ,[4]]
y = [0, 5, 4, 6]
bases = polynomial_bases(1, 1)
B = [b(x) for x in X, b in bases]
```

```
θ = pinv(B)*y
@show θ
```

```
θ = [-0.5, 1.7]
```



Упражнение 15.1. Гауссовские процессы являются непараметрическими, тогда как модели линейной регрессии являются параметрическими. Это означает, что число степеней свободы модели растет с увеличением объема данных, что позволяет гауссовскому процессу поддерживать баланс между смещением и дисперсией в процессе оптимизации.

Упражнение 15.2. Получение условного распределения гауссовского процесса требует решения уравнения (15.13). Самая дорогая операция — обращение $(m \times m)$-матрицы $\mathbf{K}(X, X)$, имеющая сложность $O(m^3)$.

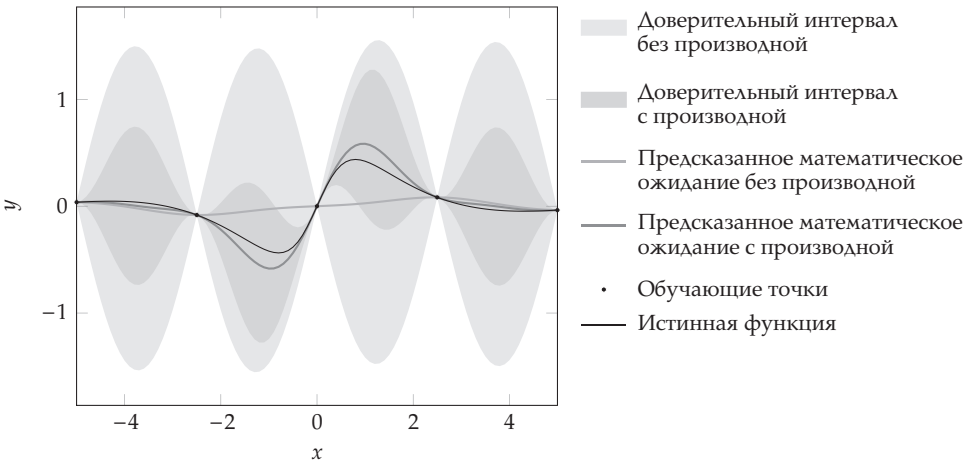
Упражнение 15.3. Производная от f имеет вид

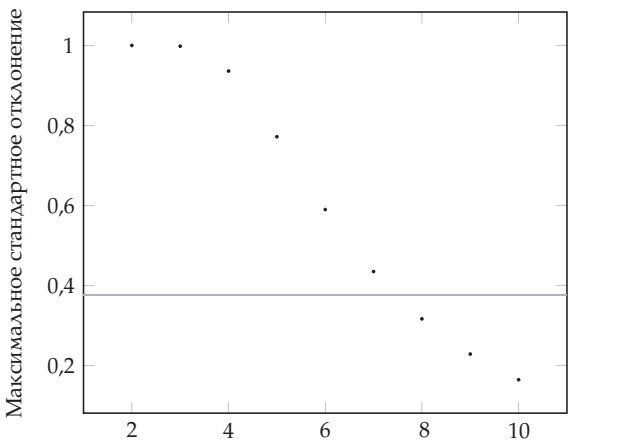
$$\frac{\left(x^2+1\right)\cos x - 2x\sin x}{\left(x^2+1\right)^2}.$$

Ниже мы построим прогностическое распределение для гауссовских процессов с информацией о производной и без нее. Максимальное стандартное отклонение в прогностическом распределении на отрезке [–5, 5] для гауссовского процесса с информацией о производной составляет приблизительно 0,377 при $x \approx \pm 3{,}8$.

Учет информации о производной значительно уменьшает доверительный интервал, потому что для прогноза теперь имеется больше информации. Ниже приведен график максимального стандартного отклонения прогностического распределения на отрезке [–5, 5] для гауссовских процессов без информации о производной с различным числом равномерно распределенных значений. Для того чтобы

превзойти гауссовский процесс с информацией о производной, необходимо как минимум восемь точек.





Упражнение 15.4. Это можно вывести с помощью следующих выражений:

$$
\begin{aligned}
k_{f\nabla}(\mathbf{x}, \mathbf{x}')_i &= \operatorname{cov}\left(f(\mathbf{x}),\ \frac{\partial}{\partial x_i'} f(\mathbf{x}') \right) = \\
&= \mathbb{E}\left[\left(f(\mathbf{x}) - \mathbb{E}\left[f(\mathbf{x}) \right] \right) \left(\frac{\partial}{\partial x_i'} f(\mathbf{x}') - \mathbb{E}\left[\frac{\partial}{\partial x_i'} f(\mathbf{x}') \right] \right) \right] = \\
&= \mathbb{E}\left[\left(f(\mathbf{x}) - \mathbb{E}\left[f(\mathbf{x}) \right] \right) \left(\frac{\partial}{\partial x_i'} f(\mathbf{x}') - \frac{\partial}{\partial x_i'} \mathbb{E}\left[f(\mathbf{x}') \right] \right) \right] = \\
&= \frac{\partial}{\partial x_i'} \mathbb{E}\left[\left(f(\mathbf{x}) - \mathbb{E}\left[f(\mathbf{x}) \right] \right) \left(f(\mathbf{x}') - \mathbb{E}\left[f(\mathbf{x}') \right] \right) \right] =
\end{aligned}
$$

$$= \frac{\partial}{\partial x_i'} \operatorname{cov}\left(f(\mathbf{x}), f(\mathbf{x}')\right) =$$
$$= \frac{\partial}{\partial x_i'} k_{ff}(\mathbf{x}, \mathbf{x}'),$$

где мы использовали тот факт, что $\mathbb{E}\left[\frac{\partial}{\partial x} f\right] = \frac{\partial}{\partial x}\mathbb{E}[f]$. Можно убедиться, что это так:

$$\mathbb{E}\left[\frac{\partial}{\partial x} f\right] = \mathbb{E}\left[\lim_{h\to 0} \frac{f(x+h) - f(x)}{h}\right] =$$
$$= \lim_{h\to 0} \mathbb{E}\left[\frac{f(x+h) - f(x)}{h}\right] =$$
$$= \lim_{h\to 0} \frac{1}{h}\left(\mathbb{E}[f(x+h)] - \mathbb{E}[f(x)]\right) =$$
$$= \frac{\partial}{\partial x}\mathbb{E}[f],$$

при условии, что целевая функция является дифференцируемой.

Упражнение 15.5. Запишем совместное нормальное распределение как

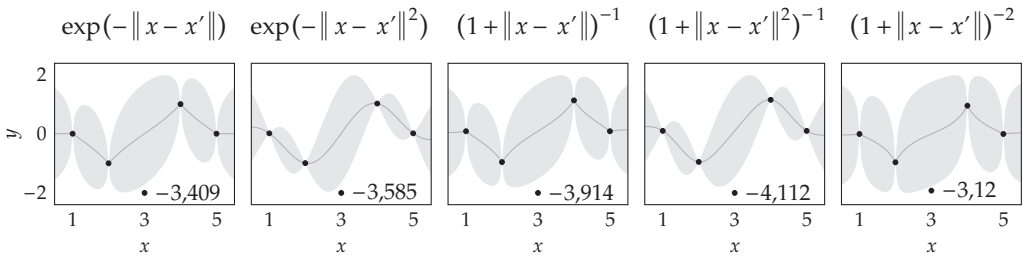
$$\begin{bmatrix} a \\ b \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_a \\ \mu_b \end{bmatrix}, \begin{bmatrix} \nu_a & \nu_c \\ \nu_c & \nu_b \end{bmatrix}\right). \tag{Г.24}$$

Маргинальное распределение по a представляет собой распределение $\mathcal{N}(\mu_a, \nu_a)$, которое имеет дисперсию ν_a. Условное распределение по a имеет дисперсию $\nu_a - \nu_c^2/\nu_b$. Мы знаем, что дисперсия ν_a должна быть положительной, чтобы исходная ковариационная матрица была положительно определенной. Таким образом, ν_c^2/ν_b — положительная величина и $\nu_a - \nu_c^2/\nu_b \leq \nu_a$.

Интуитивно понятно, что дисперсия условного распределения не может быть больше дисперсии маргинального распределения, поскольку условное распределение включает в себя больше информации о величине a. Если величины a и b коррелируют, то знание значения b увеличивает знание об a и уменьшает нашу неопределенность.

Упражнение 15.6. Можно настроить параметры для нашей функции ядра или поменять функции ядра, используя оценку ошибки обобщения или максимизируя вероятность наблюдаемых данных.

Упражнение 15.7. Максимизация произведения вероятностей эквивалентна максимизации суммы логарифмических вероятностей. Ниже приведены логарифмические вероятности третьей точки при фиксированных остальных точках для разных ядер.



Вычисление этих значений для всех пяти групп дает общие значения логарифмического правдоподобия:

$$\begin{aligned}
\exp\left(-\|x-x'\|\right) &\rightarrow -8{,}688,\\
\exp\left(-\|x-x'\|^2\right) &\rightarrow -9{,}010,\\
\left(1+\|x-x'\|\right)^{-1} &\rightarrow -9{,}579,\\
\left(1+\|x-x'\|^2\right)^{-1} &\rightarrow -10{,}195,\\
\left(1+\|x-x'\|\right)^{-2} &\rightarrow -8{,}088.
\end{aligned}$$

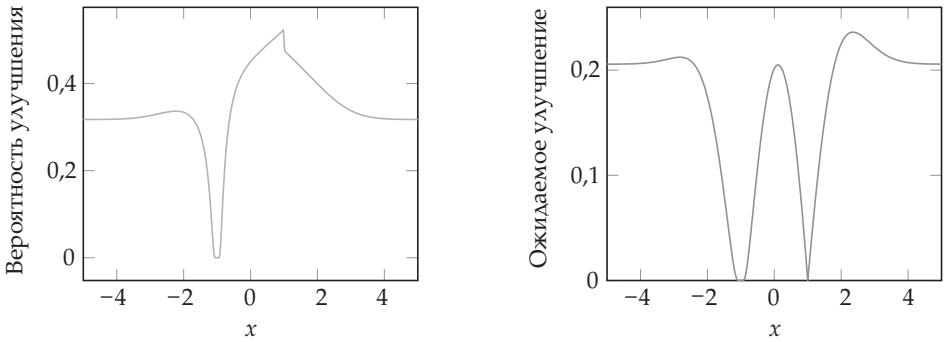
Отсюда следует, что ядром, максимизирующим правдоподобие при поэлементной перекрестной проверке, является рационально-квадратичное ядро $\left(1+\right.$ $\left.+\|x-x'\|\right)^{-2}$.

Упражнение 16.1. Предположим, что у нас есть гауссовский процесс с нулевым математическим ожиданием, и мы начинаем с одной точки $\mathbf{x}^{(1)}$, в которой $\mathbf{y}^{(1)}=-10$. Предсказанное математическое ожидание достигает единственного глобального минимума в точке $\mathbf{x}^{(1)}$. Оптимизация на основе предсказания будет продолжать выбор в точке $\mathbf{x}^{(1)}$.

Упражнение 16.2. Исследование, основанное на ошибках, тратит впустую усилия на уменьшение дисперсии и не стремится активно минимизировать функцию.

Упражнение 16.3. Гауссовский процесс выглядит следующим образом:

Вероятность улучшения и ожидаемое улучшение выглядят так.



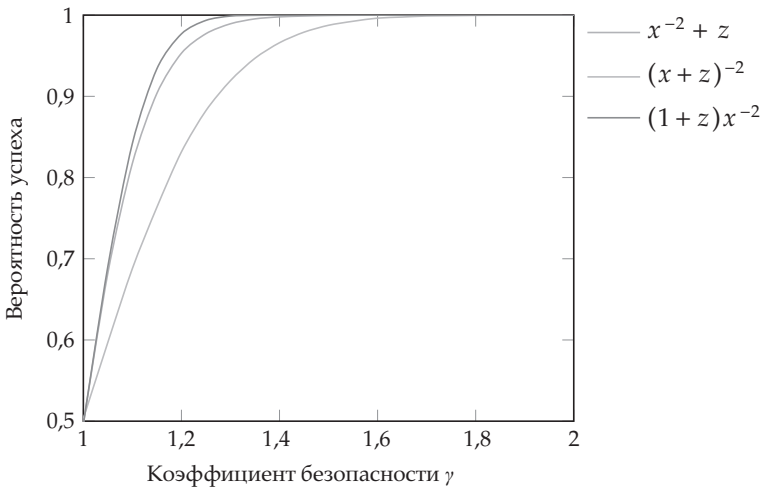
Максимальная вероятность улучшения составляет при $x = 0{,}98$ для $P = 0{,}523$. Максимальное ожидаемое улучшение достигается при $x = 2{,}36$ для $E = 0{,}236$.

Упражнение 17.1. Наша цель — минимизировать функцию $\mathbb{E}_{z\sim\mathcal{N}}\left[f\left(x+z\right)\right] - -\sqrt{\mathrm{Var}_{z\sim\mathcal{N}}\left[f\left(x+z\right)\right]}$. Первое слагаемое, соответствующее математическому ожиданию, минимизируется в расчетной точке a. Второе слагаемое, соответствующее стандартному отклонению, также максимизируется в расчетной точке a, поскольку возмущения проекта в этом месте вызывают значительные изменения в выходных данных. Таким образом, оптимальный проект достигается в точке $x^* = a$.

Упражнение 17.2. Задача детерминированной оптимизации имеет вид

$$\min_x x^2$$

при условии $\gamma x^{-2} \leq 1$.



Оптимальная длина поперечного сечения как функция коэффициента безопасности равна $x = \sqrt{\gamma}$. Таким образом, можно подставить $\sqrt{\gamma}$ вместо длины поперечного сечения в каждой формулировке неопределенности и оценке вероятности того, что проект не потерпит неудачу. Обратите внимание: все конструкции имеют 50%-ную вероятность отказа, когда коэффициент безопасности равен единице, из-за симметрии нормального распределения.

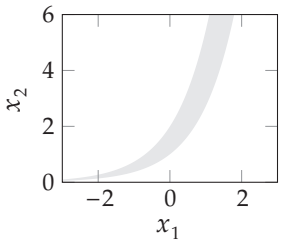


Рис. Г.1. Достижимая область без шума

Упражнение 17.3. На рис. Г.1 показана практически свободная от шума область. В отсутствие шума координата x_1 оптимальной точки является бесконечно отрицательной. У нас есть шум, и мы решили не принимать какие-либо выбросы с магнитудой больше 6. Такие выбросы встречаются примерно в $1{,}973 \cdot 10^{-7}$% случаев. Допустимая область для x_2 лежит между e^{x_1} и $2e^{x_1}$. Шум симметричный, поэтому наиболее надежный выбор для x_2 равен $1{,}5e^{x_1}$.

Ширина допустимой области для x_2 равна e^{x_1}, и эта функция растет с увеличением x_1. Целевая функция также растет с увеличением x_1, поэтому оптимальное значение x_1 является наименьшим, при котором ширина допустимой области составляет не менее 12. Это приводит к ответу: $x_1 = \ln 12 \approx 2{,}485$ и $x_2 = 18$.

Упражнение 18.1. Для вычисления этих значений может использоваться кумулятивная функция распределения.

```
julia> using Distributions
julia> N = Normal(0, 1);
julia> cdf(N, 1) - cdf(N, -1)
0.6826894921370861
julia> cdf(N, 1)
0.841344746068543
```

Таким образом, наша выборка лежит на расстоянии не более одного стандартного отклонения от математического ожидания влево (68,3%) и вправо (84,1%).

Упражнение 18.2. Начнем с подстановки в определении выборочного среднего:

$$\mathrm{Var}\left(\hat{\mu}\right) = \mathrm{Var}\left(\frac{x^{(1)} + x^{(2)} + \ \ldots \ + x^{(m)}}{m}\right) =$$

$$= \mathrm{Var}\left(\frac{1}{m}x^{(1)} + \frac{1}{m}x^{(2)} + \ \ldots \ + \frac{1}{m}x^{(m)}\right).$$

Дисперсия суммы двух независимых переменных является суммой дисперсий двух переменных. Это следует из следующих выражений:

$$\mathrm{Var}\left(\hat{\mu}\right) = \mathrm{Var}\left(\frac{1}{m}x^{(1)}\right) + \mathrm{Var}\left(\frac{1}{m}x^{(2)}\right) + \ \ldots \ + \mathrm{Var}\left(\frac{1}{m}x^{(m)}\right) =$$

$$= \frac{1}{m^2}\mathrm{Var}\left(x^{(1)}\right) + \frac{1}{m^2}\mathrm{Var}\left(x^{(2)}\right) + \ \ldots \ + \frac{1}{m^2}\mathrm{Var}\left(x^{(m)}\right) =$$

$$= \frac{1}{m^2}\left(\nu + \nu + \ \ldots \ + \nu\right) =$$

$$= \frac{1}{m^2}\left(m\nu\right) =$$

$$= \frac{\nu}{m}.$$

Упражнение 18.3. Трехчленное рекуррентное соотношение для ортогональных многочленов является центральным для их построения и использования. Ключом к выводу является то, что множитель z можно переносить с одного полинома на другой.

$$\int_{\mathcal{Z}} \left(zb_i\left(z\right)\right)b_j\left(z\right)p\left(z\right)dz = \int_{\mathcal{Z}} b_i\left(z\right)\left(zb_j\left(z\right)\right)p\left(z\right)dz.$$

Мы должны показать, что функции

$$b_{i+1}\left(z\right) = \begin{cases}\left(z - \alpha_i\right)b_i\left(z\right) & \text{для } i = 1,\\ \left(z - \alpha_i\right)b_i\left(z\right) - \beta_i b_{i-1}\left(z\right) & \text{для } i > 1,\end{cases}$$

порождают ортогональные полиномы.

Заметим, что $b_{i+1} - zb_i$ — это полином степени не более i, поэтому его можно записать как линейную комбинацию первых i ортогональных полиномов

$$b_{i+1}\left(z\right) - zb_i\left(z\right) = -\alpha_i b_i\left(z\right) - \beta_i b_{i-1}\left(z\right) + \sum_{j=0}^{i-2}\gamma_{ij}b_j\left(z\right)$$

с константами α_i, β_i и γ_{ij}.

Умножим обе части на b_i и p, а затем интегрируем результат:

$$\int_{\mathcal{Z}} \left(b_{i+1}(z) - zb_i(z)\right) b_i(z)\, p(z) dz =$$

$$= \int_{\mathcal{Z}} \left(-\alpha_i b_i(z) - \beta_i b_{i-1}(z) + \sum_{j=0}^{i-2} \gamma_{ij} b_j(z) \right) b_i(z)\, p(z) dz,$$

$$\int_{\mathcal{Z}} b_{i+1}(z) b_i(z)\, p(z) dz - \int_{\mathcal{Z}} z b_i(z) b_i(z)\, p(z) dz =$$

$$= -\int_{\mathcal{Z}} \alpha_i b_i(z) b_i(z)\, p(z)\, dz -$$

$$- \int_{\mathcal{Z}} \beta_i b_{i-1}(z) b_i(z)\, p(z) dz +$$

$$+ \sum_{j=0}^{i-2} \int_{\mathcal{Z}} \gamma_{ij} b_j(z) b_i(z)\, p(z) dz -$$

$$- \int_{\mathcal{Z}} z b_i^2(z)\, p(z) dz =$$

$$= -\alpha_i \int_{\mathcal{Z}} b_i^2(z)\, p(z) dz.$$

В результате получаем выражение для α_i:

$$\alpha_i = \frac{\int_{\mathcal{Z}} z b_i^2(z)\, p(z) dz}{\int_{\mathcal{Z}} b_i^2(z)\, p(z) dz}.$$

Выражение для β_i при $i \geq 1$ получается аналогично, только перед интегрированием умножение обеих частей производится на b_{i-1} и p.

Умножая обе стороны на b_k при $k < i - 1$, аналогично получаем:

$$-\int_{\mathcal{Z}} z b_i(z) b_k(z)\, p(z) dz = \gamma_{ik} \int_{\mathcal{Z}} b_k^2(z)\, p(z) dz.$$

Применяя перенос множителя, получаем:

$$\int_{\mathcal{Z}} z b_i(z) b_k(z)\, p(z) dz = \int_{\mathcal{Z}} b_i(z) \left(z b_k(z)\right) p(z) dz = 0.$$

Поскольку $z b_k(z)$ — полином порядка не более чем $i - 1$, то вследствие ортогональности этот интеграл равен нулю. Отсюда следует, что все γ_{ik} равны нулю, и мы получаем трехчленное рекуррентное соотношение.

Упражнение 18.4. Можно получить градиентное приближение, используя частную производную f по компоненту x_i:

$$\frac{\partial}{\partial x_i} f(\mathbf{x}, \mathbf{z}) \approx b_1(\mathbf{z}) \frac{\partial}{\partial x_i} \theta_1(\mathbf{x}) + \ \ldots \ + b_k(\mathbf{z}) \frac{\partial}{\partial x_i} \theta_k(\mathbf{x}).$$

Если у нас есть m выборок, то можно записать эти частные производные в виде матрицы.

$$\begin{bmatrix} \frac{\partial}{\partial x_i} f\left(\mathbf{x}, \mathbf{z}^{(1)}\right) \\ \vdots \\ \frac{\partial}{\partial x_i} f\left(\mathbf{x}, \mathbf{z}^{(m)}\right) \end{bmatrix} \approx \begin{bmatrix} b_1\left(\mathbf{z}^{(1)}\right) & \cdots & b_k\left(\mathbf{z}^{(1)}\right) \\ \vdots & \ddots & \vdots \\ b_1\left(\mathbf{z}^{(m)}\right) & \cdots & b_k\left(\mathbf{z}^{(m)}\right) \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial x_i} \theta_1(\mathbf{x}) \\ \vdots \\ \frac{\partial}{\partial x_i} \theta_k(\mathbf{x}) \end{bmatrix}.$$

Решая это уравнение относительно аппроксимаций $\frac{\partial}{\partial x_i}\theta_1(\mathbf{x})$, ..., $\frac{\partial}{\partial x_i}\theta_k(\mathbf{x})$ с помощью псевдообращения, получаем:

$$\begin{bmatrix} \frac{\partial}{\partial x_i} \theta_1(\mathbf{x}) \\ \vdots \\ \frac{\partial}{\partial x_i} \theta_k(\mathbf{x}) \end{bmatrix} \approx \begin{bmatrix} b_1\left(\mathbf{z}^{(1)}\right) & \cdots & b_k\left(\mathbf{z}^{(1)}\right) \\ \vdots & \ddots & \vdots \\ b_1\left(\mathbf{z}^{(m)}\right) & \cdots & b_k\left(\mathbf{z}^{(m)}\right) \end{bmatrix}^{+} \begin{bmatrix} \frac{\partial}{\partial x_i} f\left(\mathbf{x}, \mathbf{z}^{(1)}\right) \\ \vdots \\ \frac{\partial}{\partial x_i} f\left(\mathbf{x}, \mathbf{z}^{(m)}\right) \end{bmatrix}.$$

Упражнение 18.5. Расчетное математическое ожидание и дисперсия имеют коэффициенты, которые зависят от проектных переменных.

$$\begin{aligned} \hat{\mu}(\mathbf{x}) &= \theta_1(\mathbf{x}), \\ \hat{v}(\mathbf{x}) &= \sum_{i=2}^{k} \theta_i^2(\mathbf{x}) \int_{\mathcal{Z}} b_i(\mathbf{z})^2 \, p(\mathbf{z}) d\mathbf{z}. \end{aligned} \tag{Г.25}$$

Частичная производная f_{mod} по i-му компоненту имеет вид:

$$\frac{\partial}{\partial x_i} f_{\text{mod}}(\mathbf{x}) = \alpha \frac{\partial \theta_1(\mathbf{x})}{\partial x_i} + 2(1-\alpha) \sum_{i=2}^{k} \theta_i(\mathbf{x}) \frac{\partial \theta_i(\mathbf{x})}{x_i} \int_{\mathcal{Z}} b_i(\mathbf{z})^2 \, p(\mathbf{z}) d\mathbf{z}. \tag{Г.26}$$

Для вычисления уравнения (Г.26) требуется градиент коэффициентов по $\mathbf{x}$, который вычисляется в упражнении 18.4.

Упражнение 19.1. Перебор обходит все конструкции. Каждый компонент может быть истинным или ложным, что приводит к 2^n возможным проектам в худшем случае. Эта задача имеет $2^3 = 8$ возможных конструкций.

```
f(x) = (!x[1] || x[3]) && (x[2] || !x[3]) && (!x[1] || !x[2])
using IterTools
for x in product([true, false], [true, false], [true, false])
   if f(x)
      @show(x)
      break
   end
```

```
end

x = (false, true, true)
```

Упражнение 19.2. Задача о булевой выполнимости просто ищет правильное решение. Таким образом, мы устанавливаем **c** равным нулю.

Ограничения более интересны. Как и во всех задачах целочисленного линейного программирования, вектор **x** должен быть неотрицательным и целочисленным. Кроме того, мы считаем, что 1 соответствует значению `true`, а 0 — `false`, и вводим ограничение $\mathbf{x} \leq \mathbf{1}$.

Обратите внимание на то, что в целевой функции операторы ^ делят ее на отдельные логические выражения, каждое из которых должно быть истинным. Мы преобразовываем эти выражения в линейные ограничения.

$$\begin{aligned} x_1 &\Rightarrow x_1 \geq 1, \\ x_2 \vee \neg x_3 &\Rightarrow x_2 + (1 - x_3) \geq 1, \\ \neg x_1 \vee \neg x_2 &\Rightarrow (1 - x_1) + (1 - x_2) \geq 1, \end{aligned}$$

где каждое выражение должно выполняться (≥ 1), а отрицание переменной $\neg x_i$ превращается в выражение $1 - x_i$.

Искомая задача целочисленного линейного программирования имеет вид:

$$\begin{aligned} &\min_{\mathbf{x}} 0 \\ &\text{при условии} \quad x_1 \geq 1, \\ &\qquad\qquad\quad x_2 - x_3 \geq 0, \\ &\qquad\qquad -x_1 - x_2 \geq -1, \\ &\qquad\qquad\quad \mathbf{x} \in \mathbb{N}^3 \end{aligned}$$

Этот подход является общим и может использоваться для преобразования любой задачи о булевой выполнимости в задачу целочисленного линейного программирования.

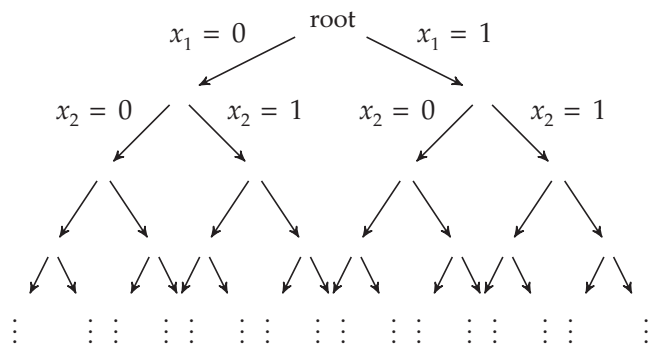
Упражнение 19.3. Вполне унимодулярные матрицы имеют обратные матрицы, которые также являются целочисленными матрицами. Задачи целочисленного программирования, для которых матрица **A** вполне унимодулярная, а **b** — целочисленный вектор, можно точно решить с помощью симплекс-метода.

Матрица является вполне унимодулярной, если каждая ее квадратная неособая подматрица унимодулярная. Отдельный элемент матрицы является квадратной подматрицей. Определитель матрицы 1×1 является абсолютным значением ее единственного элемента. Подматрица с одним элементом является унимодулярной, только если она имеет определитель, равный ± 1, что имеет место только для элементов, равных 1. Подматрица с одним элементом также может быть невырожденной, т.е. может содержать нули. Никакие другие элементы не допуска-

ются, и, следовательно, каждая вполне унимодулярная матрица содержит только элементы, которые равны 0, 1 или –1.

Упражнение 19.4. Метод ветвей и границ требует, чтобы мы могли выполнять операции ветвления и ограничения применительно к нашему проекту [87]. Принимаемые решения в задаче о рюкзаке 0–1 заключаются в том, стоит ли уложить либо удалить каждый предмет. Следовательно, каждый предмет представляет собой ветвь; либо предмет уложен в рюкзак, либо удален из него.

Для каждого такого перечисления строится дерево в соответствии с указанным ниже правилом.



Каждый узел представляет собой подзадачу, в которую определенные предметы уже уложены или удалены. Подзадача имеет связанные переменные решения, значения и веса удаленных объектов, причем емкость эффективно уменьшается на общий вес включенных элементов.

Метод ветвей и границ избегает построения всего дерева, используя информацию о границах. Оценка может быть построена путем решения ослабленной версии задачи о рюкзаке. Эта *задача о дробном ранце* позволяет укладывать в рюкзак части предметов, т.е. $0 \le x_i \le 1$.

Ослабленную задачу о рюкзаке можно эффективно решить с помощью жадного подхода. Предметы добавляются по одному, при этом на каждом шаге выбирается предмет с наибольшим отношением стоимости к весу. Если оставшейся емкости рюкзака достаточно, предмету присваивается значение $x_i = 1$. Если нет, то ему присваивается такое дробное значение, что оставшаяся емкость наполняется, а для всех остальных предметов $x_i = 0$.

Начнем ветвление с первого предмета. Поддереву с $x_1 = 0$ соответствует подзадача:

$$\min_{\mathbf{x}_{2:6}} \; -\sum_{i=2}^{6} v_i x_i$$

$$\text{при условии } \sum_{i=2}^{6} w_i x_i \le 20,$$

тогда как поддереву с $x_1 = 1$ соответствует подзадача:

$$\min_{\mathbf{x}_{2:6}} \quad -9 - \sum_{i=2}^{6} v_i x_i$$

$$\text{при условии} \sum_{i=2}^{6} w_i x_i \le 13.$$

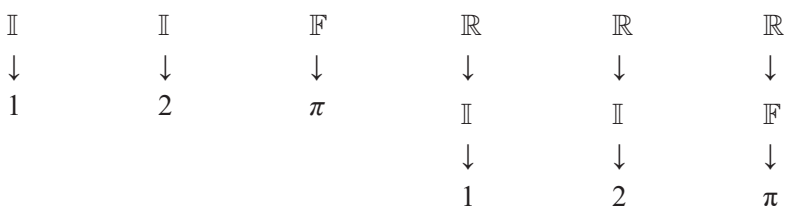
Можно построить нижнюю границу для обоих поддеревьев, используя жадный подход. Сортируем оставшиеся предметы по весу.

предмет:	6	4	5	3	2
отношение:	3/4	3/5	5/9	2/4	4/8
	0,75	0,6	0,556	0,5	0,5

В поддереве с $x_1 = 0$ мы целиком укладываем элементы 6, 4 и 5. Затем мы частично укладываем предмет 3, потому что у нас есть оставшаяся емкость, равная 2, и устанавливаем $x_3 = 2/4 = 0{,}5$. Таким образом, нижняя граница равна $-(3 + 5 + 3 + 0{,}5 \cdot 2) = -12$.

В поддереве с $x_1 = 1$ мы целиком укладываем элементы 6 и 4 и частично укладываем элемент 5 для $x_5 = 4/9$. Следовательно, нижняя граница равна $-(3 + 5 + (4/9) \cdot 3) \approx -18{,}333$. Поддерево с $x_1 = 1$ имеет лучшую нижнюю границу, поэтому алгоритм продолжает работу, разбивая эту подзадачу. Окончательное решение равно $\mathbf{x} = [1, 0, 0, 0, 1, 1]$.

Упражнение 20.1. Можно сгенерировать шесть деревьев выражений.



Упражнение 20.2. Существует только одно выражение нулевой высоты, и это пустое выражение. Обозначим его как $a_0 = 1$. Точно так же существует только одно выражение высоты 1, и это выражение {}. Обозначим его как $a_1 = 1$. Существуют три выражения глубины 2, 21 — глубины 3 и т.д.

Предположим, что мы построили все выражения до высоты h. Все выражения высоты $h + 1$ могут быть построены с использованием корневого узла с левым и правым подвыражениями, выбранными в соответствии со следующими правилами.

1. Левое выражение имеет высоту h, правое выражение — меньше h.

2. Правое выражение имеет высоту h, а левое — меньше h.

3. Левое и правое выражения имеют высоту h.

Отсюда следует, что количество выражений высоты $h + 1$ равно

$$a_{h+1} = 2a_h\left(a_0 + \ldots + a_{h-1}\right) + a_h^2.$$

Упражнение 20.3. Можно использовать следующую грамматику и начальный символ $\mathbb{I}$.

$\mathbb{I} \mapsto \mathbb{D} + 10 \times \mathbb{I}$
$\mathbb{D} \mapsto 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

Упражнение 20.4. Создание грамматик без исключений может быть сложной задачей. Таких проблем можно избежать, если перехватывать исключения в целевой функции и соответствующим образом штрафовать их.

Упражнение 20.5. Существует множество причин, по которым нужно ограничивать типы переменных, которыми манипулируют во время оптимизации выражений. Многие операторы корректны только для определенных входных данных[2], а умножение матриц требует, чтобы размеры матрицы были совместимыми. Физическая размерность переменных — это еще одна проблема. Грамматика должна основываться на единицах входных значений и допустимых операциях, которые могут быть выполнены над ними.

Например, выражение $x \times y$, где переменная x измеряется в единицах измерения $\mathrm{kg}^a\mathrm{m}^b\mathrm{s}^c$, а y — в единицах измерения $\mathrm{kg}^d\mathrm{m}^e\mathrm{s}^f$, даст значение с единицами $\mathrm{kg}^{a+d}\mathrm{m}^{b+e}\mathrm{s}^{c+f}$. Извлечение квадратного корня из x даст значение в единицах $\mathrm{kg}^{a/2}\mathrm{m}^{b/2}\mathrm{s}^{c/2}$. Кроме того, такие операции, как sin, могут быть применены только к безразмерным входным данным.

Один из подходов к обработке физических единиц состоит в том, чтобы связать n-кортеж с каждым узлом в дереве выражений. Кортеж записывает показатель степени по отношению к допустимым элементарным единицам, которые задаются пользователем. Если элементарные единицы измерения — это масса, длина и время, то каждый узел будет обладать 3-элементным кортежем (a, b, c) для представления единиц измерения $\mathrm{kg}^a\mathrm{m}^b\mathrm{s}^c$. При назначении правил вывода соответствующая грамматика должна учитывать эти единицы.[3]

Упражнение 20.6. Грамматика может быть закодирована с использованием пакета **`ExprRules.jl`** с применением строки:

```
grammar = @grammar begin
   S = NP * " " * VP
   NP = ADJ * " " * NP
   NP = ADJ * " " * N
   VP = V * " " * ADV
```

[2] Например, что квадратный корень не извлекался из отрицательных чисел.

[3] Более подробный обзор см. в [127].

```
    ADJ = |(["a", "the", "big", "little", "blue", "red"])
    N = |(["mouse", "cat", "dog", "pony"])
    V = |(["ran", "sat", "slept", "ate"])
    ADV = |(["quietly", "quickly", "soundly", "happily"])
end
```

Чтобы получить решение, можно использовать метод фенотипа.

```
eval(phenotype([2, 10, 19, 0, 6], grammar, :S)[1], grammar)
```

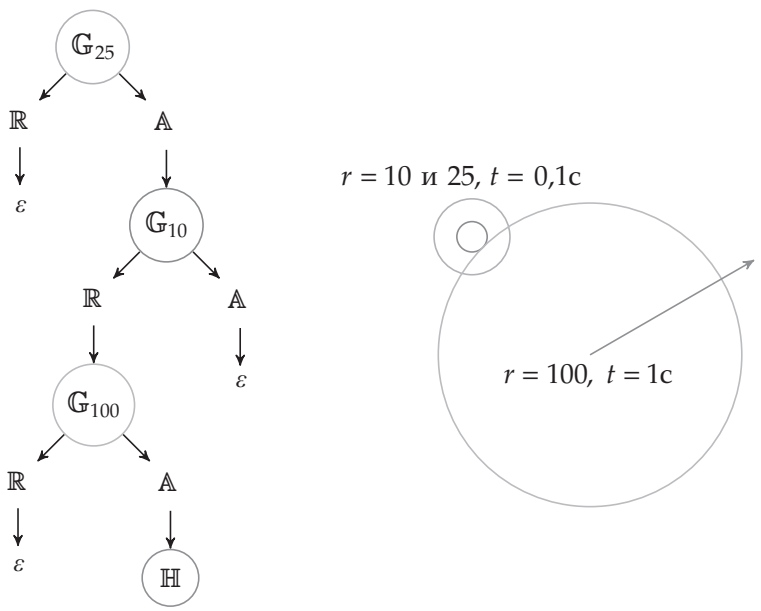
Фенотип имеет вид: “little dog ate quickly”.

Упражнение 20.7. Определим грамматику для задачи о часах. Пусть $\mathbb{G}_r$ — символ шестерни радиуса r, $\mathbb{A}$ — ось, $\mathbb{R}$ — обод, а $\mathbb{H}$ — стрелка. Наша грамматика имеет вид

$$\mathbb{G}_r \mapsto \mathbb{R}\ \mathbb{A} \mid \varepsilon$$
$$\mathbb{R} \mapsto \mathbb{R}\ \mathbb{R} \mid \mathbb{G}_r \mid \varepsilon$$
$$\mathbb{A} \mapsto \mathbb{A}\ \mathbb{A} \mid \mathbb{G}_r \mid \mathbb{H} \mid \varepsilon$$

Это позволяет каждой шестерне иметь любое количество дочерних ободов и осей. Выражение ε — это пустой терминальный символ.

Часы с одной секундной стрелкой могут быть сконструированы в соответствии с деревом:



Обратите внимание на то, что грамматика не вычисляет период вращения шестерен. Это задача решается целевой функцией. Для возврата периодов вращения всех стрелок может быть написана рекурсивная процедура. Список пе-

риодов вращения впоследствии используется для вычисления значения целевой функции.

Упражнение 20.8. Любой из методов, описанных в этой главе, можно использовать для завершения головоломки о четырех четверках. Простой подход — использовать метод Монте-Карло на подходящей грамматике. Выборочные выражения с ровно четырьмя символами вычисляются и, если необходимо, записываются. Эта процедура повторяется до тех пор, пока не будет найдено выражение для каждого целого числа.

Одна такая подходящая грамматика имеет вид:[4]

$$\mathbb{R} \mapsto 4 \mid 44 \mid 444 \mid 4444 \mid \mathbb{R} + \mathbb{R} \mid \mathbb{R} - \mathbb{R} \mid \mathbb{R} \times \mathbb{R} \mid \mathbb{R} \,/\, \mathbb{R} \mid$$
$$\mathbb{R}^{\mathbb{R}} \mid \lfloor \mathbb{R} \rfloor \mid \lceil \mathbb{R} \rceil \mid \sqrt{\mathbb{R}} \mid \mathbb{R}! \mid \Gamma(\mathbb{R})$$

Мы округляем вычисленные выражения до ближайшего целого числа. Выражение, округленное в большую сторону, может содержаться в операции округления с избытком, а выражение, округленное в меньшую сторону, может содержаться в операции округления с недостатком, поэтому все такие выражения допустимы.

Упражнение 20.9. Выражение получается путем применения правил вывода:

$$\begin{aligned}
&\mathbb{R} \mapsto \ \mathbb{R} + \mathbb{R} && P = 1/12,\\
&\mathbb{R} \mapsto \mathbb{F} && P = 5/12,\\
&\mathbb{F} \mapsto 1{,}5 && P = 4/7,\\
&\mathbb{R} \mapsto \mathbb{I} && P = 5/12,\\
&\mathbb{I} \ \mapsto && P = 1/3.
\end{aligned}$$

Вероятность этого выражения равна

$$\frac{1}{12}\frac{5}{12}\frac{4}{7}\frac{5}{12}\frac{1}{3} = \frac{25}{9072} \approx 0{,}00276.$$

Упражнение 20.10. Итерация обучения очищает все значения и затем увеличивает каждое правило вывода каждый раз, когда оно применяется. Каждое из пяти примененных правил увеличивается один раз, что приводит к

$$\begin{aligned}
&\mathbb{R} \mapsto \mathbb{R} + \mathbb{R} \mid \ \mathbb{R} \times \mathbb{R} \mid \ \mathbb{F} \mid \ \mathbb{I} && w_{\mathbb{R}} = [1, 0, 1, 1],\\
&\mathbb{F} \mapsto 1{,}5 \mid \infty && p_{\mathbb{F}} = [1, 0],\\
&\mathbb{I} \mapsto 1 \mid 2 \mid 3 && p_{\mathbb{I}} = [0, 1, 0].
\end{aligned}$$

[4] Гамма-функция $\Gamma(x)$ является расширением факториала на действительные и комплексные числа. Для положительного целого числа x она возвращает значение $(x - 1)!$

Упражнение 21.1. Максимизация коэффициента подъемной силы для аэродинамического профиля при условии ограничения конструктивной устойчивости аэродинамического профиля.

Упражнение 21.2. Рассмотрим задачу, в которой граф дисциплинарной зависимости представляет собой (направленное) дерево. Если оптимизация начинается с корня и продолжается в соответствии с топологическим порядком, то сходимость происходит после одного обхода дерева.

Упражнение 21.3. Она позволяет выполнять дисциплинарный анализ параллельно.

Упражнение 21.4. Задача о пружинном маятнике при возможной архитектуре междисциплинарного проектирования имеет вид

$$\min_k f\left(\mathrm{MDA}\left(k\right)\right)$$
$$\text{при условии } k > 0$$
$$\theta \le \theta_{\max}$$
$$\mathrm{MDA}(k) \text{ сходится,}$$

где $\mathrm{MDA}(k)$ выполняет междисциплинарный анализ двух дисциплинарных анализов — анализа нагрузок, который вычисляет $\mathcal{A}[\mathcal{M}] = mgl\cos\mathcal{A}[\theta]$, и анализа смещения, который вычисляет $\mathcal{A}[\theta] = \mathcal{A}[\mathcal{M}]/\mathcal{A}[k]$. Анализ сходимости междисциплинарного проектирования включен в качестве дополнительной переменной ответа для обеспечения сходимости.

Решение задачи оптимизации дает $k \approx 55{,}353/N$.

Упражнение 21.5. Задача о пружинном маятнике при допустимой архитектуре отдельной дисциплины имеет вид:

$$\min_{k,\theta_c,M_c} k$$
$$\text{при условиях } k > 0,$$
$$\theta_c = F_{\text{displacement}}\left(k, M_c\right),$$
$$M_c = F_{\text{loads}}\left(\theta_c\right),$$
$$\theta \le \theta_{\max},$$

где θ_c и M_c — дополнительные переменные связи, управляемые оптимизатором. Два дисциплинарных анализа могут выполняться параллельно.

Упражнение 21.6. Две дисциплинарные задачи оптимизации для задачи о пружинным маятнике в архитектуре совместной оптимизации имеют вид

$$\min_{k,M} \; J_{\text{dispacement}} = \left(k_g - \mathcal{A}\left[k_g\right]\right)^2 + \left(M_g - \mathcal{A}\left[M_g\right]\right)^2 + \left(F_{\text{displacement}}\left(k_g, M_g\right) - \theta\right)^2$$

при условиях $\theta_g \le \theta_{\max}$,

$$k > 0$$

и

$$\min_{\theta_g} J_{\text{loads}} = \left(\theta_g - \theta\right)^2 + \left(F_{\text{loads}}\left(\theta_g\right) - M\right)^2,$$

где индекс указывает на глобальную переменную. Глобальными переменными являются $\mathcal{A}_g = \{k_g, \theta_g, M_g\}$.

Задача оптимизации на системном уровне имеет вид:

$$\min_{k_g, \theta_g, M_g} \; k_g$$

при условиях $J_{\text{structures}} = 0,$

$$J_{\text{loads}} = 0.$$

Библиография

1. N. M. Alexandrov and M. Y. Hussaini, eds., *Multidisciplinary Design Optimization: State of the Art*. SIAM, 1997.
2. S. Amari, “Natural Gradient Works Efficiently in Learning”, *Neural Computation*, vol. 10, no. 2, pp. 251–276, 1998.
3. Aristotle, Metaphysics, trans. by W. D. Ross. 350 BCE, Book I, Part 5.
4. L. Armijo, “Minimization of Functions Having Lipschitz Continuous First Partial Derivatives”, *Pacific Journal of Mathematics*, vol. 16, no. 1, pp. 1–3, 1966.
5. J. Arora, *Introduction to Optimum Design*, 4th ed. Academic Press, 2016.
6. R. K. Arora, *Optimization: Algorithms and Applications*. Chapman and Hall/CRC, 2015.
7. T. W. Athan and P. Y. Papalambros, “A Note on Weighted Criteria Methods for Compromise Solutions in Multi-Objective Optimization”, *Engineering Optimization*, vol. 27, no. 2, pp. 155–176, 1996.
8. C. Audet and J. E. Dennis Jr., “Mesh Adaptive Direct Search Algorithms for Constrained Optimization”, *SIAM Journal on Optimization*, vol. 17, no. 1, pp. 188–217, 2006.
9. D. A. Bader, W. E. Hart, and C. A. Phillips, “Parallel Algorithm Design for Branch and Bound”, in *Tutorials on Emerging Methodologies and Applications in Operations Research*, H. J. Greenberg, ed., Kluwer Academic Press, 2004.
10. W. W. R. Ball, *Mathematical Recreations and Essays*. Macmillan, 1892.
11. D. Barber, *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.
12. A. G. Baydin, R. Cornish, D. M. Rubio, M. Schmidt, and F. Wood, “Online Learning Rate Adaptation with Hypergradient Descent”, in *International Conference on Learning Representations* (ICLR), 2018.
13. A. D. Belegundu and T. R. Chandrupatla, *Optimization Concepts and Applications in Engineering*, 2nd ed. Cambridge University Press, 2011.
14. R. Bellman, “On the Theory of Dynamic Programming”, Proceedings of the National Academy of Sciences of the United States of America, vol. 38, no. 8, pp. 716–719, 1952 (cit. on p. 3).
15. R. Bellman, *Eye of the Hurricane: An Autobiography*. World Scientific, 1984.
16. H. Benaroya and S. M. Han, *Probability Models in Engineering and Science*. Taylor & Francis, 2005.
17. F. Berkenkamp, A. P. Schoellig, and A. Krause, “Safe Controller Optimization for Quadrotors with Gaussian Processes”, in *IEEE International Conference on Robotics and Automation* (*ICRA*), 2016.

18. H.-G. Beyer and B. Sendhoff, “Robust OptimOverview–A Comprehensive Survey”, *Computer Methods in Applied Mechanics and Engineering*, vol. 196, no. 33, pp. 3190–3218, 2007.
19. T. L. Booth and R. A. Thompson, “Applying Probability Measures to Abstract Languages”, *IEEE Transactions on Computers*, vol. C-22, no. 5, pp. 442–450, 1973.
20. C. Boutilier, R. Patrascu, P. Poupart, and D. Schuurmans, “Constraint-Based Optimization and Utility Elicitation Using the Minimax Decision Criterion”, *Artificial Intelligence*, vol. 170, no. 8-9, pp. 686–713, 2006.
21. G. E. P. Box, W. G. Hunter, and J. S. Hunter, *Statistics for Experimenters: An Introduction to Design, Data Analysis, and Model Building*, 2nd ed. Wiley, 2005.
22. S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.
23. D. Braziunas and C. Boutilier, “Minimax Regret-Based Elicitation of Generalized Additive Utilities”, in *Conference on Uncertainty in Artificial Intelligence* (*UAI*), 2007.
24. D. Braziunas and C. Boutilier, “Elicitation of Factored Utilities”, *AI Magazine*, vol. 29, no. 4, pp. 79–92, 2009.
25. R. P. Brent, *Algorithms for Minimization Without Derivatives*. Prentice Hall, 1973.
26. S. J. Colley, *Vector Calculus*, 4th ed. Pearson, 2011.
27. V. Conitzer, “Eliciting Single-Peaked Preferences Using Comparison Queries”, *Journal of Artificial Intelligence Research*, vol. 35, pp. 161–191, 2009.
28. S. Cook, “The Complexity of Theorem-Proving Procedures”, in *ACM Symposium on Theory of Computing*, 1971.
29. W. Cook, A. M. Gerards, A. Schrijver, and E. Tardos, “Sensitivity Theorems in Integer Linear Programming”, *Mathematical Programming*, vol. 34, no. 3, pp. 251–264, 1986.
30. A. Corana, M. Marchesi, C. Martini, and S. Ridella, “Minimizing Multimodal Functions of Continuous Variables with the ‘Simulated Annealing’ Algorithm”, *ACM Transactions on Mathematical Software*, vol. 13, no. 3, pp. 262–280, 1987.
31. G. B. Dantzig, “Origins of the Simplex Method”, in *A History of Scientific Computing*, S. G. Nash, ed., ACM, 1990, pp. 141–151.
32. S. Das and P. N. Suganthan, “Differential Evolution: A Survey of the State-Of-The-Art”, *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 4–31, 2011.
33. W. C. Davidon, “Variable Metric Method for Minimization”, Argonne National Laboratory, Tech. Rep. ANL-5990, 1959.
34. W. C. Davidon, “Variable Metric Method for Minimization”, *SIAM Journal on Optimization*, vol. 1, no. 1, pp. 1–17, 1991.

35. A. Dean, D. Voss, and D. Draguljic, *Design and Analysis of Experiments*, 2nd ed. Springer, 2017.
36. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II", *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.
37. T. J. Dekker, "Finding a Zero by Means of Successive Linear Interpolation", in *Constructive Aspects of the Fundamental Theorem of Algebra*, B. Dejon and P. Henrici, eds., Interscience, 1969.
38. R. Descartes, "La Geometrie", in *Discours De La Methode*. 1637.
39. E. D. Dolan, R. M. Lewis, and V. Torczon, "On the Local Convergence of Pattern Search", *SIAM Journal on Optimization*, vol. 14, no. 2, pp. 567–583, 2003.
40. M. Dorigo, V. Maniezzo, and A. Colorni, "Ant System: Optimization by a Colony of Cooperating Agents", *IEEE Transactions on Systems, Man, and Cybernetics*, Part B (Cybernetics), vol. 26, no. 1, pp. 29–41, 1996.
41. M. Dorigo, G. Di Caro, and L. M. Gambardella, "Ant Algorithms for Discrete Optimization", *Artificial Life*, vol. 5, no. 2, pp. 137–172, 1999.
42. J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization", *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.
43. R. Eberhart and J. Kennedy, "A New Optimizer Using Particle Swarm Theory", in *International Symposium on Micro Machine and Human Science*, IEEE, 1995.
44. B. Efron, "Bootstrap Methods: Another Look at the Jackknife", *The Annals of Statistics*, vol. 7, pp. 1–26, 1979 (cit. on p. 272).
45. B. Efron, "Estimating the Error Rate of a Prediction Rule: Improvement on Cross-Validation", *Journal of the American Statistical Association*, vol. 78, no. 382, pp. 316–331, 1983.
46. B. Efron and R. Tibshirani, "Improvements on Cross-Validation: The .632+ Bootstrap Method", *Journal of the American Statistical Association*, vol. 92, no. 438, pp. 548–560, 1997.
47. Euclid, *The Elements*, trans. by D. E. Joyce. 300 BCE.
48. R. Fletcher and M. J. D. Powell, "A Rapidly Convergent Descent Method for Minimization", *The Computer Journal*, vol. 6, no. 2, pp. 163–168, 1963.
49. R. Fletcher and C. M. Reeves, "Function Minimization by Conjugate Gradients", *The Computer Journal*, vol. 7, no. 2, pp. 149–154, 1964.
50. R. Fletcher, *Practical Methods of Optimization*, 2nd ed. Wiley, 1987.
51. J. J. Forrest and D. Goldfarb, "Steepest-Edge Simplex Algorithms for Linear Programming", *Mathematical Programming*, vol. 57, no. 1, pp. 341–374, 1992.
52. A. Forrester, A. Sobester, and A. Keane, *Engineering Design via Surrogate Modelling: A Practical Guide*. Wiley, 2008.

53. A. I. J. Forrester, A. Sobester, and A. J. Keane, “Multi-Fidelity Optimization via Surrogate Modelling”, *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 463, no. 2088, pp. 3251–3269, 2007.
54. J. H. Friedman, “Exploratory Projection Pursuit”, *Journal of the American Statistical Association*, vol. 82, no. 397, pp. 249–266, 1987 (cit. on p. 323).
55. W. Gautschi, *Orthogonal Polynomials: Computation and Approximation*. Oxford University Press, 2004.
56. A. Girard, C. E. Rasmussen, J. Q. Candela, and R. Murray-Smith, “Gaussian Process Priors with Uncertain Inputs—Application to Multiple-Step Ahead Time Series Forecasting”, in *Advances in Neural Information Processing Systems* (*NIPS*), 2003.
57. D. E. Goldberg, Genetic *Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989 (cit. on p. 148).
58. D. E. Goldberg and J. Richardson, “Genetic Algorithms with Sharing for Multimodal Function Optimization”, in *International Conference on Genetic Algorithms*, Lawrence Erlbaum, 1987.
59. G. H. Golub and J. H. Welsch, “Calculation of Gauss Quadrature Rules”, *Mathematics of Computation*, vol. 23, no. 106, pp. 221–230, 1969 (cit. on p. 331).
60. R. E. Gomory, “An Algorithm for Integer Solutions to Linear Programs”, *Recent Advances in Mathematical Programming*, vol. 64, pp. 269–302, 1963.
61. I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.
62. A. Griewank and A. Walther, Evaluating Derivatives: Principles and *Techniques of Algorithmic Differentiation*, 2nd ed. SIAM, 2008 (cit. on p. 23).
63. S. Guo and S. Sanner, “Real-Time Multiattribute Bayesian Preference Elicitation with Pairwise Comparison Queries”, in *International Conference on Artificial Intelligence and Statistics* (*AISTATS*), 2010 (cit. on p. 230).
64. B. Hajek, “Cooling Schedules for Optimal Annealing”, *Mathematics of Operations Research*, vol. 13, no. 2, pp. 311–329, 1988 (cit. on p. 130).
65. T. C. Hales, “The Honeycomb Conjecture”, *Discrete & Computational Geometry*, vol. 25, pp. 1–22, 2001.
66. J. H. Halton, “Algorithm 247: Radical-Inverse Quasi-Random Point Sequence”, *Communications of the ACM*, vol. 7, no. 12, pp. 701–702, 1964.
67. N. Hansen and A. Ostermeier, “Adapting Arbitrary Normal Mutation Distributions in Evolution Strategies: The Covariance Matrix Adaptation”, in *IEEE International Conference on Evolutionary Computation*, 1996.
68. N. Hansen, “The CMA Evolution Strategy: A Tutorial”, *ArXiv*, no. 1604.00772, 2016.

69. F. M. Hemez and Y. Ben-Haim, “Info-Gap Robustness for the Correlation of Tests and Simulations of a Non-Linear Transient”, *Mechanical Systems and Signal Processing*, vol. 18, no. 6, pp. 1443–1467, 2004.
70. G. Hinton and S. Roweis, “Stochastic Neighbor Embedding”, in *Advances in Neural Information Processing Systems* (*NIPS*), 2003.
71. R. Hooke and T. A. Jeeves, “Direct Search Solution of Numerical and Statistical Problems”, *Journal of the ACM* (*JACM*), vol. 8, no. 2, pp. 212–229, 1961.
72. H. Ishibuchi and T. Murata, “A Multi-Objective Genetic Local Search Algorithm and Its Application to Flowshop Scheduling”, *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 28, no. 3, pp. 392–403, 1998.
73. V. S. Iyengar, J. Lee, and M. Campbell, “Q-EVAL: Evaluating Multiple Attribute Items Using Queries”, in *ACM Conference on Electronic Commerce, ACM*, 2001.
74. D. R. Jones, C. D. Perttunen, and B. E. Stuckman, “Lipschitzian Optimization Without the Lipschitz Constant”, *Journal of Optimization Theory and Application*, vol. 79, no. 1, pp. 157–181, 1993.
75. D. Jones and M. Tamiz, *Practical Goal Programming*. Springer, 2010.
76. A. B. Kahn, “Topological Sorting of Large Networks”, *Communications of the ACM*, vol. 5, no. 11, pp. 558–562, 1962.
77. L. Kallmeyer, *Parsing Beyond Context-Free Grammars*. Springer, 2010.
78. L. V. Kantorovich, “A New Method of Solving Some Classes of Extremal Problems”, in *Proceedings of the USSR Academy of Sciences*, vol. 28, 1940.
79. A. F. Kaupe Jr, “Algorithm 178: Direct Search”, *Communications of the ACM*, vol. 6, no. 6, pp. 313–314, 1963.
80. A. Keane and P. Nair, *Computational Approaches for Aerospace Design*. Wiley, 2005.
81. J. Kennedy, R. C. Eberhart, and Y. Shi, *Swarm Intelligence*. Morgan Kaufmann, 2001.
82. D. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization”, in *International Conference on Learning Representations* (*ICLR*), 2015.
83. S. Kiranyaz, T. Ince, and M. Gabbouj, *Multidimensional Particle Swarm Optimization for Machine Learning and Pattern Recognition*. Springer, 2014, Section 2.1.
84. S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, “Optimization by Simulated Annealing”, *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
85. T. H. Kjeldsen, “A Contextualized Historical Analysis of the Kuhn-Tucker Theorem in Nonlinear Programming: The Impact of World War II”, *Historia Mathematica*, vol. 27, no. 4, pp. 331 –361, 2000.
86. L. Kocis and W. J. Whiten, “Computational Investigations of Low-Discrepancy Sequences”, *ACM Transactions on Mathematical Software*, vol. 23, no. 2, pp. 266–294, 1997.

87. P. J. Kolesar, “A Branch and Bound Algorithm for the Knapsack Problem”, *Management Science*, vol. 13, no. 9, pp. 723–735, 1967.
88. B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, 5th ed. Springer, 2012 (cit. on p. 339).
89. J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.
90. K. W. Ku and M.-W. Mak, “Exploring the Effects of Lamarckian and Baldwinian Learning in Evolving Recurrent Neural Networks”, in *IEEE Congress on Evolutionary Computation* (*CEC*), IEEE, 1997.
91. L. Kuipers and H. Niederreiter, *Uniform Distribution of Sequences*. Dover, 2012.
92. J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright, “Convergence Properties of the Nelder–Mead Simplex Method in Low Dimensions”, *SIAM Journal on Optimization*, vol. 9, no. 1, pp. 112–147, 1998.
93. R. Lam, K. Willcox, and D. H. Wolpert, “Bayesian Optimization with a Finite Budget: An Approximate Dynamic Programming Approach”, in *Advances in Neural Information Processing Systems* (*NIPS*), 2016.
94. A. H. Land and A. G. Doig, “An Automatic Method of Solving Discrete Programming Problems”, *Econometrica*, vol. 28, no. 3, pp. 497–520, 1960.
95. C. Lemieux, *Monte Carlo and Quasi-Monte Carlo Sampling*. Springer, 2009.
96. J. R. Lepird, M. P. Owen, and M. J. Kochenderfer, “Bayesian Preference Elicitation for Multiobjective Engineering Design Optimization”, *Journal of Aerospace Information Systems*, vol. 12, no. 10, pp. 634–645, 2015 (cit. on p. 230).
97. K. Levenberg, “A Method for the Solution of Certain Non-Linear Problems in Least Squares”, *Quarterly of Applied Mathematics*, vol. 2, no. 2, pp. 164–168, 1944.
98. S. Linnainmaa, “The Representation of the Cumulative Rounding Error of an Algorithm as a Taylor Expansion of the Local Rounding Errors”, Master’s thesis, University of Helsinki, 1970.
99. M. Manfrin, “Ant Colony Optimization for the Vehicle Routing Problem”, PhD thesis, Universite Libre de Bruxelles, 2004.
100. R. T. Marler and J. S. Arora, “Survey of Multi-Objective Optimization Methods for Engineering”, *Structural and Multidisciplinary Optimization*, vol. 26, no. 6, pp. 369–395, 2004.
101. J. R.R. A. Martins and A. B. Lambe, “Multidisciplinary Design Optimization: A Survey of Architectures”, *AIAA Journal*, vol. 51, no. 9, pp. 2049–2075, 2013.
102. J. R.R. A. Martins, P. Sturdza, and J. J. Alonso, “The Complex-Step Derivative Approximation”, *ACM Transactions on Mathematical Software*, vol. 29, no. 3, pp. 245–262, 2003.

103. J. H. Mathews and K. D. Fink, *Numerical Methods Using MATLAB*, 4th ed. Pearson, 2004.
104. K. Miettinen, *Nonlinear Multiobjective Optimization*. Kluwer Academic Publishers, 1999.
105. D. J. Montana, "Strongly Typed Genetic Programming", *Evolutionary Computation*, vol. 3, no. 2, pp. 199–230, 1995.
106. D. C. Montgomery, *Design and Analysis of Experiments*. Wiley, 2017.
107. M. D. Morris and T. J. Mitchell, "Exploratory Designs for Computational Experiments", *Journal of Statistical Planning and Inference*, vol. 43, no. 3, pp. 381–402, 1995.
108. K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.
109. S. Narayanan and S. Azarm, "On Improving Multiobjective Genetic Algorithms for Design Optimization", *Structural Optimization*, vol. 18, no. 2-3, pp. 146–155, 1999.
110. S. Nash and A. Sofer, *Linear and Nonlinear Programming*. McGraw-Hill, 1996.
111. J. A. Nelder and R. Mead, "A Simplex Method for Function Minimization", *The Computer Journal*, vol. 7, no. 4, pp. 308–313, 1965.
112. Y. Nesterov, "A Method of Solving a Convex Programming Problem with Convergence Rate $O(1/k^2)$", *Soviet Mathematics Doklady*, vol. 27, no. 2, pp. 543–547, 1983.
113. J. Nocedal, "Updating Quasi-Newton Matrices with Limited Storage", *Mathematics of Computation*, vol. 35, no. 151, pp. 773–782, 1980.
114. J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. Springer, 2006.
115. J. Nocedal and S. J. Wright, "Trust-Region Methods", in *Numerical Optimization*. Springer, 2006, pp. 66–100.
116. A. O'Hagan, "Some Bayesian Numerical Analysis", *Bayesian Statistics*, vol. 4, J. M. Bernardo, J. O. Berger, A. P. Dawid, and A. F. M. Smith, eds., pp. 345–363, 1992.
117. M. Padberg and G. Rinaldi, "A Branch-And-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems", *SIAM Review*, vol. 33, no. 1, pp. 60–100, 1991.
118. P. Y. Papalambros and D. J. Wilde, *Principles of Optimal Design*. Cambridge University Press, 2017.
119. G.-J. Park, T.-H. Lee, K. H. Lee, and K.-H. Hwang, "Robust Design: An Overview", *AIAA Journal*, vol. 44, no. 1, pp. 181–191, 2006.
120. G. C. Pflug, "Some Remarks on the Value-At-Risk and the Conditional Value-At-Risk", in *Probabilistic Constrained Optimization: Methodology and Applications*, S. P. Uryasev, ed. Springer, 2000, pp. 272–281.

121. S. Piyavskii, “An Algorithm for Finding the Absolute Extremum of a Function”, *USSR Computational Mathematics and Mathematical Physics*, vol. 12, no. 4, pp. 57–67, 1972.
122. E. Polak and G. Ribiere, “Note Sur la Convergence de Methodes de Directions Conjuguees”, *Revue Francaise D’informatique et de Recherche Operationnelle, Serie Rouge*, vol. 3, no. 1, pp. 35–43, 1969.
123. M. J. D. Powell, “An Efficient Method for Finding the Minimum of a Function of Several Variables Without Calculating Derivatives”, *Computer Journal*, vol. 7, no. 2, pp. 155–162, 1964.
124. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1982, vol. 2.
125. C. E. Rasmussen and Z. Ghahramani, “Bayesian Monte Carlo”, in *Advances in Neural Information Processing Systems* (*NIPS*), 2003.
126. C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.
127. A. Ratle and M. Sebag, “Genetic Programming and Domain Knowledge: Beyond the Limitations of Grammar-Guided Machine Discovery”, in *International Conference on Parallel Problem Solving from Nature*, 2000.
128. I. Rechenberg, *Evolutionsstrategie Optimierung Technischer Systeme Nach Prinzipien Der Biologischen Evolution*. Frommann-Holzboog, 1973.
129. R. G. Regis, “On the Properties of Positive Spanning Sets and Positive Bases”, *Optimization and Engineering*, vol. 17, pp. 229–262, 1 2016.
130. R. T. Rockafellar and S. Uryasev, “Optimization of Conditional Value-At-Risk”, *Journal of Risk*, vol. 2, pp. 21–42, 2000.
131. R. T. Rockafellar and S. Uryasev, “Conditional Value-At-Risk for General Loss Distributions”, *Journal of Banking and Finance*, vol. 26, pp. 1443–1471, 2002.
132. H. H. Rosenbrock, “An Automatic Method for Finding the Greatest or Least Value of a Function”, *The Computer Journal*, vol. 3, no. 3, pp. 175–184, 1960.
133. R. Y. Rubinstein and D. P. Kroese, *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*. Springer, 2004.
134. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning Representations by Back-Propagating Errors”, *Nature*, vol. 323, pp. 533–536, 1986.
135. C. Ryan, J. J. Collins, and M. O. Neill, “Grammatical Evolution: Evolving Programs for an Arbitrary Language”, in *European Conference on Genetic Programming*, Springer, 1998.
136. T. Salimans, J. Ho, X. Chen, and I. Sutskever, “Evolution Strategies as a Scalable Alternative to Reinforcement Learning”, *ArXiv*, no. 1703.03864, 2017.

137. R. Salustowicz and J. Schmidhuber, “Probabilistic Incremental Program Evolution”, *Evolutionary Computation*, vol. 5, no. 2, pp. 123–141, 1997.
138. J. D. Schaffer, “Multiple Objective Optimization with Vector Evaluated Genetic Algorithms”, in *International Conference on Genetic Algorithms and Their Applications*, 1985.
139. C. Schretter, L. Kobbelt, and P.-O. Dehaye, “Golden Ratio Sequences for Low-Discrepancy Sampling”, *Journal of Graphics Tools*, vol. 16, pp. 95–104, 2 2016.
140. A. Shapiro, D. Dentcheva, and A. Ruszczynski, *Lectures on Stochastic Programming: Modeling and Theory*, 2nd ed. SIAM, 2014.
141. A. Shmygelska, R. Aguirre-Hernandez, and H. H. Hoos, “An Ant Colony Algorithm for the 2D HP Protein Folding Problem”, in *International Workshop on Ant Algorithms* (*ANTS*), 2002.
142. B. O. Shubert, “A Sequential Method Seeking the Global Maximum of a Function”, *SIAM Journal on Numerical Analysis*, vol. 9, no. 3, pp. 379–388, 1972.
143. D. Simon, *Evolutionary Optimization Algorithms*. Wiley, 2013.
144. J. Sobieszczanski-Sobieski, A. Morris, and M. van Tooren, *Multidisciplinary Design Optimization Supported by Knowledge Based Engineering*. Wiley, 2015.
145. I. M. Sobol, “On the Distribution of Points in a Cube and the Approximate Evaluation of Integrals”, *USSR Computational Mathematics and Mathematical Physics*, vol. 7, no. 4, pp. 86–112, 1967.
146. D. C. Sörensen, “Newton’s Method with a Model Trust Region Modification”, *SIAM Journal on Numerical Analysis*, vol. 19, no. 2, pp. 409–426, 1982.
147. K. Sörensen, “Metaheuristics — the Metaphor Exposed”, *International Transactions in Operational Research*, vol. 22, no. 1, pp. 3–18, 2015.
148. T. J. Stieltjes, “Quelques Recherches Sur la Theorie Des Quadratures Dites Mecaniques”, *Annales Scientifiques de L’Ecole Normale Superieure*, vol. 1, pp. 409–426, 1884.
149. J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis*, 3rd ed. Springer, 2002.
150. M. Stone, “Cross-Validatory Choice and Assessment of Statistical Predictions”, *Journal of the Royal Statistical Society*, vol. 36, no. 2, pp. 111–147, 1974.
151. T. Stützle, “MAX-MIN Ant System for Quadratic Assignment Problems”, Technical University Darmstadt, Tech. Rep., 1997.
152. Y. Sui, A. Gotovos, J. Burdick, and A. Krause, “Safe Exploration for Optimization with Gaussian Processes”, in *International Conference on Machine Learning* (*ICML*), vol. 37, 2015.
153. H. Szu and R. Hartley, “Fast Simulated Annealing”, *Physics Letters A*, vol. 122, no. 3-4, pp. 157–162, 1987.
154. G. B. Thomas, *Calculus and Analytic Geometry*, 9th ed. Addison-Wesley, 1968.

155. V. Torczon, “On the Convergence of Pattern Search Algorithms”, *SIAM Journal of Optimization*, vol. 7, no. 1, pp. 1–25, 1997.
156. M. Toussaint, “The Bayesian Search Game”, in *Theory and Principled Methods for the Design of Metaheuristics*, Y. Borenstein and A. Moraglio, eds. Springer, 2014, pp. 129–144.
157. R. Vanderbei, *Linear Programming, Foundations and Extensions*, 4th ed. Springer, 2014.
158. D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, and J. Schmidhuber, “Natural Evolution Strategies”, *ArXiv*, no. 1106.4487, 2011 (cit. on p. 139).
159. H. P. Williams, *Model Building in Mathematical Programming*, 5th ed. Wiley, 2013.
160. D. H. Wolpert and W. G. Macready, “No Free Lunch Theorems for Optimization”, *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.
161. P. K. Wong, L. Y. Lo, M. L. Wong, and K. S. Leung, “Grammar-Based Genetic Programming with Bayesian Network”, in *IEEE Congress on Evolutionary Computation* (*CEC*), 2014.
162. X.-S. Yang, *Nature-Inspired Metaheuristic Algorithms*. Luniver Press, 2008.
163. X.-S. Yang, “A Brief History of Optimization”, in *Engineering Optimization*. Wiley, 2010, pp. 1–13.
164. X.-S. Yang and S. Deb, “Cuckoo Search via Levy Flights”, in *World Congress on Nature & Biologically Inspired Computing* (*NaBIC*), IEEE, 2009.
165. P. L. Yu, “Cone Convexity, Cone Extreme Points, and Nondominated Solutions in Decision Problems with Multiobjectives”, *Journal of Optimization Theory and Applications*, vol. 14, no. 3, pp. 319–377, 1974.
166. Y. X. Yuan, “Recent Advances in Trust Region Algorithms”, *Mathematical Programming*, vol. 151, no. 1, pp. 249–281, 2015.
167. L. Zadeh, “Optimality and Non-Scalar-Valued Performance Criteria”, *IEEE Transactions on Automatic Control*, vol. 8, no. 1, pp. 59–60, 1963.
168. M. D. Zeiler, “ADADELTA: An Adaptive Learning Rate Method”, *ArXiv*, no. 1212.5701, 2012.

Предметный указатель

И

К

Л

М

Н

О

П

Р

ОПТИМИЗАЦИЯ ПРОГРАММ НА C++

Курт Гантерот

C++ — язык динамично развивающийся, так что методы оптимизации программ на нем тоже постоянно развиваются и совершенствуются. То, что когда-то было передовым способом оптимизации, теперь зачастую делает вместо вас компилятор. Однако в настоящем программировании всегда есть возможность усовершенствовать имеющийся код. Как это сделать? Об этом вы и узнаете из книги, написанной профессионалом для профессионалов и достойной занять свое место рядом с клавиатурой каждого серьезного программиста на C++.

www.dialektika.com

ISBN 978-5-907144-58-3 **в продаже**

JAVA:
ОПТИМИЗАЦИЯ ПРОГРАММ
ПРАКТИЧЕСКИЕ МЕТОДЫ ПОВЫШЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ ПРИЛОЖЕНИЙ В JVM

Бенджамин Эванс
Джеймс Гоф
Крис Ньюланд

www.williamspublishing.com

В этой книге вы найдете массу практической информации о том, как устроены язык Java и JVM и как заставить свои программы работать максимально быстро. Java — это язык, динамично развивающийся во всех отношениях (и JVM не является исключением), так что методы оптимизации программ на этом языке тоже постоянно развиваются и совершенствуются. То, что когда-то было передовым способом оптимизации, теперь зачастую делает вместо вас компилятор, а иногда эта методика приводит к обратному эффекту и только тормозит работу программ! Однако в настоящем промышленном программировании всегда найдется возможность выжать еще немного производительности из имеющегося кода. Как это сделать? Об этом и рассказывается в книге, написанной профессионалами для профессионалов и достойной занять свое место рядом с клавиатурой каждого серьезного программиста на Java.

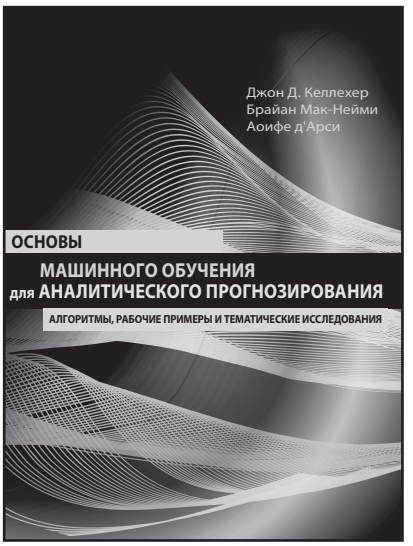
ISBN 978-5-907114-84-5

в продаже

ОСНОВЫ МАШИННОГО ОБУЧЕНИЯ ДЛЯ АНАЛИТИЧЕСКОГО ПРОГНОЗИРОВАНИЯ

АЛГОРИТМЫ, РАБОЧИЕ ПРИМЕРЫ И ТЕМАТИЧЕСКИЕ ИССЛЕДОВАНИЯ

Джон Д. Келлехер
Брайан Мак-Нейми
Аоифе д'Арси



www.williamspublishing.com

Книга представляет собой учебник по машинному обучению с акцентом на коммерческие приложения. Она предлагает подробное описание наиболее важных подходов к машинному обучению, используемых в интеллектуальном анализе данных, охватывающих как теоретические концепции, так и практические приложения. Формальный математический материал дополняется пояснительными примерами, а примеры исследований иллюстрируют применение этих моделей в более широком контексте бизнеса. В книге рассмотрены информационное обучение, обучение на основе сходства, вероятностное обучение и обучение на основе ошибок. Описанию каждого из этих подходов предшествует объяснение основополагающей концепции, за которой следуют математические модели и алгоритмы, иллюстрированные подробными рабочими примерами.

ISBN 978-5-6040044-9-4

в продаже

Алгоритмы: построение и анализ *3-е издание*

Томас Кормен,
Чарльз Лейзерсон,
Рональд Ривест,
Клиффорд Штайн

www.williamspublishing.com

Фундаментальный труд известных специалистов в области информатики достоин занять место на полке любого человека, чья деятельность так или иначе связана с вычислительной техникой и алгоритмами. Для профессионала эта книга может служить настольным справочником, для преподавателя — пособием для подготовки к лекциям и источником интересных нетривиальных задач, для студентов и аспирантов — отличным учебником.

Строгий математический анализ и обилие теорем сопровождаются большим количеством иллюстраций, элементарными рассуждениями и простыми приближенными оценками. Широта охвата материала и степень строгости его изложения дают основания считать эту книгу одной из лучших книг, посвященных разработке и анализу алгоритмов.

Третье издание этого классического труда в большой степени доработано. В нем появились новые главы, в том числе посвященные такой важной в последнее время теме, как многопоточные алгоритмы, а старые подверглись переработке, местами весьма существенной, когда уже имевшийся во втором издании материал излагается с иных позиций, чем ранее.

Данная книга будет не лишней как на столе студента и аспиранта, так и на рабочей полке практикующего программиста.

ISBN 978-5-8459-2016-4 **в продаже**