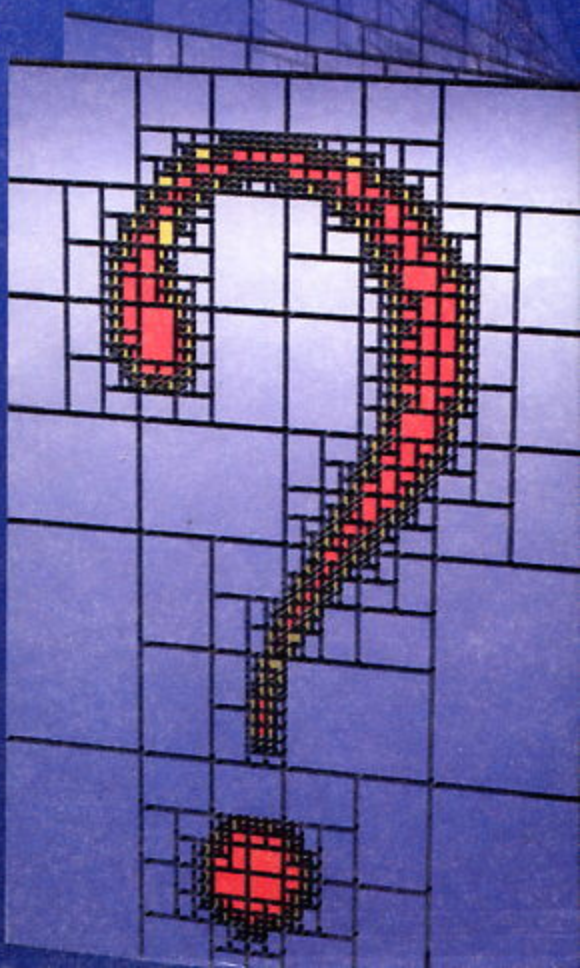
Л. Жолен, М. Кифер,
О. Дидри, Э. Вальтер

ПРИКЛАДНОЙ ИНТЕРВАЛЬНЫЙ АНАЛИЗ



Институт компьютерных исследований

Luc Jaulin, Michel Kieffer, Olivier Didrit
and Éric Walter

Applied Interval Analysis

With Examples in Parameter and State Estimation, Robust Control and Robotics

With 125 Figures

Springer

Л. Жолен, М. Кифер, О. Дидри, Э. Вальтер

ПРИКЛАДНОЙ ИНТЕРВАЛЬНЫЙ АНАЛИЗ

С примерами по оцениванию параметров и состояний, робастному управлению и робототехнике

Со 125 иллюстрациями

Перевод с англ. к. т. н. С. И. Кумкова
Под редакцией д. т. н. Б. Т. Поляка

Москва ♦ Ижевск

2005

УДК 518+519.4

РФФИ

Издание осуществлено при финансовой поддержке Российского фонда фундаментальных исследований по проекту №03-01-14151.

Жолен Л., Кифер М., Дидри О., Вальтер Э.
Прикладной интервальный анализ. — М.–Ижевск: Институт компьютерных исследований, 2005. — 468 с.

Книга посвящена теории и численным методам гарантированного оценивания и аппроксимации множеств. Техника и математический аппарат интервального анализа строго обобщаются на процедуры работы с множествами. Впервые в монографической литературе детально рассматривается приложение разработанных методов к решению систем нелинейных уравнений и неравенств, задачам оптимизации, оценивания параметров и состояний, робастного управления и робототехники. Приводится подборка примеров и упражнений.

Книга рассчитана на широкий круг читателей, студентов, аспирантов, инженеров и научных работников, занятых исследованием и проектированием систем обработки информации и систем управления, а также на математиков, вычислителей и программистов.

ISBN 5-93972-384-5

http://rcd.ru
http://ics.org.ru

Оглавление

РАЗДЕЛ III. ПРИЛОЖЕНИЯ

РАЗДЕЛ IV. РЕАЛИЗАЦИЯ

Предисловие

В основе многих инженерных задач лежит решение систем уравнений, неравенств и оптимизация целевых функций. К сожалению, за исключением особых случаев, в которых система уравнений линейна относительно неизвестных или когда минимизируемая целевая функция выпукла, а минимизация выполняется при выпуклых ограничениях, результаты, получаемые с помощью обычных численных методов, носят локальный характер и не являются гарантированными оценками. Это может означать, например, что действительный глобальный минимум целевой функции может и не быть найден, или что процедуры поиска глобального минимума его не находят. В противоположность этому, интервальный анализ дает возможность получать гарантированные аппроксимации множества *всех действительных* (*допустимых*) решений рассматриваемой задачи.

Это, вместе с отсутствием литературы, где методы интервального анализа представлялись бы в виде, который позволил бы им стать частью удобных инженерных численных методов, побудило авторов к написанию данной книги.

Начало было положено в 1991 году Люком Жоленом при подготовке им своей диссертации под руководством Эрика Вальтера. Под их совместным руководством работа была продолжена при подготовке диссертаций Оливье Дидри и Мишеля Кифера. Более двух лет назад, когда проект нашей книги был представлен в издательство Springer, мы наивно полагали, что ее подготовка к печати будет простой на основе того, что было уже сделано... В действительности же, эта книга является результатом острых дискуссий между авторами о том, что и как должно быть написано! Временами мы опасались, что можем вообще не написать эту книгу, но мы чувствовали, что в процессе работы результаты улучшались.

Однако было по крайней мере две идеи, по которым мы легко пришли к согласию. Во-первых, книга должна быть как можно более простой и понятной, вот почему в ней так много иллюстраций и примеров. Во-вторых, читателю, который захочет поупражняться в интервальном анализе применительно к своим задачам, нужно дать конкретный инструмент, чтобы он мог это сделать.

Многие коллеги помогали нам приобщиться к интервальному анализу, и невозможно упомянуть их всех, но, во всяком случае, мы хотели бы

поблагодарить Владика Крейновича за всю его энергию, которую он вложил в подготовку web-сайта Interval Computations, и за все, чему мы там научились.

Особо мы благодарны Мишелю Петито за его помощь в постижении тайн языка ADA и платформы Стюарта–Гофа, Доминику Майзелю за ознакомление с задачами слежения за роботом и определения его местоположения, Олафу Кнюппелю и Зигфриду М. Рампу за адаптацию PROFIL/BIAS и INTLAB, Изабель Брэм, Мартине Себерио, Рамону Муру, Стефану Ратшану и Натали Риволь за их конструктивные замечания при вычитке рукописи и нашему редактору Оливеру Джексону, чья дружеская требовательность была конструктивной для выхода в свет данной книги в этом тысячелетии.

Мы хотели бы выразить нашу признательность Гаю Деноману, руководителю Лаборатории сигналов и систем, и Жан-Луи Ферье, руководителю Лаборатории проектирования автоматизированных систем, за их поддержку и создание условий для работы.

Мы признательны также Национальному центру научных исследований Франции за создание идеальной рабочей обстановки, и частичная поддержка гранта ИНТАС также была полезна.

Обозначения

Ниже приводятся таблицы, описывающие основные используемые обозначения и символы.

Точечные величины

x	—	точечный скаляр
x^*	—	истинное значение неизвестной переменной
$\check{x}$	—	априорное значение неизвестной переменной
$\hat{x}$	—	апостериорное значение неизвестной переменной
$\mathbf{x}$	—	точечный вектор-столбец
$\mathbf{x}^{\mathrm{T}}$	—	точечный вектор-строка
$\mathbf{0}$	—	нуль-вектор, вектор с компонентами из нулей
$\mathbf{1}$	—	единичный вектор, вектор с компонентами из единиц
$\mathbf{X}$	—	точечная матрица
$\mathbf{O}$, $\mathbf{O}_{n\times m}$	—	нулевая матрица, $n \times m$ матрица с нулевыми элементами
$\mathbf{I}$, $\mathbf{I}_n$	—	единичная матрица (матрица тождественного преобразования), $n \times n$ матрица
$\mathrm{Im}(s)$	—	мнимая часть числа s
$\mathrm{Re}(s)$	—	вещественная часть числа s

Интервалы

$[x] = [\underline{x}, \overline{x}]$	—	интервальный скаляр
$[\mathbf{x}] = [\underline{\mathbf{x}}, \overline{\mathbf{x}}]$	—	интервальный вектор
$[\mathbf{X}] = [\underline{\mathbf{X}}, \overline{\mathbf{X}}]$	—	интервальная матрица
$[x_i] = ([\mathbf{x}])_i$	—	i-й интервальный элемент интервального вектора $[\mathbf{x}]$
$[x_{ij}] = ([\mathbf{X}])_{ij}$	—	интервальный элемент интервальной матрицы $[\mathbf{X}]$ ее i-й строки и j-го столбца
$\mathrm{lb}([x])$	—	нижняя граница интервала $[x]$
$\mathrm{ub}([x])$	—	верхняя граница интервала $[x]$
$w([x])$	—	ширина интервала $[x]$
$\mathrm{mid}([x])$	—	центр интервала $[x]$

Множества

$\varnothing$	—	пустое множество
$\mathbb{S}$	—	множество
$\mathbb{N}$	—	множество всех положительных целых чисел
$\mathbb{Z}$	—	множество всех целых чисел
$\mathbb{R}$	—	множество всех вещественных чисел
$\mathbb{IR}$	—	множество всех интервальных вещественных чисел
$\mathbb{C}$	—	множество всех комплексных чисел
$\mathbb{C}^-$	—	множество всех комплексных чисел со строго отрицательной вещественной частью
$\mathbb{B}$	—	множество всех булевских переменных
$\mathbb{IB}$	—	множество всех интервальных булевских переменных
$\partial\mathbb{S}$	—	граница множества $\mathbb{S}$
$[\mathbb{S}]$	—	интервальная оболочка множества $\mathbb{S}$
$\overline{\mathbb{S}}$	—	аппроксимация множества $\mathbb{S}$ сверху (внешняя аппроксимация)
$\underline{\mathbb{S}}$	—	аппроксимация множества $\mathbb{S}$ снизу (внутренняя аппроксимация)
$\mathcal{L}$	—	список, стек, очередь величин, дерево или граф

Другие обозначения

$\triangleq$	—	равенство по определению
$:=$	—	оператор присваивания
$\forall$	—	квантор всеобщности (*для всех*)
$\exists$	—	квантор существования (*существует*)
$\neg$	—	логическое дополнение
$\wedge$	—	логическое “И”
$\vee$	—	логическое “ИЛИ”
$\mathbb{A}\times\mathbb{B}$	—	прямое произведение множеств $\mathbb{A}$ и $\mathbb{B}$
$\mathbb{A}\backslash\mathbb{B}$	—	разность множеств, т. е. совокупность элементов, содержащая элементы $\{x : (x \in \mathbb{A}) \wedge (x \notin \mathbb{B})\}$
$\mathbb{A}\sqcup\mathbb{B}$	—	интервальное объединение $\mathbb{A}$ и $\mathbb{B}$, т. е. совокупность $[\mathbb{A}\cup\mathbb{B}]$

Функции

Функции обозначаются теми же символами, что и элементы пространств их образов; так $[f](\cdot)$ обозначает скалярную интервальную функцию, а $[\mathbf{f}](\cdot)$ обозначает векторную интервальную функцию.

Если $\mathbf{f}(\cdot)$ — однократно дифференцируемая функция, отображающая из $\mathbb{R}^{n_\mathbf{x}}$ в $\mathbb{R}^{n_\mathbf{y}}$, то ее *матрица Якоби* по переменной $\mathbf{x}$ имеет вид:

$$\mathbf{J_f}(\mathbf{x}) \triangleq \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_{n_\mathbf{x}}}(\mathbf{x}) \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \frac{\partial f_{n_y}}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial f_{n_y}}{\partial x_{n_\mathbf{x}}}(\mathbf{x}) \end{pmatrix}.$$

Если $f(\cdot)$ — однократно дифференцируемая функция, отображающая из $\mathbb{R}^{n_x}$ в $\mathbb{R}$, то ее *градиент* в точке $\mathbf{x}$ обозначается как:

$$\mathbf{g}_f(\mathbf{x}) \triangleq \begin{pmatrix} \frac{\partial f}{\partial x_1}(\mathbf{x}) \\ \cdot \\ \cdot \\ \cdot \\ \frac{\partial f}{\partial x_{n_\mathbf{x}}}(\mathbf{x}) \end{pmatrix}.$$

Если $f(\cdot)$ — дважды дифференцируемая функция, то ее *матрица Гессе* в точке $\mathbf{x}$ является симметричной матрицей Якоби, связанной с ее градиентом, и обозначается как:

$$\mathbf{H}_f(\mathbf{x}) \triangleq \begin{pmatrix} \frac{\partial^2 f}{\partial^2 x_1}(\mathbf{x}) & \cdots & \frac{\partial^2 f}{\partial x_{n_\mathbf{x}} \partial x_1}(\mathbf{x}) \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \frac{\partial^2 f}{\partial x_1 \partial x_{n_\mathbf{x}}}(\mathbf{x}) & \cdots & \frac{\partial f}{\partial^2 x_{n_\mathbf{x}}}(\mathbf{x}) \end{pmatrix}.$$

Алгоритмы

Алгоритмы описываются в псевдокоде, позволяющем использовать обычные математические обозначения. Большинство важных аргументов приводится после имени (NAME) алгоритма как входные переменные (вход:), выходные персменные (выход:) или как входные–выходные переменные

(вход/выход:). Для облегчения чтения мы оставили за собой право опускать обозначения некоторых из них, таких как функции включения, градиента, матриц Гессе и т. д. Блоки с утверждениями выделяются специальным знаком. Любой оператор возврата вызывает немедленный выход из текущего алгоритма. В конце алгоритмов подразумевается наличие операторов возврата.

Подробности реализации этих алгоритмов приводятся в Главе 11, где текст программ на **C++** выделен машинописным шрифтом.

Дополнение: специальные термины

(Добавлено переводчиком и научным редактором перевода)

Для адаптации терминологии, используемой Авторами, к терминологии, принятой в российской литературе, приняты следующие специальные термины:

параллелотоп — многомерный параллелепипед с ребрами, ориентированными по осям соответствующей декартовой системы;

подпараллелотоп — параллелотоп, получаемый в результате измельчения некоторого исходного параллелотопа (например, с помощью бисекции) и целиком содержащийся в исходном параллелотопе;

покрытие — объединение множеств (или просто множество), имеющее непустое пересечение с оцениваемым множеством;

покрытие сверху — верхнее покрытие, целиком содержащее оцениваемое множество;

покрытие снизу — нижнее покрытие, целиком содержащееся в оцениваемом множестве;

граничное покрытие — покрытие границы, целиком содержащее границу оцениваемого множества; состоит из параллелотопов, содержащих границу, а также некоторые малые области, лежащие вокруг границы как внутри, так и вне оцениваемого множества;

подпокрытие — покрытие, получаемое в результате измельчения некоторого исходного покрытия (например, с помощью бисекции), целиком содержащееся в исходном покрытии;

регулярное покрытие — покрытие (или подпокрытие), составленное из параллелотопов, пересекающихся между собой только границами (гранями, ребрами или вершинами, когда данные параллелотопы являются соседствующими), или непересекающихся (когда данные параллелотопы не являются соседствующими в данном покрытии);

пустое покрытие — покрытие (или подпокрытие), целиком составленное из точек (и параллелотопов), не удовлетворяющих заданному критерию принадлежности (критерию проверки).

Раздел I

Введение

Глава 1
Введение

Книга посвящена численным методам гарантированной аппроксимации множеств и применению этих методов в инженерной практике. Здесь под гарантированностью понимается нахождение внешней (иногда внутренней) аппроксимации исследуемых множеств, которая может быть, по крайней мере в принципе, выполнена с желаемой точностью. При этом становится возможным решать задачи, которые, как часто представляется, не могут быть решены численными методами. Это такие задачи, как нахождение всех решений системы нелинейных уравнений и неравенств или всех возможных глобальных оптимумов при векторных критериях.

Фон рисунка на обложке книги поясняет эту идею. Знаком вопроса иллюстрируется проблема, порожденная последовательностью неравенств, связанных логическими операторами. Исходная исследуемая область разбивается на три группы прямоугольников (двумерный аналог того, что мы называем параллелотопом, т. е. многомерным параллелепипедом с ребрами, ориентированными по осям соответствующей декартовой системы). Первая группа (отмечена красным цветом) составлена из прямоугольников, заведомо содержащихся в исследуемом множестве. Вторая группа (отмечена синим цветом) состоит из прямоугольников, заведомо имеющих пустое пересечение. Последняя группа (отмечена желтым цветом) содержит прямоугольники, относительно которых нельзя ничего доказать. Ценою большего объема вычислений размеры области, окрашенной в желтый цвет, могут быть достаточно просто уменьшены.

Основной инструмент, используемый в интервальном анализе, основан на очень простой идее окружения вещественных чисел интервалами и вещественных векторов областями прямоугольной формы — параллелотопами. При этом впервые появляется возможность получить гарантированную оценку результатов компьютерных вычислений прямым переходом к интервальным переменным в классических численных алгоритмах, применяемых обычно в вычислениях с числами с плавающей точкой. Совсем недавно интервальный анализ позволил построить алгоритмы дифференцирования, не имевшие ранее аналогов, специально предназначенные для работы с множествами. Это дало возможность применять *численные* методы для дока-

зательства утверждений относительно множеств. Таким образом, алгоритмы, основанные на интервальном анализе, дополняют алгоритмы на основе компьютерной алгебры. Алгоритмы на основе интервального анализа обладают тем преимуществом, что могут работать с более общими классами задач и с задачами, которые могут быть решены только численно (например, нахождение корней полиномиального уравнения высокой степени), но при этом решены гарантированно. Отметим, что обычные численные методы со статистическим поиском (Монте-Карло) или подстройкой сетки не могут быть использованы для доказательства даже таких простых свойств, как пустота или неодносвязность множества.

1.1. Основные идеи

Многие исследователи полагают, что некоторые из рассматриваемых задач аппроксимации и оценивания все еще не могут быть решены. Возникает вопрос: как интервальный анализ позволит это сделать? Вкратце, ответ, который дается в настоящей книге, состоит в том, что интервальный анализ позволяет получить гарантированные выводы о свойствах параллелотопов в исследуемом пространстве после конечного числа операций, хотя количество векторов в каждой из этих областей может быть даже несчетным. Это достигается покрытием исследуемых множеств параллелотопами или объединениями параллелотопов, вычисления на которых могут быть выполнены более просто, чем на исходном множестве.

Рассмотрим параллелотоп $[\mathbf{x}]$ в $\mathbb{R}^n$, некоторую функцию f, отображающую из $\mathbb{R}^n$ в $\mathbb{R}$, и некоторое подмножество $\mathbb{S}$ из $\mathbb{R}^n$, определенное последовательностью ограничений, связанных с логическими операторами, такими как "И" или "ИЛИ". На обложке книги знак вопроса представляет такое двумерное множество $\mathbb{S}$. Интервальный анализ делает возможным выполнение трех принципиально важных операций.

Первая операция — *ограничение образа* параллелотопа $[\mathbf{x}]$ в области значений функции f, т. е. вычисление некоторого интервала, который содержит $\{f(\mathbf{x}) \mid \mathbf{x} \in [\mathbf{x}]\}$. В этой операции ключевым моментом, прямо вытекающим из свойств интервального анализа, является понятие *функции включения*.

Вторая операция — *проверка принадлежности* параллелотопа $[\mathbf{x}]$ множеству $\mathbb{S}$, или, более точно, проверка $[\mathbf{x}] \subset \mathbb{S}$ или $[\mathbf{x}] \cap \mathbb{S} = \varnothing$. С этой целью вводится понятие операции *проверки включения*. Третья операция состоит *в сужении* параллелотопа $[\mathbf{x}]$ по отношению к $\mathbb{S}$, т. е., в замене $[\mathbf{x}]$ некоторым меньшим параллелотопом $[\mathbf{z}]$, таким, что $[\mathbf{x}] \cap \mathbb{S} = [\mathbf{z}] \cap \mathbb{S}$. Если обнаруживается, что параллелотоп $[\mathbf{z}]$ пуст, а $\mathbb{S}$ определяет допустимое множество значений аргумента по отношению к конкретному решению некоторой задачи, то $[\mathbf{x}]$ исключается из списка параллелотопов, которые могут содержать это решение.

Если не удается получить заключение о некотором заданном параллелотопе, то он может быть разбит на подпараллелотопы, и каждый из них может быть исследован отдельно. Эта операция соответствует алгоритму *ветвей и границ*. Главным недостатком этого алгоритма является экспоненциальное возрастание сложности относительно числа интервальных переменных. *Сжимающие операторы* могут использоваться для снижения (а иногда и исключения) необходимости разбиения параллелотопов на подпараллелотопы, и, таким образом, могут играть принципиальную роль в преодолении проклятья размерности.

Указанные выше понятия и операции составляют ядро основных рассматриваемых алгоритмов, и любая математическая теория, допускающая их реализацию, могла бы быть включена в интервальный анализ. Например, эллипсоиды уже используются для гарантированных вычислений на множествах. Интервальный анализ, однако, может использоваться на значительно более широком классе функций f и ограничений, определяющих множество $\mathbb{S}$. Кроме того, это может быть сделано так, что ошибки округления из-за неточного представления вещественных чисел в компьютере учитываются в получаемых гарантированных результатах.

1.2. Предыстория

В 1962 году Мур написал свою докторскую диссертацию по использованию интервальных оценок при анализе ошибок вычислений (и контроля за ними) на компьютерах. В 1966 году он опубликовал свою первую книгу «Интервальный анализ» [Moore, 1966], которая и поныне остается цитируемым источником. В тот же период Хансен изучал операции с интервалами в линейной алгебре [Hansen, 1965], а группа немецких исследователей, в том числе Алефельд, Кравчик и Никель, развила многие вопросы компьютерной реализации таких методов [Nickel, 1966]. Таким образом, интервальный анализ имеет нескольких родителей.

В течение первых двадцати лет распространение методологии интервального анализа ограничивалось местом своего возникновения, в основном в Германии, в Университете Карлсруэ [Kulisch и Miranker, 1981]. Среди новых поклонников, существенно продвинувших интервальный анализ, можно отметить Немайера [1985] по решению систем линейных и нелинейных уравнений, Ратшека и Рокне [1984] и Киерфотта [1989a] по задачам оптимизации.

За 90-е годы интервальный анализ привлек гораздо бóльший круг исследователей. Сообщество имеет теперь основанный в 1991 году свой соб-

ственный журнал «Интервальные вычисления», в 1995 году переименованный в журнал «Надежные вычисления», и несколько регулярно проводимых международных конференций. Ссылки на множество статей могут быть найдены на web-сайте:

http://liinwww.ira.uka.de/bibliography/?query=interval&case=off&partial=on.

Читателю рекомендуется также ознакомиться с очень популярным сайтом:

http://www.cs.uter.edu/interval-comp/main.html,

который целиком посвящен интервальному анализу.

1.3. О сложности вычислений

Интервальные алгоритмы будут требовать всегда большего времени, чем их аналоги с вещественными числами, если такие аналоги есть. Иногда приходится разбивать интервалы на подинтервалы, что может вызывать необходимость большего объема машинной памяти и приводить вскоре к проклятью размерности. Показатели увеличения машинного времени и необходимой памяти существенно меняются от задачи к задаче. Примеры, рассматриваемые в Главах 6–8, должны убедить читателя, что указанные сложности не являются сдерживающим фактором в большом числе практически интересных задач. Эти задачи были рассмотрены за последнее десятилетие на множестве персональных компьютеров и в программах, написанных на языках PASCAL, ADA и C++. Мы не нашли полезным пересчитывать задачи заново на одном компьютере с целью получить единую оценку времени счета, а просто приводим сведения о затратах машинного времени на исходных компьютерах. Читатель при этом может оставаться уверенным в том, что эти показатели пессимистичны — и иногда очень пессимистичны — гораздо выше затрат, которые могут быть получены на современных персональных компьютерах.

1.4. Структура книги

Раздел II посвящен техническим средствам интервального анализа. Все сделано для того, чтобы представить их как можно более просто, в конкретной и готовой к использованию форме. Некоторые из технических приемов впервые описываются в настоящей книге.

В Главе 2 напоминаются несколько простых положений из теории множеств, которые составляют методологическую основу интервального

анализа. Далее освещаются основные идеи интервального анализа, в том числе очень важные понятия функций включения и проверки включения.

В Главе 3 описываются покрытия, т. е. наборы неперекрывающихся параллелотопов, которые используются для аппроксимации компактных множеств. Напоминается несколько положений из топологии, которые необходимы для количественной оценки расстояния между покрытиями и компактными множествами. Поясняется представление одного полезного класса покрытий в виде двоичных деревьев, и расчет покрытий применяется в двух принципиально важных операциях, а именно, в процедуре обращения множества и в процедуре прямой количественной оценки образа.

Глава 4 посвящена сжимающим операторам, т. е. операторам, используемым для уменьшения размеров областей, на которых переменным разрешается изменяться при условии выполнения ими заданного набора ограничений. Сжимающие операторы уже упоминались выше в связи с их принципиальной ролью в преодолении проклятья размерности.

В Главе 5 рассматриваются методы решения — процедуры разрешающих операторов. Все интересующие нас задачи нельзя решить с использованием только сжимающих операторов, и иногда необходимо прибегать к разбиению параллелотопов пополам для получения с помощью новых покрытий более точных аппроксимаций множеств решений. Рассматриваются задачи нахождения множеств решений систем нелинейных уравнений или неравенств, оптимизации мультимодальных и минимаксных критериев.

Способность технических средств и приемов, рассмотренных в Разделе II, решать нетривиальные инженерные задачи демонстрируется в Разделе III. По каждому примеру приводится достаточный объем технических деталей, чтобы читатели, имея в виду другие применения этих приемов, могли ухватить суть каждого примера.

Глава 6 посвящена оцениванию, т. е. использованию экспериментальных данных для получения информации о численной величине некоторых переменных с неопределенностью, которые могут полагаться константами (идентификация параметров), или изменяющимися по времени (оценка текущего состояния или отслеживание меняющегося параметра). Оценивание выполняется или путем оптимизации целевой функции (в этом случае, например, используется оценивание по методу наименьших квадратов) или путем просмотра всех значений вектора рассматриваемых неопределенностей, которые совместны с имеющимися данными, включая априорные ограничения на допустимые ошибки. В обоих случаях интервальный анализ позволяет получать гарантированные результаты.

В Главе 7 рассматриваются две базовые задачи робастного управления. Первая — задача робастности некоторой заданной управляемой системы по отношению к неопределенности модели процесса, которым нужно управлять. Вторая, более сложная, заключается в проектировании системы

управления, обеспечивающей удовлетворяющий уровень показателя качества в условиях неопределенности.

Глава 8 посвящена трем трудным задачам из робототехники. Первая из них является задачей оценивания всех возможных конфигураций параллельного робота, известного как платформа Стюарта-Гофа при заданной длине его звеньев — нынешний классический пример применения компьютерной алгебры. Вторая задача — планирование бесконфликтной траектории движения робота в среде с препятствиями. Последняя задача — определение местоположения и слежение за движением аппарата на основе бортовых замеров дальности в среде с частично известными характеристиками.

Углубленное рассмотрение вопросов реализации в Разделе IV улучшает понимание читателем этого аспекта и применения свободно доступного программного обеспечения, которое делает интервальные вычисления настолько простыми, насколько просты обычные вычисления с числами с плавающей точкой.

Глава 9 представляет автоматическое дифференцирование — численный инструмент, который может быть использован для получения гарантированных оценок производных от функций по их аргументам, как это необходимо в некоторых алгоритмах, рассмотренных выше.

Глава 10 описывает средства, представляемые стандартом IEEE 754 для двоичной арифметики с плавающей точкой, которая используется в большинстве современных компьютеров, а также ограничения, которые она накладывает. Приводятся также указания на готовые программы для интервальных вычислений.

В Главе 11 читателям дается основная информация, необходимая для разработки на языке **C++** их собственных библиотек для интервальных вычислений или использования библиотеки `PROFILE/BIAS`. Реализация основных алгоритмов, описанных в книге, и их применение в иллюстрационных примерах детально рассматривается в упражнениях. Тексты программ, соответствующих решениям всех примеров из упражнений, могут быть загружены с web-сайта:

http:/www.lss.supelec.fr/books/intervals.

Раздел II

Аппарат интервального анализа

Глава 2

Основные понятия интервального анализа

2.1. Введение

Прежде чем использовать интервальный анализ в последующих главах как рабочее средство, мы введем его главные положения. В параграфе 2.2 напоминаются основные положения теории операторов и функций над множествами и исчисления множеств. Далее, в параграфе 2.3 представляются основные понятия интервального анализа. Параграф 2.4 посвящен важнейшему понятию функции включения. Наконец, в параграфе 2.5 рассматривается расширение на интервалы операций логических проверок, которые практически без изменений представлены в интерссующих нас алгоритмах.

2.2. Операции над множествами

Интервальное вычисление является специальным случаем вычислений над множеством, и теория множеств лежит в основе интервального анализа. Читатель, интересующийся только интервальными вычислениями, может пропустить этот параграф и перейти прямо к параграфу 2.3.

Операции над множествами разделяются на два типа. Первый, рассматриваемый в параграфе 2.2.1, содержит сведения об операциях, имеющих чисто теоретико-множественный смысл (такие как объединение, пересечение, прямое произведение и т. д.). Второй тип, рассматриваемый в параграфе 2.2.2, касается расширений на множества операций, которые были введены для работы над числами (или векторами).

2.2.1. Теоретико-множественные операции

Рассмотрим два множества $\mathbb{X}$ и $\mathbb{Y}$. Их *пересечение* представляется

$$\mathbb{X} \cap \mathbb{Y} \triangleq \{x \mid x \in \mathbb{X} \text{ и } x \in \mathbb{Y}\}, \tag{2.1}$$

их *объединение* записывается

$$\mathbb{X} \cup \mathbb{Y} \triangleq \{x \mid x \in \mathbb{X} \text{ или } x \in \mathbb{Y}\}. \tag{2.2}$$

Разность от вычитания множества $\mathbb{Y}$ из множества $\mathbb{X}$ определяется как

$$\mathbb{X} \setminus \mathbb{Y} \triangleq \{x \mid x \in \mathbb{X} \text{ и } x \notin \mathbb{Y}\}. \tag{2.3}$$

Прямое произведение множеств $\mathbb{X}$ и $\mathbb{Y}$ есть

$$\mathbb{X} \times \mathbb{Y} \triangleq \{(x, y) \mid x \in \mathbb{X} \text{ и } y \in \mathbb{Y}\}. \tag{2.4}$$

Если $\mathbb{Z} = \mathbb{X} \times \mathbb{Y}$, то *проекция* подмножества $\mathbb{Z}_1$ множества $\mathbb{Z}$ на множество $\mathbb{X}$ (по отношению к $\mathbb{Y}$) определяется как

$$\text{proj}_{\mathbb{X}}(\mathbb{Z}_1) \triangleq \{x \in \mathbb{X} \mid \exists y \in \mathbb{Y}, \text{ такое, что } (x, y) \in \mathbb{Z}_1\}.$$

Суть данных операций поясняется на рис. 2.1.

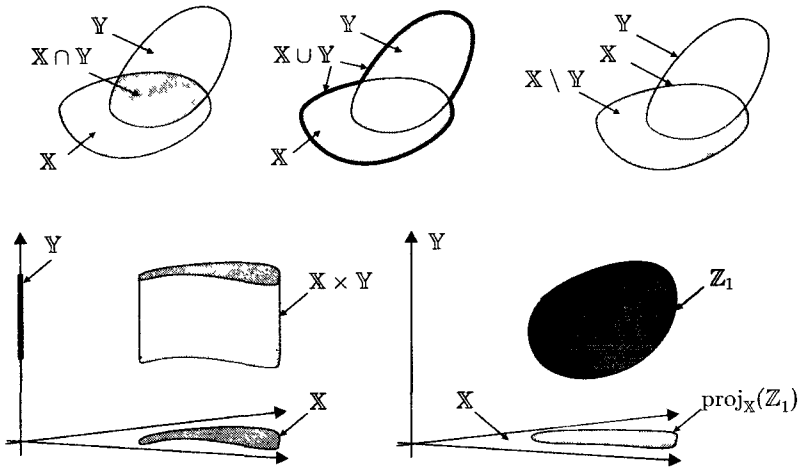


Рис. 2.1. Операции над множествами

Включение множества $\mathbb{X}$ в $\mathbb{Y}$ определяется как

$$\mathbb{X} \subset \mathbb{Y} \Leftrightarrow \forall x \in \mathbb{X},\ x \in \mathbb{Y}, \tag{2.5}$$

равенство множеств $\mathbb{X}$ и $\mathbb{Y}$ определяется как

$$\mathbb{X} = \mathbb{Y} \Leftrightarrow \mathbb{X} \subset \mathbb{Y} \text{ и } \mathbb{Y} \subset \mathbb{X}. \tag{2.6}$$

2.2.2. Расширенные операции

Рассмотрим два множества $\mathbb{X}$ и $\mathbb{Y}$ и некоторую функцию над ними $f : \mathbb{X} \to \mathbb{Y}$. Если $\mathbb{X}_1 \subset \mathbb{X}$, то *прямой образ* множества $\mathbb{X}_1$, определяемый отображением f, представляется

$$f(\mathbb{X}_1) \triangleq \{f(x) \mid x \in \mathbb{X}_1\}. \tag{2.7}$$

Если $\mathbb{Y}_1 \subset \mathbb{Y}$, то *прообраз* множества $\mathbb{Y}_1$ для f записывается как

$$f^{-1}(\mathbb{Y}_1) \triangleq \{x \in \mathbb{X} \mid f(x) \in \mathbb{Y}_1\}. \tag{2.8}$$

Если $\varnothing$ обозначает пустое множество, то из определений выше следует, что

$$f(\varnothing) = f^{-1}(\varnothing) = \varnothing. \tag{2.9}$$

Тривиально показать, что если $\mathbb{X}_1$ и $\mathbb{X}_2$ подмножества множества $\mathbb{X}$ и если $\mathbb{Y}_1$ и $\mathbb{Y}_2$ — подмножества $\mathbb{Y}$, то

$$\begin{gathered}
f(\mathbb{X}_1 \cap \mathbb{X}_2) \subset f(\mathbb{X}_1) \cap f(\mathbb{X}_2), \\
f(\mathbb{X}_1 \cup \mathbb{X}_2) = f(\mathbb{X}_1) \cup f(\mathbb{X}_2), \\
f^{-1}(\mathbb{Y}_1 \cap \mathbb{Y}_2) = f^{-1}(\mathbb{Y}_1) \cap f^{-1}(\mathbb{Y}_2), \\
f^{-1}(\mathbb{Y}_1 \cup \mathbb{Y}_2) = f^{-1}(\mathbb{Y}_1) \cup f^{-1}(\mathbb{Y}_2), \\
f\left(f^{-1}(\mathbb{Y})\right) \subset \mathbb{Y}, \\
f^{-1}(f(\mathbb{X})) \supset \mathbb{X}, \\
\mathbb{X}_1 \subset \mathbb{X}_2 \Rightarrow f(\mathbb{X}_1) \subset f(\mathbb{X}_2), \\
\mathbb{Y}_1 \subset \mathbb{Y}_2 \Rightarrow f^{-1}(\mathbb{Y}_1) \subset f^{-1}(\mathbb{Y}_2), \\
\mathbb{X} \subset \mathbb{Y}_1 \times \mathbb{Y}_2 \Rightarrow \mathbb{X} \subset \text{proj}_{\mathbb{Y}_1}(\mathbb{X}) \times \text{proj}_{\mathbb{Y}_2}(\mathbb{X}).
\end{gathered} \tag{2.10}$$

Тем же способом можно расширить операции над числами (или векторами) на операции над множествами. Обозначим через $\mathcal{P}(\mathbb{X})$ *мощность* множества $\mathbb{X}$, т. е. множество всех подмножеств $\mathbb{X}$. Пусть $\diamond$ — бинарное отображение из $\mathbb{X} \times \mathbb{Y}$ в $\mathbb{Z}$. Оно может быть расширено как операция над множествами следующим образом:

$$\mathbb{X}_1 \diamond \mathbb{Y}_1 \triangleq \{x_1 \diamond y_1 \mid x_1 \in \mathbb{X}_1,\ y_1 \in \mathbb{Y}_1\}, \tag{2.11}$$

где $\diamond$ — теперь отображение из $\mathcal{P}(\mathbb{X} \times \mathbb{Y})$ в $\mathcal{P}(\mathbb{Z})$. Например, если $\mathbb{X}_1$ и $\mathbb{Y}_1$ являются подмножествами множества $\mathbb{R}^n$, то

$$\mathbb{X}_1 + \mathbb{Y}_1 = \{\mathbf{x} + \mathbf{y} \mid \mathbf{x} \in \mathbb{X}_1,\ \mathbf{y} \in \mathbb{Y}_1\}, \tag{2.12}$$

$$\mathbb{X}_1 - \mathbb{Y}_1 = \{\mathbf{x} - \mathbf{y} \mid \mathbf{x} \in \mathbb{X}_1,\ \mathbf{y} \in \mathbb{Y}_1\}. \tag{2.13}$$

Заметим, что не надо путать множество $\mathbb{X}_1 - \mathbb{X}_1 = \{\mathbf{x} - \mathbf{y} \mid \mathbf{x} \in \mathbb{X}_1, \mathbf{y} \in \mathbb{X}_1\}$ с множеством $\{\mathbf{x} - \mathbf{x} \mid \mathbf{x} \in \mathbb{X}_1\} = \{\mathbf{0}\}$.

Когда оператор $\diamond$ действует на некоторый элемент x_1 из $\mathbb{X}$ и $\mathbb{Y}_1$ — подмножество из $\mathbb{Y}$, то x_1 преобразуется в одиночный элемент $\{x_1\}$, такой, что

$$x_1 \diamond \mathbb{Y}_1 \triangleq \{x_1\} \diamond \mathbb{Y}_1 = \{x_1 \diamond y_1 \mid y_1 \in \mathbb{Y}_1\}. \tag{2.14}$$

Например, если $\mathbb{D}$ круг из $\mathbb{R}^2$ с центром $\mathbf{c}$ и радиусом 4, то $3*\mathbb{D}$ является кругом в $\mathbb{R}^2$ с центром $3\mathbf{c}$ и радиусом 12. Если $\mathbb{Y}_1$ — одиночный элемент $\{y_1\}$, то

$$x_1 \diamond y_1 = \{x_1\} \diamond \{y_1\} = \{x_1 \diamond y_1\}. \tag{2.15}$$

Для обозначений точечных аргументов в операторе $\diamond$ используются обычные правила.

2.2.3. Свойства операторов над множествами

Некоторые свойства операторов над числами расширяются на их аналоги, действующие над множествами. Рассмотрим в качестве иллюстрации некоторое множество $\mathbb{X}$ с бинарным отображением $\diamond$ на нем. Полагаем, что $\mathbb{X}$ *замкнуто* относительно оператора $\diamond$ (т. е. если x и y принадлежат $\mathbb{X}$, то $x \diamond y$ принадлежит $\mathbb{X}$). Положим также, что $\diamond$ расширен на $\mathcal{P}(\mathbb{X})$, как это показано в предыдущем параграфе. Некоторые свойства, имеющие место для $(\mathbb{X}, \diamond)$, имеют место и для $(\mathcal{P}(\mathbb{X}), \diamond)$. Например, если оператор $\diamond$ коммутативен на $\mathbb{X}$, то он также коммутативен на $\mathcal{P}(\mathbb{X})$, т. е.

$$\begin{aligned}(\forall x_1 \in \mathbb{X},\ \forall x_2 \in \mathbb{X},\ x_1 \diamond x_2 = x_2 \diamond x_1) \Rightarrow \\ \Rightarrow (\forall \mathbb{X}_1 \in \mathcal{P}(\mathbb{X}),\ \forall \mathbb{X}_2 \in \mathcal{P}(\mathbb{X}),\ \mathbb{X}_1 \diamond \mathbb{X}_2 = \mathbb{X}_2 \diamond \mathbb{X}_1).\end{aligned} \tag{2.16}$$

Если оператор $\diamond$ ассоциативен на $\mathbb{X}$, то он также ассоциативен на $\mathcal{P}(\mathbb{X})$, т. е.

$$\begin{aligned}&\left(\forall (x_1, x_2,\ x_3) \in \mathbb{X}^3, x_1 \diamond (x_2 \diamond x_3) = (x_1 \diamond x_2) \diamond x_3\right) \Rightarrow \\ &\Rightarrow \left(\forall (\mathbb{X}_1, \mathbb{X}_2, \mathbb{X}_3) \in (\mathcal{P}(\mathbb{X}))^3,\ \mathbb{X}_1 \diamond (\mathbb{X}_2 \diamond \mathbb{X}_3) = (\mathbb{X}_1 \diamond \mathbb{X}_2) \diamond \mathbb{X}_3\right).\end{aligned} \tag{2.17}$$

Если оператор $\diamond$ допускает существование единичного элемента (обозначим его как 0) в $\mathbb{X}$, то этот оператор также допускает существование единичного элемента в $\mathcal{P}(\mathbb{X})$, который является единичным элементом $\{0\}$.

С другой стороны, некоторые свойства, имеющие место на $(\mathbb{X}, \diamond)$, могут отсутствовать на $(\mathcal{P}(\mathbb{X}), \diamond)$. Например, если каждый элемент x в $(\mathbb{X}, \diamond)$ имеет

симметричный элемент, то это уже не имеет место в $(\mathcal{P}(\mathbb{X}), \diamond)$. Имеет место только

$$\begin{aligned}(\forall x_1 \in \mathbb{X},\ \exists y_1 \in \mathbb{X} \mid x_1 \diamond y_1 = 0) \Rightarrow \\ \Rightarrow (\forall \mathbb{X}_1 \in \mathcal{P}(\mathbb{X}),\ \exists \mathbb{Y}_1 \in \mathcal{P}(\mathbb{X}) \mid \mathbb{X}_1 \diamond \mathbb{Y}_1 \ni 0).\end{aligned} \tag{2.18}$$

Таким образом, если $(\mathbb{X}, \diamond)$ является группой, то $(\mathcal{P}(\mathbb{X}), \diamond)$ — уже только моноид.

Нашей целью не является дать исчерпывающий обзор расширений операторов над числами на операторы над множествами, отметим только, что надо действовать очень осторожно, когда имеешь дело с арифметическими действиями. Например, если $\mathbb{X}_1$ является подмножеством группы $(\mathbb{X}, +)$, выражение вида

$$\mathbb{Z} = \mathbb{X}_1 - \mathbb{X}_1 \tag{2.19}$$

не должно истолковываться как

$$\mathbb{Z} = \{\mathbf{x} - \mathbf{x} \mid \mathbf{x} \in \mathbb{X}_1\}, \tag{2.20}$$

но как

$$\mathbb{Z} = \{\mathbf{x} - \mathbf{y} \mid \mathbf{x} \in \mathbb{X}_1,\ \mathbf{y} \in \mathbb{X}_1\}, \tag{2.21}$$

(см. соотношение (2.13)). Чтобы избежать подобной путаницы, использование алгебраических операций по тексту книги максимально ограничено.

Следующая теорема будет иметь важные последствия по отношению к интервальным вычислениям.

Теорема 2.1. *Рассмотрим функцию*

$$f: \begin{array}{ccc} \mathbb{X}(1) \times \ldots \times \mathbb{X}(n) & \to & \mathbb{Y} \\ (x_1, \ldots, x_n) & \mapsto & y, \end{array}$$

в формальном выражении которой имеются операторы и функции. Пусть $\mathbb{F}(\mathbb{X}_1, \ldots, \mathbb{X}_n)$ *— отображение из* $\mathcal{P}(\mathbb{X}(1)) \times \ldots \times \mathcal{P}(\mathbb{X}(n))$ *в* $\mathcal{P}(\mathbb{Y})$. *Предположим, что* $\mathbb{F}(\mathbb{X}_1, \ldots, \mathbb{X}_n)$ *имеет то же самое формальное выражение, что и* $f(\mathbb{X}_1, \ldots, \mathbb{X}_n)$. *Тогда*

$$f(\mathbb{X}_1, \ldots, \mathbb{X}_n) \subset \mathbb{F}(\mathbb{X}_1, \ldots, \mathbb{X}_n), \tag{2.22}$$

где

$$f(\mathbb{X}_1, \ldots, \mathbb{X}_n) = \{f(x_1, \ldots, x_n) \mid x_1 \in \mathbb{X}_1, \ldots, x_n \in \mathbb{X}_n\}.$$

Кроме того, если каждый элемент x_i по крайней мере однажды появляется в формальном выражении f, то

$$f(\mathbb{X}_1, \ldots, \mathbb{X}_n) = \mathbb{F}(\mathbb{X}_1, \ldots, \mathbb{X}_n). \tag{2.23}$$

■

Доказательство Теоремы 2.1 для случая, кода все функции, включенные в формальную запись f, являются непрерывными, а множества $\mathbb{X}_i$ являются замкнутыми интервалами, можно найти в [Moore, 1979]. Расширение Теоремы 2.1 на множество более общего вида (в той форме, как это указано в теореме) является прямым следствием теоремы распространения ограничений [Jaulin, 2001b] (на основе метода вперед-назад), суть которой будет пояснена далее в параграфе 6.4.4, (с. 224).

В общем случае, к сожалению, $\mathbb{F}(\mathbb{X}_1, \ldots, \mathbb{X}_n)$ является только аппроксимацией сверху для $f(\mathbb{X}_1, \ldots, \mathbb{X}_n)$ вследствие многократности появления переменных в формальном выражении для f. Вычисление множества при этом становится *проблематичным* из-за *эффекта зависимости* между переменными, как это показывается в нижеследующем Примере 2.1.

Пример 2.1. *Рассмотрим функцию* $f : \mathbb{R}^2 \to \mathbb{R}$, $f(x_1, x_2) = x_1 + x_2 - x_1$. *Тогда*

$$\begin{array}{rcl} f(\mathbb{X}_1, \mathbb{X}_2) & = & \{x_1 + x_2 - x_1 \mid x_1 \in \mathbb{X}_1, x_2 \in \mathbb{X}_2\} \\ & = & \{x_2 \mid x_2 \in \mathbb{X}_2\} = \mathbb{X}_2, \end{array} \tag{2.24}$$

и

$$\begin{array}{rcl} \mathbb{F}(\mathbb{X}_1, \mathbb{X}_2) & = & \mathbb{X}_1 + \mathbb{X}_2 - \mathbb{X}_1 \\ & = & \{x_1 + x_2 - x_3 \mid x_1 \in \mathbb{X}_1, x_2 \in \mathbb{X}_2, x_3 \in \mathbb{X}_1\}. \end{array} \tag{2.25}$$

Ясно, что $f(\mathbb{X}_1, \mathbb{X}_2) \subset \mathbb{F}(\mathbb{X}_1, \mathbb{X}_2)$. *Зависимость между* x_1 *и* x_3 (при $x_1 = x_3$) *пропадает в* (2.25), *но при этом добавляется одна степень свободы в выборе множества* $\mathbb{F}(\mathbb{X}_1, \mathbb{X}_2)$; *это и приводит к упомянутому ухудшению оценивания.* ■

Отметим, что многократность появления переменных, связанных с единичными элементами в образе, сама по себе не вызывает ухудшения оценивания.

2.2.4. Оболочки

Рассмотрим множество $\mathbb{X}$ и некоторое множество $\mathbb{IX}$ подмножеств $\mathbb{X}$. $\mathbb{IX}$ является *множеством оболочек* для $\mathbb{X}$, если $\mathbb{X}$ и каждый отдельный

элемент из $\mathbb{X}$ принадлежат множеству $\mathbb{IX}$, и если $\mathbb{IX}$ является замкнутым по отношению к операции пересечения (т. е., если $\mathbb{X}_1 \in \mathbb{IX}$ и $\mathbb{X}_2 \in \mathbb{IX}$, то $\mathbb{X}_1 \cap \mathbb{X}_2 \in \mathbb{IX}$). Пустое множество $\varnothing$ при этом должно принадлежать множеству $\mathbb{IX}$, если $\mathbb{X}$ не является одиночным элементом.

Пример 2.2. *Некоторое множество оболочек для* $\mathbb{X} = \{a, b, c, d\}$ *имеет вид*

$$\mathbb{IX} = \{\varnothing,\ a, b, c,\ \{a, b\},\ \{a, d\},\ \{a, b, c, d\}\}. \tag{2.26}$$

■

Пусть $\mathbb{X}_1$ — некоторое подмножество множества $\mathbb{X}$; *наименьшая оболочка* $[\mathbb{X}_1]$ множества $\mathbb{X}_1$ является наименьшим элементом из $\mathbb{IX}$, содержащим $\mathbb{X}_1$:

$$[\mathbb{X}_1] = \bigcap \{\mathbb{Y} \in \mathbb{IX} \mid \mathbb{X}_1 \subset \mathbb{Y}\}. \tag{2.27}$$

Пример 2.3. *Множество* $\mathcal{P}(\mathbb{X})$ *всех подмножеств множества* $\mathbb{X}$ *является множеством оболочек для* $\mathbb{X}$. *Если* $\mathbb{X}_1$ *является подмножеством* $\mathbb{X}$, *то* $[\mathbb{X}_1] = \mathbb{X}_1$. ■

Пример 2.4. *Когда* $\mathbb{X} = \mathbb{R}^n$, *множество* $\mathbb{IX}$ *всех выпуклых подмножеств* $\mathbb{R}^n$ *является множеством оболочек для* $\mathbb{X}$. *Тогда* $[\mathbb{X}_1]$ *есть выпуклая оболочка множества* $\mathbb{X}_1$. ■

На практике *оболочки* по своей структуре должны быть достаточно простыми, чтобы позволить выполнять вычисления над внешними аппроксимациями множеств. Оболочки, которые рассматриваются в данной книге, могут быть интервалами из $\mathbb{R}$, параллелотопами из $\mathbb{R}^n$ с ребрами, ориентированными по осям $\mathbb{R}^n$, подмножествами булевского множества {ДА, НЕТ}, а также могут быть объединениями интервалов или параллелотопов. Можно было бы также использовать другие виды оболочек, таких как политопы, многомерные области, выпуклые множества или их объединения. Хотя такие множества не являются замкнутыми по отношению к операции пересечения, множества в форме эллипсоидов также могут использоваться [Milanese, 1996].

Пусть $\diamond$ — бинарное отображение из $\mathbb{X} \times \mathbb{Y}$ в $\mathbb{Z}$. Пусть $\mathbb{IX}$, $\mathbb{IY}$ и $\mathbb{IZ}$ — множества оболочек $\mathbb{X}$, $\mathbb{Y}$ и $\mathbb{Z}$. Оператор $\diamond$ может быть расширен на эти множества оболочек следующим образом. Если $\mathbb{X}_1 \in \mathbb{IX}$ и $\mathbb{Y}_1 \in \mathbb{IY}$, то

$$\mathbb{X}_1\ [\diamond]\ \mathbb{Y}_1 \triangleq [\{x_1 \diamond y_1 \mid x_1 \in \mathbb{X}_1, y_1 \in \mathbb{Y}_1\}], \tag{2.28}$$

где оператор построения оболочки $[\cdot]$ над правой частью определяется по (2.27). Поэтому, по определению $\mathbb{X}_1 \diamond \mathbb{Y}_1$, даваемому выражением (2.11),

имеем

$$\mathbb{X}_1 \ [\diamond] \ \mathbb{Y}_1 \ \supset \ \mathbb{X}_1 \diamond \mathbb{Y}_1. \tag{2.29}$$

Рассмотрим два множества $\mathbb{X}$ и $\mathbb{Y}$, некоторое отображение $f : \mathbb{X} \to \mathbb{Y}$ и два множества оболочек $\mathbb{I}\mathbb{X}$ и $\mathbb{I}\mathbb{Y}$ для $\mathbb{X}$ и $\mathbb{Y}$, соответственно. Если $\mathbb{X}_1 \in \mathbb{I}\mathbb{X}$, то f может быть расширено на оболочки как

$$[f](\mathbb{X}_1) \triangleq [\{f(x) \mid x \in \mathbb{X}_1\}]. \tag{2.30}$$

Снова получаем

$$[f](\mathbb{X}_1) \supset f(\mathbb{X}_1). \tag{2.31}$$

Рассмотрим три множества $\mathbb{X}$, $\mathbb{Y}$, $\mathbb{Z}$, множества их оболочек $\mathbb{I}\mathbb{X}$, $\mathbb{I}\mathbb{Y}$, $\mathbb{I}\mathbb{Z}$ и три отображения $f : \mathbb{X} \to \mathbb{Y}$, $g : \mathbb{Y} \to \mathbb{Z}$, $h = g \circ f$ (где $\circ$ обозначает оператор-суперпозицию). Тогда $g(f(\mathbb{X}_1)) = h(\mathbb{X}_1)$, но при этом можно записать только

$$[g]([f](\mathbb{X}_1)) \supset [h](\mathbb{X}_1). \tag{2.32}$$

Данное действие известно как *эффект обертывания*, когда оболочки являются выпуклыми множествами в $\mathbb{R}^2$. Это явление иллюстрируется на рис. 2.2.

Как следствие эффекта зависимости между переменными, эффект обертывания приводит к ухудшению аппроксимации в операциях с оболочками. Некоторые свойства вычислений, которые имеют место при работе над множествами, не сохраняются при работе с оболочками. Это как раз тот случай. Применительно к Теореме 2.1, когда имеем дело с оболочками, свойство включения (2.22) еще сохраняется; но это не значит, что оно перейдет в равенство (2.23), когда каждая из входных переменных появляется более одного раза в формальном выражении функции f.

Замечание 2.1. В настоящей книге рассматриваются вычисления с оболочками, а не с порождающими их множествами. В тех местах, где это не приведет к двусмысленности, выражение $\mathbb{X}_1 \, [\diamond] \, \mathbb{Y}_1$ записывается в более краткой форме $\mathbb{X}_1 \diamond \mathbb{Y}_1$. ■

2.3. Интервальный анализ

В интервальном анализе оболочки, которые будут использоваться, являются интервалами, когда работаем с множеством вещественных чисел $\mathbb{R}$ и параллелотопами с ребрами, ориентированными по осям многомерного вещественного пространства $\mathbb{R}^n$. Используя основные идеи параграфа 2.2 по вычислениям над множествами, введем интервальные вычисления.

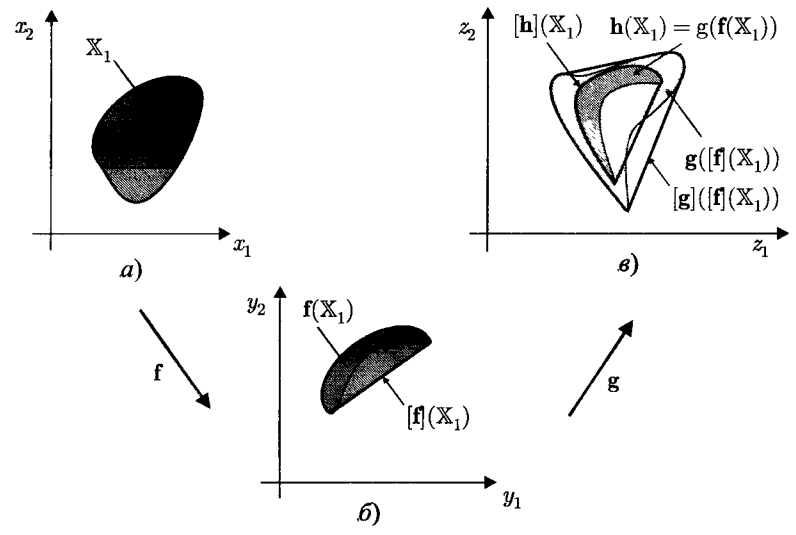


Рис. 2.2. Эффект обертывания, когда оболочки являются выпуклыми множествами в $\mathbb{R}^2$: *а*) $\mathbb{X}_1$ выпукло и является оболочкой; *б*) $\mathbf{f}(\mathbb{X}_1)$, отмечено темно-серой заливкой, невыпукло, а его выпуклой оболочкой является обертка $[\mathbf{f}](\mathbb{X}_1)$; *в*) поскольку $\mathbf{f}(\mathbb{X}_1) \subset [\mathbf{f}](\mathbb{X}_1)$, имеем $\mathbf{g}(\mathbf{f}(\mathbb{X}_1)) \subset \mathbf{g}([\mathbf{f}](\mathbb{X}_1)) \subset [\mathbf{g}]([\mathbf{f}](\mathbb{X}_1))$, следовательно, $[\mathbf{g}]([\mathbf{f}](\mathbb{X}_1))$ является выпуклым множеством, которое содержит выпуклую оболочку $[\mathbf{h}](\mathbb{X}_1)$ множества $\mathbf{g}(\mathbf{f}(\mathbb{X}_1))$; нежелательные точки в $[\mathbf{g}]([\mathbf{f}](\mathbb{X}_1))$, которые лежат вне $[\mathbf{h}](\mathbb{X}_1)$, являются прямым следствием эффекта обертывания

2.3.1. Интервалы

Вещественное интервальное число $[x]$ — это некоторое односвязное подмножество из $\mathbb{R}$. Мы будем придерживаться этого обозначения $[x]$, даже если это подмножество незамкнуто. Там, где путаница не возникает, интервальное число $[x]$ часто будет просто именоваться *интервалом*. Должно ли пустое множество $\varnothing$ считаться интервалом, если оно находится в рассмотрении? Ответ положительный, если мы убеждаемся, что рассматриваемое множество интервалов замкнуто по отношению к операции пересечения, а также если это множество $\varnothing$ показывает на отсутствие решений некоторой задачи.

Нижняя граница интервала $[x]$ обозначается как $\mathrm{lb}([x])$ или $\underline{x}$ и определяется

$$\underline{x} = \mathrm{lb}([x]) \triangleq \sup\{a \in \mathbb{R} \cup \{-\infty, \infty\} \mid \forall x \in [x], \ a \leqslant x\}. \tag{2.33}$$

Верхняя граница интервала обозначается как $\mathrm{ub}([x])$ или $\overline{x}$ и определяется как

$$\overline{x} = \mathrm{ub}([x] \triangleq \inf\{b \in \mathbb{R} \cup \{-\infty, \infty\} \mid \forall x \in [x],\ x \leqslant b\}. \tag{2.34}$$

Таким образом, $\underline{x}$ есть наибольшее число слева от интервала $[x]$, а $\overline{x}$ есть наименьшее число справа от этого интервала. Например, если $[x] =]-3, 7]$, то $\underline{x} = -3$ и $\overline{x} = 7$; если $[x] =]-\infty, \infty[$, то $\underline{x} = -\infty$ и $\overline{x} = \infty$.

Ширина любого непустого интервала $[x]$ определяется как

$$w([x]) \triangleq \overline{x} - \underline{x}, \tag{2.35}$$

поэтому $w(]3, \infty[) = \infty$.

Средняя точка (или *центр*) любого ограниченного и непустого интервала $[x]$ определяется как

$$\mathrm{mid}([x]) \triangleq \frac{\underline{x} + \overline{x}}{2}. \tag{2.36}$$

Теоретико-множественные операции параграфа 2.2.1 могут быть применены к интервалам. *Пересечение* двух интервалов $[x]$ и $[y]$, определенное в виде

$$[x] \cap [y] \triangleq \{z \in \mathbb{R} \mid z \in [x] \text{ и } z \in [y]\}, \tag{2.37}$$

всегда является интервалом. Это не имеет места для их *объединения*

$$[x] \cup [y] \triangleq \{z \in \mathbb{R} \mid z \in [x] \text{ или } z \in [y]\}. \tag{2.38}$$

Чтобы сделать множество интервалов замкнутым по отношению к операции их объединения, определим *интервальную оболочку* некоторого подмножества $\mathbb{X}$ множества $\mathbb{R}$ как наименьший интервал $[\mathbb{X}]$, который содержит это подмножество. Это согласуется с (2.27). Например, интервальная оболочка множества $]2, 3] \cup [5, 7]$ есть интервал $]2, 7]$. Определим *интервальный оператор объединения* множеств $[x]$ и $[y]$, обозначим его $[x] \sqcup [y]$, как интервальную оболочку множества $[x] \cup [y]$, т. е.

$$[x] \sqcup [y] \triangleq [[x] \cup [y]]. \tag{2.39}$$

Аналогично,

$$[x]\ [\backslash]\ [y] = [[x] \setminus [y]] = [\{x \in [x] \mid x \notin [y]\}]. \tag{2.40}$$

Например, $[0, 5[\ [\backslash]\ ]3, 4[= [0, 5[$ и $[0, 5[\ [\backslash]\ ]3, 5[= [0, 3]$.

Прямое произведение двух интервалов есть уже не интервал, а прямоугольник $\mathbb{R}^2$, при этом он соответствует некоторой внешней операции (этот вопрос рассматривается в параграфе 2.3.4).

2.3.2. Интервальные вычисления

Четыре классические арифметические операции над вещественными числами, а именно, сложение $(+)$, вычитание $(-)$, умножение $(*)$ и деление $(/)$ могут быть расширены на действия с интервалами. Для любого бинарного оператора, обозначенного как $\diamond$, выполнение операции с ним связано с интервалами $[x]$ и $[y]$ и означает вычисление

$$[x] \diamond [y] = [\{x \diamond y \in \mathbb{R} \mid x \in [x], y \in [y]\}], \tag{2.41}$$

что является прямым следствием (2.28). Например,

$$(]1, 2.2] * [0, 2[) +]1, 3] = [0, 4.4[+]1, 3] =]1, 7.4[, \tag{2.42}$$

$$1/[-2, 2[=]-\infty, \infty[, \tag{2.43}$$

$$[3, 4]/[0, 0] = \varnothing. \tag{2.44}$$

Правило (2.41) было впервые введено при работе с ограниченными замкнутыми интервалами в [Moore, 1959] и потом расширено на действия с неограниченными интервалами с незамкнутыми концами [Hanson, 1968; Kahan, 1968; Davis, 1987].

ЗАМЕЧАНИЕ 2.2. Результат (2.44) основывается на математической интерпретации интервала $[0, 0]$. В Главе 10 мы покажем, что при выполнении интервальных вычислений выделяются и используются два типа нулей в представлениях с числами с плавающей точкой, и они могут приводить к различным результатам. ■

ЗАМЕЧАНИЕ 2.3. Определение (2.41) отличается от определения, данного для множеств в (2.11), так как требуется, чтобы оператор $\diamond$ давал интервал, а не какое-нибудь, может быть даже несвязное, подмножество из $\mathbb{R}$. Вследствие эффекта обертывания, мы были вынуждены написать $[x]\ [\diamond]\ [y]$ вместо $[x] \diamond [y]$, но также, как и для исходных оболочек, мы будем придерживаться здесь упрощенного обозначения $[x] \diamond [y]$. Заметим, что (2.41) и (2.11) имеют место для непрерывного оператора $\diamond$, так как множество $\{x \diamond y \in \mathbb{R} \mid x \in [x], y \in [y]\}$ является интервалом. ■

ЗАМЕЧАНИЕ 2.4. Оператор $\diamond$, определенный на интервалах, может быть расширен на объединения интервалов (называемые также *неодносвязными интервалами*) [Hyvonen, 1992]. Например, можно записать

$$1/[-2, 2[=]-\infty, -1/2[\cup [1/2, \infty[, \tag{2.45}$$

вместо $]-\infty, \infty[$ как в (2.43). Вычисления на объединениях интервалов могут, однако, привести к взрывному увеличению числа подинтервалов, с которыми приходится работать. Поэтому далее в книге такие вычисления не рассматриваются. ■

Как уже отмечалось в отношении исходных множеств, свойства основных операторов над интервалами отличаются от их свойств над всем множеством $\mathbb{R}$. Например, $[x] - [x]$ в общем случае не равно $[0, 0]$. Это имеет

место вследствие того, что $[x]-[x]=\{x-y \mid x\in[x],\ y\in[x]\}$ отличается от $\{x-x \mid x\in[x]\}$. Таким образом, операция вычитания не учитывает зависимость результата от двух появлений интервала $[x]$. Операции сложения и умножения остаются ассоциативными и коммутативными, но умножение перестает быть дистрибутивным по отношению к сложению. Вместо этого имеем

$$[x]*([y]+[z])\subset[x]*[y]+[x]*[z], \tag{2.46}$$

это — свойство, известное как *полудистрибутивность*; оно есть прямое следствие эффекта зависимости переменных, так как $[x]$ появляется только один раз в левой части операции, но дважды — в правой части. Как результат, рекомендуется как можно больше факторизовать расширенные формы операций.

Элементарные функции, такие как $\exp, \mathrm{tg}, \sin, \cos \ldots$, расширяются на вычисления с интервалами как показано в (2.30). Если f есть отображение из $\mathbb{R}$ в $\mathbb{R}$, то его интервальный аналог $[f]$ удовлетворяет соотношению

$$[f]([x])=[\{f(x) \mid x\in[x]\}]. \tag{2.47}$$

Для любой непрерывной элементарной функции множество $[f]([x])$ совпадает с множеством образа $f([x])$. Например,

$$\begin{aligned}
[\mathrm{arctg}]([0,\infty[) &= [0,\pi/2[,\\
[\mathrm{sqr}]([-1,3]) &= [0,9[,\\
[\exp]([0,1[) &= [\exp(0),\exp(1)[= [1,e[,\\
[\mathrm{sqrt}]([4,25]) &= [\mathrm{sqrt}(4),\ \mathrm{sqrt}(25)] = [2,5],\\
[\mathrm{sqrt}]([-25,-4]) &= \varnothing,\\
[\mathrm{sqrt}]([-50,1]) &= [\mathrm{sqrt}]([0,1]) = [0,1],\\
[\ln]([-50,1[) &=]-\infty,0[.
\end{aligned} \tag{2.48}$$

2.3.3. Замкнутые интервалы

В данном параграфе описываются правила выполнения вычислений над подмножествами множества $\mathbb{R}$ для специального случая, когда все оболочки являются замкнутыми интервалами в $\mathbb{R}$. Обозначим как $\mathbb{IR}$ множество всех таких замкнутых интервалов. Поскольку $\mathbb{R}$ и $\varnothing$ оба одновременно являются открытыми и замкнутыми, они оба принадлежат множеству $\mathbb{IR}$, и любой элемент из $\mathbb{IR}$ может быть записан в одной из следующих форм: $[a,b],]-\infty,a], [a,\infty[,]-\infty,\infty[$ или $\varnothing$, где a и b — вещественные числа, такие, что $a\leqslant b$. Любой интервал $[x]$ из $\mathbb{IR}$ может быть единственным

образом задан своей нижней $\underline{x}$ и верхней $\overline{x}$ границами. Для простоты, мы будем часто писать $[x]=[\underline{x},\overline{x}]$, даже если эти границы бесконечны. Таким образом, выражение $[0,\infty]$ должно интерпретироваться как $[0.\infty[$. Отметим двойную природу замкнутых интервалов, которые могут рассматриваться и как *множества* — тогда к ним применяются стандартные операции над множествами, и как *пары элементов* из $\mathbb{R}$ — тогда на них можно ввести соответствующие арифметические действия. Пары вида $[\infty,\infty], [-\infty,-\infty]$ и $[a,b]$ при $a>b$ не являются интервалами (см. *Модальная интервальная арифметика* [Gerdenes, 1985], где, однако, обозначению $[a,b]$ при $a>b$ присваивается некий смысл). Когда $\underline{x}$ и $\overline{x}$ совпадают, интервал $[x]$ называется *точечным* (или вырожденным). Таким образом, любое вещественное число может быть представлено как точечный интервал и наоборот.

Операции из параграфа 2.3.2 могут быть переопределены по отношению к замкнутым интервалам как операции над их границами: границы результата выполнения интервальной операции выражаются в виде значений функций от границ их интервальных аргументов. Оставшаяся часть этого параграфа посвящена описанию того, как это можно сделать.

Интервальное объединение двух непустых замкнутых интервалов $[x]$ и $[y]$, определенное соотношением (2.39), записывается как

$$\forall[x]\in\mathbb{IR},\ \forall[y]\in\mathbb{IR},\quad [x]\sqcup[y]=\left[\min\{\underline{x},\underline{y}\},\max\{\overline{x},\overline{y}\}\right]. \tag{2.49}$$

Интервальное пересечение двух непустых замкнутых интервалов $[x]$ и $[y]$, определяемое по (2.37), удовлетворяет соотношениям

$$\begin{aligned}
[x]\cap[y] &= \left[\max\{\underline{x},\underline{y}\},\min\{\overline{x},\overline{y}\}\right], \text{ если } \max\{\underline{x},\underline{y}\}\leqslant\min\{\overline{x},\overline{y}\},\\
&= \varnothing, \text{ в остальных случаях.}
\end{aligned} \tag{2.50}$$

Если α — некоторое вещественное число и $[x]$ — некоторый непустой интервал, то интервал

$$\alpha[x]\triangleq\{\alpha x \mid x\in[x]\} \tag{2.51}$$

задается как

$$\alpha[x]=\begin{cases}[\alpha\underline{x},\ \alpha\overline{x}], & \text{если } \alpha\geqslant 0,\\ [\alpha\overline{x},\ \alpha\underline{x}], & \text{если } \alpha<0.\end{cases} \tag{2.52}$$

Для непустых замкнутых интервалов

$$\begin{aligned}
[x]+[y] &= [\underline{x}+\underline{y},\overline{x}+\overline{y}],\\
[x]-[y] &= [\underline{x}-\overline{y},\overline{x}-\underline{y}],\\
[x]*[y] &= [\min\{\underline{x}\underline{y},\underline{x}\overline{y},\overline{x}\underline{y},\overline{x}\overline{y}\},\ \max\{\underline{x}\underline{y},\underline{x}\overline{y},\overline{x}\underline{y},\overline{x}\overline{y}\}].
\end{aligned} \tag{2.53}$$

Интервальное произведение двух интервалов равноправно может быть написано двояко, либо как $[x]*[y]$, либо как $[x][y]$.

Операция *интервального деления* по (2.41) определяется соотношениями

$$1/[y] = \begin{cases} \varnothing, & \text{если } [y] = [0,0], \\ [1/\overline{y}, 1/\underline{y}], & \text{если } 0 \notin [y], \\ [1/\overline{y}, \infty[, & \text{если } \underline{y} = 0 \text{ и } \overline{y} > 0, \\ ]-\infty, 1/\underline{y}], & \text{если } \underline{y} < 0 \text{ и } \overline{y} = 0, \\ ]-\infty, \infty[, & \text{если } \underline{y} < 0 \text{ и } \overline{y} > 0, \end{cases} \tag{2.54}$$

и

$$[x]/[y] \quad = \quad [x] * (1/[y]). \tag{2.55}$$

Разумеется, в случае работы с точечными интервалами $[x]$ и $[y]$, указанные правила превращаются в обычные арифметические действия с вещественными числами. Вот почему интервальные арифметические действия могут считаться расширением обычных арифметических действий.

Элементарные интервальные функции также могут быть выражены в терминах границ. Например, для любого непустого интервала $[x]$

$$[\exp]([x]) = [\exp(\underline{x}), \exp(\overline{x})]. \tag{2.56}$$

Для немонотонных функций ситуация несколько сложнее. Например, интервал $[\sin]([0,\pi]) = [0,1]$ отличается от интервала $[\sin(0), \sin(\pi)] = [0,0]$. Следовательно, в этом случае должны быть разработаны специальные алгоритмы. Алгоритм для интервальных вычислений с функцией синуса приводится в табл. 2.1.

Таблица 2.1. Алгоритм интервальных вычислений с функцией синуса

Алгоритм sin(вход: $[x]$; выход: $[r]$)	
1	если $\exists k \in \mathbb{Z} \mid 2k\pi - \pi/2 \in [x]$, то $\underline{r} = -1$;
2	иначе $\underline{r} = \min \sin(\underline{x}), \sin(\overline{x})$;
3	если $\exists k \in \mathbb{Z} \mid 2k\pi + \pi/2 \in [x]$, то $\overline{r} = 1$;
4	иначе $\overline{r} = \max \sin(\underline{x}), \sin(\overline{x})$.

Рис. 2.3 иллюстрирует интервальные вычисления с функцией синуса $\sin([2{,}6.\ 7{,}2]) = [-1,\ 0{,}7937]$. Функции включения для других интервальных тригонометрических функций могут быть получены аналогично или путем выражения их через функцию синуса. Интервальные гиперболические функции строятся подобным образом.

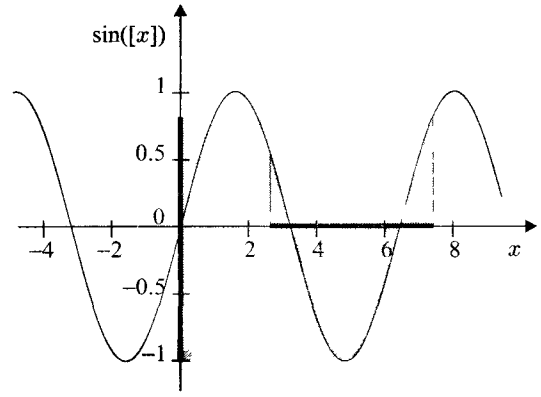


Рис. 2.3. Интервальное вычисление с функцией sin([x])

ЗАМЕЧАНИЕ 2.5. В большинстве случаев интервальные операции с функциями или оценивание интервалов их значений при пустом множестве значений аргумента должны давать пустое множество. Очевидными исключениями из этого являются интервальные объединения с оператором $\cup$ и интервальные объединения с оператором $\sqcup$, так как $[x] \cup \varnothing = [x]$ и $[x] \sqcup \varnothing = [x]$. ■

Как уже отмечалось, правила интервальных арифметических действий отличаются от аналогичных арифметических действий с вещественными числами. Например, для вещественных чисел $x^2 - x = (x - \frac{1}{2})^2 - \frac{1}{4}$, в то время как $[x]^2 - [x]$ отличается от $([x] - \frac{1}{2})^2 - \frac{1}{4}$, что иллюстрируется следующим примером.

Пример 2.5. *Для* $[x] = [-1,3]$,

$$[x]^2 - [x] = [-1,3]^2 - [-1,3] = [0,9] + [-3,1] = [-3,10], \tag{2.57}$$

$$\left([x] - \tfrac{1}{2}\right)^2 - \tfrac{1}{4} = \left[-\tfrac{3}{2}, \tfrac{5}{2}\right]^2 - \tfrac{1}{4} = \left[0, \tfrac{25}{4}\right] - \tfrac{1}{4} = \left[-\tfrac{1}{4}, 6\right]. \tag{2.58}$$

Первый результат дает плохую аппроксимацию образа множества $x^2 - x$ *для* $[-1,3]$, *в то время как второй результат совпадает с его точным образом* (см. Теорему 2.1). ■

Таким образом, достаточно ясно показано, как преобразовывать выражения с интервалами, чтобы максимально уменьшить число появлений каждой переменной; что иногда гораздо проще сказать, чем сделать.

Поскольку интервальные вычисления над замкнутыми интервалами сводятся к вычислениям над их границами, то традиционное программное обеспечение интервальных вычислений рассматривает только действия с замкнутыми интервалами (за исключением таких пакетов, как INC++ или PROLOG 4, которые также поддерживают операции с открытыми и полуоткрытыми интервалами ценой выполнения более сложных действий).

2.3.4. Интервальные векторы

Вещественный интервальный вектор $[\mathbf{x}]$ – это подмножество $\mathbb{R}^n$, которое определяется как прямое произведение n замкнутых интервалов. Когда путаница не возникает, будем называть $[\mathbf{x}]$ просто интервальным вектором, или *параллелотопом*. Будем записывать его как

$$[\mathbf{x}] = [x_1] \times [x_2] \times \cdots \times [x_n], \text{ где } [x_i] = [\underline{x}_i, \overline{x}_i] \text{ для } i = 1, \ldots, n. \quad (2.59)$$

Его i-я *интервальная компонента* $[x_i]$ есть проекция $[\mathbf{x}]$ на i-ю ось. Пустое множество из $\mathbb{R}^n$ также должно быть записано как $\varnothing \times \ldots \times \varnothing$, поскольку все его интервальные компоненты пусты. Следовательно, выражения вида

$$[\mathbf{x}] = \varnothing \times [0,\ 1] \quad (2.60)$$

запрещены, так как $[0, 1]$ не является проекцией интервального вектора $[\mathbf{x}]$ на вторую ось. Придерживаясь такого правила, мы гарантируем единообразие обозначений по всей книге. Множество всех n-мерных параллелотопов будем обозначать как $\mathbb{IR}^n$. Непустые параллелотопы являются n-мерными параллелепипедами с ребрами, ориентированными по соответствующим осям декартовой системы координат в n-мерном пространстве. Рис. 2.4 иллюстрирует случай двумерного параллелотопа.

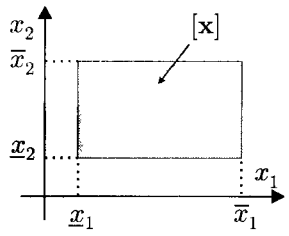


Рис. 2.4. Двумерный ($n = 2$) параллелотоп $[\mathbf{x}] = [x_1] \times [x_2]$

Многие понятия, введенные в параграфе 2.3.2 для интервалов, без труда расширяются на параллелотопы. Например, параллелотоп называется

точечным, если *все* его интервальные компоненты являются точечными. Любой параллелотоп, у которого по крайней мере одна интервальная компонента является точечной, имеет нулевую величину, поэтому параллелотоп нулевой величины может не быть точечным.

Нижняя граница $\mathrm{lb}([\mathbf{x}])$ параллелотопа $[\mathbf{x}]$ есть точечный вектор, составленный из нижних границ его интервальных компонент:

$$\underline{\mathbf{x}} = \mathrm{lb}([\mathbf{x}]) \triangleq (\mathrm{lb}([x_1]), \mathrm{lb}([x_2]), \cdots, \mathrm{lb}([x_n]))^{\mathrm{T}} = (\underline{x}_1, \underline{x}_2, \cdots, \underline{x}_n)^{\mathrm{T}}.$$

Подобным образом, *верхняя граница* параллелотопа $[\mathbf{x}]$ есть точечный вектор

$$\overline{\mathbf{x}} = \mathrm{ub}([\mathbf{x}]) \triangleq (\mathrm{ub}([x_1]), \mathrm{ub}([x_2]), \cdots, \mathrm{ub}([x_n]))^{\mathrm{T}} = (\overline{x}_1, \overline{x}_2, \cdots, \overline{x}_n)^{\mathrm{T}}.$$

Ширина параллелотопа $[\mathbf{x}] = ([x_1], [x_2], \ldots, [x_n])^{\mathrm{T}}$ есть

$$w([\mathbf{x}]) \triangleq \max_{1 \leqslant i \leqslant n} w([x_i]). \quad (2.61)$$

Если параллелотоп $[\mathbf{x}]$ является ограниченным и непустым, то его *средняя точка* (или *центр*) есть

$$\mathrm{mid}([\mathbf{x}]) \triangleq (\mathrm{mid}([x_1]), \ldots, \mathrm{mid}([x_n]))^{\mathrm{T}}. \quad (2.62)$$

Интервальная оболочка некоторого подмножества $\mathbb{A}$ из $\mathbb{R}^n$ есть наименьший параллелотоп из $\mathbb{IR}^n$, который содержит $\mathbb{A}$; он обозначается как $[\mathbb{A}]$.

Пересечение параллелотопов $[\mathbf{x}]$ и $[\mathbf{y}]$ из $\mathbb{IR}^n$ определяется как

$$[\mathbf{x}] \cap [\mathbf{y}] \triangleq ([x_1] \cap [y_1]) \times \ldots \times ([x_n] \cap [y_n]), \quad (2.63)$$

что обеспечивается непустотой пересечения $[\mathbf{x}] \cap [\mathbf{y}]$. Применяя это правило к пересечению параллелотопа $[1, 3] \times [-1, 2]$ с параллелотопом $[2, 4] \times [3, 7]$, получаем $[2, 3] \times \varnothing$, что является совершенно правильным, хотя и несовместно с тем, что интервал $[2, 3]$ должен быть проекцией результирующего параллелотопа на первую ось.

В большинстве случаев *объединение* двух параллелотопов $[\mathbf{x}]$ и $[\mathbf{y}]$ не является параллелотопом. Счастливое исключение может быть тогда, когда существует некоторый единственный индекс $i \in \{1, \ldots, n\}$, такой, что $[x_j] = [y_j]$ для всех $j \neq i$ и $[x_i] \cap [y_i] \neq \varnothing$, так как при этом $[\mathbf{x}] \cup [\mathbf{y}] = ([x_1], \ldots, [x_{i-1}], [x_i] \cup [y_i], [x_{i+1}], \ldots, [x_n])^{\mathrm{T}}$. Представляется, что этот, кажущийся неправдоподобным, случай может появляться в алгоритмах, рассматриваемых в данной книге. Во всех остальных случаях интервальная

оболочка $[[\mathbf{x}] \cup [\mathbf{y}]]$ объединения параллелотопов $[\mathbf{x}]$ и $[\mathbf{y}]$ может быть рассчитана как

$$[\mathbf{x}] \sqcup [\mathbf{y}] \triangleq ([x_1] \sqcup [y_1]) \times \ldots \times ([x_n] \sqcup [y_n]), \tag{2.64}$$

и этот процесс распространяется на любое число объединяемых параллелотопов, причем некоторые их них могут быть пустыми множествами.

Для параллелотопов отношения *вложения* и *принадлежности* записываются как

$$[\mathbf{x}] \subset [\mathbf{y}] \Leftrightarrow [x_1] \subset [y_1] \text{ и } \ldots \text{ и } [x_n] \subset [y_n], \tag{2.65}$$

и

$$\mathbf{x} \in [\mathbf{y}] \Leftrightarrow x_1 \in [y_1] \text{ и } \ldots \text{ и } x_n \in [y_n]. \tag{2.66}$$

Классические операции с интервальными векторами являются прямыми расширениями тех же операций с точечными векторами. Например, если $[\mathbf{x}]$ и $[\mathbf{y}]$ два интервальных вектора (параллелотопа) из $\mathbb{IR}^n$, и если α — некоторое вещественное число, то

$$\begin{aligned} &\alpha[\mathbf{x}] \triangleq (\alpha[x_1]) \times \cdots \times (\alpha[x_n]), \\ &[\mathbf{x}]^{\mathrm{T}} * [\mathbf{y}] \triangleq [x_1] * [y_1] + \cdots + [x_n] * [y_n], \\ &[\mathbf{x}] + [\mathbf{y}] \triangleq ([x_1] + [y_1]) \times \cdots \times ([x_n] + [y_n]). \end{aligned} \tag{2.67}$$

Эти определения согласуются с операцией над множествами, записанной в более общем виде (2.11) на стр. 29.

2.3.5. Интервальные матрицы

Пусть $\mathbb{R}^{m\times n}$ — множество всех матриц с вещественными коэффициентами из m строк и n столбцов. Интервальная матрица размерности $(m \times n)$ является подмножеством из $\mathbb{R}^{m\times n}$ и может быть определена как прямое произведение mn замкнутых интервалов. Интервальная матрица $[\mathbf{A}]$ может быть записана в любой из двух форм:

$$\begin{aligned} [\mathbf{A}] &= \begin{pmatrix} [a_{11}] & \ldots & [a_{1n}] \\ \vdots & & \vdots \\ [a_{m1}] & \ldots & [a_{mn}] \end{pmatrix} \\ &= [a_{11}] \times [a_{12}] \times \ldots \times [a_{mn}] = ([a_{ij}])_{1\leqslant i\leqslant m, 1\leqslant j\leqslant n}, \end{aligned} \tag{2.68}$$

где $[a_{ij}] = [\underline{a}_{ij}, \overline{a}_{ij}]$ является проекцией интервальной матрицы $[\mathbf{A}]$ на (i,j)-ю ось. Такая договоренность делает единственным способ представления пустой матрицы. Например, запись

$$[\mathbf{A}] = \begin{pmatrix} \varnothing & [0,1] \\ \varnothing & \varnothing \end{pmatrix} \tag{2.69}$$

запрещена, так как при представлении этой матрицы в виде прямого произведения $\varnothing \times [0,1] \times \varnothing \times \varnothing$ ее проекция на первую и вторую $(1,2)$ оси является пустой и не равна интервалу $[0,1]$.

Множество всех интервальных матриц размерности $m \times n$ обозначается как $\mathbb{IR}^{m\times n}$. Интервальная матрица называется *точечной*, если все ее элементы являются точечными. *Нижняя граница* $\mathrm{lb}([\mathbf{A}])$ интервальной матрицы $[\mathbf{A}]$ есть точечная матрица, составленная из нижних границ ее интервальных компонент:

$$\underline{\mathbf{A}} = \mathrm{lb}([\mathbf{A}]) \triangleq \begin{pmatrix} \underline{a}_{11} & \cdots & \underline{a}_{1n} \\ \vdots & & \vdots \\ \underline{a}_{m1} & \cdots & \underline{a}_{mn} \end{pmatrix}. \tag{2.70}$$

Подобным образом, *верхняя граница* $\mathrm{ub}([\mathbf{A}])$ интервальной матрицы есть точечная матрица

$$\overline{\mathbf{A}} = \mathrm{ub}([\mathbf{A}]) \triangleq \begin{pmatrix} \overline{a}_{11} & \cdots & \overline{a}_{1n} \\ \vdots & & \vdots \\ \overline{a}_{m1} & \cdots & \overline{a}_{mn} \end{pmatrix}. \tag{2.71}$$

Ширина $w([\mathbf{A}])$ интервальной матрицы определяется как

$$w([\mathbf{A}]) \triangleq \max_{1\leqslant i\leqslant m, 1\leqslant j\leqslant n} w([a_{ij}]). \tag{2.72}$$

Если $[\mathbf{A}] \in \mathbb{IR}^{m\times n}$ ограничена и непуста, то ее *средняя точка* (или *центр*) задается как

$$\mathrm{mid}([\mathbf{A}]) = (\mathrm{mid}([a_{ij}]))_{1\leqslant i\leqslant m, 1\leqslant j\leqslant n}. \tag{2.73}$$

Для $[\mathbf{A}]$ и $[\mathbf{B}]$ из $\mathbb{IR}^{m\times n}$ и $\mathbf{C}$ из $\mathbb{R}^{m\times n}$,

$$[\mathbf{A}] \subset [\mathbf{B}] \Leftrightarrow [a_{ij}] \subset [b_{ij}] \text{ для } 1 \leqslant i \leqslant m,\ 1 \leqslant j \leqslant n, \tag{2.74}$$

$$\mathbf{C} \in [\mathbf{B}] \Leftrightarrow c_{ij} \in [b_{ij}] \text{ для } 1 \leqslant i \leqslant m,\ 1 \leqslant j \leqslant n. \tag{2.75}$$

Интервальной оболочкой множества $\mathbb{A}$ матриц из $\mathbb{R}^{n\times m}$ является наименьший элемент из $\mathbb{IR}^{n\times m}$, который содержит $\mathbb{A}$.

Если $[\mathbf{A}]$ и $[\mathbf{B}]$ являются интервалами, интервальными векторами или интервальными матрицами соответствующих размерностей и если $\diamond$ является бинарным оператором, то

$$[\mathbf{A}]\diamond[\mathbf{B}] = [\{\mathbf{A}\diamond\mathbf{B} \mid \mathbf{A}\in[\mathbf{A}] \text{ и } \mathbf{B}\in[\mathbf{B}]\}]\,. \tag{2.76}$$

Например, если $[\mathbf{A}]$ и $[\mathbf{B}]$ принадлежат $\mathbb{IR}^{n\times n}$, $[\mathbf{x}]$ принадлежит $\mathbb{IR}^n$ и α принадлежит $\mathbb{R}$, то

$$\begin{aligned}
\alpha[\mathbf{A}] &= (\alpha[a_{11}])\times\cdots\times(\alpha[a_{nn}])\,,\\
[\mathbf{A}]+[\mathbf{B}] &= ([a_{ij}]+[b_{ij}])_{1\leqslant i\leqslant n,1\leqslant j\leqslant n}\,,\\
[\mathbf{A}]*[\mathbf{B}] &= \left(\sum_{k=1}^{n}[a_{ik}]*[b_{kj}]\right)_{1\leqslant i\leqslant n,1\leqslant j\leqslant n}\,,\\
[\mathbf{A}]*[\mathbf{x}] &= \left(\sum_{j=1}^{n}[a_{ij}]*[x_j]\right)_{1\leqslant i\leqslant n}\,.
\end{aligned} \tag{2.77}$$

Как и в случае с интервалами, произведение двух интервальных матриц будет обозначаться следующими двумя равносильными способами: $[\mathbf{A}]*[\mathbf{B}]$ или $[\mathbf{A}][\mathbf{B}]$. Некоторые классические свойства матриц в точечном смысле здесь уже не имеют места. Например, произведение не является ассоциативным

$$([\mathbf{A}][\mathbf{B}])[\mathbf{C}] \neq [\mathbf{A}]([\mathbf{B}][\mathbf{C}]) \tag{2.78}$$

или коммутативным по отношению к скалярам

$$[\alpha]([\mathbf{A}][\mathbf{x}]) \neq [\mathbf{A}]([\alpha][\mathbf{x}]). \tag{2.79}$$

Некоторые из этих особенностей могут быть объяснены эффектом обертывания; это иллюстрируется следующим примером, где показывается, что

$$\mathbf{A}[\mathbf{x}] \supset \{\mathbf{A}\mathbf{x} \mid \mathbf{x}\in[\mathbf{x}]\}. \tag{2.80}$$

Пример 2.6. *Пусть*

$$\mathbf{A} = \begin{pmatrix} 1 & 1\\ 0 & 1\end{pmatrix}, \quad [\mathbf{x}] = \begin{pmatrix} [-1,0]\\ [1,2]\end{pmatrix}. \tag{2.81}$$

Тогда

$$\mathbf{A}[\mathbf{x}] = \begin{pmatrix} [0,2]\\ [1,2]\end{pmatrix}, \tag{2.82}$$

откуда следует, что вектор $(0,2)^{\mathrm{T}}$ *принадлежит* $\mathbf{A}[\mathbf{x}]$, *в то время как на самом деле он не принадлежит фактическому множеству* $\mathbb{B} = \{\mathbf{A}\mathbf{x} \mid \mathbf{x}\in[\mathbf{x}]\}$, *что поясняется на* рис. 2.5. ■

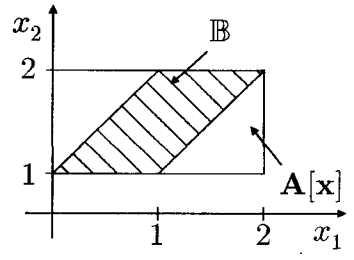


Рис. 2.5. Ухудшение аппроксимации, вносимое эффектом обертывания

Разумеется, более тонкие операции из линейной алгебры (такие как обращение матрицы, расчет собственных значений и собственных векторов) тоже приводят к определенным трудностям. Однако данные вопросы лежат вне этой вводной главы. Более детально с ними можно ознакомиться в [Neumaier, 1990].

2.4. Функции включения

2.4.1. Определения

Рассмотрим некоторую функцию $\mathbf{f}$, отображающую из $\mathbb{R}^n$ в $\mathbb{R}^m$. Интервальная функция $[\mathbf{f}]$, отображающая из $\mathbb{IR}^n$ в $\mathbb{IR}^m$, является *функцией включения* для $\mathbf{f}$, если

$$\forall[\mathbf{x}]\in\mathbb{IR}^n, \quad \mathbf{f}([\mathbf{x}])\subset[\mathbf{f}]([\mathbf{x}])\,. \tag{2.83}$$

Интервальная функция $[\mathbf{f}]([\mathbf{x}]) = \mathbb{R}^m$, для всех $[\mathbf{x}]\in\mathbb{IR}^n$, является примером некоторой (не очень полезной) функции включения для всех функций $\mathbf{f}$, отображающих из $\mathbb{R}^n$ в $\mathbb{R}^m$. Одной из целей интервального анализа является разработка для большого класса функций $\mathbf{f}$ таких функций включения, которые могут быть оценены достаточно быстро, и таких, что

параллелотоп $[\mathbf{f}]([\mathbf{x}])$ не является слишком большим. Такая функция $\mathbf{f}$ может быть, например, полиномом [Malan, 1992; Garloff, 2000] или задаваться некоторым алгоритмом [Moore, 1979]. Она может быть даже определена как решение системы дифференциальных уравнений [Lohner, 1987; Berz, 1998; Kuhn, 1998].

Чтобы проиллюстрировать понятие функции включения, рассмотрим некоторую функцию $\mathbf{f}$, отображающую из $\mathbb{R}^2$ в $\mathbb{R}^2$, от переменных x_1 и x_2, которые могут изменяться в интервалах $[x_1]$ и $[x_2]$. Множество-образ $\mathbf{f}([\mathbf{x}])$ может принимать любую форму. Образ может быть невыпуклым, (т. е. содержать такие точки из $\mathbf{f}([\mathbf{x}])$, что линия, их соединяющая, не лежит в $\mathbf{f}([\mathbf{x}])$), или даже образ может быть неодносвязным множеством (т. е. $\mathbf{f}([\mathbf{x}])$ является объединением несвязных множеств), если $\mathbf{f}$ является разрывной. Какой бы ни была форма образа $\mathbf{f}([\mathbf{x}])$, некоторая функция включения $[\mathbf{f}]$ функции $\mathbf{f}$ делает возможным вычисление прямоугольной области $[\mathbf{f}]([\mathbf{x}])$, гарантированно содержащей этот образ (рис. 2.6).

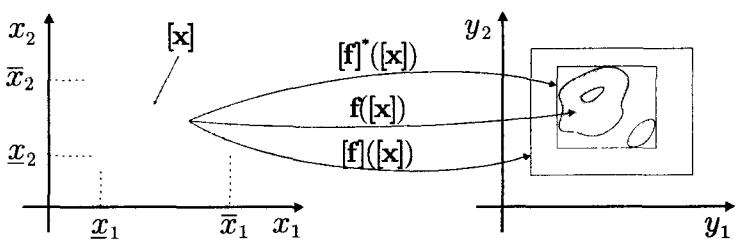


Рис. 2.6. Образы прямоугольной области по функции $\mathbf{f}$ и по двум ее функциям включения $[\mathbf{f}]$ и $[\mathbf{f}]^*$; $[\mathbf{f}]^*$ является минимальной

В действительности, как показано на рис. 2.6, функция включения $[\mathbf{f}]([\mathbf{x}])$ может давать очень плохую оценку образа $\mathbf{f}([\mathbf{x}])$. Но некоторые замечательные свойства функции $\mathbf{f}$, например такое, как положительность некоторой из ее компонент, могут сохраняться функцией включения $[\mathbf{f}]$. Если это так, то становится значительно легче работать с прямоугольными областями (параллелотопами), чем с исходными множествами; и это очень интересный момент при наблюдении за векторной функцией.

Некоторая функция включения $[\mathbf{f}]$ для функции $\mathbf{f}$ является *точной*, если для любого точечного $[\mathbf{x}] = \mathbf{x}$, интервального вектора $[\mathbf{f}](\mathbf{x}) = \mathbf{f}(\mathbf{x})$. Функция включения является *сходящейся*, если для любой последовательности параллелотопов $[\mathbf{x}](k)$

$$\lim_{k\to\infty} w([\mathbf{x}](k)) = 0 \Rightarrow \lim_{k\to\infty} w([\mathbf{f}]([\mathbf{x}](k)) = 0. \tag{2.84}$$

Это свойство иллюстрируется на рис. 2.7. Заметим, что если $[\mathbf{f}]$ является сходящейся, то необходимо она является точной. Сходимость функций включения требуется для обеспечения сходимости интервальных методов решения (процедур разрешающих операторов), таких, как представлены в Главе 5. Функция включения $[\mathbf{f}]$ является *минимальной*, если для любого $[\mathbf{x}]$ $[\mathbf{f}]([\mathbf{x}])$ есть минимальный параллелотоп, который содержит образ $\mathbf{f}([\mathbf{x}])$. Минимальная функция включения для $\mathbf{f}$ единственна и будет обозначаться как $[\mathbf{f}]^*$ (см. рис. 2.6).

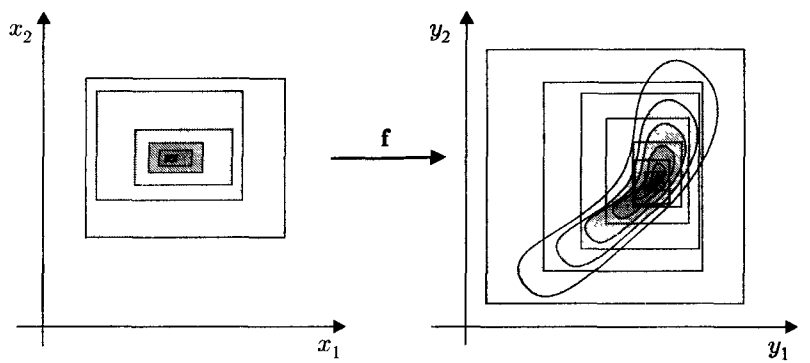


Рис. 2.7. Сходящаяся функция включения

Функция $[\mathbf{f}]$ является *монотонной по включению*, если

$$[\mathbf{x}] \subset [\mathbf{y}] \Rightarrow [\mathbf{f}]([\mathbf{x}]) \subset [\mathbf{f}]([\mathbf{y}]). \tag{2.85}$$

Тривиально проверяется, что минимальная функция включения является монотонной по включению, но необязательно сходящейся (вследствие того, что $\mathbf{f}$ может быть разрывной). Сходящаяся функция включения может не быть монотонной по включению (см. рис. 2.8).

Рассмотрим некоторую функцию $\mathbf{f}$, отображающую из $\mathbb{R}^n$ в $\mathbb{R}^m$, и пусть $[f_j]$, $j = 1, \ldots, m$, есть m функций включения, отображающих из $\mathbb{IR}^n$ в $\mathbb{IR}$, и связанных с координатными функциями f_j функции $\mathbf{f}$. При этом функция включения для $\mathbf{f}$ задается как

$$[\mathbf{f}]([\mathbf{x}]) = [f_1]([\mathbf{x}]) \times \ldots \times [f_m]([\mathbf{x}]). \tag{2.86}$$

Функция $[\mathbf{f}]$ является сходящейся (точной, минимальной, монотонной по включению, соответственно), если все ее координатные функции $[f_i]([\mathbf{x}])$

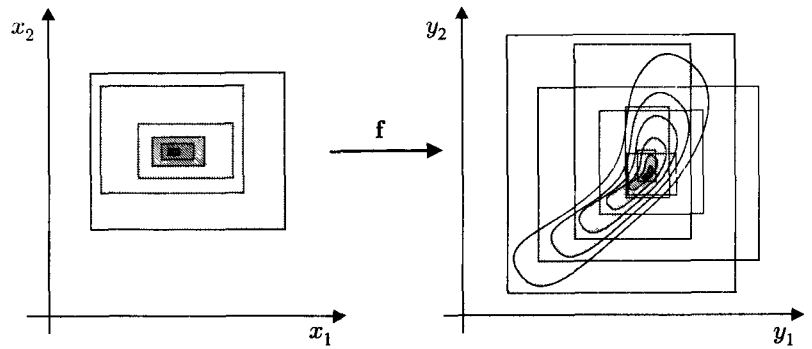


Рис. 2.8. Сходящаяся функция включения, которая не является монотонной по включению

являются сходящимися (точными, минимальными, монотонными по включению, соответственно). Построение функций включения для **f** может быть выполнено, следовательно, за счет построения функций включения для каждой из ее координатных функций. Вот почему мы сосредоточим наше внимание на получении функций включения для вещественно-значных функций.

2.4.2. Естественные функции включения

Первая идея, которая приходит в голову при построении функции включения для $f : \mathbb{R}^n \to \mathbb{R}$, состоит в том, чтобы выполнить две операции оптимизации для вычисления *infimum* и *supremum* функции f, когда каждая из переменных x_i ограничивается своим интервалом $[x_i]$. По крайней мере в принципе, таким образом можно получить наименьший интервал, содержащий $f([x_1], [x_2], \ldots, [x_n])$ и обозначаемый как $[f]^*([x_1], [x_2], \ldots, [x_n])$. Однако эти задачи оптимизации в общем случае представляются далекими от тривиальных.

Альтернативным и более удобным в работе оказывается подход, который является прямым следствием Теоремы 2.1 (см. параграф 2.2.3).

Теорема 2.2. *Рассмотрим функцию*

$$\begin{aligned} f: \ & \mathbb{R}^n \to \mathbb{R}, \\ & (x_1, \ldots, x_n) \mapsto f(x_1, \ldots, x_n), \end{aligned} \tag{2.87}$$

выражаемую как композицию операторов $+, -, *, /$ *и элементарных функций* $\sin, \cos, \exp, \mathrm{sqrt} \ldots$ *Монотонная по включению и точная функция включения* $[f] : \mathbb{IR}^n \to \mathbb{IR}$ *для* f *получается замещением каждой вещественной переменной* x_i *интервальной переменной* $[x_i]$ *и каждого оператора или функции их соответствующим интервальным аналогом. Эта функция называется естественной функцией включения для* f. *Если* f *содержит только непрерывные операторы и непрерывные элементарные функции, то* $[f]$ *является сходящейся. Если, кроме того, каждая из переменных* $(x_1, \ldots, x_n)$ *появляется не более одного раза в формальном выражении функции* f, *то* $[f]$ *является минимальной.* ■

ЗАМЕЧАНИЕ 2.6. Иначе говоря, для вычисления множества оказывается недостаточным, чтобы каждая входная переменная x_i появлялась по крайней мере однажды для того, чтобы естественная функция включения была минимальной. Вследствие эффекта обертывания требуется также, чтобы все функции и операторы, входящие в выражение для f, были непрерывны. Рассмотрим, например, непрерывную функцию $f(x) = (\mathrm{sign}(x))^2$, где $\mathrm{sign}(x)$ равно 1, если $x \geqslant 0$ и равно -1 в остальных случаях. Ее естественная функция включения $[f]$ удовлетворяет $[f]([-1, 1]) = [-1, 1]^2 = [0, 1]$. Хотя x появляется только однажды в формальном выражении для $f(x)$, $[f]$ не является минимальной, так как $f([-1, 1]) = 1$. ■

Естественные функции включения в общем случае не являются минимальными вследствие эффектов зависимости и обертывания. Точность результирующего интервала сильно зависит от выражения функции f, как это иллюстрируется следующими, тремя примерами. Первый пример представляет функцию одной переменной для возможности графической иллюстрации. Второй пример представляет функцию двух переменных. Последний пример показывает, как необходимо работать с трансцендентными функциями.

Пример 2.7. *Рассмотрим следующие четыре формальных выражения одной и той же функции* $f(x)$:

$$f_1(x) = x(x + 1), \tag{2.88}$$

$$f_2(x) = x * x + x, \tag{2.89}$$

$$f_3(x) = x^2 + x, \tag{2.90}$$

$$f_4(x) = (x + \frac{1}{2})^2 - \frac{1}{4}. \tag{2.91}$$

Оценим их естественные функции включения для $[x] = [-1, 1]$:

$$[f_1]([x]) = [x]([x] + 1) = [-2,\ 2], \tag{2.92}$$

$$[f_2]([x]) = [x] * [x] + [x] = [-2,\ 2], \tag{2.93}$$

$$[f_3]([x]) = [x]^2 + [x] = [-1,\ 2], \tag{2.94}$$

$$[f_4]([x]) = ([x] + \frac{1}{2})^2 - \frac{1}{4} = [-\frac{1}{4},\ 2]. \tag{2.95}$$

Таким образом, точность интервального результата зависит от формального выражения функции f (см. рис. 2.9).

Поскольку $[x]$ появляется только один раз в f_4 и f_4 является непрерывной, то $[f_4]$ является минимальной. Таким образом, $[f_4]([x]) = f([x]) = [-\frac{1}{4},\ 2]$. ■

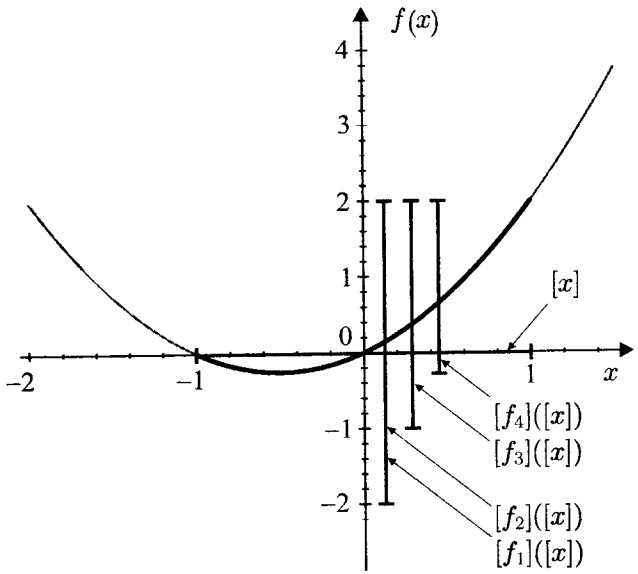


Рис. 2.9. Четыре естественных функции включения для одной и той же исходной функции

Пример 2.8. *Рассмотрим вещественную функцию $f : \mathbb{R}^2 \to \mathbb{R}$ определяемую выражением*

$$f(x_1, x_2) = \frac{x_1 - x_2}{x_1 + x_2}, \quad \text{при } x_1 \in [-1,\ 2] \text{ и } x_2 \in [3,\ 5]. \tag{2.96}$$

Естественная функция включения $[f]_1$ для f получается заменой каждой вещественной переменной ее интервальной переменной, и каждой вещественной операции ее интервальным аналогом:

$$[f]_1([x_1], [x_2]) = \frac{[x_1] - [x_2]}{[x_1] + [x_2]}, \tag{2.97}$$

поэтому

$$\begin{aligned}[f]_1([-1,\ 2], [3,\ 5]) &= \frac{[-1,\ 2] - [3,\ 5]}{[-1,\ 2] + [3,\ 5]} = \frac{[-6, -1]}{[2,\ 7]} \\ &= [-6, -1] * [\tfrac{1}{7},\ \tfrac{1}{2}] = [-3, -\tfrac{1}{7}].\end{aligned} \tag{2.98}$$

Второе интервальное расширение $[f]_2$ может быть подобным же образом получено после перезаписи функции f таким образом, чтобы каждая из переменных x_1 и x_2 появлялась только один раз:

$$[f]_2([x_1], [x_2]) = 1 - \frac{2}{1 + [x_1]/[x_2]}. \tag{2.99}$$

Тогда

$$\begin{aligned}[f]_2([-1,\ 2], [3,\ 5]) &= 1 - \frac{2}{1 + [-1,\ 2]/[3,\ 5]} = 1 - \frac{2}{1 + [-1/3, 2/3]} \\ &= 1 - \frac{2}{[2/3, 5/3]} = 1 - [6/5,\ 3] \\ &= [-2, -1/5].\end{aligned} \tag{2.100}$$

Функции включения $[f]_1$ и $[f]_2$ обе являются интервальными расширениями f. Функция $[f]_2$ является более точной, чем $[f]_1$, которая оказывается более плохой из-за эффекта зависимости. Интервал, рассчитанный по $[f]_2$, является минимальным и, таким образом, равен множеству значений образа $f([-1, 2], [3, 5])$. ■

Пример 2.9. *Рассмотрим вещественную функцию f, определенную выражением*

$$f(x_1, x_2) = \ln(e^{x_1} + \sin(x_2)), \tag{2.101}$$

при $x_1 \in [0, 1]$ и $x_2 \in [\pi/4, 4\pi/3]$. Ее естественная функция включения имеет вид:

$$[f]([x_1], [x_2]) = \ln(\exp([x_1]) + \sin([x_2])), \tag{2.102}$$

поэтому

$$\begin{aligned}[f]([0, 1], [\pi/4, 4\pi/3]) &= \ln(\exp([0, 1]) + \sin([\pi/4, 4\pi/3])) \\ &= \ln([1,\ e] + [-\sqrt{3}/2,\ 1]) \\ &= \ln([1 - \sqrt{3}/2, e + 1]) \\ &= [\ln(1 - \sqrt{3}/2), \ln(e + 1)] \\ &\subset [-2{,}0101,\ 1{,}3133].\end{aligned} \tag{2.103}$$

Поскольку каждая переменная появляется только один раз и поскольку все функции и операторы, включенные в формальное описание функции f являются непрерывными, предпоследний интервал является точным множеством значений образа $f([0,1],[\pi/4,4\pi/3])$. Заметим, что минимальное и максимальное значения функции f вычислены без выполнения операций оптимизации, хотя f не является монотонной. Окончательный интервал является гарантированной численной оценкой предыдущего интервала, полученной округлением с избытком (см. Главу 10). ■

Однако использование естественных функций включения не всегда может быть рекомендовано. Их эффективность сильно зависит от количества появлений каждой переменной, которое часто бывает трудно уменьшить. Таким образом, важной областью исследований в интервальном анализе является поиск других типов функций включения, которые бы давали не очень плохие результаты [Ratschek, 1984], как это показано в параграфах 2.4.3–2.4.5.

2.4.3. Центрированные функции включения

Пусть $f : \mathbb{R}^n \to \mathbb{R}$ — некоторая скалярная функция от вектора $\mathbf{x} = (x_1, \ldots, x_n)^{\mathrm{T}}$. Положим, что функция f дифференцируема на параллелотопе $[\mathbf{x}]$, и обозначим $\mathrm{mid}([\mathbf{x}])$ как $\mathbf{m}$. При этом, теорема о среднем дает

$$\forall \mathbf{x} \in [\mathbf{x}],\ \exists \mathbf{z} \in [\mathbf{x}] \mid f(\mathbf{x}) = f(\mathbf{m}) + \mathbf{g}^{\mathrm{T}}(\mathbf{z})(\mathbf{x} - \mathbf{m}), \tag{2.104}$$

где $\mathbf{g}$ — *градиент* функции f, т. е. вектор-столбец с элементами $g_i = \partial f / \partial x_i$, $i = 1, \ldots, n$. Таким образом,

$$\forall \mathbf{x} \in [\mathbf{x}],\ f(\mathbf{x}) \in f(\mathbf{m}) + [\mathbf{g}^{\mathrm{T}}]([\mathbf{x}])(\mathbf{x} - \mathbf{m}), \tag{2.105}$$

где $[\mathbf{g}^{\mathrm{T}}]$ является функцией включения для $\mathbf{g}^{\mathrm{T}}$, поэтому

$$f([\mathbf{x}]) \subseteq f(\mathbf{m}) + [\mathbf{g}^{\mathrm{T}}]([\mathbf{x}])([\mathbf{x}] - \mathbf{m}). \tag{2.106}$$

Таким образом, интервальная функция

$$[f_{\mathrm{c}}]([\mathbf{x}]) \triangleq f(\mathbf{m}) + [\mathbf{g}^{\mathrm{T}}]([\mathbf{x}])([\mathbf{x}] - \mathbf{m}) \tag{2.107}$$

является функцией включения для $\mathbf{f}$. Функцию (2.107) мы будем называть *центрированной функцией включения*. Чтобы проиллюстрировать интересные свойства такой функции в одномерном случае, рассмотрим функцию $[f_{\mathrm{c}}](x)$, отображающую из $\mathbb{R}$ в $\mathbb{IR}$ и определенную соотношением

$$[f_{\mathrm{c}}](x) \triangleq f(m) + [f'](x - m) \tag{2.108}$$

для любого заданного $[x]$. Эта функция может рассматриваться как афинное преобразование над x при неопределенном коэффициенте наклона, принадлежащем $[f'](x])$. Таким образом, график $[f_{\mathrm{c}}]\,(x)$ может быть представлен некоторым конусом с центром $(m, f(m))$, что показано на рис. 2.10 для уменьшающейся длины интервала $[x]$. Чем меньше ширина $w([x])$, тем лучше конус аппроксимирует функцию. Этот рисунок иллюстрирует тот факт, что

$$\frac{w([f_c]([x]))}{w(f([x]))} \to 1, \tag{2.109}$$

когда ширина интервала $[x]$ сходится к 0, что в общем случае не имеет места при использовании естественной функции включения.

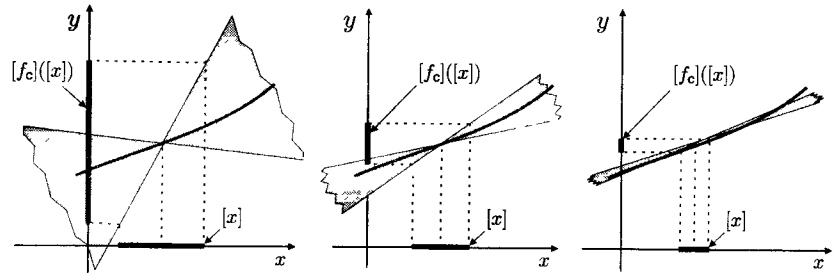


Рис. 2.10. Интерпретация центрированной функции включения

Когда ширина интервала $[\mathbf{x}]$ мала, эффект ухудшения оценки, являющийся возможным результатом интервальной оценки $[\mathbf{g}]([\mathbf{x}])$, уменьшается скалярным произведением на $[\mathbf{x}] - \mathbf{m}$, который является малым интервалом, центрированным относительно нуля.

2.4.4. Смешанные центрированные функции включения

Центрированная функция включения для функции f, отображающей из $\mathbb{R}^n$ в $\mathbb{R}$, может быть значительно улучшена за счет некоторого усложнения ее формулировки [Hansen, 1968]. Напомним, что для функции φ, отображающей из $\mathbb{R}$ в $\mathbb{R}$,

$$\varphi(x) \in \varphi(m) + \varphi'([x])([x] - m), \tag{2.110}$$

где $m = \mathrm{mid}([x])$. Главная идея получения смешанной центрированной функции включения состоит в том, чтобы применить (2.110) n раз, рас-

сматривая поочередно каждую из переменных в функции f. Чтобы упростить пояснения, рассмотрим сперва случай $n = 3$. Рассмотрим функцию $f(x_1, x_2, x_3)$ как функцию только переменной x_3 и положим $m_3 = \text{mid}([x_3])$; тогда (2.110) влечет

$$f(x_1, x_2, x_3) \in f(x_1, x_2, m_3) + g_3(x_1, x_2, [x_3]) * ([x_3] - m_3). \quad (2.111)$$

Рассмотрим теперь $f(x_1, x_2, m_3)$ как функцию только переменной x_2 и положим $m_2 = \text{mid}([x_2])$; тогда (2.110) дает

$$f(x_1, x_2, m_3) \in f(x_1, m_2, m_3) + g_2(x_1, [x_2], m_3) * ([x_2] - m_2). \quad (2.112)$$

Наконец, рассмотрим $f(x_1, m_2, m_3)$ как функцию переменной x_1 и положим $m_1 = \text{mid}([x_1])$; тогда (2.110) приводит к выражению

$$f(x_1, m_2, m_3) \in f(m_1, m_2, m_3) + g_1([x_1], m_2, m_3) * ([x_1] - m_1). \quad (2.113)$$

Комбинируя эти три уравнения, получаем

$$\begin{aligned} f(x_1, x_2, x_3) \in\ & f(m_1, m_2, m_3) + g_1([x_1], m_2, m_3) * ([x_1] - m_1) \\ & + g_2(x_1, [x_2], m_3) * ([x_2] - m_2) \\ & + g_3(x_1, x_2, [x_3]) * ([x_3] - m_3). \end{aligned} \quad (2.114)$$

Таким образом,

$$\begin{aligned} f([x_1], [x_2], [x_3]) \subset\ & f(m_1, m_2, m_3) + g_1([x_1], m_2, m_3) * ([x_1] - m_1) \\ & + g_2([x_1], [x_2], m_3) * ([x_2] - m_2) \\ & + g_3([x_1], [x_2], [x_3]) * ([x_3] - m_3). \end{aligned} \quad (2.115)$$

Это выражение можно обобщить на функцию f от n переменных. При $\mathbf{x} = (x_1, \ldots, x_n)^{\mathrm{T}}$ и $\mathbf{m} = \text{mid}([\mathbf{x}])$, получаем

$$f([\mathbf{x}]) \subset f(\mathbf{m}) + \sum_{i=1}^{n} g_i([x_1], \ldots, [x_i], m_{i+1}, \ldots, m_n) * ([x_i] - m_i), \quad (2.116)$$

и правая часть (2.116) определяет *смешанную центрированную функцию включения*. Ее основное отличие от (2.107) состоит в аргументах градиента. В (2.116) используются как интервальные, так и точечные аргументы, что позволяет снизить неточность оценивания, именно:

$$[\mathbf{g}](\text{mid}([\mathbf{x}]), [\mathbf{x}]) \subset [\mathbf{g}]([\mathbf{x}]). \quad (2.117)$$

2.4.5. Тейлоровские функции включения

Работая с идеей, которая привела к центрированной функции включения, можно придти к мысли применить разложение в ряд Тейлора, чтобы с большим числом членов разложения аппроксимировать функцию f, отображающую из $\mathbb{R}^n$ в $\mathbb{R}$. Это приводит к *тейлоровской функции включения*. В качестве иллюстрации рассмотрим разложение второго порядка:

$$[f]_{\mathrm{T}}([\mathbf{x}]) = f(\mathbf{m}) + \mathbf{g}^{\mathrm{T}}(\mathbf{m})([\mathbf{x}] - \mathbf{m}) + \frac{1}{2}([\mathbf{x}] - \mathbf{m})^{\mathrm{T}}[\mathbf{H}]([\mathbf{x}])([\mathbf{x}] - \mathbf{m}), \quad (2.118)$$

где $\mathbf{m} = \text{mid}([\mathbf{x}])$, $\mathbf{g}$ опять является градиентом функции f и $[\mathbf{H}]([\mathbf{x}])$ является интервальной *матрицей Гессе*. Элемент $[\mathbf{H}]_{ij}$ матрицы $[\mathbf{H}]$ является функцией включения

$$h_{ij} = \begin{cases} \partial^2 f/\partial x_i^2, & \text{если } j = i \quad (i = 1, \ldots, n), \\ 2\partial^2 f/\partial x_i x_j, & \text{если } j < i \quad (i = 1, \ldots, n), \\ 0, & \text{в остальных случаях.} \end{cases} \quad (2.119)$$

Симметричность матрицы Гессе ($h_{ij} = \partial^2 f/\partial x_i x_j$ для всех i и j) также может быть использована, но связанное с этим увеличение числа интервальных компонент в $[\mathbf{H}]([\mathbf{x}])$ может привести к ухудшению оценки $[f]_{\mathrm{T}}$. Это ухудшение может быть уменьшено заменой $[\mathbf{H}]([\mathbf{x}])$ на смешанное выражение $[\mathbf{H}](\text{mid}([\mathbf{x}]), [\mathbf{x}])$, как это было сделано для градиента в смешанной центрированной форме в параграфе 2.4.4.

Когда функция f зависит только от одной переменной, тейлоровская функция включения n-ного порядка задается выражением

$$\begin{aligned} [f]^{\mathrm{T}}([x]) = & f(m) + f'(m)([x] - m) + \cdots + f^{n-1}(m)\frac{([x] - m)^{n-1}}{(n-1)!} \\ & + [f^n]([x])\frac{([x] - m)^n}{n!}. \end{aligned} \quad (2.120)$$

Таким образом, построение тейлоровской функции включения порядка n требует расчета производных функции f до n-й включительно, что может привести к громоздким вычислениям.

2.4.6. Сравнение

При слабых технических ограничениях [Moore, 1979] естественная, центрированная и тейлоровская функции включения являются сходящими-

ся. Грубо говоря, *скорость сходимости* сходящейся функции включения есть наибольшая величина α такая, что

$$\exists \beta \mid w([f]([\mathbf{x}])) - w(f([\mathbf{x}])) \leqslant \beta w([\mathbf{x}])^{\alpha}, \tag{2.121}$$

когда $w([\mathbf{x}])$ сходится к 0.

Если функция включения минимальна, то ее скорость сходимости бесконечна. Скорость сходимости естественной функции включения по крайней мере линейна ($\alpha \geqslant 1$), в то время как скорость сходимости центрированной функции включения по крайней мере квадратична ($\alpha \geqslant 2$) . Скорость сходимости тейлоровской функции включения также по крайней мере квадратична для любого порядка $n \geqslant 2$. Квадратичная скорость сходимости, разумеется, выглядит более привлекательной, чем линейная, но необходимо помнить, что это означает лишь то, что более точные результаты могут быть получены только в случае бесконечно малых параллелотопов. Ничего подобного нельзя сказать о поведении функций включения при параллелотопах реальных размеров. Когда параллелотоп, с которым работаем, является бо́льшим, естественная функция включения в общем случае более предпочтительна, чем центрированная функция включения, в то время как последняя работает лучше, если параллелотоп мал, а смешанная центрированная функция включения дает лучший результат по сравнению с естественной функцией включения.

Не удается указать подхода к построению функции включения, который был бы равномерно наилучшим, и необходимо часто прибегать к компромиссу между сложностью и эффективностью вычислений. Можно также использовать сразу несколько функций включения и брать пересечение их множеств образов, чтобы получить более лучшую аппроксимацию множества образа исходной функции.

Пример 2.10. *Рассмотрим функцию f, определенную выражением*

$$f(x) = x^2 + \sin(x), \tag{2.122}$$

и интервалы

$$[x] = [\frac{2\pi}{3}, \frac{4\pi}{3}] \quad \text{и} \quad [y] = [\frac{99\pi}{100}, \frac{101\pi}{100}]. \tag{2.123}$$

Сравним аппроксимации $f([x])$ и $f([y])$, получаемые при использовании естественной функции включения, центрированной функции включения, тейлоровской функции включения второго порядка и минимальной функции включения, обозначая, соответственно, эти аппроксимации как $[f]_{\mathrm{n}}$, $[f]_{\mathrm{c}}$, $[f]_{\mathrm{T}}$ и $[f]^*$ (см. также Упражнение 11.10, стр. 391). *Первые три из этих функций задаются как*

$$[f]_n([x]) = [x]^2 + \sin([x]), \tag{2.124}$$

$$[f]_c([x]) = f(\pi) + ([x] - \pi)[f']([x]), \tag{2.125}$$

$$[f]_T([x]) = f(\pi) + ([x] - \pi)f'(\pi) + \frac{([x] - \pi)^2}{2}[f'']([x]), \tag{2.126}$$

при

$$f'(x) = 2x + \cos(x) \quad \text{и} \quad f''(x) = 2 - \sin(x). \tag{2.127}$$

Минимальная функция включения указывается тривиально с учетом того, что f является возрастающей на $[x]$ (и, следовательно, на $[y] \subset [x]$). Поэтому

$$[f]^*([x]) = [\underline{x}^2 + \sin(\underline{x}),\ \overline{x}^2 + \sin(\overline{x})]. \tag{2.128}$$

Результаты, полученные применением каждой из указанных функций на интервалах $[x]$ и $[y]$, приведены в табл. 2.2, *где $\Delta([f]([x]))$ обозначает величину $w([f]([x])) - w(f([x]))$. Численные значения приведены с точностью 10^{-5}. Из таблицы следует, что естественная функция включения обеспечивает приемлемый результат при бо́льшем интервале $[x]$, что, кроме ее простоты, дает дополнительный повод применять эту функцию. Центрированная и тейлоровская функции включения оказываются более эффективны, чем естественная функция включения, для меньшего интервала $[y]$. Тейлоровская функция включения дает заметное улучшение по сравнению с естественной и центрированной функциями даже для большего интервала. Наконец, этот пример напоминает, что оказывается полезным проверить, является ли рассматриваемая функция монотонной, так как, если это имеет место, получение минимальной функции включения оказывается тривиальным. К сожалению, зависимости, встречающиеся в инженерных приложениях, редко оказываются согласованными, как в рассматриваемом примере, что снижает практическую ценность данного замечания.* ■

Пример 2.11. *Рассмотрим теперь векторную функцию* **f**, *отображающую из $\mathbb{R}^2$ в $\mathbb{R}^2$, задаваемую выражениями*

$$\begin{aligned} f_1(x_1, x_2) &= x_1^2 + x_1 \exp(x_2) - x_2^2, \\ f_2(x_1, x_2) &= x_1^2 - x_1 \exp(x_2) + x_2^2, \end{aligned} \tag{2.129}$$

Таблица 2.2. Сравнение функций включения

	$[x] = \left[\frac{2\pi}{3}, \frac{4\pi}{3}\right]$		$[y] = \left[\frac{99\pi}{100}, \frac{101\pi}{100}\right]$	
$[f]$	$[f]([x])$	$\Delta([f]([x]))$	$[f]([y])$	$\Delta([f]([y]))$
$[f]_{\mathrm{n}}$	[3,52046, 18,41199]	3,46410	[9,64178, 10,09940]	0,12564
$[f]_{\mathrm{c}}$	[1,62022, 18,11899]	5,07134	[9,70163, 10,03758]	0,00397
$[f]_{\mathrm{T}}$	[4,33706, 16,97362]	1,20913	[9,70362. 10,03659]	0,00099
$[f]^*$	[5,25251, 16.67994]	0	[9,70461, 10,03658]	0

где x_1 и x_2 принадлежат $[x_1]$ и $[x_2]$, соответственно. Естественная функция включения $[\mathbf{f}]_{\mathrm{n}}$ для функции $\mathbf{f}$ описывается как

$$[\mathbf{f}]_{\mathrm{n},1}([\mathbf{x}]) = [x_1]^2 + [x_1]\exp([x_2]) - [x_2]^2, \tag{2.130}$$

$$[\mathbf{f}]_{\mathrm{n},2}([\mathbf{x}]) = [x_1]^2 - [x_1]\exp([x_2]) + [x_2]^2. \tag{2.131}$$

Центрированная функция включения $[\mathbf{f}]_{\mathrm{c}}$ задается как

$$[\mathbf{f}]_{\mathrm{c}}([\mathbf{x}]) = \mathbf{f}(\mathrm{mid}([\mathbf{x}])) + [\mathbf{J_f}]([\mathbf{x}]) * ([\mathbf{x}] - \mathrm{mid}([\mathbf{x}])). \tag{2.132}$$

Все аргументы интервальной матрицы Якоби $[\mathbf{J_f}]$ являются интервалами, и i-я строка $[\mathbf{J_f}]([\mathbf{x}])$ задается как $[\mathbf{g}_i^{\mathrm{T}}]([\mathbf{x}])$, при $\mathbf{g}_i$ градиенте i-й компоненты $\mathbf{f}$ (см. (2.107), стр. 54):

$$[\mathbf{J_f}]([\mathbf{x}]) = \begin{pmatrix} 2[x_1] + \exp([x_2]) & -2[x_2] + [x_1]\exp([x_2]) \\ 2[x_1] - \exp([x_2]) & 2[x_2] - [x_1]\exp([x_2]) \end{pmatrix}. \tag{2.133}$$

Смешанная центрированная функция включения $[\mathbf{f}]_{\mathrm{m}}$ задается как

$$[\mathbf{f}]_{\mathrm{m}}([\mathbf{x}]) = \mathbf{f}(\mathrm{mid}([\mathbf{x}])) + [\mathbf{J_f}](\mathrm{mid}([\mathbf{x}]), [\mathbf{x}]) * ([\mathbf{x}] - \mathrm{mid}([\mathbf{x}])), \tag{2.134}$$

где $[\mathbf{J_f}]$ теперь зависит смешанным образом от точечных и интервальных аргументов:

$$[\mathbf{J_f}](\mathrm{mid}([\mathbf{x}]), [\mathbf{x}]) = \begin{pmatrix} 2[x_1] + \exp(\mathrm{mid}([x_2])) & -2[x_2] + [x_1]\exp([x_2]) \\ 2[x_1] - \exp(\mathrm{mid}([x_2])) & 2[x_2] - [x_1]\exp([x_2]) \end{pmatrix}. \tag{2.135}$$

Как и ожидалось, (2.135) *содержит меньше интервальных аргументов, чем* (2.133). *В* табл. 2.3 *сравнивается эффективность естественной, центрированной и смешанной центрированной функций включения на двух параллелотопах* $[\mathbf{x}]$ *и* $[\mathbf{y}]$. *Величина* Δ *имеет тот же смысл, что и в* Примере 2.10.

Выводы здесь подобны выводам для скалярного случая: для большего параллелотопа $[\mathbf{x}]$ естественная функция включения более предпочтительна, в то время как центрированная функция включения работает лучше на меньшем параллелотопе $[\mathbf{y}]$, а смешанная центрированная функция дает лучший результат, чем просто центрированная. ■

Таблица 2.3. Сравнение векторных функций включения

	$[x_1] = [0.5, 1.5]; [x_2] = [1.5, 2.5]$		$[y_1] = [0.9, 1.1]; [y_2] = [1.9, 2.1]$	
$[\mathbf{f}]$	$[\mathbf{f}]([\mathbf{x}])$	$\Delta([\mathbf{f}]([\mathbf{x}]))$	$[\mathbf{f}]([\mathbf{y}])$	$\Delta([\mathbf{f}]([\mathbf{y}]))$
$[f_1]_{\mathrm{n}}$	[−3,75916, 18,2737]	7,67904	[2,41730, 6,58279]	1,60000
$[f_2]_{\mathrm{n}}$	[−15,7737, 6,25916]	11,67904	[−4,56279, −0,39730]	2,40001
$[f_1]_{\mathrm{c}}$	[−10,8391, 19,6172]	16,10248	[2,83416, 5,94395]	0,54430
$[f_2]_{\mathrm{c}}$	[−15,6172, 10,8391]	16,10248	[−3,54395, −1,23416]	0,54431
$[f_1]_{\mathrm{m}}$	[−8,44234, 17,2205]	11,30902	[2,91187, 5,86624]	0,38888
$[f_2]_{\mathrm{m}}$	[−13,2205, 8,44234]	11,30902	[−3,46624, −1,31187]	0,38889
$[f_1]^*$	[−0,08008, 14,27374]	0	[3,21730, 5,78279]	0
$[f_2]^*$	[−9,77374, 0,58008]	0	[−3,36279, −1,59731]	0

2.5. Проверки включения

Проверки включения могут использоваться для доказательства того, что все точки из заданного параллелотопа удовлетворяют заданному свойству, или для доказательства того, что ни одна из них ему не удовлетворяет. Эти проверки включают интервальные булевские переменные, которые и представляются прежде всего.

2.5.1. Интервальные булевские переменные

Операции над множеством, определенные в параграфе 2.2, могут быть использованы для получения множества значений булевских переменных

$$\mathbb{B} \triangleq \{\mathrm{false}, \mathrm{true}\}, \tag{2.136}$$

но нет необходимости использовать оболочки для аппроксимации сверху множеств булевских переменных, чтобы с ними работать, поскольку $\mathbb{B}$ является конечным. Следовательно, эффект обертывания здесь будет отсутствовать, но эффект зависимости еще будет присутствовать. *Булевское*

число является элементом множества $\mathbb{B}$. По расширению[1] *интервальная булевская переменная* есть некоторое подмножество множества $\mathbb{B}$. Таким образом, множество всех интервальных булевских переменных есть

$$\mathbb{IB} = \{\varnothing, 0, 1, [0,1]\}, \tag{2.137}$$

где $\varnothing$ соответствует *невозможному,* 0 для *ложного*, 1 для *истинного* и $[0,1]$ для *неопределенного* случая. Операции над булевскими интервальными переменными просто определяются в рамках операций над множествами:

$$\begin{aligned} [a] \vee [b] &= \{a \vee b \mid a \in [a], b \in [b]\}, \\ [a] \wedge [b] &= \{a \wedge b \mid a \in [a], b \in [b]\}, \\ \neg [a] &= \{\neg a \mid a \in [a]\}, \\ [a] \cap [b] &= \{\max(\underline{a}, \underline{b}),\ \min(\overline{a}, \overline{b})\}, \\ [a] \cup [b] &= \{\min(\underline{a}, \underline{b}),\ \max(\overline{a}, \overline{b})\}, \end{aligned} \tag{2.138}$$

где $\wedge$ и $\vee$, соответственно, обозначают логические операторы И и ИЛИ, и где $\neg$ есть оператор дополнения,такой, что $\neg 0 = 1$ и $\neg 1 = 0$. Например,

$$([0,1] \vee 1) \wedge ([0,1] \wedge 1) = 1 \wedge [0,1] = [0,1]. \tag{2.139}$$

Если $[a] \in \mathbb{IB}$, то

$$0 \wedge [a] = 0;\ 1 \wedge [a] = [a];\ [a] \wedge [a] = [a]; \tag{2.140}$$

$$0 \vee [a] = [a];\ 1 \vee [a] = 1;\ [a] \vee [a] = [a]. \tag{2.141}$$

Эффект зависимости все еще присутствует, когда применяется оператор дополнения. Например,

$$[a] \subset ([a] \wedge [b]) \vee ([a] \wedge \neg [b]), \tag{2.142}$$

и величины обеих частей в (2.142) различны при $a = 0$ (или 1) и $[b] = [0,\ 1]$, в то время как $a = (a \wedge b) \vee (a \wedge \neg b)$.

Произвольная функция β, отображающая из $\mathbb{B}^n$ в $\mathbb{B}$, называется *булевской функцией*. Понятие функции включения, введенное для вещественных функций, сразу расширяется на булевские функции. $[\beta] : \mathbb{IB}^n \to \mathbb{IB}$ является *функцией включения* для β, если

$$\forall ([b_1], \ldots, [b_n]) \in \mathbb{IB}^n,\ \beta([b_1], \ldots, [b_n]) \subset [\beta]([b_1], \ldots, [b_n]). \tag{2.143}$$

[1] Для любого множества с частичным упорядочением $(\mathbb{S}, \leqslant)$ можно всегда определить множество $\mathbb{IS}$ пар $[a, b]$ таких, что $a \in \mathbb{S}$, $b \in \mathbb{S}$ и $a \leqslant b$. Элементы множества $\mathbb{IS}$ будем называть интервалами. $\mathbb{S}$ может, например, представлять собой $\mathbb{R}^n$, множество всех булевых чисел $\mathbb{B}$ или множество всех компактных множеств. В последнем случае частичное упорядочение представляется как включение $\subset$.

Естественная функция включения $[\beta]$ функции β получается заменой всех аргументов и операторов в функции β их интервальными аналогами. $[\beta]([b_1], \ldots, [b_n])$ является *минимальной*, если

$$\forall ([b_1], \ldots, [b_n]) \in \mathbb{IB}^n,\ \beta([b_1], \ldots, [b_n]) = [\beta]([b_1], \ldots, [b_n]). \tag{2.144}$$

Как и в случае вещественных функций, всегда, когда функция $\beta(b_1, \ldots, b_n)$ является неубывающей относительно всех своих аргументов, минимальная функция включения задается как

$$[\beta]^*([b_1], \ldots, [b_n]) = [\beta(\underline{b}_1, \ldots, \underline{b}_n),\ \beta(\overline{b}_1, \ldots, \overline{b}_n)]. \tag{2.145}$$

Заметим, что эта ситуация встречается часто. Это имеет место, например, когда оператор дополнения не используется, т. е. когда β является полиномом. Тем не менее есть много булевских функций, которые не являются монотонными, такие как *исключенное* ИЛИ

$$\beta(b_1, b_2) = (b_1 \wedge \neg b_2) \vee (\neg b_1 \wedge b_2). \tag{2.146}$$

Но даже тогда, когда булевская функция немонотонна, всегда есть возможность, по крайней мере в принципе, построить для нее минимальную функцию включения, так как каждая из интервальных булевских переменных может принимать не более четырех значений. Рассмотрим, например, $\beta([b_1], [b_2])$, при $\beta(b_1, b_2) = (b_1 \wedge b_2) \vee (b_1 \wedge \neg b_2)$, для $[b_1] = 1$ и $[b_2] = [0,\ 1]$. Естественное интервальное расширение функции β дает $[\beta](1,\ [0,\ 1]) = (1 \wedge [0,\ 1]) \vee (1 \wedge \neg [0,\ 1]) = [0,\ 1]$. Минимальная оценка описывается выражением $\beta(1, [0,1]) = \beta(1,0) \cup \beta(1,1) = 1$. Разумеется, этот подход ведет к взрывному увеличению числа комбинаций, когда число переменных увеличивается. Этого иногда удается избежать, манипулируя булевскими выражениями, например, с помощью таблиц Карно или используя преимущества широко известных правил упрощения числа появлений булевских переменных.

2.5.2. Проверки

Проверка — это некоторая функция t, отображающая из $\mathbb{R}^n$ в $\mathbb{B}$. *Проверка включения* для функции t есть функция $[t]$, отображающая из $\mathbb{IR}^n$ в $\mathbb{IB}$, такая, что для любого $[\mathbf{x}] \in \mathbb{IR}^n$

$$\begin{aligned} ([t]([\mathbf{x}]) = 1) &\Rightarrow (\forall \mathbf{x} \in [\mathbf{x}],\ t(\mathbf{x}) = 1), \\ ([t]([\mathbf{x}]) = 0) &\Rightarrow (\forall \mathbf{x} \in [\mathbf{x}],\ t(\mathbf{x}) = 0). \end{aligned} \tag{2.147}$$

Проверка включения $[t]$ является *точной*, если $[t](\mathbf{x}) = t(\mathbf{x})$ для любого $\mathbf{x} \in \mathbb{R}^n$. Эта проверка является *минимальной*, если

$$\forall [\mathbf{x}] \in \mathbb{IR}^n,\ [t]([\mathbf{x}]) = \{t(\mathbf{x}) \mid \mathbf{x} \in [\mathbf{x}]\}. \tag{2.148}$$

Минимальная проверка необходимо является точной.

Пример 2.12. *Рассмотрим проверку*

$$t: \begin{array}{ccc} \mathbb{R}^2 & \to & \{0,1\} \\ (x_1,x_2)^{\mathsf{T}} & \mapsto & (x_1+x_2 \leqslant 5), \end{array} \tag{2.149}$$

что означает, что

$$t(\mathbf{x}) = \begin{cases} 1, & \text{если } x_1+x_2 \leqslant 5, \\ 0, & \text{если } x_1+x_2 > 5. \end{cases} \tag{2.150}$$

Минимальная проверка включения $[t]$, связанная с функцией t, задается как

$$[t]([\mathbf{x}]) = \begin{cases} 1, & \text{если } \overline{x}_1+\overline{x}_2 \leqslant 5, \\ 0, & \text{если } \underline{x}_1+\underline{x}_2 > 5, \\ [0,\,1], & \text{в остальных случаях,} \end{cases} \tag{2.151}$$

выражение можно переписать в более понятной форме как

$$[t]([\mathbf{x}]) \Leftrightarrow ([x_1]+[x_2] \leqslant 5). \tag{2.152}$$

Проверка является минимальной и точной. ■

Любой булевский оператор над вещественными числами, такой как $(\leqslant, \geqslant, <, >$, integer, odd, even, prime$\ldots)$, может быть подобным же образом расширен на интервалы. Например,

$$([a,b] \leqslant [c,d]) = \begin{cases} 1, & \text{если } b \leqslant c, \\ 0, & \text{если } a > d, \\ [0,1], & \text{если ни } b \leqslant c \text{ ни } a > d. \end{cases} \tag{2.153}$$

Булевский оператор сравнения $=$ не может быть расширен подобным образом, так как по теории множеств ему уже задан двузначный смысл:

$$([a,b] = [c,d]) = \begin{cases} 1, & \text{если } a = c \text{ и } b = d, \\ 0, & \text{в остальных случаях.} \end{cases} \tag{2.154}$$

С помощью интервального анализа и понятия функции включения легко построить некоторую проверку включения для любой проверки, которая может быть представлена в форме:

$$t(\mathbf{x}) = \beta(t_1(\mathbf{x}), \ldots, t_n(\mathbf{x})), \tag{2.155}$$

при

$$t_i(\mathbf{x}) \Leftrightarrow (f_i(\mathbf{x}) \geqslant 0),\ i = 1, \ldots, n, \tag{2.156}$$

и $\beta : \mathbb{B}^n \to \mathbb{B}$ в булевском выражении. Эта проверка включения задается в виде:

$$[t]([\mathbf{x}]) = [\beta]([t_1]([\mathbf{x}]), \ldots, [t_n]([\mathbf{x}])), \tag{2.157}$$

при

$$[t_i]([\mathbf{x}]) \Leftrightarrow ([f_i]([\mathbf{x}]) \geqslant 0),\ i = 1, \ldots, n, \tag{2.158}$$

и $[\beta]$ некоторой функции включения для функции β. Заметим, что даже если проверки $[t_i]([\mathbf{x}])$ все являются минимальными и если $[\beta]$ тоже минимальна, то эффект зависимости еще тайно присутствует, так что ухудшение оценки может наступать. Например, проверка $t(x) = (x \leqslant 7) \vee (x \geqslant 6)$ допускает проверку включения $[t]([x]) = ([x] \leqslant 7) \vee ([x] \geqslant 6)$. Несмотря на тот факт, что $[t]([x])$ состоит из двух минимальных проверок включения и является полиномом и, следовательно, растущим выражением $\beta(b_1, b_2) = b_1 \vee b_2$, вся эта проверка является плохой. Например, для $[x] = [5,\,8]$

$$[t]([x]) = ([5,\,8] \leqslant 7) \vee ([5,\,8] \geqslant 6) = [0,\,1] \vee [0,\,1] = [0,\,1], \tag{2.159}$$

в то время как $t([x]) = \{t(x) \mid x \in [5,\,8]\} = 1$.

2.5.3. Проверки включений для множеств

Пусть $\mathbb{A}$ — множество из $\mathbb{R}^n$; *проверка включения* $[t_{\mathbb{A}}]$ *для* $\mathbb{A}$ является проверкой включения для проверки $t_{\mathbb{A}}(\mathbf{x}) \Leftrightarrow (\mathbf{x} \in \mathbb{A})$, т. е. $[t_{\mathbb{A}}]$ удовлетворяет соотношениям

$$\begin{array}{lll} [t_{\mathbb{A}}]([\mathbf{x}]) = 1 & \Rightarrow & (\forall \mathbf{x} \in [\mathbf{x}],\ t_{\mathbb{A}}(\mathbf{x}) = 1) \Leftrightarrow ([\mathbf{x}] \subset \mathbb{A}), \\ [t_{\mathbb{A}}]([\mathbf{x}]) = 0 & \Rightarrow & (\forall \mathbf{x} \in [\mathbf{x}],\ t_{\mathbb{A}}(\mathbf{x}) = 0) \Leftrightarrow ([\mathbf{x}] \cap \mathbb{A} = \varnothing). \end{array} \tag{2.160}$$

Когда $[t_{\mathbb{A}}]([\mathbf{x}]) = [0,\,1]$, нельзя сделать никаких заключений относительно включения $[\mathbf{x}]$ в $\mathbb{A}$.

Понятие проверки включения множества упростит представление алгоритмов в последующих главах.

Предыдущие определения могут быть скорректированы для проверок включений множеств:

$[t_{\mathbb{A}}]([\mathbf{x}])$ *включение монотонно* тогда и только тогда, когда	$([\mathbf{x}] \subset [\mathbf{y}]) \Rightarrow ([t_{\mathbb{A}}]([\mathbf{x}]) \subset [t_{\mathbb{A}}]([\mathbf{y}]))$
$[t_{\mathbb{A}}]$ *минимально* тогда и только тогда, когда	$\forall [\mathbf{x}] \in \mathbb{IR}^n,\ [t_{\mathbb{A}}]([\mathbf{x}]) = t_{\mathbb{A}}([\mathbf{x}])$
$[t_{\mathbb{A}}]$ *точное* тогда и только тогда, когда	$\forall \mathbf{x} \in \mathbb{R}^n,\ [t_{\mathbb{A}}](\mathbf{x}) \neq [0, 1]$

Будем говорить, что проверка включения $[t_{\mathbb{A}}]$ *более точна*, чем проверка включения $[t'_{\mathbb{A}}]$, тогда и только тогда, когда

$$\forall [\mathbf{x}] \in \mathbb{IR}^n,\ [t_{\mathbb{A}}]([\mathbf{x}]) \subset [t'_{\mathbb{A}}]([\mathbf{x}]). \tag{2.161}$$

Следующие свойства могут быть использованы для построения проверок включения множеств на основе элементарных множественных операций, таких как объединение, пересечение или дополнение. Если $[t_{\mathbb{A}}]([\mathbf{x}])$ и $[t_{\mathbb{B}}]([\mathbf{x}])$ — точные проверки включения множеств $\mathbb{A}$ и $\mathbb{B}$, то определим

$$\begin{array}{ll} [t_{\mathbb{A}\cap\mathbb{B}}]([\mathbf{x}]) & \triangleq ([t_{\mathbb{A}}] \cap [t_{\mathbb{B}}])([\mathbf{x}]) = [t_{\mathbb{A}}]([\mathbf{x}]) \cap [t_{\mathbb{B}}]([\mathbf{x}]), \\ [t_{\mathbb{A}\cup\mathbb{B}}]([\mathbf{x}]) & \triangleq ([t_{\mathbb{A}}] \cup [t_{\mathbb{B}}])([\mathbf{x}]) = [t_{\mathbb{A}}]([\mathbf{x}]) \cup [t_{\mathbb{B}}]([\mathbf{x}]), \\ [t_{\neg\mathbb{A}}]([\mathbf{x}]) & \triangleq \neg[t_{\mathbb{A}}]([\mathbf{x}]) = 1 - [t_{\mathbb{A}}]([\mathbf{x}]). \end{array} \tag{2.162}$$

Проверки $[t_{\mathbb{A}\cap\mathbb{B}}]$, $[t_{\mathbb{A}\cup\mathbb{B}}]$ и $[t_{\neg\mathbb{A}}]$ являются точными проверками включения множеств $\mathbb{A} \cap \mathbb{B}$, $\mathbb{A} \cup \mathbb{B}$ и $\neg\mathbb{A} \triangleq \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{x} \notin \mathbb{A}\}$, соответственно.

2.6. Выводы

Вычисления на множествах должны в общем случае рассматриваться как идеализация, так как исходные множества не могут быть точно представлены и рассчитаны на компьютере. Вот, собственно, почему и было введено множество оболочек. Оболочки являются простыми множествами, с которыми просто работать на компьютере, и которые используются для аппроксимации множеств более сложного вида.

Интервальный анализ использует интервалы и параллелотопы в качестве оболочек, которые дают возможность рассчитывать аппроксимации сверху для областей значения функций. Когда оцениваются интервальные

аргументы, функции включения дают аппроксимации сверху множеств интересующего нас действительного образа. Вычисление только интервалов дает, таким образом, более плохой результат. Такое ухудшение происходит вследствие *эффектов зависимости и обертывания*. Эффект зависимости, уже присутствующий, когда вычисления производятся на множествах, имеет место и когда переменные появляются несколько раз в формальном выражении функции, которую необходимо оценивать. Эффект обертывания появляется вследствие того факта, что исходные множества содержатся в интервалах или параллелотопах.

Ухудшение оценивания может быть уменьшено путем преобразования формального выражения функций для уменьшения числа появлений переменных. Ухудшение оценивания может быть уменьшено также использованием функций включения более утонченных, чем естественная функция включения, и получаемых с помощью замещения каждого оператора и каждой элементарной функции их интервальными аналогами. Только интервалы и параллелотопы сами по себе не могут описать с достаточной точностью все интересующие нас множества. В следующей главе мы покажем, как это может быть сделано с использованием объединений интервалов или параллелотопов.

Глава 3

Покрытия

3.1. Введение

Как мы видели в предыдущей главе, интервалы и параллелотопы образуют привлекательный класс оболочек, с которым просто работать. Однако эти оболочки сами по себе не являются в общем достаточно подходящими для описания всех типов интересующих нас множеств. Эти типы, разумеется, не ограничиваются интервалами и параллелотопами и могут быть, например, объединениями несвязных подмножеств.

Путь, которым мы пойдем, основывается на покрытии интересующего нас множества $\mathbb{X}$ подмножествами из $\mathbb{R}^n$, которые легко представлять и с которыми легко работать. В качестве таких классов подмножеств могут быть эллипсоиды, параллелотопы, многогранники, области специального вида и т. д. (см. [Schweppe, 1968; Fogel и Huang, 1982; Milanese и Belforte, 1982; Walter и Piet-Lahanier, 1989; Milanese et al., 1996; Kuhn, 1998 и ссылки на литературу в этих работах]). С помощью такого покрытия могут быть доказаны важные свойства множества $\mathbb{X}$. Если, например, это покрытие пусто, то пустым является и множество $\mathbb{X}$.

В данной книге множество $\mathbb{X}$ будет покрываться наборами неперекрывающихся параллелотопов из $\mathbb{R}^n$ или *покрытиями* (см. принятую терминологию, с. 16). Мы также будем заключать оцениваемое множество $\mathbb{X}$ между его аппроксимацией снизу (нижней аппроксимацией) и верхней аппроксимацией [Jaulin и Walter, 1993a; 1993в; Jaulin, 1994]. Необходимо будет вычислять два покрытия $\underline{\mathbb{X}}$ и $\overline{\mathbb{X}}$, такие, что

$$\underline{\mathbb{X}} \subset \mathbb{X} \subset \overline{\mathbb{X}}. \tag{3.1}$$

Знание пары $[\underline{\mathbb{X}}, \overline{\mathbb{X}}]$ дает ценную информацию о $\mathbb{X}$. Например, $\mathrm{vol}(\underline{\mathbb{X}}) \leqslant \leqslant \mathrm{vol}(\mathbb{X}) \leqslant \mathrm{vol}(\overline{\mathbb{X}})$ (здесь символ $\mathrm{vol}(\cdot)$ — обозначает некоторую количественную характеристику размера множества); если $\overline{\mathbb{X}}$ пусто, то $\mathbb{X}$ также пусто, и если $\underline{\mathbb{X}}$ не пусто, то $\mathbb{X}$ также не пусто (см. термин «пустое покрытие», с. 16). Становится возможным доказать связность или несвязность

множества $\mathbb{X}$. Например, если $\underline{\mathbb{X}}$ и $\overline{\mathbb{X}}$ оба состоят из двух несвязных множеств: $\underline{\mathbb{X}} = \underline{\mathbb{X}}_1 \cup \underline{\mathbb{X}}_2$ при $\underline{\mathbb{X}}_1 \cap \underline{\mathbb{X}}_2 = \varnothing$ и $\overline{\mathbb{X}} = \overline{\mathbb{X}}_1 \cup \overline{\mathbb{X}}_2$ при $\overline{\mathbb{X}}_1 \cap \overline{\mathbb{X}}_2 = \varnothing$ и, кроме того, $\underline{\mathbb{X}}_1 \subset \overline{\mathbb{X}}_1$ и $\underline{\mathbb{X}}_2 \subset \overline{\mathbb{X}}_2$, то $\mathbb{X}$ *не является связным*. Этот тип информации не может быть получен из представления множества $\mathbb{X}$ в виде облака точек, полученных, например, при использовании метода Монте-Карло или систематической сетки.

В параграфе 3.2 вводится понятие расстояния между множествами, которое используется для оценки качества аппроксимации одного множества другим. В параграфе 3.3. вводится аппроксимация множества на основе покрытий и подпокрытий и поясняется, как эти объекты устроены. Наконец, в параграфе 3.4 представляются алгоритмы оценивания прямых образов и прообразов некоторого компактного множества, определяемых некоторой заданной функцией.

3.2. Топология множеств

3.2.1. Расстояние между компактными множествами

Пусть $\mathbf{C}(\mathbb{R}^n)$ — совокупность всех компактных множеств из $\mathbb{R}^n$. Для количественной оценки качества заданного представления некоторого компактного множества потребуется мера расстояния между двумя множествами $\mathbb{A}$ и $\mathbb{B}$ из $\mathbf{C}(\mathbb{R}^n)$. Напомним, что компактные множества в $\mathbb{R}^n$ являются замкнутыми и ограниченными подмножествами $\mathbb{R}^n$. Снабдим $\mathbb{R}^n$ расстоянием

$$L_\infty(\mathbf{x}, \mathbf{y}) \triangleq \max_{i \in \{1, \ldots, n\}} \{|y_i - x_i|\}. \tag{3.2}$$

При этом единичный шар $\mathbb{U} \triangleq [-1.1]^{\times n}$ в $(\mathbb{R}^n, L_\infty)$ является гиперкубом с длиной ребра два. *Близость* $\mathbb{A}$ к $\mathbb{B}$ оценивается

$$h^0_\infty(\mathbb{A}, \mathbb{B}) \triangleq \inf\{r \in \mathbb{R}^+ \mid \mathbb{A} \subset \mathbb{B} + r\mathbb{U}\}. \tag{3.3}$$

Рис. 3.1 поясняет это определение. Чтобы получить $h^0_\infty(\mathbb{A}, \mathbb{B})$, нужно «раздувать» множество $\mathbb{B}$, пока оно не поглотит множество $\mathbb{A}$, а чтобы получить $h^0_\infty(\mathbb{B}, \mathbb{A})$, нужно «раздувать» множество $\mathbb{A}$, пока оно не поглотит множество $\mathbb{B}$. Заметим, что такое определение расстояния h^0_∞ может быть применено и к некомпактным множествам, однако в этом случае такое расстояние может оказаться бесконечным.

Хаусдорфово расстояние [Berger, 1979] между множествами $\mathbb{A}$ и $\mathbb{B}$ определяется как

$$h_\infty(\mathbb{A}, \mathbb{B}) \triangleq \max\{h^0_\infty(\mathbb{A}, \mathbb{B}), h^0_\infty(\mathbb{B}, \mathbb{A})\}. \tag{3.4}$$

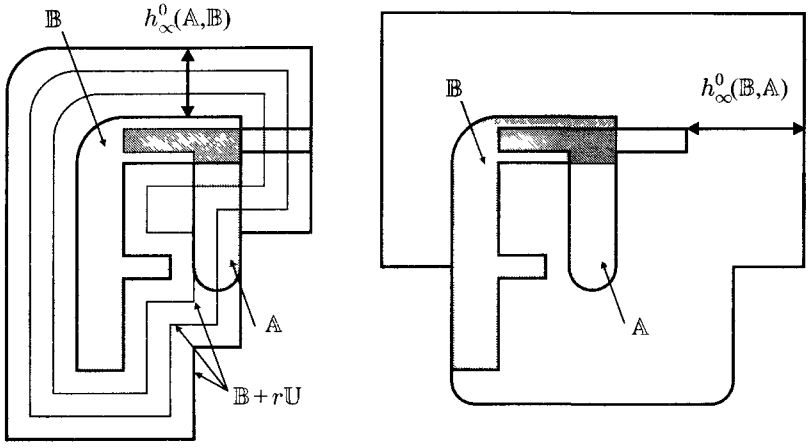


Рис. 3.1. Хаусдорфово расстояние $h_\infty(\mathbb{A},\mathbb{B})$ равно $\max\{h^0_\infty(\mathbb{A},\mathbb{B}), h^0_\infty(\mathbb{B},\mathbb{A})\}$

Так определенная величина является расстоянием на $\mathbf{C}(\mathbb{R}^n)$, поскольку удовлетворяются следующие условия:

1) сепарабельность: $h_\infty(\mathbb{A},\mathbb{B})=0\Rightarrow\mathbb{A}=\mathbb{B}$,
2) симметричность: $h_\infty(\mathbb{A},\mathbb{B})=h_\infty(\mathbb{B},\mathbb{A})$, (3.5)
3) неравенство треугольника: $h_\infty(\mathbb{A},\mathbb{C})\leqslant h_\infty(\mathbb{A},\mathbb{B})+h_\infty(\mathbb{B},\mathbb{C})$.

Рассмотрим некоторое компактное множество $\mathbb{A}$ и некоторую точку $\mathbf{a}$ вне множества $\mathbb{A}$. Тогда множество $\mathbb{A}_1=\mathbb{A}\cup\{\mathbf{a}\}$ также является h_∞-удаленным от $\mathbb{A}$. С другой стороны, множество, полученное «высверливанием» маленьких отверстий в $\mathbb{A}$, остается h_∞-близким к $\mathbb{A}$. Это подчеркивает грубость описания различий между компактными множествами, даваемого хаусдорфовым расстоянием. Потребуется более тонкое определение расстояния для того, чтобы анализировать свойства сходимости алгоритма SIVIA, представленного в параграфе 3.4.1.

Определим *дополнительное хаусдорфово полурасстояние* $\bar{h}_\infty$ между $\mathbb{A}$ и $\mathbb{B}$ из $\mathbf{C}(\mathbb{R}^n)$ следующим образом:

$$\begin{aligned}\bar{h}_\infty(\mathbb{A},\mathbb{B}) &\triangleq h_\infty(\mathbb{R}^n\backslash\mathbb{A},\,\mathbb{R}^n\backslash\mathbb{B})\\ &=\max\{h^0_\infty(\mathbb{R}^n\backslash\mathbb{A},\,\mathbb{R}^n\backslash\mathbb{B}),\,h^0_\infty(\mathbb{R}^n\backslash\mathbb{B},\,\mathbb{R}^n\backslash\mathbb{A})\}\\ &=\max\{\bar{h}^0_\infty(\mathbb{A},\,\mathbb{B},\,\bar{h}^0_\infty(\mathbb{B},\,\mathbb{A})\},\end{aligned}\qquad(3.6)$$

где $\mathbb{R}^n\backslash\mathbb{A}$ является дополнением $\mathbb{A}$ в $\mathbb{R}^n$ и $\bar{h}^0_\infty(\mathbb{A},\mathbb{B})\triangleq h^0_\infty(\mathbb{R}^n\backslash\mathbb{A},\mathbb{R}^n\backslash\mathbb{B})$.

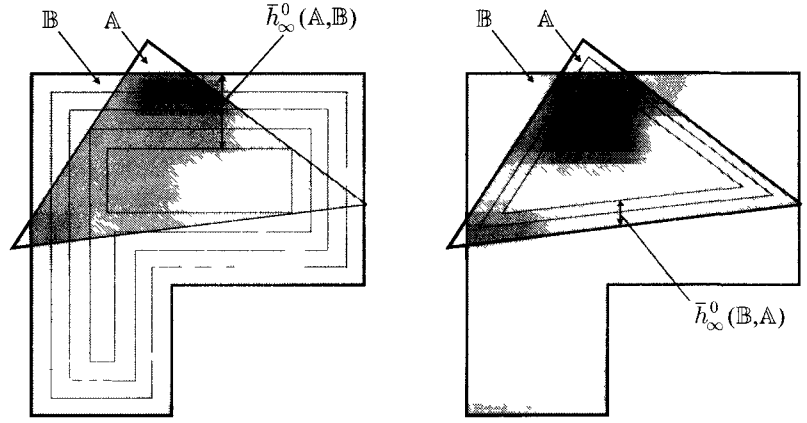


Рис. 3.2. Дополнительное хаусдорфово полурасстояние равно $\max\{\bar{h}^0_\infty(\mathbb{A},\mathbb{B}), \bar{h}^0_\infty(\mathbb{B},\mathbb{A})\}$

Рис. 3.2 поясняет это определение. Расстояние $\bar{h}^0_\infty(\mathbb{A},\mathbb{B})$ определяется «сдуванием» множества $\mathbb{B}$, пока оно не поглотится множеством $\mathbb{A}$, а чтобы получить $\bar{h}^0_\infty(\mathbb{B},\mathbb{A})$, нужно «сдувать» множество $\mathbb{A}$, пока оно не поглотится множеством $\mathbb{B}$. Для ситуации, представленной на рис. 3.2, расстояние $\bar{h}_\infty(\mathbb{A},\mathbb{B})$ равно расстоянию $\bar{h}^0_\infty(\mathbb{A},\mathbb{B})$, так как $\bar{h}^0_\infty(\mathbb{A},\mathbb{B})$ оказывается больше расстояния $\bar{h}^0_\infty(\mathbb{B},\mathbb{A})$.

Оператор $\bar{h}_\infty$ является полурасстоянием на $\mathbf{C}(\mathbb{R}^n)$, так как он не удовлетворяет условию сепарабельности, поскольку $\bar{h}_\infty(\mathbb{B},\mathbb{A})=0$ всегда, когда $\mathbb{A}$ и $\mathbb{B}$ одноточечные множества.

Как и расстояние h_∞, дополнительное хаусдорфово полурасстояние h_∞ не может дать тонкого описания различий между компактными множествами; если компактное множество, полученное «высверливанием» одиночной маленькой дырочки в $\mathbb{A}$, не является $\bar{h}_\infty$-близкой к $\mathbb{A}$, то компактное множество, полученное добавлением к $\mathbb{A}$ конечного числа векторов, отдаленных от $\mathbb{A}$, остается $\bar{h}_\infty$-близкой к $\mathbb{A}$.

На основе h_∞ и $\bar{h}_\infty$ можно построить [Jaulin и Walter, 1993в] некоторое новое расстояние без недостатков, присущих каждому из исходных расстояний:

$$m_\infty(\mathbb{A},\mathbb{B})\triangleq\max\{h_\infty(\mathbb{A},\mathbb{B}),\bar{h}_\infty(\mathbb{B},\mathbb{A})\}.\qquad(3.7)$$

Пример 3.1. *Рассмотрим три компактных подмножества из* $\mathbb{R}$, *заданные следующим образом.* $\mathbb{X}=[1,7]$, $\mathbb{Y}=[1,7]\cup[9-\varepsilon,9]$ *и* $\mathbb{Z}=[1,5]\cup$

$\cup[5+\varepsilon, 7]$, *где* ε *— некоторое положительное число, сходящееся к нулю. Тогда*

$$h_\infty(\mathbb{X},\mathbb{Y}) = 2; \quad \bar{h}_\infty(\mathbb{X},\mathbb{Y}) = {}^{\varepsilon}/_{2}; \quad m_\infty(\mathbb{X},\mathbb{Y}) = \max(2, {}^{\varepsilon}/_{2}) = 2, \tag{3.8}$$

$$h_\infty(\mathbb{X},\mathbb{Z}) = {}^{\varepsilon}/_{2}; \quad \bar{h}_\infty(\mathbb{X},\mathbb{Z}) = 2; \quad m_\infty(\mathbb{X},\mathbb{Z}) = \max({}^{\varepsilon}/_{2}, 2) = 2, \tag{3.9}$$

$$h_\infty(\mathbb{Y},\mathbb{Z}) = 2; \quad \bar{h}_\infty(\mathbb{Y},\mathbb{Z}) = 2; \quad m_\infty(\mathbb{Y},\mathbb{Z}) = \max(2, 2) = 2. \tag{3.10}$$

Хаусдорфово расстояние h_∞ *не выявляет различия между* $\mathbb{X}$ *и* $\mathbb{Z}$*, дополнительное хаусдорфово полурасстояние* $\bar{h}_\infty$ *не выявляет различия между* $\mathbb{X}$ *и* $\mathbb{Y}$*, а расстояние* m_∞ *выявляет оба эти отличия.* ■

3.2.2. Помещение компактных множеств между покрытиями

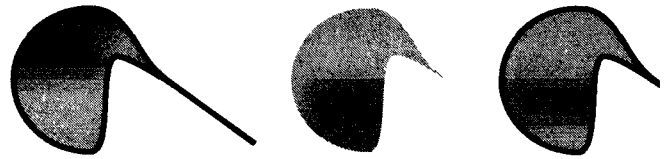
Назовем *покрытием* параллелотопа $[\mathbf{x}] \subset \mathbb{R}^n$ объединение неперекрывающихся подпараллелотопов из $[\mathbf{x}]$, имеющих ненулевую ширину. Два параллелотопа из одного и того же покрытия могут иметь непустое пересечение, если они имеют общую границу, но при этом их внутренние части должны иметь пустое пересечение. Покрытия могут быть использованы для гарантированной аппроксимации компактных множеств. Вычисление на покрытиях позволяет аппроксимировать вычисление на этих компактных множествах, и формирует основу алгоритмов оценивания параметров и состояний; данные алгоритмы будут рассмотрены в Главе 6.

Когда некоторое покрытие $\mathbb{P}$ охватывает параллелотоп $[\mathbf{x}]$, то говорим, что оно является *покрытием* $[\mathbf{x}]$ сверху. *Множество накопления* покрытия $\mathbb{P}$ является пределом подмножества из $\mathbb{R}^n$, образованного объединением всех параллелотопов из $\mathbb{P}$ с шириной меньшей чем ε, при ε, стремящемся к нулю. Поскольку покрытия содержат только параллелотопы ненулевой ширины, множество накопления конечного покрытия необходимо является пустым.

Пусть $(\mathbf{C}(\mathbb{R}^n), \subset, m_\infty)$ — совокупность всех компактных множеств из $\mathbb{R}^n$, снабженная правилом частичного упорядочения $\subset$ и расстоянием m_∞. Множество конечных покрытий является плотным сверху в $(\mathbf{C}(\mathbb{R}^n), \subset, m_\infty)$, т. е. для любого компактного множества $\mathbb{X}$ можно найти некоторое покрытие $\overline{\mathbb{X}}$, содержащее $\mathbb{X}$ и насколько угодно m_∞-близкое к $\mathbb{X}$. Рассмотрим, например, отрезок линии в $\mathbb{R}^2$. Он может быть аппроксимирован покрытием из $\mathbb{R}^2$ насколько угодно близко сверху, но не снизу — изнутри. Чтобы этого избежать, мы иногда ограничиваем рассмотрение лишь на (весьма широком) классе *полных* (*телесных*) *компактных множеств*, т. е. таких компактных множеств, которые равны замыканию их внутренней части. Рис. 3.3 дает пример компактного множества, которое не является полным. Обозначим через $\mathbf{C}_f(\mathbb{R}^n)$ совокупность всех полных компактных множеств в $\mathbb{R}^n$. Множество всех конечных покрытий пространства $\mathbb{R}^n$ является плотным снизу (изнутри) и сверху (снаружи) в $(\mathbf{C}_f(\mathbb{R}^n), \subset, m_\infty)$ [Jaulin и Walter, 1993в]. Таким образом, для любого полного компактного множества $\mathbb{X}$ можно найти пару конечных покрытий $\underline{\mathbb{X}}$ (снизу) и $\overline{\mathbb{X}}$ (сверху), насколько угодно m_∞-близких к $\mathbb{X}$ и таких, что $\underline{\mathbb{X}} \subset \mathbb{X} \subset \overline{\mathbb{X}}$. Набор компактных множеств

$$[\underline{\mathbb{X}}, \overline{\mathbb{X}}] \triangleq \{\mathbb{X}' \in \mathbf{C}_f(\mathbb{R}^n) \mid \underline{\mathbb{X}} \subset \mathbb{X}' \subset \overline{\mathbb{X}}\} \tag{3.11}$$

является некоторой окрестностью множества $\mathbb{X}$, диаметр $m_\infty(\underline{\mathbb{X}}, \overline{\mathbb{X}})$ которой может быть сделан сколь угодно малым [Jaulin, 1994].



а) *б)* *в)*

Рис. 3.3. *а)* компактное множество $\mathbb{A}$; *б)* его внутренняя часть $\mathbb{B}$; *в)* замыкание $\mathbb{C}$ внутренней части $\mathbb{B}$; поскольку множества $\mathbb{A}$ и $\mathbb{C}$ различны, множество $\mathbb{A}$ не является полным

3.3. Регулярные покрытия

Прежде чем более точно определять покрытия (и подпокрытия) и пояснять, как полезный класс регулярных покрытий может быть представлен в компьютере, введем некоторые дополнительные понятия. Далее мы представим несколько элементарных алгоритмов для работы с регулярными покрытиями.

Рассмотрим параллелотоп

$$[\mathbf{x}] = [\underline{x}_1, \overline{x}_1] \times \ldots \times [\underline{x}_n, \overline{x}_n] = [x_1] \times \ldots \times [x_n] \tag{3.12}$$

и выделим индекс j его первой компоненты с максимальной шириной, т. е.

$$j = \max\{i \mid \omega([x_i]) = \omega([\mathbf{x}])\}. \tag{3.13}$$

Определим параллелотопы $L[\mathbf{x}]$ и $R[\mathbf{x}]$ следующим образом:

$$\begin{aligned} L[\mathbf{x}] &\triangleq [\underline{x}_1, \overline{x}_1] \times \ldots \times [\underline{x}_j, (\underline{x}_j + \overline{x}_j)/2] \times \ldots \times [\underline{x}_n, \overline{x}_n], \\ R[\mathbf{x}] &\triangleq [\underline{x}_1, \overline{x}_1] \times \ldots \times [(\underline{x}_j + \overline{x}_j)/2, \underline{x}_j] \times \ldots \times [\underline{x}_n, \overline{x}_n]. \end{aligned} \tag{3.14}$$

Например, если $[\mathbf{x}] = [1,\ 2] \times [2,\ 4] \times [1,\ 3]$, то получаем $w([\mathbf{x}]) = 2$, $j = 2$, $L[\mathbf{x}] = [1,\ 2] \times [2,\ 3] \times [1,\ 3]$ и $R[\mathbf{x}] = [1,\ 2] \times [3,\ 4] \times [1,\ 3]$. $L[\mathbf{x}]$ является *левым отростком* множества $[\mathbf{x}]$, а $R[\mathbf{x}]$ является *правым отростком* множества $[\mathbf{x}]$. L и R могут рассматриваться как операторы из $\mathbb{IR}^n$ в $\mathbb{IR}^n$. Генерирование этих двух отростков по $[\mathbf{x}]$ называется *бисекцией* $[\mathbf{x}]$. Эти два параллелотопа $L[\mathbf{x}]$ и $R[\mathbf{x}]$ являются *родственными*. *Слиянием* назовем операцию объединения двух родственных параллелотопов $L[\mathbf{x}]$ и $R[\mathbf{x}]$ в их родителя $[\mathbf{x}]$. Эту операцию будем обозначать как $[\mathbf{x}] := (L[\mathbf{x}] \mid R[\mathbf{x}])$.

3.3.1. Покрытия и подпокрытия

Покрытие множества $[\mathbf{x}]$ является *регулярным* (см. термины, с. 16), если каждый из его параллелотопов получен из $[\mathbf{x}]$ конечной последовательностью бисекций и отборов. Регулярные покрытия [Jaulin, 1994; Sam-Haroud и Faltings, 1996] называются также *n-деревьями* [Samet, 1990] и являются классом подмножеств из $\mathbb{R}^n$, с которыми, как мы увидим в дальнейшем, легко работать на компьютере. Нерегулярные покрытия также будут использоваться, но такие операции, как пересечение двух покрытий, становятся значительно более трудоемкими в вычислительном плане. Оба типа покрытий дают возможность аппроксимировать полные компактные подмножества из $\mathbb{R}^n$ с желаемой точностью (см., [Lozano-Pérez, 1981; Pruski, 1996; Pruski и Rohmer, 1997]).

Совокупность всех регулярных подпокрытий параллелотопа $[\mathbf{x}]$ будем обозначать как $\mathcal{RSP}([\mathbf{x}])$. Рис. 3.4 показывает *исходный двумерный параллелотоп* $[\mathbf{x}] = [0,\ 8] \times [0,\ 8]$ (обведен жирной рамкой) и его *регулярное покрытие* $\mathbb{P}$ (набор всех белых и серых прямоугольников). Серые прямоугольники образуют *регулярное подпокрытие* $\mathbb{Q}$ параллелотопа $[\mathbf{x}] = [0,\ 8] \times [0,\ 8]$. По любому регулярному подпокрытию (или покрытию) $\mathbb{Q} \in \mathcal{RSP}([\mathbf{x}])$ определим $L\mathbb{Q} \in \mathcal{RSP}(\mathcal{L}[\mathbf{x}])$ как регулярное подпокрытие, содержащее все параллелотопы из $\mathbb{Q}$, также содержащиеся в $L[\mathbf{x}]$. Подобным же образом определим $R\mathbb{Q} \in \mathcal{RSP}(\mathcal{R}[\mathbf{x}])$ как регулярное подпокрытие, содержащее все параллелотопы из $\mathbb{Q}$, также содержащиеся в $R[\mathbf{x}]$. $L\mathbb{Q}$ и $R\mathbb{Q}$ являются, соответственно, левым и правым отростками подпокрытия $\mathbb{Q}$. Если, например, $\mathbb{Q}$ определено как на рис. 3.4, то

$$\begin{aligned} L\mathbb{Q} &= LL[\mathbf{x}] = [0,\ 4] \times [0,\ 4] \\ R\mathbb{Q} &= LRR[\mathbf{x}] \cup LLRRR[\mathbf{x}] \cup LLRRRR[\mathbf{x}] = \\ &= [4,\ 6] \times [4,\ 8] \cup [6,\ 7] \times [4,\ 6] \cup [6,\ 7] \times [6,\ 7]. \end{aligned} \tag{3.15}$$

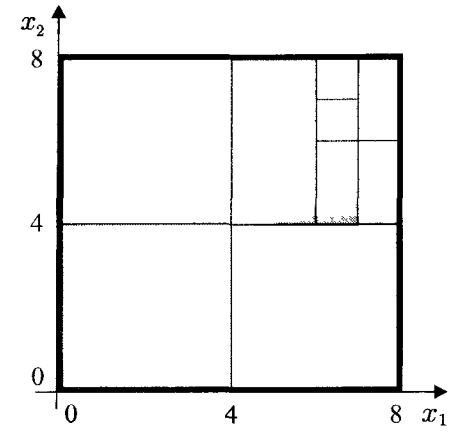


Рис. 3.4. Регулярное покрытие параллелотопа; параллелотопы, отмеченные серым, образуют регулярное подпокрытие

Параллелотоп, из которого $\mathbb{Q}$ строится последовательностью бисекций и отборов параллелотопов, является корнем для $\mathbb{Q}$. Таким образом, $\mathrm{root}(\mathbb{Q}) = [\mathbf{x}]$ и $\mathrm{root}(L\mathbb{Q}) = L[\mathbf{x}]$.

ЗАМЕЧАНИЕ 3.1. Подпокрытие $\mathbb{Q}$ имеет двойственную природу. Оно может рассматриваться как некоторое подмножество в $\mathbb{R}^2$, и мы можем написать $[0,\ 1]^2 \subset \subset \mathbb{Q} \subset \mathbb{R}^2$. С другой стороны, $\mathbb{Q}$ может рассматриваться как конечный список параллелотопов

$$\begin{aligned} &\{LL[\mathbf{x}], LRR[\mathbf{x}], LLRRR[\mathbf{x}], LLRRRR[\mathbf{x}]\} = \\ &\quad = \{[0,\ 4] \times [0,\ 4];\ [4,\ 6] \times [4,\ 8];\ [6,\ 7] \times [4,\ 6];\ [6,\ 7] \times [6,\ 7]\}. \end{aligned} \tag{3.16}$$

Обозначение $\mathbb{Q}$ будет использоваться, когда это подпокрытие рассматривается как множество, а вместо него обозначение $\mathcal{Q}$ будет использоваться, когда это подпокрытие рассматривается как список параллелотопов. ■

Рис. 3.5 показывает погружение (захват) множества

$$\mathbb{S} = \{(x, y)\} \mid x^2 + y^2 \in [1,\ 2]\} \tag{3.17}$$

в вилку между двумя покрытиями с увеличивающейся точностью (слева направо). Рамка соответствует квадрату $[-2,\ 2] \times [-2,\ 2]$. Отмеченные серым цветом прямоугольники образуют *граничное покрытие* (см. термины,

с. 16) $\Delta\mathbb{Q}$, которое содержит границу множества $\mathbb{S}$, в то время как покрытие снизу $\underline{\mathbb{S}}$, отмеченное прямоугольниками белого цвета, является внутренностью множества $\mathbb{S}$. Таким образом,

$$\underline{\mathbb{S}} \subset \mathbb{S} \subset \overline{\mathbb{S}} \quad \text{при} \quad \overline{\mathbb{S}} \triangleq \underline{\mathbb{S}} \cup \Delta\mathbb{S}. \tag{3.18}$$

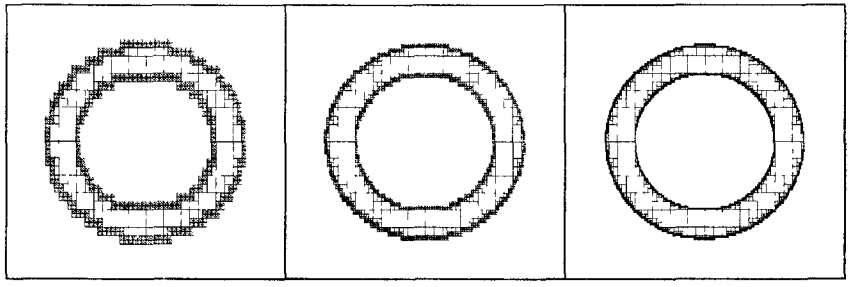


Рис. 3.5. Заключение множества в вилку между двумя покрытиями; точность аппроксимации увеличивается слева направо

3.3.2. Представление регулярного покрытия в виде двоичного дерева

В компьютере регулярное покрытие может быть представлено в виде двоичного дерева. Двоичное дерево содержит конечный набор *узлов*. Этот набор может быть пустым, может содержать единственный узел, *корень* дерева, или может содержать два двоичных дерева с пустым пересечением, а именно, *левое* и *правое поддеревья*. Таким образом, подпокрытие $\mathbb{Q}$ на рис. 3.4 описывается двоичным деревом, представленным на рис. 3.6, где цифра 1 означает, что соответствующий узел принадлежит этому подпокрытию. На данном рисунке множество A является корнем дерева. Множества B и C являются его, соответственно, левым и правым отростками. Они являются родственными, так как имеют один и тот же родительский узел A. Множество A имеет левое и правое поддеревья; правое поддерево узла B является пустым. Узлы A, B и C действительно являются узлами, так как они имеют по крайней мере по одному непустому поддереву. Наконец, так как узел D не имеет поддеревьев, то он является вырожденным узлом или *листом*.

Двоичное дерево, связанное с некоторым покрытием, может быть построено по списку его параллелотопов. Рост количества его ветвей зависит

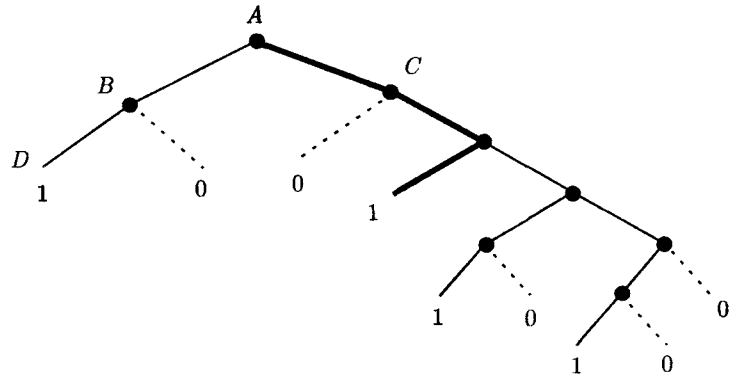


Рис. 3.6. Дерево, связанное с регулярным подпокрытием рис. 3.4

от того, как выполняется бисекция начального параллелотопа $[\mathbf{x}_0]$, который соответствует корню дерева. Любой невырожденный узел соответствует параллелотопу, на которым выполнена бисекция. Любой лист показывает, что параллелотоп, которому он соответствует, принадлежит покрытию. Например, ветвь, отмеченная жирной линией на рис. 3.6, соответствует параллелотопу $LRR[\mathbf{x}_0] = [4,\ 6] \times [4,\ 8]$. *Глубиной* некоторого параллелотопа является число бисекций, необходимых для его получения их корневого параллелотопа. Таким образом, глубина параллелотопа $[4, 6] \times [4, 8]$ равна трем.

Дерево (или соответствующее покрытие) является *минимальным*, если оно не имеет родственных листьев. Любое представление некоторого покрытия в виде неминимального дерева может быть сделано минимальным деревом путем исключения родственных листьев так, чтобы их родители стали листьями. Это достигается слиянием родственных параллелотопов покрытия в соответствующие одинарные параллелотопы.

Поскольку понятия бинарного дерева и регулярного покрытия эквивалентны, терминология деревьев будет также использоваться и для регулярных покрытий. В последующем, представление регулярных покрытий бинарными деревьями будет использоваться вследствие рекурсивности этой информационной структуры.

3.3.3. Основные операции на регулярных покрытиях

Рассматриваются следующие основные операции: слияние родственных покрытий, объединение или пересечение покрытий и проверка факта принадлежности параллелотопа покрытию. Выполнение всех этих операций облегчается использованием бинарных деревьев. Для нерегулярных покры-

тий эти операции стали бы более трудоемкими в вычислениях. Компьютерная реализация регулярных покрытий рассматривается в параграфе 11.12.

Операция слияния родственных покрытий. Рассмотрим некоторый параллелотоп $[\mathbf{x}]$ и два регулярных покрытия $\mathbb{X} \in \mathcal{RSP}(L[\mathbf{x}])$ и $\mathbb{Y} \in \in \mathcal{RSP}(R[\mathbf{x}])$. Эти покрытия в качестве родителя имеют один и тот же параллелотоп $[\mathbf{x}]$. *Слитое* покрытие $\mathbb{Z} \triangleq (\mathbb{X}|\mathbb{Y}) \in \mathcal{RSP}([\mathbf{x}])$ вычисляется следующим образом:

Алгоритм REUNITE(вход: $\mathbb{X}, \mathbb{Y}$; выход: $\mathbb{Z}$)
1 если $\mathbb{X} = L[\mathbf{x}]$ и $\mathbb{Y} = R[\mathbf{x}]$, то $\mathbb{Z} := L[\mathbf{x}]$;
2 иначе если $\mathbb{X} = \varnothing$ и $\mathbb{Y} = \varnothing$, то $\mathbb{Z} := \varnothing$;
3 иначе $L\mathbb{Z} := \mathbb{X}$ и $R\mathbb{Z} := \mathbb{Y}$.

Когда используется представление в виде двоичного дерева, каждая из процедур в алгоритме REUNITE является тривиальной в исполнении. Например, процедуры $L\mathbb{Z} := \mathbb{X}$ и $R\mathbb{Z} := \mathbb{Y}$ приводят к сращиванию двух деревьев $\mathbb{X}$ и $\mathbb{Y}$ на некотором узле для образования дерева $\mathbb{Z}$ (см. рис. 3.7, *б*). Заметим, что число $\#\mathbb{Z}$ параллелотопов в покрытии $\mathbb{Z}$ необязательно равно сумме $\#\mathbb{X} + \#\mathbb{Y}$. Если, например, $[\mathbf{x}] = [0,\ 2]^2$, $\mathbb{X} = [0,\ 1] \times [0,\ 2]$ и $\mathbb{Y} = [1,\ 2] \times [0,\ 2]$, то $\mathbb{X} = L[\mathbf{x}]$ и $\mathbb{Y} = R[\mathbf{x}]$. Таким образом, $\#\mathbb{Z} = 1$ в то время как $\#\mathbb{X} + \#\mathbb{Y} = 2$ (см. на рис. 3.7, *а*). В дальнейшем алгоритм REUNITE$(\mathbb{X}, \mathbb{Y})$ будет просто записываться как $(\mathbb{X} \mid \mathbb{Y})$. Заметим, что операция слияния может рассматриваться как операция, обратная применению операторов L и R, поскольку

$$\mathbb{Z} = (\mathbb{X} \mid \mathbb{Y}) \Leftrightarrow \mathbb{X} = L\mathbb{Z} \ \text{ и } \ \mathbb{Y} = R\mathbb{Z}. \tag{3.19}$$

Операция пересечения покрытий. Если $\mathbb{X} \in \mathcal{RSP}([\mathbf{x}])$ и $\mathbb{Y} \in \in \mathcal{RSP}([\mathbf{x}])$, то покрытие $\mathbb{Z} = \mathbb{X} \cap \mathbb{Y}$ также является покрытием $\mathcal{RSP}([\mathbf{x}])$. Оно содержит только узлы, являющиеся общими для бинарных деревьев, представляющих $\mathbb{X}$ и $\mathbb{Y}$, и может быть вычислено по рекурсивному алгоритму INTER:

Алгоритм INTER(вход: $\mathbb{X}, \mathbb{Y}, [\mathbf{x}]$; выход: $\mathbb{Z}$)
1 если $\mathbb{X} = \varnothing$ или $\mathbb{Y} = \varnothing$, то $\mathbb{Z} := \varnothing$;
2 иначе если $\mathbb{X} = [\mathbf{x}]$, то $\mathbb{Z} := \mathbb{Y}$;
3 иначе если $\mathbb{Y} = [\mathbf{x}]$, то $\mathbb{Z} := \mathbb{X}$;
4 иначе $\mathbb{Z} := (\text{INTER}(L\mathbb{X}, L\mathbb{Y}, L[\mathbf{x}]) \mid$ INTER$(R\mathbb{X}, R\mathbb{Y}, R[\mathbf{x}]))$.

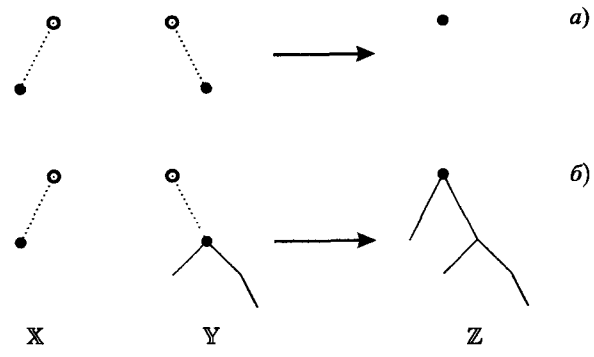


Рис. 3.7. Слияние родственных покрытий: *а* — родственные листья; *б* — родственные покрытия

Операция объединения покрытий. Если $\mathbb{X} \in \mathcal{RSP}([\mathbf{x}])$ и $\mathbb{Y} \in \in \mathcal{RSP}([\mathbf{x}])$, то покрытие $\mathbb{Z} = \mathbb{X} \cup \mathbb{Y}$ также является покрытием $\mathcal{RSP}([\mathbf{x}])$. Оно рассчитывается объединением всех узлов двух бинарных деревьев, представляющих $\mathbb{X}$ и $\mathbb{Y}$. Снова, это выполняется по рекурсивному алгоритму UNION.

Алгоритм UNION(вход: $\mathbb{X}, \mathbb{Y}, [\mathbf{x}]$; выход: $\mathbb{Z}$)
1 если $\mathbb{X} = \varnothing$ или $\mathbb{Y} = [\mathbf{x}]$, то $\mathbb{Z} := \mathbb{Y}$;
2 иначе если $\mathbb{Y} = \varnothing$ или $\mathbb{X} := [\mathbf{x}]$, то $\mathbb{Z} := \mathbb{X}$;
3 иначе $\mathbb{Z} := (\text{UNION}(L\mathbb{X}, L\mathbb{Y}, L[\mathbf{x}]) \mid$ UNION$(R\mathbb{X}, R\mathbb{Y}, R[\mathbf{x}]))$.

Операция проверки факта принадлежности параллелотопа покрытию $\mathbb{X}$ покрытия $\mathcal{RSP}([\mathbf{x}])$. Эта проверка прямо выполняется в четырех случаях. Она верна, если параллелотоп $[\mathbf{z}]$ пуст или если $\mathbb{X}$ сводится к одиночному параллелотопу $[\mathbf{x}]$ и $[\mathbf{z}] \in [\mathbf{x}]$. Она неверна, если $\mathbb{X}$ пусто, а $[\mathbf{z}]$ непуст, или если $[\mathbf{z}]$ не является корневым параллелотопом $\mathbb{X}$. Эти базовые проверки будут применяться прежде всего к корню дерева, представляющего покрытие. Если ни один из этих простых случаев не имеет место, то эти базовые проверки рекурсивно выполняются на левом и правом поддеревьях. Следующий алгоритм INSIDE описывает этот процесс в целом:

Алгоритм INSIDE(вход: $[\mathbf{z}], \mathbb{X}$; выход: t)
1 если $[\mathbf{z}] = \varnothing$ или если ($\mathbb{X}$ есть параллелотоп $[\mathbf{x}]$ и $[\mathbf{z}] \subset [\mathbf{x}]$), то $t := 1$;
2 иначе если $\mathbb{X} = \varnothing$, то $t := 0$;
3 иначе $t :=$ (INSIDE($[\mathbf{z}] \cap L[\mathbf{x}], L\mathbb{X}) \cup$ INSIDE($[\mathbf{z}] \cap R[\mathbf{x}], R\mathbb{X})$).

Когда $[\mathbf{z}] \in \mathbb{X}$, выдается признак $t = 1$; когда $[\mathbf{z}] \cap \mathbb{X} = \varnothing$, то признак равен 0, а в случае, когда $[\mathbf{z}]$ перехлестывает границу $\mathbb{X}$, выдается значение признака $[0, 1]$.

ЗАМЕЧАНИЕ 3.2. Было бы интересно рассмотреть и многие другие алгоритмы, работающие на покрытиях. Например, вычисление окрестностей заданного параллелотопа из некоторого покрытия может быть выполнено по алгоритму Самета [Samet, 1982]. Этот алгоритм мог бы быть очень полезным для изучения вопроса: является ли некоторое покрытие связным, как это требуется в контексте планирования траектории (см. параграф 8.3). ■

3.4. Выполнение вычислений с множествами

Покажем, как два основных блока операций над множествами могут выполняться приближенным, но гарантирующим способом, основанным на понятиях функции включения и проверки включения, описанных в Главе 2, и использующем регулярные покрытия в качестве основного класса объектов для представления множеств. Реализация более утонченных алгоритмов, работающих с множествами, отнесена в последующие главы.

Первый основной блок, который будет рассмотрен, — это вычисление прообраза

$$\mathbb{X} = \mathbf{f}^{-1}(\mathbb{Y}) \tag{3.20}$$

регулярного покрытия $\mathbb{Y}$ из $\mathbb{R}^m$ по функции $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$. Будем называть эту операцию *обращением множества*. Метод вычисления двух покрытий $\underline{\mathbb{X}}$ и $\overline{\mathbb{X}}$ в $\mathbb{R}^n$, таких, что

$$\underline{\mathbb{X}} \subset \mathbb{X} \subset \overline{\mathbb{X}} \tag{3.21}$$

предлагается в параграфе 3.4.1.

Второй основной блок, который будет рассмотрен, — это вычисление прямого образа

$$\mathbb{Y} = \mathbf{f}(\mathbb{X}), \tag{3.22}$$

покрытия $\mathbb{X}$ из $\mathbb{R}^n$ по функции $\mathbf{f}$. Будем называть эту операцию *оцениванием образа*. Алгоритм, который рассчитывает покрытие сверху $\overline{\mathbb{Y}}$ для

покрытия $\mathbb{Y}$, предлагается в параграфе 3.4.2. Будет установлено, что такое оценивание образа является более сложной операцией, чем обращение множества. Кроме того, до настоящего времени не существует другого метода расчета внутренней (снизу) аппроксимации $\underline{\mathbb{Y}}$ для покрытия $\mathbb{Y}$, как только через операцию обращения множества. Это является причиной, по которой можно рекомендовать строить оценку образа в рамках использования операции обращения множества в случае, когда функция $\mathbf{f}$ обратима.

3.4.1. Обращение множества

Пусть $\mathbf{f}$ — некоторая (возможно нелинейная) функция, отображающая из $\mathbb{R}^n$ в $\mathbb{R}^m$, и пусть $\mathbb{Y}$ — некоторое подмножество в $\mathbb{R}^m$ (например, это — некоторое покрытие). Обращение множества — это его описание в следующем виде:

$$\mathbb{X} = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{f}(\mathbf{x}) \in \mathbb{Y}\} = \mathbf{f}^{-1}(\mathbb{Y}). \tag{3.23}$$

Для любого $\mathbb{Y} \subset \mathbb{R}^m$ и для любой функции $\mathbf{f}$, допускающей сходящуюся функцию включения $[\mathbf{f}](\cdot)$, два регулярных покрытия, такие, что

$$\underline{\mathbb{X}} \subset \mathbb{X} \subset \overline{\mathbb{X}}, \tag{3.24}$$

можно найти по алгоритму SIVIA (Set Invertor Via Interval Analysis — инвертор множества на основе интервального анализа [Jaulin и Walter, 1993а; 1993в]). Опишем теперь этот алгоритм.

Алгоритм SIVIA требует некоторого (возможно, очень большого) начального параллелотопа $[\mathbf{x}](0)$, в который покрытие $\overline{\mathbb{X}}$ гарантированно вкладывается. Для облегчения описания на рис. 3.8 показаны основные шаги алгоритма SIVIA в предположении, что $\mathbb{Y}$ — некоторое регулярное покрытие. Могут встретиться следующие четыре случая.

1. Если $[\mathbf{f}](\mathbf{x})$ имеет непустое пересечение с $\mathbb{Y}$, но не целиком находится в $\mathbb{Y}$, то параллелотоп $[\mathbf{x}]$ может содержать некоторую часть множества решения (рис. 3.8, *а*); говорится, что при этом $[\mathbf{x}]$ *недоопределен*. Если этот параллелотоп имеет ширину, большую, чем заранее предписанная величина параметра точности ε, то он должен быть подвергнут бисекции (это подразумевает возникновение двух отростков из $[\mathbf{x}]$), и необходимо рекурсивно применить к этим новообразованным параллелотопам правило проверки принадлежности.

2. Если параллелотоп $[\mathbf{f}](\mathbf{x})$ имеет пустое пересечение с $\mathbb{Y}$, то параллелотоп $[\mathbf{x}]$ не принадлежит покрытию $\mathbb{X}$ и может быть исключен из дерева решения (рис. 3.8, *б*).

3. Если параллелотоп $[\mathbf{f}](\mathbf{x})$ целиком заключен в $\mathbb{Y}$, то параллелотоп $[\mathbf{x}]$ принадлежит искомому решению — покрытию $\mathbb{X}$, и он запоминается в покрытиях $\underline{\mathbb{X}}$ и $\overline{\mathbb{X}}$ (рис. 3.8, *в*).

4. Последний случай изображен на рис. 3.8,*г*. Если рассматриваемый параллелотоп недоопределен, но его ширина меньше, чем параметр точности ε, то он представляется достаточно малым и запоминается в покрытии сверху $\overline{\mathbb{X}}$, аппроксимирующем покрытие $\mathbb{X}$.

Таблица 3.1. Вариант процедуры SIVIA, основанный на использовании функции включения

Алгоритм SIVIA(вход: $\mathbf{f}, \mathbb{Y}, [\mathbf{x}], \varepsilon$; вход/выход: $\underline{\mathbb{X}}, \overline{\mathbb{X}}$)	
1 если $[\mathbf{f}]([\mathbf{x}]) \cap \mathbb{Y} = \varnothing$, то возврат;	// Рис. 3.8*б*
2 если $[\mathbf{f}]([\mathbf{x}]) \subset \mathbb{Y}$, то	
3 $\{\underline{\mathbb{X}} := \underline{\mathbb{X}} \cup [\mathbf{x}]; \overline{\mathbb{X}} := \overline{\mathbb{X}} \cup [\mathbf{x}]$; возврат$\}$;	// Рис. 3.8*в*
4 если $w([\mathbf{x}]) < \varepsilon$, то $\{\overline{\mathbb{X}} := \overline{\mathbb{X}} \cup [\mathbf{x}]\}$; возврат$\}$;	// Рис. 3.8*г*
5 SIVIA($\mathbf{f}, \mathbb{Y}, L[\mathbf{x}], \varepsilon, \underline{\mathbb{X}}, \overline{\mathbb{X}}$); SIVIA($\mathbf{f}, \mathbb{Y}, R[\mathbf{x}], \varepsilon, \underline{\mathbb{X}}, \overline{\mathbb{X}}$).	// Рис. 3.8*а*

SIVIA является рекурсивным алгоритмом, полностью описываемым в табл. 3.1, где покрытия $\underline{\mathbb{X}}$ и $\overline{\mathbb{X}}$ в начале засылаются пустыми множествами.

Покрытие $\Delta\mathbb{X} \triangleq \overline{\mathbb{X}} \setminus \underline{\mathbb{X}}$ является граничным (см. термины, с. 16) и содержит все параллелотопы из $\overline{\mathbb{X}}$, не входящие в $\underline{\mathbb{X}}$; $\Delta\mathbb{X}$ может быть названо также *слоем неопределенности*. Оно является регулярным покрытием, все параллелотопы которого имеют ширину, меньшую, чем параметр точности ε.

Теорема 3.1. [Jaulin и Walter, 1993в]. *Если функция* $\mathbf{f}^{-1}$ *является* m_∞*-непрерывной в окрестности* $\mathbb{Y}$*, то при* ε *стремящемся к нулю*

$$\begin{aligned} &1)\quad \Delta\mathbb{X} \xrightarrow{\supset} \partial\mathbb{X}, \\ &2)\quad \overline{\mathbb{X}} \xrightarrow{\supset} \mathbb{X} \\ &3)\quad \underline{\mathbb{X}} \xrightarrow{\subset} \mathbb{X} \quad \text{(если } \mathbb{X} \text{ является полным)}, \end{aligned} \tag{3.25}$$

где символы $\xrightarrow{\supset}$ *и* $\xrightarrow{\subset}$*, соответственно, обозначают* h_∞*-сходимость сверху и снизу, а* $\partial\mathbb{X}$ *обозначает границу компактного множества* $\mathbb{X}$. ■

С учетом того, что $\mathbb{X}$ является полным, эта теорема означает, что пара $[\underline{\mathbb{X}}, \overline{\mathbb{X}}]$ определяет некоторую окрестность $\mathbb{X}$ с диаметром, который может быть выбран сколь угодно малым. Алгоритм SIVIA заканчивает работу после менее, чем

$$\left(\frac{(w([\mathbf{x}](0))}{\varepsilon} + 1\right)^n$$

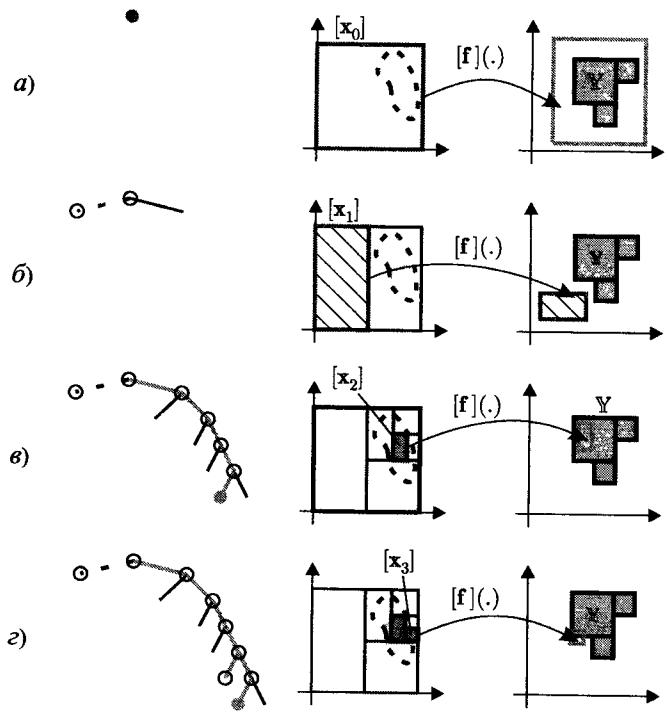


Рис. 3.8. Четыре случая, встречающиеся в алгоритме SIVIA; в левом столбце множество $\mathbb{X} = \mathbf{f}^{-1}(\mathbb{Y})$, отмеченное светло-серой заливкой и штриховой границей, является искомым: *а*) параллелотоп $[\mathbf{x}_0]$, который подлежит проверке, является недоопределенным и должен подвергнуться бисекции; *б*) параллелотоп $[\mathbf{f}]([\mathbf{x}_1])$ не пересекается с $\mathbb{Y}$ и $[\mathbf{x}_1]$ и исключается; *в*) параллелотоп $[\mathbf{f}]([\mathbf{x}_2])$ целиком лежит в $\mathbb{Y}$ и $[\mathbf{x}_2]$ и запоминается в покрытиях $\underline{\mathbb{X}}$ и $\overline{\mathbb{X}}$; *г*) параллелотоп $[\mathbf{x}_3]$ является недоопределенным и представляется слишком малым, чтобы подвергаться дальнейшей бисекции, и он запоминается только в $\overline{\mathbb{X}}$, а в $\underline{\mathbb{X}}$ не запоминается

бисекций и при затратах машинного времени, экспоненциально растущем в зависимости от размерности $\mathbf{x}$ [Jaulin и Walter, 1993а].

Когда интерес представляет только расчет некоторой заданной характеристики покрытия $\mathbb{X}$, например, его интервальной оболочки $[\mathbb{X}]$ или его

размера, то машинная память занимается только этим рекурсивным стеком. Как показано [Jaulin и Walter, 1993a], это чрезвычайно малый объем памяти, оцениваемый как

$$\#\text{stack} \leqslant n(\log_2(w([\mathbf{x}](0)) - \log_2(\varepsilon) + 1). \tag{3.26}$$

Например, для $n = 100, w([\mathbf{x}](0)) = 10^4$ и $\varepsilon = 10^{-10}$ (3.26) дает оценку $\#\text{stack} \leqslant 100(\log_2(10^4) - \log_2(10^{-10}) + 1) = 4751$.

Этот алгоритм может быть обобщен на случай, в котором искомая часть пространства, полагавшаяся ранее некоторым параллелотопом $[\mathbf{x}](0)$, заменяется на покрытие более общего вида [Kieffer, 1999; Kieffer *et al.*, 2001a].

Таблица 3.2 Вариант алгоритма SIVIA, основанный на использовании операции проверки включения

Алгоритм SIVIA(вход: t, $[\mathbf{x}], \varepsilon$; вход/выход: $\underline{\mathbb{X}}, \overline{\mathbb{X}}$)
1 если $[t]([\mathbf{x}]) = 0$, то возврат;
2 если $[t]([\mathbf{x}]) = 1$, то $\{\underline{\mathbb{X}} := \underline{\mathbb{X}} \cup [\mathbf{x}]; \overline{\mathbb{X}} := \overline{\mathbb{X}} \cup [\mathbf{x}];$ возврат$\}$;
3 если $w([\mathbf{x}]) < \varepsilon$, то $\{\overline{\mathbb{X}} := \overline{\mathbb{X}} \cup [\mathbf{x}];$ возврат$\}$;
4 SIVIA$(t, L[\mathbf{x}], \varepsilon, \underline{\mathbb{X}}, \overline{\mathbb{X}})$; SIVIA$(t, R[\mathbf{x}], \varepsilon, \underline{\mathbb{X}}, \overline{\mathbb{X}})$.

Алгоритм SIVIA может также быть представлен через проверку включения $[t](\cdot)$, принимающей значения из набора $\{0, 1, [0, 1]\}$ на месте расположения $[\mathbf{f}]$ и $\mathbb{Y}$, как показано в табл. 3.2. Оба алгоритма запускаются одним и тем же образом.

Пример 3.2. *Пусть $\mathbb{X}$ — множество всех двумерных векторов $\mathbf{x}$ в $\mathbb{R}^2$, удовлетворяющих условиям*

$$\begin{cases} \exp(x_1) + \exp(x_2) & \in [10,\ 11], \\ \exp(2x_1) + \exp(2x_2) & \in [62,\ 72]. \end{cases} \tag{3.27}$$

Описание покрытия $\mathbb{X}$ есть задача обращения множества

$$\mathbb{X} = \mathbf{f}^{-1}([10,\ 11] \times [62,\ 72]), \tag{3.28}$$

при

$$\mathbf{f}(\mathbf{x}) = \begin{pmatrix} \exp x_1 + \exp(x_2) \\ \exp(2x_1) + \exp(2x_2) \end{pmatrix}. \tag{3.29}$$

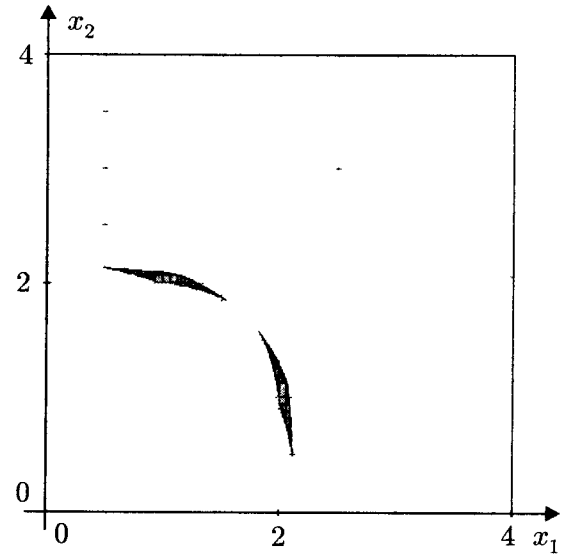


Рис. 3 9 Покрытие, полученное по алгоритму SIVIA для Примера 3.2

Для $[\mathbf{x}](0) = [0,\ 4] \times [0,\ 4]$ и $\varepsilon = 0{,}001$ алгоритм SIVIA *строит регулярное покрытие $\underline{\mathbb{X}}$, показанное на* рис. 3.9, *менее чем за* 2 *секунды на компьютере* Pentium 133. ■

При использовании некоторых операторов сжатия, описываемых в Главе 4, становится возможным улучшить качество описания множества, найденного по алгоритму SIVIA, при заданном числе бисекций. Цена, которую за это приходится платить, это то, что получаемое покрытие может не быть регулярным.

3.4.2. Оценивание образа

Расчет прямого образа покрытия по функции является более сложным, чем расчет прообраза, так как интервальный анализ прямо не обеспечивает какую-либо проверку включения для точечной проверки $t(\mathbf{y}) = (\mathbf{y} \in \mathbf{f}(\mathbb{X}))$. Заметим, что в общем случае трудно оценить точечную проверку, в противоположность точечной проверке $t(\mathbf{x}) = (\mathbf{x} \in \mathbf{f}^{-1}(\mathbb{Y}))$, входящей в операцию обращения множества. Действительно, чтобы проверить $\mathbf{x} \in \mathbf{f}^{-1}(\mathbb{Y})$, доста-

точно рассчитать $\mathbf{f}(\mathbf{x})$ и проверить, принадлежит ли результат $\mathbb{Y}$. С другой стороны, чтобы проверить $\mathbf{y} \in \mathbf{f}(\mathbb{X})$, нужно исследовать, допускает ли набор уравнений $\mathbf{f}(\mathbf{x}) = \mathbf{y}$ по крайней мере одно решение при ограничении $\mathbf{x} \in \mathbb{X}$, что обычно далеко не просто.

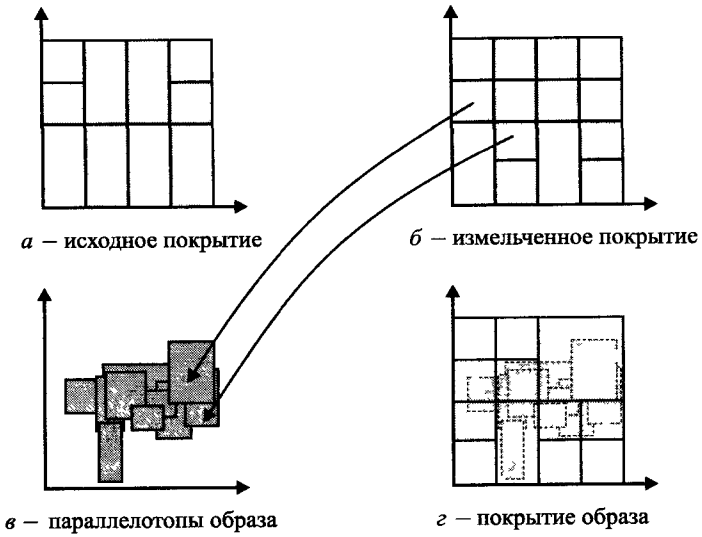


Рис. 3.10. Три шага процедуры IMAGESP: *а* → *б*: измельчение покрытия; *б* → *в*: оценивание; *в* → *г*: регуляризация

Положим, что функция $\mathbf{f}$ непрерывна и что для $\mathbf{f}$ существует сходящаяся функция включения $[\mathbf{f}]$. Алгоритм, который сейчас будет представлен, вырабатывает регулярное покрытие $\overline{\mathbb{Y}}$, содержащее образ $\mathbb{Y}$ регулярного покрытия $\mathbb{X}$ по функции $\mathbf{f}$ (см. также [Kieffer *et al.*, 1998; 2001б]). Множество $\mathbb{Y}$ включено в параллелотоп $[\mathbf{f}](\mathbb{X})$, т. е. в образ, даваемый функцией включения $[\mathbf{f}]$ для наименьшего параллелотопа, содержащего $\mathbb{X}$. Этот алгоритм выполняется за три шага, а именно: *измельчение покрытия, оценивание и регуляризация* (рис. 3.10). Как и в алгоритме SIVIA, точность аппроксимации сверху будет регулироваться вещественным числом $\varepsilon > 0$, которое выбирается пользователем. При измельчении покрытия строится неминимальное регулярное покрытие $\mathbb{X}_\varepsilon$, такое, что ширина каждого из его параллелотопов меньше, чем ε. При оценивании параллелотоп $[\mathbf{f}]([\mathbf{x}])$ вычисляется для каждого параллелотопа $[\mathbf{x}]$ из покрытия $\mathbb{X}_\varepsilon$, и все результирующие параллелотопы запоминаются в списке $\mathcal{U}$. При регуляризации рассчитывается некоторое регулярное покрытие $\overline{\mathbb{Y}}$, которое содержит объединение $\mathbb{U}$ всех параллелотопов из $\mathcal{U}$. Алгоритм регуляризации может рассматриваться как вызов алгоритма SIVIA для обращения объединения $\mathbb{U}$ по тождественной функции с использованием преимущества того факта, что $\mathbf{f}(\mathbb{X}) \subset \mathrm{Id}^{-1}(\mathbb{U})$.

Результирующий алгоритм описывается в табл. 3.3.

Таблица 3.3. Алгоритм оценивания образа, основанный на покрытиях

Алгоритм IMAGESP(вход: $\mathbf{f}, \mathbb{X}, \varepsilon$; выход: $\overline{\mathbb{Y}}$)
1 $\mathbb{X}_\varepsilon :=$ измельчение$(\mathbb{X}, \varepsilon)$;
2 $\mathcal{U} := \varnothing$; //$\mathcal{U}$ есть список, а $\mathbb{U}$ есть набор параллелотопов из списка $\mathcal{U}$
3 для каждого $[\mathbf{x}] \in \mathbb{X}_\varepsilon$, и $[\mathbf{f}]([\mathbf{x}])$ вносится в список $\mathcal{U}$;
4 SIVIA$([\mathbf{y}] \in \mathbb{U}, [\mathbf{f}]([\mathbb{X}]), \varepsilon, \underline{\mathbb{Y}}, \overline{\mathbb{Y}})$. // SIVIA по табл. 3.2

Вычислительная сложность и свойства сходимости алгоритма IMAGESP изучены в [Kieffer, 1999].

Замечание 3.3. Так как алгоритм IMAGESP выдает только аппроксимацию сверху $\overline{\mathbb{Y}}$, то, разумеется, можно использовать упрощенный вариант алгоритма SIVIA без вычисления аппроксимации снизу $\underline{\mathbb{Y}}$. ■

Замечание 3.4. Поскольку $\mathbb{U}$ не является покрытием, реализация алгоритма не является тривиальной (см. подробности в Параграфе 11.12.3). ■

Теорема 3.2. [Jaulin, 2000б; Kieffer *et al.*, 2001б] *Если функция* $\mathbf{f}$ *допускает некоторую сходящуюся функцию включения* $[\mathbf{f}](\cdot)$, *то множества* $\overline{\mathbb{Y}}$ *и* $\mathbb{U}$, *построенные алгоритмом* IMAGESP$(\mathbf{f}, \mathbb{X}, \varepsilon)$, *удовлетворяют следующим свойствам:*

1) $\mathbf{f}(\mathbb{X}) \subset \overline{\mathbb{Y}}$,
2) $h_\infty(\mathbb{U}, \mathbf{f}(\mathbb{X})) \leqslant \max\limits_{[\mathbf{x}] \in \mathbb{X}_\varepsilon} \omega([\mathbf{f}]([\mathbf{x}]))$,
3) $h_\infty(\mathbb{U}, \mathbf{f}(\mathbb{X})) \to 0$, когда $\varepsilon \to 0$,
4) $h_\infty(\overline{\mathbb{Y}}, \mathbf{f}(\mathbb{X})) \to 0$, когда $\varepsilon \to 0$.

■

Пример 3.3. *Рассмотрим регулярное покрытие* $\overline{\mathbb{X}}$, *показанное на* рис. 3.11, *а. Это покрытие охватывает множество*

$$\mathbb{X} = \{(x_1, x_2) \in \mathbb{R}^2 \mid x_1^2 + x_2^2 \in [1,\ 2]\}. \tag{3.30}$$

Алгоритм IMAGESP *используется для вычисления аппроксимации сверху образа покрытия* $\mathbb{X}$ *по функции*

$$\mathbf{f}(\mathbf{x}) = \begin{pmatrix} xy \\ x+y \end{pmatrix}. \tag{3.31}$$

Алгоритм измельчения покрытия вырабатывает регулярное покрытие $\overline{\mathbb{X}}_\varepsilon$ (рис. 3.11, *б*). *Заметим, что хотя* $\overline{\mathbb{X}}_\varepsilon$ *и* $\overline{\mathbb{X}}$ *являются одним и тем же множеством* ($\overline{\mathbb{X}}_\varepsilon = \overline{\mathbb{X}}$), *они не содержат одинаковые списки параллелотопов* ($\overline{\chi}_\varepsilon \neq \overline{\chi}$), *поскольку число параллелотопов в* $\mathbb{X}_\varepsilon$ *больше числа параллелотопов в* $\mathbb{X}$ (см. Замечание 3.1). *На шаге оценивания вырабатывается список* $\mathcal{Y}$ *параллелотопов и объединение этих параллелотопов содержит* $\mathbf{f}(\mathbb{X}_\varepsilon)$ (рис. 3.11, *в*). *Наконец, регуляризация дает регулярное покрытие* $\overline{\mathbb{Y}}$ (рис. 3.11, *г*). ■

Следующий пример использует алгоритмы SIVIA и IMAGESP.

Пример 3.4. *Этот пример разделен на три части. Первая — построение описания множества*

$$\mathbb{X}_1 = \{(x_1, x_2) \in \mathbb{R}^2 \mid x_1^4 - x_1^2 + 4x_2^2 \in [-0{,}1,\ 0,\ 1]\}. \tag{3.32}$$

Эта задача обращения множества решается алгоритмом SIVIA *в параллелотопе поиска* $[\mathbf{x}]_1 = [-3, 3] \times [-3, 3]$ *при* $\varepsilon = 0{,}1$. *Получаемое покрытие* $\overline{\mathbb{X}}_1$ *представлено на* рис. 3.12, *а*.

Вторая часть примера состоит в оценивании некоторой аппроксимации сверху образа $\mathbb{X}_2$ *покрытия* $\overline{\mathbb{X}}_1$ *по функции*

$$\mathbf{f}(x_1, x_2) = \begin{pmatrix} (x_1-1)^2 - 1 + x_2 \\ -x_1^2 + (x_2-1)^2 \end{pmatrix}.$$

При $\varepsilon = 0{,}1$ *алгоритм* IMAGESP *дает покрытие* $\overline{\mathbb{X}}_2$, *изображенное на* рис. 3.12, *б*.

Последняя часть примера состоит в описании образа $\overline{\mathbb{X}}_2$ *обращением функции* $\mathbf{f}(\cdot)$, *т. е. в нахождении* $\mathbb{X}_3 = \{\mathbf{f}^{-1}(\overline{\mathbb{X}}_2)\}$. *Функция* $\mathbf{f}(\cdot)$ *необратима* (*в обычном смысле*) *в* $\mathbb{R}^2$. *Поэтому явный вид обратной функции* $\mathbf{f}^{-1}(\cdot)$ *недоступен на всей области поиска и задача решается как задача обращения множества. Алгоритм* SIVIA *используется, следовательно, для поиска параллелотопа* $[\mathbf{x}]_3 = [-5,\ 5] \times [-5,\ 5]$ *снова при* $\varepsilon = 0{,}1$.

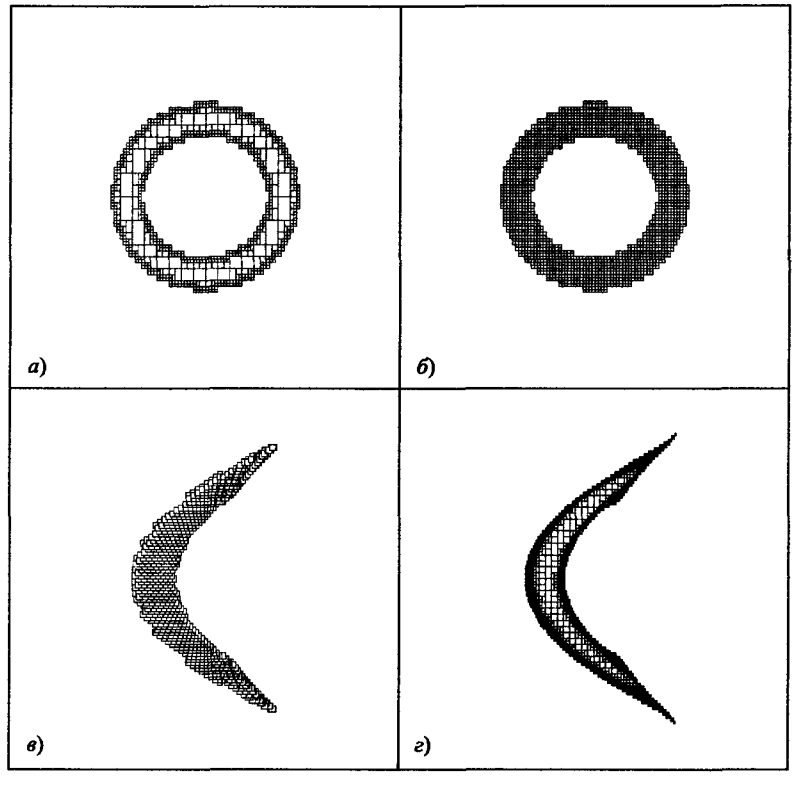


Рис. 3.11. Суть алгоритма IMAGESP: *а*) исходное покрытие; *б*) измельченное покрытие; *в*) оцененное множество; *г*) регуляризованное покрытие, содержащее оцениваемое множество; все квадратные рамки соответствуют параллелотопу $[-3,\ 3] \times [-3,\ 3]$

Верхнее аппроксимирующее покрытие $\overline{\mathbb{X}}_3$ *показано на* рис. 3.12, *в*. *Имеем* $\overline{\mathbb{X}}_1 \subset \mathbf{f}^{-1}(\mathbf{f}(\overline{\mathbb{X}}_1)) \subset \overline{\mathbb{X}}_3$). *Исходное множество* $\overline{\mathbb{X}}_1$ *очевидно присутствует в* $\overline{\mathbb{X}}_3$. *Результат оказывается несколько более «толстым» вследствие ошибки, накапливающейся при вычислениях обратного и прямого оценивания образа. Дополнительные области появились вследствие того, что функция* $\mathbf{f}(\cdot)$ *обращаема только в теоретико-множественном смысле.* ■

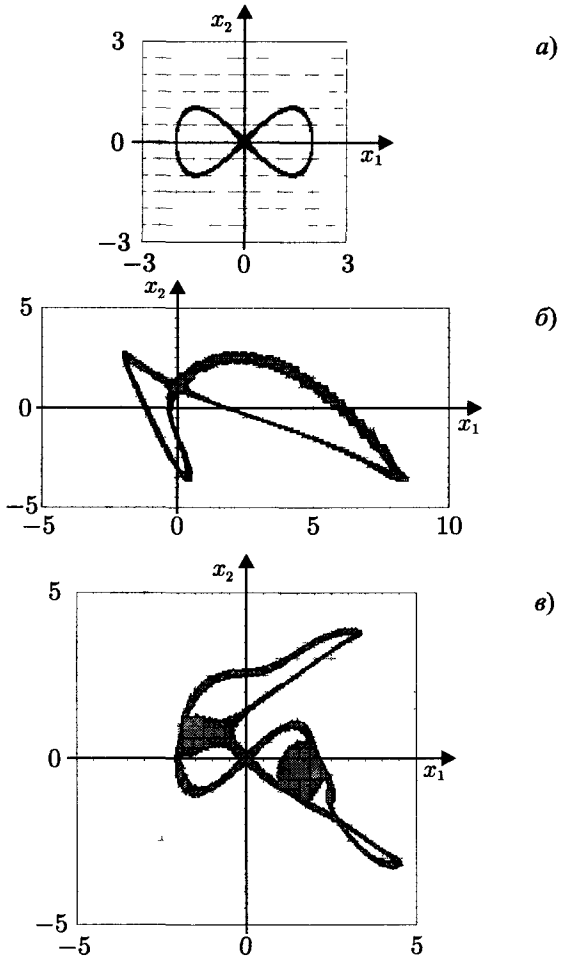


Рис. 3.12. Обратное и прямое оценивание образа. *а)* $\overline{\mathbb{X}}_1$, *б)* $\mathbf{f}(\overline{\mathbb{X}}_1)$, *в)* $\mathbf{f}^{-1}(\mathbf{f}(\overline{\mathbb{X}}_1))$

3.5. Выводы

Понятие покрытия, введенное в этой главе, делает возможным получать аппроксимации компактных множеств снизу и сверху, запоминать их и работать с ними. Покрытия образуют некоторый полезный класс объектов, на которых в дальнейшем будут выполняться вычисления. Были представлены два основных алгоритма для выполнения прямого и обратного оценивания функций на покрытиях. Проблема получения внутренней аппроксимации для образа множеств остается нерешенной в случае отсутствия обратной функции из-за невозможности построения ограничения на обращение множества.

Регулярные покрытия оказываются более простыми в запоминании и работе, чем исходные покрытия, но приводят к более сложным представлениям с точки зрения затрат машинной памяти. Таким образом, они более подходят к задачам с малой размерностью. Для множеств более высокой размерности требование регулярности покрытия может быть ослаблено использованием сжимающих операторов, рассматриваемых в следующей главе.

Глава 4

Сжимающие операторы

4.1. Введение

Рассмотрим $n_{\mathbf{x}}$ переменных $x_i \in \mathbb{R}$, $i = 1, \ldots, n_{\mathrm{x}}$, связанных $n_{\mathbf{f}}$ соотношениями (или ограничениями) вида:

$$f_j(x_1, x_2, \ldots, x_{n_{\mathbf{x}}}) = 0, \quad j \in \{1, \ldots, n_{\mathbf{f}}\}. \tag{4.1}$$

Известно, что каждая переменная x_i принадлежит *области* $\mathbb{X}_i$. Для простоты, эти области полагаются интервалами, обозначаемыми как $[x_i]$, но могут также рассматриваться и объединения этих интервалов. Обозначим вектор $\mathbf{x}$ как

$$\mathbf{x} = (x_1, \ldots, x_{n_{\mathbf{x}}})^{\mathrm{T}}, \tag{4.2}$$

а соответствующую область значений вектора $\mathbf{x}$ как

$$[\mathbf{x}] = [x_1] \times \ldots \times [x_{n_{\mathbf{x}}}]. \tag{4.3}$$

Пусть $\mathbf{f}$ — некоторая вектор-функция, координатами которой являются функции f_j. При этом уравнение (4.1) можно записать в векторной форме как $\mathbf{f}(\mathbf{x}) = \mathbf{0}$. Оно соответствует *задаче выполнения ограничений* $\mathcal{H}$, которая формулируется как

$$\mathcal{H} : (\mathbf{f}(\mathbf{x}) = \mathbf{0},\ \mathbf{x} \in [\mathbf{x}]). \tag{4.4}$$

Множество решения задачи (4.4) определяется как

$$\mathbb{S} = \{\mathbf{x} \in [\mathbf{x}],\ \mathbf{f}(\mathbf{x}) = \mathbf{0}). \tag{4.5}$$

Задачи выполнения ограничений могут включать ограничения в виде равенств и неравенств. Например, набор ограничений

$$\begin{cases} x_1 + \sin x_2 \leqslant 0, \\ x_1 - x_2 = 3, \end{cases} \tag{4.6}$$

может быть сведен к задаче выполнения ограничений введением *вспомогательной переменной* x_3 для получения набора ограничений

$$\begin{cases} x_1 + \sin(x_2) + x_3 = 0, \\ x_1 - x_2 - 3 = 0, \end{cases} \tag{4.7}$$

где переменные имеют области значений $[x_1] = \mathbb{R}, [x_2] = \mathbb{R}, [x_3] = [0, \infty[$, а координатные функции f_j имеют вид: $f_1(\mathbf{x}) = x_1 + \sin(x_2) + x_3$ и $f_2(\mathbf{x}) = x_1 - x_2 - 3$. Описание множества решения $\mathbb{S}$ в общем случае имеет полиномиальную сложность (NP-сложность), что означает, что, в наихудшем случае, никакой алгоритм с полиномиальной сложностью по числу переменных не может дать точную аппроксимацию множества $\mathbb{S}$.

Первоначально такие задачи были определены на областях с дискретными значениями, т. е. когда значения аргументов x_i принадлежали конечному набору [Clowes, 1975; Waltz, 1975; Mackwotrh, 1977a; 1977б; Freuder, 1978; Mackwoth и Freuder, 1985; Dechter и Dechter, 1987]. Позднее, эти задачи были расширены на области с непрерывными значениями (подмножества из $\mathbb{R}$ и интервалы) [Cleary, 19874 Davis; 1987; Hyönen, 1992; Lhomme, 1993; Benhamou *et al.*, 1994; Sam-Haroud, 1995; Sam-Haroud и Faltings, 1996; Deville *et al.*, 1997; van Hentenryck *et al.*, 1998; Lottaz *et al.*, 1998]. Большинство алгоритмов, описанных в данных работах, используют аппарат совместности, как это делается в алгоритмах, описываемых ниже, для нахождения аппроксимации сверху множества $\mathbb{S}$ всех решений задачи $\mathcal{H}$. Главное преимущество этого аппарата состоит в том, что он вырабатывает гарантированное замыкание множества $\mathbb{S}$ при полиномиальной сложности алгоритмов по памяти и времени счета.

Сжатие задачи $\mathcal{H}$ означает замену параллелотопа $[\mathbf{x}]$ некоторой меньшей областью $[\mathbf{x}']$, такой, что множество решения остается неизменным, т. е. $\mathbb{S} \subset [\mathbf{x}'] \subset [\mathbf{x}]$. Существует оптимальное сжатие $\mathcal{H}$, которое соответствует замещению $[\mathbf{x}]$ наименьшим параллелотопом, содержащим $\mathbb{S}$. *Сжимающим оператором* (или — оператором сжатия) для $\mathcal{H}$ является любой оператор, который может быть применен для сжатия $\mathcal{H}$. Чтобы держать сложность вычислений (по времени и памяти) на уровне полиномиальной, нельзя позволять сжимающим операторам производить бисекцию областей. Будем обозначать сжимающий оператор символом $\mathcal{C}$ с нижним индексом, показывающим принцип, на котором оператор основывается. Сжимающие операторы, которые будут рассмотрены в данной главе, перечислены в табл. 4.1.

Замечание 4.1. Иногда бывает удобно выделить несколько типов областей для величин с неопределенностью, таких как $[\mathbf{A}]$, $[\mathbf{b}]$ и $[\mathbf{p}]$ в табл. 4.1, *вместо того, чтобы объединять их в некоторый единый параллелотоп* $[\mathbf{x}]$. ■

Таблица 4.1. Рассматриваемые операторы сжатия

Оператор	Основан на	Параграф
$\mathcal{C}_{\mathrm{GE}}(\mathbf{Ap}-\mathbf{b}=\mathbf{0},[\mathbf{A}],[\mathbf{p}],[\mathbf{b}])$	Метод исключения Гаусса	4.2.2
$\mathcal{C}_{\mathrm{GS}}(\mathbf{Ap}-\mathbf{b}=\mathbf{0},[\mathbf{A}],[\mathbf{p}],[\mathbf{b}])$	Алгоритм Гаусса–Зейделя	4.2.3
$\mathcal{C}_{\mathrm{K}}(\mathbf{f}(\mathbf{x})=\mathbf{0},[\mathbf{x}])$	Метод Кравчика	4.2.3
$\mathcal{C}_{\uparrow\downarrow}(\mathbf{f}(\mathbf{x})=\mathbf{0},[\mathbf{x}])$	Метод вперед-назад	4.2.4
$\mathcal{C}_{\mathrm{LP}}(\mathbf{Ap}-\mathbf{b}=\mathbf{0},[\mathbf{A}],[\mathbf{p}],[\mathbf{b}])$	Линейное программирование	4.2.5
$\mathcal{C}_{\mathrm{GSP}}(\mathbf{Ap}-\mathbf{b}=\mathbf{0},[\mathbf{A}],[\mathbf{p}],[\mathbf{b}])$	Алгоритм Гаусса–Зейделя с улучшением обусловленности	4.3.2
$\mathcal{C}_{\mathrm{GEP}}(\mathbf{Ap}-\mathbf{b}=\mathbf{0},[\mathbf{A}],[\mathbf{p}],[\mathbf{b}])$	Метод исключения Гаусса с улучшением обусловленности	4.3.2
$\mathcal{C}_{\mathrm{N}}(\mathbf{f}(\mathbf{x})=\mathbf{0},[\mathbf{x}])$	Метод Ньютона с улучшением обусловленности	4.3.3
$\mathcal{C}_{\|}(\mathbf{f}(\mathbf{x})=\mathbf{0},[\mathbf{x}])$	Параллельная линеаризация	4.3.4

Основными сжимающими операторами, рассматриваемыми в параграфе 4.2, являются операторы $\mathcal{C}_{\mathrm{GE}}, \mathcal{C}_{\mathrm{GS}}, \mathcal{C}_{\mathrm{K}}, \mathcal{C}_{\uparrow\downarrow}$ и $\mathcal{C}_{\mathrm{LP}}$.

Эти операторы оказываются эффективными только на специальных классах задач. Параграф 4.3 показывает некоторые приемы преобразования задачи выполнения ограничений так, чтобы указанные основные операторы стали более широко применимыми. Введение преобразования в процедуры этих операторов сжатия приводит к тому, что новые операторы $\mathcal{C}_{\mathrm{GSP}}, \mathcal{C}_{\mathrm{GEP}}, \mathcal{C}_{\mathrm{N}}$ и $\mathcal{C}_{\|}$ могут работать на гораздо более широком классе задач. В параграфе 4.4 имеющиеся операторы преобразуются так, чтобы работать совместно для повышения их эффективности. Параграф 4.5 дает понятие оператора сжатия на множествах. Это понятие не привносит ничего нового в методическом плане, но позволяет просто работать с операторами сжатия независимо от типа ограничений, которые определяют рассматриваемое множество. В следующей главе мы покажем, как операторы сжатия можно применять для получения эффективных разрешающих методов для работы с различными задачами оптимизации и решения систем уравнений и неравенств.

4.2. Основные сжимающие операторы

В этом параграфе будут представлены основные сжимающие операторы. Некоторые из них являются интервальными аналогами, классических

точечных алгоритмов таких, как метод исключения Гаусса, алгоритмов метода Гаусса–Зейделя и Ньютона. В других используется метод вперед-назад распространения ограничения. Каждый их этих алгоритмов эффективен только для конкретных задач выполнения ограничений. Но, как мы увидим в параграфе 4.3, подходящая комбинация этих сжимающих операторов, с возможным дополнением их некоторым формальным преобразованием или улучшением обусловленности, делает их эффективными для гораздо более широкого класса таких задач. Чтобы показать полезность некоторого сжимающего оператора, достаточно показать, что он расширяет класс задач, решаемых эффективно, даже если все еще остаются задачи, на которых оператор не работает.

В параграфе 4.2.1 дается понятие конечных решающих операторов, используемых в параграфе 4.2.2 для построения сжимающих операторов на основе интервализации. В параграфе 4.2.3 рассматриваются операторы сжатия, получаемые интервализацией методов с неподвижной точки. Оператор, основанный на методе вперед-назад, описывается в параграфе 4.2.4. Последний основной оператор сжатия, который представляется в параграфе 4.2.5, использует сильные стороны аппарата линейного программирования.

4.2.1. Конечные разрешающие операторы

Конечный разрешающий подоператор задачи выполнения ограничений $\mathcal{H} : (\mathbf{f}(\mathbf{x}) = \mathbf{0}, \mathbf{x} \in [\mathbf{x}])$ является конечным алгоритмом, вычисляющим величины *некоторых* переменных x_i, когда некоторые другие переменные x_j неизвестны. (*Прим. перев.*: далее, без потери смысла, для упрощения изложения вместо термина *подоператор* используется просто термин *оператор*, более привычный российскому читателю.) Рис. 4.1. иллюстрирует некоторый разрешающий оператор ϕ, вычисляющий x_8 и x_9 по x_1, x_2, x_3 и x_4. Формальное определение разрешающего оператора потребует определения подвектора заданного вектора.

Определение 4.1. *Вектор* $\mathbf{u} = (u_1, \ldots, u_{n_\mathbf{u}})^{\mathrm{T}}$ *является подвектором вектора* $\mathbf{x} = (x_1, \ldots, x_{n_\mathbf{x}})^{\mathrm{T}}$, *если существует некоторое подмножество индексов* $\mathcal{I} = \{i_1, \ldots, i_{n_\mathbf{u}}\}$ *множества* $\{1, \ldots, n_\mathbf{x}\}$, *такое, что* $\mathbf{u} = (x_{i_1}, \ldots, x_{i_{n_\mathbf{u}}})^{\mathrm{T}}$. *При этом* $\mathcal{I}$ *называется множеством индексов вектора* $\mathbf{u}$, *и мы будем записывать* $\mathbf{u} = \mathbf{x}_\mathcal{I}$. ■

Определение 4.2. *Рассмотрим* $\mathcal{I} = \{i_1, \ldots, i_{n_\mathbf{u}}\}$ *и* $\mathcal{J} = \{j_1, \ldots, j_{n_\phi}\}$, *два таких множества индексов, что* $\mathcal{I} \cap \mathcal{J} = \varnothing$, *и два подвектора* $\mathbf{u} = \mathbf{x}_\mathcal{I}$ *и* $\mathbf{v} = \mathbf{x}_\mathcal{J}$ *одного и того же вектора* $\mathbf{x}$. *Конечный разрешающий оператор, связанный с* $\mathcal{H}$, *является множественно-значным алгорит-*

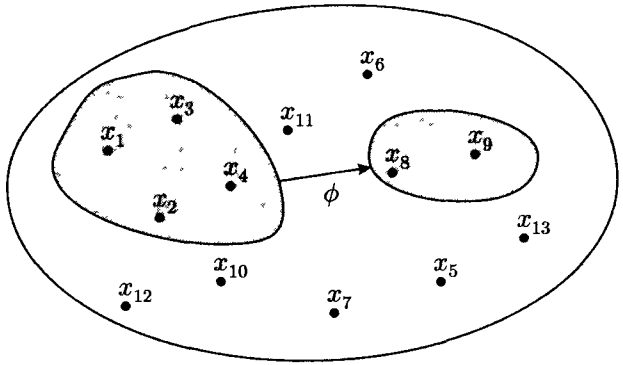


Рис. 4.1. Разрешающие операторы, вычисляющие x_8 и x_9, когда x_1, x_2, x_3 и x_4 известны

мом $\phi : \mathbf{u} \mapsto \phi(\mathbf{u})$, *таким, что следующее вложение оказывается верным:*

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \Rightarrow \mathbf{v} \in \phi(\mathbf{u}). \tag{4.8}$$

Компоненты $\mathbf{u}$ *называются входами* ϕ, *а* $\mathbf{v}$ *называются его выходами. В ситуации, показанной на* рис. 4.1, *входами являются* x_1, x_2, x_3 *и* x_4, *а выходами —* x_8 *и* x_9. ■

Часто функция $\phi(\mathbf{u})$ является векторно-значной (синглтоном) и взаимная связь в правой части (4.8) может пониматься как $\mathbf{v} = \phi(\mathbf{u})$. Следующий пример показывает, что определение функции ϕ как множественно-значной функции может быть полезным и поясняет концепцию разрешающего оператора.

Пример 4.1. *Рассмотрим задачу выполнения ограничений*

$$\mathcal{H} : \begin{pmatrix} x_1 x_2 - x_3 = 0 \\ x_2 - \sin(x_4) = 0 \\ [x_1] = [x_2] =]-\infty, 0], \ [x_3] = [x_4] = \mathbb{R} \end{pmatrix}. \tag{4.9}$$

Многие операторы, разрешающие задачу (4.9), *могут быть получены на основе элементарных алгебраических вычислений. Вот пять таких*

вычислений:

$$\begin{array}{ll}
\phi_a(\text{вход: } x_1, x_2; \text{ выход: } x_3) & \{x_3 := x_1 x_2\}, \\
\phi_b(\text{вход: } x_1, x_3; \text{ выход: } x_2) & \{x_2 := x_3 / x_1 \\
 & \text{если } x_1 \neq 0, \text{и } \mathbb{R} \text{ в противном случае}\}, \\
\phi_c(\text{вход: } x_4; \text{ выход: } x_2) & \{x_2 := \sin(x_4)\}, \\
\phi_d(\text{вход: } x_1, x_3, x_4; \text{ выход: } x_2) & \{x_2 := \phi_b(x_1, x_3) \cap \phi_c(x_4)\}, \\
\phi_e(\text{вход: } x_3, x_4; \text{ выход: } x_1, x_2) & \{x_2 := \sin(x_4), \\
 & x_1 := x_3 / x_2 \\
 & \text{если } x_2 \neq 0, \text{и } \mathbb{R} \text{ в противном случае}\}.
\end{array} \tag{4.10}$$

Заметим, что ϕ_d *может дать пустое множество, когда* x_1, x_3 *и* x_4 *несовместны с* $\mathcal{H}$. *Но это необязательно тот случай, в котором некоторые несовместные значения* x_1, x_3 *и* x_4 *таковы, что* $\phi_d(x_1, x_3, x_4) \neq \varnothing$. *Например, значения* $x_1 = x_4 = 0$ *и* $x_3 = 1$ *являются несовместными (так как* $x_1 x_2 \neq 0$), *но* $\phi_d(0, 1, 0) = \mathbb{R} \cap \{0\} = 0$. ■

Пример 4.2. *Рассмотрим задачу выполнения ограничений*

$$\mathcal{H} : \begin{pmatrix} \mathbf{A}\mathbf{p} - \mathbf{b} = \mathbf{0} \\ \mathbf{p} \in \mathbb{R}^{n_\mathbf{p}}, \ \mathbf{b} \in [\mathbf{b}], \ \mathbf{A} \in [\mathbf{A}] \end{pmatrix}, \tag{4.11}$$

где $\mathbb{A}$ *является квадратной матрицей. Множество всех переменных в задаче* (4.11) *есть*

$$\mathbf{x} = (a_{11}, \ldots, a_{n_\mathbf{p} n_\mathbf{p}}, p_1, \ldots, p_{n_\mathbf{p}}, b_1, \ldots, b_{n_\mathbf{p}})^\mathrm{T}. \tag{4.12}$$

Возможный разрешающий оператор имеет вид:

$$\phi_f(\text{вход: } \mathbf{A}, \mathbf{p}; \text{ выход: } \mathbf{b}) \ \{\mathbf{b} := \mathbf{A}\mathbf{p}\}, \tag{4.13}$$

где входной подвектор состоит из коэффициентов $\mathbf{A}$ *и* $\mathbf{p}$, *а* $\mathbf{b}$ *является выходным подвектором. Еще один разрешающий оператор имеет вид*:

$$\begin{array}{l}
\phi_g(\text{вход : } \mathbf{A}, \mathbf{b}; \text{ выход : } \mathbf{p}) \\
\quad \{\text{формула линейного разрешающего оператора}\}.
\end{array} \tag{4.14}$$

Линейный разрешающий оператор может, например, использовать метод исключения Гаусса, который будет описан ниже. ■

4.2.2. Интервальные конечные разрешающие операторы

В этом параграфе мы покажем, как знание интервального аналога некоторого конечного разрешающего оператора для задачи $\mathcal{H} : (\mathbf{f}(\mathbf{x}) = \mathbf{0},\ \mathbf{x} \in [\mathbf{x}])$ делает возможным сжать $\mathcal{H}$.

Пусть ϕ — некоторый конечный разрешающий оператор для $\mathcal{H}$ с вектором входа $\mathbf{u}$ и вектором выхода $\mathbf{v}$. *Функция включения* $[\phi]$ для ϕ является функцией, отображающей из $\mathbb{IR}^{n_\mathbf{u}}$ в $\mathbb{IR}^{n_\phi}$, так, что для всех параллелотопов $[\mathbf{u}]$ из $\mathbb{IR}^{n_\mathbf{u}}$ имеет место

$$\phi([\mathbf{u}]) \subset [\phi]([\mathbf{u}]), \quad \text{где } \phi([\mathbf{u}]) \triangleq \bigcup_{\mathbf{u}\in[\mathbf{u}]} \phi(\mathbf{u}). \tag{4.15}$$

Это определение является некоторым расширением определения, данного в Главе 2, если принять во внимание тот факт, что функция $\phi([\mathbf{u}])$ может не быть векторно-значной (синглтоном). Например, интервализация разрешающего оператора $\phi_e(x_3, x_4)$ из Примера 4.1 задается следующим образом:

$$\begin{aligned}
&[\phi_e](\text{вход} : [x_3], [x_4];\ \text{выход} : [x_1], [x_2]) \\
&\quad \{[x_2] := \sin([x_4]); \\
&\quad\ [x_1] := [x_3]/[x_2], \quad \text{если } 0 \notin [x_2]; \\
&\quad\ [x_1] := \mathbb{R} \text{ в противном случае}\}.
\end{aligned} \tag{4.16}$$

Теорема 4.1. *Рассмотрим некоторую задачу выполнения ограничений* $\mathcal{H} : (\mathbf{f}(\mathbf{x}) = \mathbf{0}, \mathbf{x} \in [\mathbf{x}])$ *и один из ее конечных разрешающих операторов* ϕ *с входным* $\mathbf{u} = \mathbf{x}_\mathcal{I}$ *и выходным* $\mathbf{v} = \mathbf{x}_\mathcal{J}$ *векторами. Если* $[\phi]$ *является функцией включения для* ϕ*, то сжатие* $\mathcal{H}$ *может быть выполнено замещением каждой области* $[x_j]$, $j \in \mathcal{J}$, *областью* $[x_j] \cap [\phi_j]([\mathbf{u}])$. ■

Доказательство. Пусть $\mathbb{S}$ — множество решения задачи $\mathcal{H}$.

$$\begin{aligned}
\mathbf{x} \in \mathbb{S} &\overset{(4.5)}{\Leftrightarrow} \mathbf{x} \in [\mathbf{x}] \text{ и } \mathbf{f}(\mathbf{x}) = \mathbf{0} \\
&\overset{(4.8)}{\Leftrightarrow} \mathbf{x} \in [\mathbf{x}] \text{ и } \mathbf{f}(\mathbf{x}) = \mathbf{0} \text{ и } \mathbf{v} \in \phi(\mathbf{u}) \\
&\overset{(4.15)}{\Leftrightarrow} \mathbf{x} \in [\mathbf{x}] \text{ и } \mathbf{f}(\mathbf{x}) = \mathbf{0} \text{ и } [\mathbf{v}] \in [\phi]([\mathbf{u}]).
\end{aligned} \tag{4.17}$$

Таким образом, это замещение не изменяет множества решения. ■

ЗАМЕЧАНИЕ 4.2. Результирующий оператор сжатия может, конечно, оставить неизменным параллелотоп $[\mathbf{x}]$, но в этом случае оператор не сможет дать какой-либо полезный результат. ■

Пример 4.3. *Рассмотрим снова ситуацию, описанную в* Примере 4.1 *с разрешающим оператором* ϕ_e. Теорема 4.1 *и* (4.16) *дают*

$$\begin{aligned}
[x_2] &=]-\infty, 0] \cap \sin(\mathbb{R}) = [-1, 0], \\
[x_1] &=]-\infty, 0] \cap \mathbb{R} =]-\infty, 0].
\end{aligned}$$

■

Интервальный метод исключения Гаусса. Важным классом задач выполнения ограничений, для которого можно реализовать интервальные конечные разрешающие операторы, являются *линейные системы* (*с квадратной матрицей*) *интервальных уравнений*. Задача состоит в расчете параллелотопа, содержащего множество решения:

$$\mathcal{H} : \begin{pmatrix} \mathbf{A} \in [\mathbf{A}],\ \mathbf{b} \in [\mathbf{b}],\ \mathbf{p} \in [\mathbf{p}] \\ \mathbf{A}\mathbf{p} - \mathbf{b} = \mathbf{0} \end{pmatrix}, \tag{4.18}$$

при

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n_\mathbf{p}} \\ \vdots & & & \vdots \\ a_{n_\mathbf{p}1} & a_{n_\mathbf{p}2} & \dots & a_{n_\mathbf{p}n_\mathbf{p}} \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_{n_\mathbf{p}} \end{pmatrix}. \tag{4.19}$$

Переменные в задаче (4.18) образуют вектор

$$\mathbf{x} = (a_{11}, \dots, a_{n_\mathbf{p}n_\mathbf{p}}, p_1, \dots, p_{n_\mathbf{p}}, b_1, \dots, b_{n_\mathbf{p}})^\mathrm{T}. \tag{4.20}$$

ЗАМЕЧАНИЕ 4.3. При том, что выражение линейных интервальных уравнений является классическим в интервальном анализе, связи между переменными являются билинейными, поскольку входы $\mathbf{A}$ и $\mathbf{p}$ принадлежат $\mathbf{x}$. Мы будем придерживаться классического выражения линейных интервальных уравнений, принятого в литературе, но будем говорить о линейной задаче выполнения ограничений только когда все связи между фигурирующими переменными линейны. Это имеет место в случае, когда область $[\mathbf{A}]$ является точечной матрицей, что влечет за собой, что вектор всех переменных сводится к $\mathbf{x} = (p_1, \dots, p_{n_\mathbf{p}}, b_1, \dots, b_{n_\mathbf{p}})^\mathrm{T}$ и, следовательно, произведения переменных отсутствуют. ■

Классическая процедура метода исключения Гаусса может быть использована для (4.18) в качестве конечного разрешающего оператора. Она дает возможность рассчитывать вектор $\mathbf{p}$, когда вектор

$$\mathbf{u} = (a_{11}, a_{12}, \dots, a_{n_\mathbf{p}n_\mathbf{p}}, b_1, \dots, b_{n_\mathbf{p}})^\mathrm{T}. \tag{4.21}$$

известен. Простая реализация этой процедуры приведена в табл. 4.2, но можно было бы рассматривать и более эффективные реализации. Функция включения для этого конечного разрешающего оператора приводится в

Таблица 4.2. Метод исключения Гаусса

Алгоритм ϕ(вход : $a_{11}, \ldots, a_{n_{\mathbf{p}}n_{\mathbf{p}}}, b_1, \ldots, b_{n_{\mathbf{p}}}$; выход: $p_1, \ldots, p_{n_{\mathbf{p}}}$)	
1	для $i := 1$ to $n_{\mathbf{p}} - 1$
2	если $a_{ii} := 0$, то $(p_1, \ldots, p_{n_{\mathbf{p}}}) := \mathbb{R}^{n_{\mathbf{p}}}$; возврат;
3	для $j := i + 1$ до $n_{\mathbf{p}}$
4	$\alpha_j := a_{ji}/a_{ii}$; $b_j := b_j - \alpha_j * b_i$;
5	для $k := i + 1$ до $n_{\mathbf{p}}$
6	$a_{jk} := a_{jk} - \alpha_j * a_{ik}$;
7	для $i := n_{\mathbf{p}}$ уменьшая до 1
8	$p_i := \left(b_i - \sum_{j=i+1}^{n_{\mathbf{p}}} a_{ij} * p_j\right) / a_{ii}$.

табл. 4.3. Разумеется, можно использовать также центрированную форму и другой тип функции включения.

По Теореме 4.1, оператор

$$\mathcal{C}_{\mathrm{GE}}([\mathbf{A}], [\mathbf{b}], [\mathbf{p}]) \mapsto ([\mathbf{A}], [\mathbf{b}], [\mathbf{p}] \cap [\phi]([\mathbf{A}], [\mathbf{b}], [\mathbf{p}])), \tag{4.22}$$

где $[\phi](\cdot)$ задается табл. 4.3, а индекс GE указывает на использование метода исключения Гаусса.

Оператор (4.22) является сжимающим для задачи (4.18). Вследствие того, что условие $0 \in [a_{ii}]$ часто выполняется для некоторых i, оператор $\mathcal{C}_{\mathrm{GE}}$ часто неспособен выполнить сжатие (4.18). Оператор $\mathcal{C}_{\mathrm{GE}}$ эффективен, например, когда интервальная матрица $[\mathbf{A}]$ близка к тождественной (единичной).

Заметим, что область $[\mathbf{b}]$ для $\mathbf{b}$ может быть сжата применением сжимающего оператора

$$\phi(\mathbf{A}, \mathbf{p}) = \mathbf{A}\mathbf{p}. \tag{4.23}$$

Пример 4.4. *Для*

$$[\mathbf{A}] = \begin{pmatrix} [4,\ 5] & [-1,\ 1] & [1{,}5,\ 2{,}5] \\ [-0{,}5,\ 0{,}5] & [-7,\ -5] & [1,\ 2] \\ [-1{,}5,\ -0{,}5] & [-0{,}7,\ -0{,}5] & [2,\ 3] \end{pmatrix},$$
$$[\mathbf{b}] = \begin{pmatrix} [3,\ 4] \\ [0,\ 2] \\ [3,\ 4] \end{pmatrix} \text{ и } [\mathbf{p}] = \begin{pmatrix} [-\infty, \infty] \\ [-\infty, \infty] \\ [-\infty, \infty] \end{pmatrix}, \tag{4.24}$$

Таблица 4.3. Интервальный метод исключения Гаусса

Алгоритм $[\phi]$(вход : $[a_{11}], \ldots, [a_{n_{\mathbf{p}}n_{\mathbf{p}}}], [b_1], \ldots, [b_{n_{\mathbf{p}}}]$; выход: $[p_1], \ldots, [p_{n_{\mathbf{p}}}]$)	
1	для $i := 1$ до $n_{\mathbf{p}} - 1$
2	если $0 \in [a_{ii}]$
3	$\left([p_1], \ldots, [p_{n_{\mathbf{p}}}]\right) := \mathbb{R}^{n_{\mathbf{p}}}$; возврат;
4	для $j := i + 1$ до $n_{\mathbf{p}}$
5	$[\alpha_j] := [a_{ji}]/[a_{ii}]$; $[b_j] := [b_j] - [\alpha_i] * [b_i]$;
6	для $k := i + 1$ до $n_{\mathbf{p}}$
7	$[a_{jk}] := [a_{jk}] - [\alpha_j] * [a_{ik}]$;
8	для $i := n_{\mathbf{p}}$ уменьшая до 1
9	$[p_i] := \left([b_i] - \sum_{j=i+1}^{n_{\mathbf{p}}} [a_{ij}] * [p_j]\right) / [a_{ii}]$.

оператор $\mathcal{C}_{\mathrm{GE}}([\mathbf{A}], [\mathbf{b}], [\mathbf{p}])$ *дает*

$$[\mathbf{p}] = \begin{pmatrix} [-1{,}81928,\ 1{,}16873] \\ [-0{,}414071,\ 1{,}72523] \\ [0{,}700233,\ 3{,}42076] \end{pmatrix}. \tag{4.25}$$

Этот пример решается в Упражнении 11.25, (с. 409). *Можно проверить, что, если снова применить оператор* $\mathcal{C}_{\mathrm{GE}}$, *область* $\mathbf{p}$ *больше не сжимается. При этом говорится, что оператор* $\mathcal{C}_{\mathrm{GE}}$ *является идемпотентным. Некоторый оператор сжатия, связанный с разрешающим оператором* (4.23), *должен сжимать и* $\mathbf{b}$, *что может позволить новое сжатие* $\mathbf{p}$ *оператором* $\mathcal{C}_{\mathrm{GE}}$. ■

4.2.3. Метод неподвижной точки

Разрешающий оператор с неподвижной точкой для задачи выполнения ограничений $\mathcal{H} : (\mathbf{f}(\mathbf{x}) = \mathbf{0}, \mathbf{x} \in [\mathbf{x}])$ — это алгоритм ψ, такой, что

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \Leftrightarrow \mathbf{x} = \psi(\mathbf{x}). \tag{4.26}$$

Если последовательность

$$\mathbf{x}_{k+1} = \psi(\mathbf{x}_k) \tag{4.27}$$

сходится к некоторой точке $\mathbf{x}_\infty$ при некотором заданном начальном значении $\mathbf{x}_0$, то $\mathbf{x}_\infty$ является решением уравнения $\mathbf{f}(\mathbf{x}) = \mathbf{0}$. Например, поскольку для любого $a \neq 0$

$$x = a(x^2 - 2) + x) \Leftrightarrow x^2 - 2 = 0, \tag{4.28}$$

то разрешающий оператор с неподвижной точкой для задачи выполнения ограничений ($x^2 - 2 = 0$, $x \in \mathbb{R}$) имеет вид:

$$\psi(x) = a(x^2 - 2) + x. \tag{4.29}$$

Разрешающий оператор дает итерационную процедуру, которая может сходиться к одному из решений уравнения $\mathbf{f}(\mathbf{x}) = \mathbf{0}$. Например, когда $n_\mathbf{f} = n_\mathbf{x}$, возможный разрешающий оператор с неподвижной точкой для задачи $\mathcal{H}$ имеет вид:

$$\boldsymbol{\psi}(\mathbf{x}) = \mathbf{x} - \mathbf{M}\mathbf{f}(\mathbf{x}), \tag{4.30}$$

где $\mathbf{M}$ — некоторая обратимая матрица, возможно зависящая от $\mathbf{x}$.

Теорема 4.2. *Пусть $\boldsymbol{\psi} : \mathbb{R}^{n_\mathbf{x}} \to \mathbb{R}^{n_\mathbf{x}}$ — некоторый разрешающий оператор с неподвижной точкой для задачи выполнения ограничений* (4.26) *и $[\boldsymbol{\psi}] : \mathbb{IR}^{n_\mathbf{x}} \to \mathbb{IR}^{n_\mathbf{x}}$ — некоторая функция включения для $\boldsymbol{\psi}$. Некоторый сжимающий оператор для $\mathcal{H}$ получается замещением $[\mathbf{x}]$ в $\mathcal{H}$ на*

$$[\mathbf{x}] \cap [\boldsymbol{\psi}]([\mathbf{x}]). \tag{4.31}$$

Этот сжимающий оператор будем называть сжимающим оператором с неподвижной точкой, связанным с $\boldsymbol{\psi}$. ■

Доказательство. Пусть $\mathbb{S}$ — решение задачи $\mathcal{H}$ (4.26). Для любого $\mathbf{x} \in \mathbb{S}$

$$\begin{aligned} \mathbf{f}(\mathbf{x}) = \mathbf{0} \text{ и } \mathbf{x} \in [\mathbf{x}] &\overset{(4.26)}{\Leftrightarrow} \mathbf{x} \in [\mathbf{x}] \text{ и } \mathbf{x} = \boldsymbol{\psi}(\mathbf{x}) \\ &\Rightarrow \mathbf{x} \in [\mathbf{x}] \text{ и } \mathbf{x} = \boldsymbol{\psi}([\mathbf{x}]) \\ &\Rightarrow \mathbf{x} \in [\mathbf{x}] \cap [\boldsymbol{\psi}]([\mathbf{x}]). \end{aligned}$$

Следовательно, $\mathbb{S} \subset [\mathbf{x}] \cap [\boldsymbol{\psi}]([\mathbf{x}])$. ■

Чтобы проиллюстрировать данный подход, опишем три сжимающих оператора с неподвижной точкой, а именно, интервальные операторы Гаусса–Зейделя, Кравчика и Ньютона.

Сжимающий оператор Гаусса–Зейделя. Рассмотрим снова задачу выполнения ограничений

$$\mathcal{H} : \begin{pmatrix} \mathbf{A} \in [\mathbf{A}],\ \mathbf{b} \in [\mathbf{b}],\ \mathbf{p} \in [\mathbf{p}] \\ \mathbf{A}\mathbf{p} - \mathbf{b} = \mathbf{0} \end{pmatrix}, \tag{4.32}$$

в которой матрица $\mathbf{A}$ полагается квадратной. $\mathbf{A}$ можно разложить на сумму диагональной матрицы и матрицу с нулями на диагонали:

$$\mathbf{A} = \mathrm{diag}(\mathbf{A}) + \mathrm{extdiag}(\mathbf{A}). \tag{4.33}$$

Теперь, уравнение $\mathbf{A}\mathbf{p} - \mathbf{b} = \mathbf{0}$ эквивалентно уравнению

$$\mathrm{diag}(\mathbf{A})\mathbf{p} + \mathrm{extdiag}(\mathbf{A})\mathbf{p} = \mathbf{b}. \tag{4.34}$$

С учетом того, что матрица $\mathrm{diag}(\mathbf{A})$ обратима (т. е. $\mathbf{A}$ имеет ненулевые коэффициенты на ее диагонали), это уравнение можно переписать в следующем виде:

$$\mathbf{p} = (\mathrm{diag}(\mathbf{A}))^{-1}(\mathbf{b} - \mathrm{extdiag}(\mathbf{A})\mathbf{p}). \tag{4.35}$$

Таким образом, разрешающий оператор с неподвижной точкой для задачи (4.32) имеет вид:

$$\boldsymbol{\psi}\begin{pmatrix} \mathbf{A} \\ \mathbf{b} \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{A} \\ \mathbf{b} \\ (\mathrm{diag}(\mathbf{A}))^{-1}(\mathbf{b} - \mathrm{extdiag}(\mathbf{A})\mathbf{p}) \end{pmatrix}. \tag{4.36}$$

Некоторая функция включения для $\boldsymbol{\psi}$ имеет вид:

$$[\boldsymbol{\psi}]\begin{pmatrix} [\mathbf{A}] \\ [\mathbf{b}] \\ [\mathbf{p}] \end{pmatrix} = \begin{pmatrix} [\mathbf{A}] \\ [\mathbf{b}] \\ (\mathrm{diag}([\mathbf{A}]))^{-1}([\mathbf{b}] - \mathrm{extdiag}([\mathbf{A}])[\mathbf{p}]) \end{pmatrix}. \tag{4.37}$$

По Теореме 4.2 сжимающий оператор задается как

$$\mathcal{C}_{\mathrm{GS}} : \begin{pmatrix} [\mathbf{A}] \\ [\mathbf{b}] \\ [\mathbf{p}] \end{pmatrix} \mapsto \begin{pmatrix} [\mathbf{A}] \\ [\mathbf{b}] \\ [\mathbf{p}] \cap (\mathrm{diag}([\mathbf{A}]))^{-1}([\mathbf{b}] - \mathrm{extdiag}([\mathbf{A}])[\mathbf{p}]) \end{pmatrix}. \tag{4.38}$$

$\mathcal{C}_{\mathrm{GS}}$ является *сжимающим оператором Гаусса–Зейделя*. Этот оператор эффективен, например, когда матрица $[\mathbf{A}]$ близка к тождественной (единичной).

Пример 4.5. *Рассмотрим снова ситуацию* Примера 4.4, *где*

$$[\mathbf{A}] = \begin{pmatrix} [4,\ 5] & [-1,\ 1] & [1{,}5,\ 2{,}5] \\ [-0{,}5,\ 0{,}5] & [-7,\ -5] & [1,\ 2] \\ [-1{,}5,\ -0{,}5] & [-0{,}7,\ -0{,}5] & [2,\ 3] \end{pmatrix}, \tag{4.39}$$

$$[\mathbf{b}] = \begin{pmatrix} [3,\ 4] \\ [0,\ 2] \\ [3,\ 4] \end{pmatrix} \text{ и } [\mathbf{p}] = \begin{pmatrix} [10,\ 10] \\ [10,\ 10] \\ [10,\ 10] \end{pmatrix}. \tag{4.40}$$

Тогда

$$(\mathrm{diag}([\mathbf{A}]))^{-1} = \begin{pmatrix} [0{,}2,\ 0{,}25] & [0,\ 0] & [0,\ 0] \\ [0,\ 0] & [-0{,}2,\ -0{,}1429] & [0,\ 0] \\ [0,\ 0] & [0,\ 0] & [0{,}3333,\ 0{,}5] \end{pmatrix} \tag{4.41}$$

и

$$\mathrm{extdiag}([\mathbf{A}]) = \begin{pmatrix} [0,\ 0] & [-1,\ 1] & [1{,}5,\ 2{,}5] \\ [-0{,}5,\ 0{,}5] & [0,\ 0] & [1,\ 2] \\ [-1{,}5,\ -0,5] & [-0{,}7,\ -0{,}5] & [0,\ 0] \end{pmatrix}. \tag{4.42}$$

Оператор $\mathcal{C}_{\mathrm{GS}}$: $([\mathbf{A}], [\mathbf{b}], [\mathbf{p}])$ *дает*

$$[\mathbf{p}]) = \begin{pmatrix} [-8,\ 9{,}75] \\ [-5{,}4001,\ 5{,}0001] \\ [-9{,}5,\ 10] \end{pmatrix}. \tag{4.43}$$

Таблица 4.4. Результаты итераций сжимающего оператора Гаусса – Зейделя

k	$[p_1](k)$	$[p_2](k)$	$[p_3](k)$
0	[−10, 10]	[−10, 10]	[−10, 10]
1	[−8, 9,75]	[−5,40001, 5,00001]	[−9,5, 10]
2	[−6,85001, 8,28751]	[−5,17501, 4,97501]	[−6,39001, 10]
5	[−5,66909, 5,24052]	[−3,03602, 4,03031]	[−4,65079, 7,84124]
10	[−3,82831, 3,40254]	[−1,86306, 2,85556]	[−2,27608, 5,79364]
20	[−2,48786, 2,03998]	[−0,994123, 2,00077]	[−0,775045, 4,29151]
100	[−2,08452, 1,63673]	[−0,736983, 1,74357]	[−0,321329, 3,83781]

Применим итерационно сжатие оператором $\mathcal{C}_{\mathrm{GS}}$. *Результаты, полученные для некоторых номеров* k *итераций, приведены в* табл. 4.4, *где* $[\mathbf{p}](k)$ — *параллелотоп, полученный для* $[\mathbf{p}]$ *на итерации номера* k. *Результаты, полученные на этом примере, менее точны по сравнению с результатами при работе оператора* $\mathcal{C}_{\mathrm{GE}}$ *в* Примере 4.4 (с. 100), *но такое бывает не всегда. Этот пример решается в* Упражнении 11.26 (с. 410). ■

Сжимающий оператор Кравчика. Рассмотрим задачу выполнения ограничений $\mathcal{H}$: $(\mathbf{f}(\mathbf{x}) = \mathbf{0}, \mathbf{x} \in [\mathbf{x}])$, где $n_{\mathbf{f}} = n_{\mathbf{x}}$ и $\mathbf{f}$ полагается дифференцируемым. Так как для любой обратимой матрицы $\mathbf{M}$ имеем $\mathbf{f}(\mathbf{x}) = \mathbf{0} \Leftrightarrow \mathbf{x} - \mathbf{M}\mathbf{f}(\mathbf{x}) = \mathbf{x}$, то функция $\psi(\mathbf{x}) = \mathbf{x} - \mathbf{M}\mathbf{f}(\mathbf{x})$ является разрешающим оператором с неподвижной точкой для этой задачи выполнения ограничений. Центрированная функция включения для ψ имеет вид:

$$[\psi]([\mathbf{x}]) = \psi(\mathbf{x}_0) + [\mathbf{J}_\psi]([\mathbf{x}]) * ([\mathbf{x}] - \mathbf{x}_0), \tag{4.44}$$

где $[\mathbf{J}_\psi]$ является функцией включения для матрицы Якоби ψ, и $\mathbf{x}_0 = \mathrm{mid}([\mathbf{x}])$ (напомним, что выражение $\mathrm{mid}([\mathbf{x}])$ обозначает центр $[\mathbf{x}]$). По Теореме 4.2 получается следующий оператор сжатия с неподвижной точкой, классически называемый *оператором сжатия Кравчика* [Neumaier, 1990]:

$$\mathcal{C}_{\mathrm{K}} : [\mathbf{x}] \mapsto [\mathbf{x}] \cap (\psi(\mathbf{x}_0) + [\mathbf{J}_\psi]([\mathbf{x}]) * ([\mathbf{x}] - \mathbf{x}_0)). \tag{4.45}$$

Заменим $\psi(\mathbf{x})$ на $\mathbf{x} - \mathbf{M}\mathbf{f}(\mathbf{x})$ в (4.45), чтобы получить

$$\mathcal{C}_{\mathrm{K}} : [\mathbf{x}] \mapsto [\mathbf{x}] \cap (\mathbf{x}_0 - \mathbf{M}\mathbf{f}(\mathbf{x}_0) + (\mathbf{I} - \mathbf{M}[\mathbf{J}_\mathbf{f}]([\mathbf{x}])) * ([\mathbf{x}] - \mathbf{x}_0)), \tag{4.46}$$

где $\mathbf{I}$ — тождественная (единичная) матрица, а $[\mathbf{J}_\mathbf{f}]$ — некоторая функция включения матрицы Якоби для $\mathbf{f}$. Матрица $\mathbf{M}$ часто выбирается равной обратной матрице $\mathbf{J}_\mathbf{f}^{-1}(\mathbf{x}_0)$ Якоби для $\mathbf{f}$, рассчитываемой в точке $\mathbf{x}_0$. Она может рассматриваться как обусловливающая матрица (см. параграф 4.3.2, с. 115). Алгоритм, реализующий оператор $\mathcal{C}_{\mathrm{K}}$, представлен в табл. 4.5.

Таблица 4.5. Сжимающий оператор Кравчика

Алгоритм $\mathcal{C}_{\mathrm{K}}$(вход : $\mathbf{f}$; вход/выход : $[\mathbf{x}]$)	
1	$\mathbf{x}_0 := \mathrm{mid}([\mathbf{x}])$;
2	$\mathbf{M} := \mathbf{J}_\mathbf{f}^{-1}(\mathbf{x}_0)$;
3	$[\mathbf{J}_\psi] := \mathbf{I} - \mathbf{M}[\mathbf{J}_\mathbf{f}]([\mathbf{x}])$;
4	$[\mathbf{r}] := \mathbf{x}_0 - \mathbf{M}\mathbf{f}(\mathbf{x}_0) + [\mathbf{J}_\psi] * ([\mathbf{x}] - \mathbf{x}_0)$;
5	$[\mathbf{x}] := [\mathbf{x}] \cap [\mathbf{r}]$.

Пример 4.6. *Рассмотрим задачу выполнения ограничений* $\mathcal{H} : (\mathbf{f}(\mathbf{x}) = \mathbf{0},\ \mathbf{x} \in [\mathbf{x}])$, *описываемую как*

$$\mathcal{H} : \begin{pmatrix} f_1(x_1, x_2) = x_1^2 - 4x_2 \\ f_2(x_1, x_2) = x_2^2 - 2x_1 + 4x_2 \\ [\mathbf{x}] = [-0{,}1,\ 0{,}1] \times [-0{,}1,\ 0{,}3] \end{pmatrix}. \tag{4.47}$$

задача (4.47) *имеет единственное решение* $\mathbf{x} = (0,\ 0)^{\mathrm{T}}$. *Матрица Якоби для* $\mathbf{f}$ *имеет вид*:

$$\mathbf{J_f} = \begin{pmatrix} 2x_1 & -4 \\ -2 & 2x_2 + 4 \end{pmatrix}, \tag{4.48}$$

а обусловливающая матрица $\mathbf{M}$ *описывается как*

$$\mathbf{M} = \mathbf{J_f}^{-1}(0, 0{,}1) = \begin{pmatrix} -0{,}525 & -0{,}5 \\ -0{,}25 & 0 \end{pmatrix}. \tag{4.49}$$

Оператор сжатия Кравчика дает последовательность

$$\begin{aligned} \mathcal{C}_{\mathrm{K}}([\mathbf{x}]) &= \begin{pmatrix} [-0{,}0555,\ 0{,}0455] \\ [-0{,}005,\ 0{,}005] \end{pmatrix}, \\ \mathcal{C}_{\mathrm{K}}(\mathcal{C}_{\mathrm{K}}([\mathbf{x}])) &= \begin{pmatrix} [-0{,}00258,\ 0{,}00255] \\ [-0{,}00128,\ 0{,}00127] \end{pmatrix}, \\ \mathcal{C}_{\mathrm{K}}(\mathcal{C}_{\mathrm{K}}(\mathcal{C}_{\mathrm{K}}([\mathbf{x}]))) &= \begin{pmatrix} [-0{,}00000818,\ 0{,}00000817] \\ [-0{,}00000329,\ 0{,}00000329] \end{pmatrix}, \end{aligned} \tag{4.50}$$

сходящуюся к единственному решению. Этот пример рассчитывается в Упражнении 11.27 (с. 411). ■

Сжимающий оператор Ньютона. Рассмотрим снова задачу выполнения ограничений

$$\mathcal{H} : (\mathbf{f}(\mathbf{x}) = \mathbf{0},\ \mathbf{x} \in [\mathbf{x}]),$$

где $n_{\mathbf{f}} = n_{\mathbf{x}}$ и разрешающий оператор с неподвижной точкой задается как $\psi(\mathbf{x}) = \mathbf{x} - \mathbf{M}\mathbf{f}(\mathbf{x})$ (см. (4.30)). Если функция $\mathbf{f}(\mathbf{x})$ афинна, например, $\mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$, то разрешающий оператор с неподвижной точкой принимает вид: $\psi(\mathbf{x}) = \mathbf{x} - \mathbf{M}(\mathbf{A}\mathbf{x} + \mathbf{b})$. При $\mathbf{M} = \mathbf{A}^{-1}$ последовательность $\mathbf{x}_{k+1} = \psi(\mathbf{x}_k)$ прямо сходится к решению $\mathbf{x}^* = -\mathbf{A}^{-1}\mathbf{b}$ за один шаг. Теперь, если $\mathbf{f}$ нелинейна, но дифференцируема, ее можно аппроксимировать

первым членом разложения в ряд Тейлора для получения разрешающего оператора с неподвижной точкой в виде $\psi(\mathbf{x}) = \mathbf{x} - \mathbf{J_f}^{-1}(\mathbf{x}) * \mathbf{f}(\mathbf{x})$. Тогда, последовательность (4.27) приводит к методу Ньютона. Функция включения имеет вид:

$$[\psi](\mathbf{x}) = [\mathbf{x}] - [\mathbf{J_f}]^{-1}([\mathbf{x}]) * [\mathbf{f}]([\mathbf{x}]) \tag{4.51}$$

и дает сжимающий оператор Ньютона (см. Теорему 4.2.)

$$\mathcal{C}_{\mathrm{N}} : [\mathbf{x}] \mapsto [\mathbf{x}] \cap ([\mathbf{x}] - [\mathbf{J_f}]^{-1}([\mathbf{x}]) * [\mathbf{f}]([\mathbf{x}])). \tag{4.52}$$

Может, конечно, использоваться и какая-нибудь другая, например, центрированная функция включения. Классически используется более эффективная версия этого сжимающего оператора, как это показывается в параграфе 4.3.3 (с. 117).

4.2.4. Метод вперед-назад

Оператор сжатия $\mathcal{C}_{\downarrow\uparrow}$ на основе метода вперед-назад [Benhamou it et al., 1999; Jaulin, 2000б] основан на двустороннем распространении ограничения [Waltz, 1975; Cleary, 1987; Davis, 1987].

Этот оператор делает возможным сжимать области решений задачи выполнения ограничений $\mathcal{H} : (\mathbf{f}(\mathbf{x}) = \mathbf{0},\ \mathbf{x} \in [\mathbf{x}])$, учитывая по отдельности любое из $n_{\mathbf{f}}$ ограничений, например, $f_i(x_1, \ldots, x_{n_{\mathbf{x}}})$. Заметим, что $n_{\mathbf{f}}$ теперь необязательно равно $n_{\mathbf{x}}$. Следующий пример показывает, как некоторое заданное ограничение может использоваться для сжатия области.

Пример 4.7. *Рассмотрим ограничение* $x_3 = x_1 x_2$ *и параллелотоп* $[\mathbf{x}] = [1, 4] \times [1, 4] \times [8, 40]$. *Это ограничение можно переписать тремя способами*:

$$\begin{aligned} x_1 &= x_3/x_2, \\ x_2 &= x_3/x_1, \\ x_3 &= x_1 x_2. \end{aligned} \tag{4.53}$$

Каждое из этих уравнений получено выделением одной из переменных в исходном ограничении. Таким образом, получаем три конечных разрешающих оператора:

$$\begin{aligned} &\phi_1(\text{вход: } x_2, x_3;\ \text{выход: } x_1)\ \{x_1 := x_3/x_2 \text{ если } x_2^0 \neq 0, \\ &\qquad\qquad \mathbb{R} \text{ в противном случае}\}, \\ &\phi_2(\text{вход: } x_1, x_3;\ \text{выход: } x_2)\ \{x_2 := x_3/x_1 \text{ если } x_1^0 \neq 0, \\ &\qquad\qquad \mathbb{R} \text{ в противном случае}\}, \\ &\phi_3(\text{вход: } x_1, x_2;\ \text{выход: } x_3)\ \{x_3 := x_1 x_2\}. \end{aligned} \tag{4.54}$$

Интервализация этих конечных разрешающих операторов приводит к следующим сжатиям:

$$\begin{aligned}([x_3]/[x_2]) \cap [x_1] &= \frac{[8, 40]}{[1, 4]} \cap [1, 4] = [2, 4],\\ ([x_3]/[x_1]) \cap [x_2] &= [2, 4],\\ ([x_1] * [x_2]) \cap [x_3] &= ([1, 4] * [1, 4]) \cap [8, 40] = [8, 16].\end{aligned} \tag{4.55}$$

Следовательно, новая область равна $[2, 4] \times [2, 4] \times [8, 16]$. ■

Предположим, что каждое ограничение имеет вид $f_i(x_1, \ldots, x_{n_x}) = 0$, где f_i может быть разложено в последовательность операций, включая элементарные операции и функции (такие как $+, -, *, /, \sin, \cos \ldots$). При этом возможно разложить данное ограничение на *элементарные ограничения* [Lhomme, 1993]. Грубо говоря, элементарное ограничение — это ограничение, включающее в себя единственный оператор (такой как $+, -, *,$ или, $/$) или простую функцию (такую как sin, cos или exp). Например, ограничение $x_1 \exp(x_2) + \sin(x_3) = 0$ можно разложить в следующий набор элементарных ограничений:

$$\begin{cases} a_1 &= \exp(x_2),\\ a_2 &= x_1 a_1,\\ a_3 &= \sin(x_3),\\ a_2 + a_3 &= 0.\end{cases} \tag{4.56}$$

Области, связанные со всеми промежуточными переменными a_1, a_2 и a_3, определены как $]-\infty, \infty[$. Метод сжатия задачи $\mathcal{H}$ по отношению к ограничению $x_1 \exp(x_2) + \sin(x_3) = 0$ состоит в том, чтобы сжимать каждое элементарное ограничение из (4.56) до тех пор, пока каждый из операторов сжатия станет неэффективным. В этом и заключается принцип *распространения ограничения — метода вперед-назад* [Waltz, 1975], первоначально использовавшийся без помощи интервального анализа.

Метод вперед-назад выделяет элементарные ограничения, которые используются для сжатий, в оптимальном порядке в смысле размера окончательно получаемых областей [Benhamou *et al.*, 1999]. Это иллюстрируется следующим примером.

Пример 4.8. *Рассмотрим уравнение*

$$f(\mathbf{x}) = 0, \tag{4.57}$$

где

$$f(\mathbf{x}) = x_1 \exp(x_2) + \sin(x_3). \tag{4.58}$$

Области значений переменных x_1, x_2 и x_3 обозначим $[x_1], [x_2]$ и $[x_3]$. Чтобы получить алгоритм сжатия этих областей, прежде всего запишем алгоритм, который вычисляет $y = f(\mathbf{x})$ по последовательности элементарных операций, таких, как предлагаются в (4.56),

$$\begin{aligned} a_1 &:= \exp(x_2),\\ a_2 &:= x_1 a_1,\\ a_3 &:= \sin(x_3),\\ y &:= a_2 + a_3.\end{aligned} \tag{4.59}$$

Тогда интервальные аналоги этих алгоритмов записываются как

$$\begin{aligned} 1\ [a_1] &:= \exp([x_2]),\\ 2\ [a_2] &:= [x_1] * [a_1],\\ 3\ [a_3] &:= \sin([x_3]),\\ 4\ [y] &:= [a_2] + [a_3].\end{aligned} \tag{4.60}$$

Поскольку $f(\mathbf{x}) = 0$, то область для y должна быть взята точечной (синглтоном) $\{0\}$. Таким образом, в алгоритме можно добавить еще один шаг:

$$5\ [y] := [y] \cap \{0\}. \tag{4.61}$$

Если интервал $[y]$, рассчитанный на Шаге 5, *оказывается пустым, то мы обнаруживаем, что данная задача не имеет решения. Иначе, $[y]$ замещается на $\{0\}$. В заключение выполняется обратный этап метода вперед-назад, обновляющий области, связанные со всеми переменными, чтобы получить*

$$\begin{aligned} 6\ [a_2] &:= ([y] - [a_3]) \cap [a_2], && \text{см. Шаг 4},\\ 7\ [a_3] &:= ([y] - [a_2]) \cap [a_3], && \text{см. Шаг 4},\\ 8\ [x_3] &:= \sin^{-1}([a_3]) \cap [x_3], && \text{см. Шаг 3},\\ 9\ [a_1] &:= ([a_2]/[x_1]) \cap [a_1], && \text{см. Шаг 2},\\ 10\ [x_1] &:= ([a_2]/[a_1]) \cap [x_1], && \text{см. Шаг 2},\\ 11\ [x_2] &:= \log([a_1]) \cap [x_2], && \text{см. Шаг 1}.\end{aligned} \tag{4.62}$$

На Шаге 8 *операция* $\sin^{-1}([a_3]) \cap [x_3]$ *выдает наименьший интервал, содержащий* $\{x_3 \in [x_3] \mid \sin(x_3) \in [a_3]\}$. *Соответствующий оператор сжатия описывается* в табл. 4.6. ■

Пример построения оператора сжатия на основе метода вперед-назад для функции, определяемой алгоритмом, содержащим циклы, будет рассмотрен в параграфе 6.4.3 (с. 220), в связи с задачей оценивания состояния. Следующий пример иллюстрирует реализацию оператора $\mathcal{C}_{\downarrow\uparrow}$ на линейной задачи выполнения ограничений (см. также Упражнение 11.11, с. 391).

Таблица 4.6. Сжимающий оператор на основе метода вперед-назад

Алгоритм $C_{\downarrow\uparrow}$ (вход/выход : $[\mathbf{x}]$)
1 $[a_1] := \exp([x_2])$;
2 $[a_2] := [x_1] * [a_1]$;
3 $[a_3] := \sin([x_3])$;
4 $[y] := [a_2] + [a_3]$;
5 $[y] := [y] \cap \{0\}$;
6 $[a_2] := ([y] - [a_3]) \cap [a_2]$;
7 $[a_3] := ([y] - [a_2]) \cap [a_3]$;
8 $[x_3] := \sin^{-1}([a_3]) \cap [x_3]$;
9 $[a_1] := ([a_2]/[x_1]) \cap [a_1]$;
10 $[x_1] := ([a_2]/[a_1]) \cap [x_1]$;
11 $[x_2] := \log([a_1]) \cap [x_2]$.

Пример 4.9. *Рассмотрим задачу выполнения ограничений*

$$\mathcal{H}: \begin{pmatrix} x_1 + 2x_2 - x_3 = 0 \\ x_1 - x_2 - x_4 = 0 \\ [\mathbf{x}] \in [-10, 10] \times [10, 10] \times [-1, 1] \times [-1, 1] \end{pmatrix}. \tag{4.63}$$

При использовании оператора $C_{\downarrow\uparrow}$ ограничение $x_1 + 2x_2 - x_3 = 0$ дает

$$[x_1] = [-10, 10], \tag{4.64}$$
$$[x_2] = [-11/2, 11/2], \tag{4.65}$$
$$[x_3] = [-1, 1], \tag{4.66}$$

а ограничение $x_1 - x_2 - x_4 = 0$ дает

$$[x_1] = [-13/2, 13/2], \tag{4.67}$$
$$[x_2] = [-11/2, 11/2], \tag{4.68}$$
$$[x_4] = [-1, 1]. \tag{4.69}$$

Выполняя итерационно эту процедуру, хотелось бы получить последовательность параллелотопов $[\mathbf{x}](k)$, сходящуюся к наименьшей области. К сожалению, это не так. Результаты сжатий для некоторых значений номера k итерации представлены в табл. 4.7.

Таблица 4.7. Итерации при выполнении оператора сжатия на основе метода вперед-назад

k	$[x_1](k)$	$[x_2](k)$	$[x_3](k)$	$[x_4](k)$
0	[−10, 10]	[−10, 10]	[−1, 1]	[−1, 1]
1	[−6,5, 6,5]	[−5,5, 5,5]	[−1, 1]	[−1, 1]
2	[−4,75, 4,75]	[−5,5, 5,5]	[−1, 1]	[−1, 1]
5	[−3,21875, 3,21875]	[−2,21875, 2,21875]	[−1, 1]	[−1, 1]
10	[−3,00684, 3,00684]	[−2,00684, 2,00684]	[−1, 1]	[−1, 1]
∞	[−3, 3]	[−2, 2]	[−1, 1]	[−1, 1]

Оператор $C_{\downarrow\uparrow}$ заходит в тупик. Возможным выходом является выполнение бисекции каждого параллелотопа на два подпараллелотопа:

$$\begin{cases} [\mathbf{x}](1) = [-10, 0] \times [-10, 10] \times [-1, 1] \times [-1, 1], \\ [\mathbf{x}](2) = [0, 10] \times [-10, 10] \times [-1, 1] \times [-1, 1], \end{cases} \tag{4.70}$$

и последовательное применение оператора $C_{\downarrow\uparrow}$ к каждому из этих подпараллелотопов. Для $[\mathbf{x}](1)$ получаем

k	$[x_1](k)$	$[x_2](k)$	$[x_3](k)$	$[x_4](k)$
0	[−10, 0]	[−10, 10]	[−1, 1]	[−1, 1]
1	[−1,5, 0]	[−0,5, 1]	[−1, 1]	[−1, 1]
∞	[−1,5, 0]	[−0,5, 1]	[−1, 1]	[−1, 1]

, (4.71)

а для $[\mathbf{x}](2)$ получаем

k	$[x_1](k)$	$[x_2](k)$	$[x_3](k)$	$[x_4](k)$
0	[0, 10]	[−10, 10]	[−1, 1]	[−1, 1]
1	[0, 1,5]	[−1, 0, 5]	[−1, 1]	[−1, 1]
∞	[0, 0,5]	[−1, 0,5]	[−1, 1]	[−1, 1]

. (4.72)

Объединение (слияние) областей, полученных для $[\mathbf{x}](1)]$ и $[\mathbf{x}](2)]$, дает область

$$[x] = [-1{,}5, 1{,}5] \times [-1, 1] \times [-1, 1] \times [-1, 1], \tag{4.73}$$

которая соответствует оптимальному сжатию задачи (4.63) [Jaulin, 2000б], т. е. $[\mathbf{x}]$ *является интервальной оболочкой множества решения.* ■

ЗАМЕЧАНИЕ 4.4. В задаче (4.32) оператор $\mathcal{C}_{\downarrow\uparrow}$, примененный к каждому уравнению системы $\mathbf{Ap} = \mathbf{b}$, приводит к такому же сжатию параллелепипеда $[\mathbf{p}]$, что и оператор $\mathcal{C}_{\mathrm{GS}}$, но перед оператором $\mathcal{C}_{\mathrm{GS}}$ он имеет то преимущество, что обеспечивает также сжатия для $[\mathbf{A}]$ и $[\mathbf{b}]$. ■

4.2.5. Подход на основе линейного программирования

Рассмотрим снова задачу сжатия областей решения задачи выполнения ограничений

$$\mathcal{H} : (\mathbf{Ap} - \mathbf{b} = 0,\ \mathbf{A} \in [\mathbf{A}],\ \mathbf{b} \in [\mathbf{b}],\ \mathbf{p} \in [\mathbf{p}]), \tag{4.74}$$

где теперь матрица $\mathbf{A}$ необязательно квадратная. Известно, что алгоритм определения наименьшего параллелотопа $[\mathbf{p}]$, который содержит все векторы $\mathbf{p}$, совместные с задачей (4.74), является полиномиально-сложным (NP-сложным) (см. Введение 4.1 и [Rohn, 1994]). Мы рассмотрим два специальных случая, в которых эта задача может быть сведена к $2n_{\mathbf{p}}$ задачам линейного программирования, так, что можно применить эффективный аппарат линейного программирования. Результирующий оператор сжатия будем обозначать $\mathcal{C}_{\mathrm{LP}}$.

Случай 1: Предположим, что *все компоненты* $\mathbf{p}$ *положительны* ($\mathbf{p} > \mathbf{0}$). Вектор $\mathbf{p} \in [\mathbf{p}]$ совместен задачей (4.74) тогда и только тогда, когда

$$\begin{aligned} &\exists \mathbf{A} \in [\mathbf{A}],\ \exists \mathbf{b} \in [\mathbf{b}] \mid \mathbf{Ap} - \mathbf{b} = \mathbf{0} \\ \Leftrightarrow &\exists \mathbf{A} \in [\mathbf{A}] \mid \mathbf{Ap} \in [\mathbf{b}] \\ \Leftrightarrow &\exists \mathbf{A} \in [\mathbf{A}] \mid \mathbf{Ap} \geqslant \underline{\mathbf{b}} \text{ и } \mathbf{Ap} \leqslant \overline{\mathbf{b}}. \end{aligned} \tag{4.75}$$

Теперь, поскольку $\mathbf{p} > \mathbf{0}$, имеют место следующие эквивалентные соотношения

$$(\exists \mathbf{A} \in [\mathbf{A}] \mid \mathbf{Ap} \geqslant \underline{\mathbf{b}}) \Leftrightarrow \overline{\mathbf{A}}\mathbf{p} \geqslant \underline{\mathbf{b}}, \tag{4.76}$$

$$(\exists \mathbf{A} \in [\mathbf{A}] \mid \mathbf{Ap} \leqslant \overline{\mathbf{b}}) \Leftrightarrow \underline{\mathbf{A}}\mathbf{p} \leqslant \overline{\mathbf{b}}. \tag{4.77}$$

ЗАМЕЧАНИЕ 4.5. Если хотя бы одна из компонент вектора $\mathbf{p}$ неположительна, то соотношения эквивалентности (4.76),(4.77) не имеют места. Например, положим, что $\mathbf{A}$ — скаляр и является вещественным числом, и вычислим (4.76) при $[a] = [-4, -1]$, $\underline{b} = 4$ и $p = -2$. Предположение $\exists a \in [-4, -1] \mid ap \geqslant \underline{b}$ оказывается верным (возьмем $a = -3$), в то время как предположение $\overline{a}p \geqslant \underline{b}$ неверно, так как неверно $-1 * -2 \geqslant 4$. ■

Таким образом, вектор $\mathbf{p} \in [\mathbf{p}]$, где $[p_i] \subset \mathbb{R}^+$, $i = 1, \ldots, n_{\mathbf{p}}$, совместен с задачей (4.74) тогда и только тогда, когда

$$\overline{\mathbf{A}}\mathbf{p} \geqslant \underline{\mathbf{b}} \text{ и } \underline{\mathbf{A}}\mathbf{p} \leqslant \overline{\mathbf{b}}. \tag{4.78}$$

Наименьший параллелотоп $[\mathbf{q}]$, содержащий все векторы $\mathbf{p}$, совместные с задачей (4.74), может, таким образом, быть вычислен решением следующих $2n_{\mathbf{p}}$ задач линейного программирования:

$$\begin{cases} \operatorname{opt} p_i,\ i = 1, \ldots, n_{\mathbf{p}}, \\ \begin{pmatrix} -\overline{\mathbf{A}} \\ \underline{\mathbf{A}} \end{pmatrix} \mathbf{p} \leqslant \begin{pmatrix} -\underline{\mathbf{b}} \\ \overline{\mathbf{b}} \end{pmatrix}, \\ \mathbf{p} \in [\mathbf{p}], \end{cases} \tag{4.79}$$

где opt, соответственно, операции min и max, применяемые для получения $\underline{\mathbf{q}}$ и $\overline{\mathbf{q}}$.

Случай 2: Область $[\mathbf{A}]$ *полагается точечной* (т. е. $\underline{\mathbf{A}} = \overline{\mathbf{A}}$). Вектор $\mathbf{p} \in [\mathbf{p}]$ совместен с задачей (4.74) тогда и только тогда, когда существует $\mathbf{b} \in [\mathbf{b}]$ такое, что $\mathbf{Ap} - \mathbf{b} = \mathbf{0}$, т. е.

$$\exists \mathbf{b} \in [\mathbf{b}] \mid \mathbf{Ap} - \mathbf{b} = 0 \Leftrightarrow \mathbf{Ap} \in [\mathbf{b}] \Leftrightarrow \mathbf{Ap} \geqslant \underline{\mathbf{b}} \quad \text{и} \quad \mathbf{Ap} \leqslant \overline{\mathbf{b}}. \tag{4.80}$$

Наименьший параллелотоп $[\mathbf{q}]$, содержащий все векторы $\mathbf{p}$, совместные с задачей (4.74) $\mathcal{H}$, может, следовательно, быть вычислен решением следующих $2n_{\mathbf{p}}$ задач линейного программирования:

$$\begin{cases} \operatorname{opt} p_i,\ i = 1, \ldots, n_{\mathbf{p}}, \\ \begin{pmatrix} -\mathbf{A} \\ \mathbf{A} \end{pmatrix} \mathbf{p} \leqslant \begin{pmatrix} -\underline{\mathbf{b}} \\ \overline{\mathbf{b}} \end{pmatrix}, \\ \mathbf{p} \in [\mathbf{p}], \end{cases} \tag{4.81}$$

где опять opt, соответственно, операции min и max.

4.3. Аппроксимация сверху

Теперь представим операторы сжатия, использующие основные операторы сжатия из параграфа 4.2 и способные выполнять сжатие на значительно более широких классах задач выполнения ограничений.

4.3.1. Основная идея

Некоторый подвектор $\mathbf{p}$ из $\mathbf{x}$ *совместен* с задачей выполнения ограничений $\mathcal{H}$: $(\mathbf{f}(\mathbf{x}) = \mathbf{0}, \mathbf{x} \in [\mathbf{x}])$, если $\mathbf{p}$ может быть дополнен другим подвектором, чтобы образовать вектор решения $\mathbf{x}$ (т. е. $\mathbf{x} \in \mathbb{S}$). Пусть $\mathcal{H}_1$ и $\mathcal{H}_2$ — две такие задачи, и $\mathbf{x}$ — вектор всех переменных задачи $\mathcal{H}$.

Запишем $\mathcal{H}_1 \Rightarrow \mathcal{H}_2$, если

- все переменные задачи $\mathcal{H}_1$ являются также переменными задачи $\mathcal{H}_2$, и
- если $\mathbf{x}$ является вектором решения задачи $\mathcal{H}_1$, то он совместен и с задачей $\mathcal{H}_2$.

Обозначение $\mathcal{H}_1 \Rightarrow \mathcal{H}_2$ показывает, что задача $\mathcal{H}_2$ выведена из задачи $\mathcal{H}_1$ (например, введением вспомогательных переменных с учетом значений некоторых выражений в ограничениях на задачу $\mathcal{H}_1$). Будем говорить, что задача $\mathcal{H}_2$ является *аппроксимацией сверху* (внешней аппроксимацией) задачи $\mathcal{H}_1$, так как множество решения задачи $\mathcal{H}_2$ гарантированно содержит множество решения задачи $\mathcal{H}_1$.

Пример 4.10. *Рассмотрим три задачи выполнения ограничений*

$$\begin{aligned}
\mathcal{H}_1 &: \begin{pmatrix} 3x_1 - \exp(x_1) = 0 \\ x_1 \in [0,\ 2] \end{pmatrix}, \\
\mathcal{H}_2 &: \begin{pmatrix} 3x_1 - x_2 = 0 \\ x_2 - \exp(x_1) = 0 \\ x_1 \in [0,\ 2],\ x_2 \in [-\infty, +\infty] \end{pmatrix}, \\
\mathcal{H}_3 &: \begin{pmatrix} 3x_1 - \exp(1) - \exp(\xi)(x_1 - 1) = 0 \\ x_1 \in [0,\ 2],\ \xi \in [0, 2] \end{pmatrix}.
\end{aligned} \tag{4.82}$$

Проверим, что $\mathcal{H}_1 \Rightarrow \mathcal{H}_2$ *и* $\mathcal{H}_1 \Rightarrow \mathcal{H}_3$. *Тот факт, что* $\mathcal{H}_1 \Rightarrow \mathcal{H}_3$ *является прямым следствием теоремы о среднем, которая дает, что для любого* $x_1 \in [0,\ 2]$ *существует* $\xi \in [0,\ 2]$, *такое, что* $\exp(x_1) = \exp(1) + \exp(\xi)(x_1 - 1)$. ■

Сжатие областей задачи $\mathcal{H}$ с помощью аппроксимаций сверху состоит из трех шагов:

- нахождение для $\mathcal{H}$ некоторой аппроксимации $\mathcal{H}_1$ сверху, для которой оказываются эффективными некоторые разрешающие операторы;
- сжатие $\mathcal{H}_1$;
- обновление областей решения задачи $\mathcal{H}$.

Проиллюстрируем этот подход прежде всего на двух классических методах, а именно, на методе улучшения обусловленности линейных интервальных систем (в параграфе 4.3.2) и на методе интервальных ньютоновских итераций (ньютоновский сжимающий оператор) в параграфе 4.3.3, перед тем, как в параграфах 4.3.4 и 4.3.5 описать некоторые альтернативные подходы.

4.3.2. Улучшение обусловленности

Рассмотрим задачу выполнения ограничений $\mathcal{H}$: $(\mathbf{Ap} - \mathbf{b} = \mathbf{0},\ \mathbf{A} \in [\mathbf{A}],\ \mathbf{p} \in [\mathbf{p}],\ \mathbf{b} \in [\mathbf{b}])$ и предположим, что матрица $[\mathbf{A}]$ такова, что оператор сжатия Гаусса–Зейделя, описанный в параграфе 4.2.3 (с. 103), и оператор сжатия, основанный на методе исключения Гаусса, описанный в параграфе 4.2.2 (с. 99), оказываются неэффективными. Если $\mathbf{A}_0$ — некоторая обратимая матрица из $[\mathbf{A}]$, то

$$\begin{pmatrix} \mathbf{Ap} - \mathbf{b} = \mathbf{0} \\ \mathbf{p} \in [\mathbf{p}],\ \mathbf{A} \in [\mathbf{A}],\ \mathbf{b} \in [\mathbf{b}] \end{pmatrix} \Rightarrow \begin{pmatrix} 1)\ \mathbf{A}' = \mathbf{A}_0^{-1}\mathbf{A} \\ 2)\ \mathbf{b}' = \mathbf{A}_0^{-1}\mathbf{b} \\ 3)\ \mathbf{A}'\mathbf{p} - \mathbf{b}' = \mathbf{0} \\ 4)\ \mathbf{p} \in [\mathbf{p}],\ \mathbf{A} \in [\mathbf{A}],\ \mathbf{b} \in [\mathbf{b}] \end{pmatrix}. \tag{4.83}$$

Основное сжатие по ограничениям 1 и 2 дает области $[\mathbf{A}']$ и $[\mathbf{b}']$ для $\mathbf{A}'$ и $\mathbf{b}'$. При условии, что ширина $w([\mathbf{A}])$ достаточно мала и $\mathbf{A}_0$ хорошо обусловлена, интервальная матрица $[\mathbf{A}']$ оказывается близкой к тождественной (единичной) матрице. Такие сжимающие операторы, как $\mathcal{C}_{\mathrm{GS}}$ или $\mathcal{C}_{\mathrm{GE}}$, могут оказаться эффективными на ограничении 3. В табл. 4.8 это поясняется оператором сжатия $\mathcal{C}_{\mathrm{GSP}}$ (индекс GSP обозначает, что оператор основан на методе Гаусса–Зейделя с улучшением обусловленности).

ЗАМЕЧАНИЕ 4.6. Если на Шаге 4 оператор $\mathcal{C}_{\mathrm{GE}}$ был бы использован вместо оператора $\mathcal{C}_{\mathrm{GS}}$, то оператор, описываемый в табл. 4.8, превратился бы в оператор $\mathcal{C}_{\mathrm{GEP}}$ (индекс GEP обозначает, что оператор основан на методе исключения Гаусса с улучшением обусловленности). ■

Пример 4.11. *Рассмотрим снова ситуацию* Примера 4.4 (с. 100), *где*

$$[\mathbf{A}] = \begin{pmatrix} [4,\ 5] & [-1,\ 1] & [1{,}5,\ 2{,}5] \\ [-0{,}5,\ 0{,}5] & [-7,\ -5] & [1,\ 2] \\ [-1{,}5,\ -0{,}5] & [-0{,}7,\ -0{,}5] & [2,\ 3] \end{pmatrix} \text{ и } [\mathbf{b}] = \begin{pmatrix} [3,\ 4] \\ [0,\ 2] \\ [3,\ 4] \end{pmatrix}, \tag{4.84}$$

Таблица 4.8. Сжимающий оператор Гаусса – Зейделя с улучшением обусловленности

Алгоритм $\mathcal{C}_{\mathrm{GSP}}$(вход/выход : $[\mathbf{A}]$, $[\mathbf{p}]$, $[\mathbf{b}]$)
1 $\mathbf{A}_0 := \mathrm{mid}([\mathbf{A}])$;
2 $[\mathbf{A}'] := \mathbf{A}_0^{-1}[\mathbf{A}]$;
3 $[\mathbf{b}'] := \mathbf{A}_0^{-1}[\mathbf{b}]$;
4 $\mathcal{C}_{\mathrm{GS}}(\mathbf{A}'\mathbf{p} - \mathbf{b}' = \mathbf{0},\ [\mathbf{A}'],\ [\mathbf{p}],\ [\mathbf{b}'])$;
5 $[\mathbf{b}] := \mathbf{A}_0[\mathbf{b}'] \cap [\mathbf{b}]$;
6 $[\mathbf{A}] := \mathbf{A}_0[\mathbf{A}'] \cap [\mathbf{A}]$.

но, предположим, что

$$[\mathbf{p}] = \begin{pmatrix} [-10,\ 10] \\ [-10,\ 10] \\ [-10,\ 10] \end{pmatrix}. \tag{4.85}$$

На Шаге 2 $[\mathbf{A}'] = (\mathrm{mid}([\mathbf{A}]))^{-1} * \mathbf{A}$ *задается интервальной матрицей*

$$\begin{pmatrix} [0{,}81909,\ 1{,}18092] & [-0{,}21869,\ 0{,}21869] & [-0{,}18092,\ 0{,}18092] \\ [-0{,}14248,\ 0{,}14248] & [-0{,}79556,\ 1{,}20445] & [-0{,}14248,\ 0{,}14248] \\ [-0{,}23659,\ 0{,}23659] & [-0{,}15110,\ 0{,}15110] & [0{,}76342,\ 1{,}23659] \end{pmatrix}, \tag{4.86}$$

а на Шаге 3 *получаем*:

$$[\mathbf{b}'] = (\mathrm{mid}(\mathbf{A}))^{-1} * [\mathbf{b}] = \begin{pmatrix} [-0{,}0755468,\ 0{,}302187] \\ [-0{,}0231942,\ 0{,}437376] \\ [1{,}24056,\ 1{,}74951] \end{pmatrix}. \tag{4.87}$$

Оператор $\mathcal{C}_{\mathrm{GS}}(\mathbf{A}'\mathbf{p} - \mathbf{b}' = \mathbf{0},\ [\mathbf{A}'],\ [\mathbf{p}],\ [\mathbf{b}'])$ *окончательно дает*

$$[\mathbf{p}] = \begin{pmatrix} [-4{,}97088,\ 5{,}24758] \\ [-3{,}611,\ 4{,}13162] \\ [-3{,}4532,\ 7{,}3698] \end{pmatrix}. \tag{4.88}$$

Результаты, приведенные в табл. 4.9, *получены итерационным применением оператора* $\mathcal{C}_{\mathrm{GSP}}$. *Они — лучше результатов, полученных без улучшения обусловленности в* Примере 4.5 (с. 104). *Этот пример рассчитывается в* Упражнении 11.28 (с. 411). ■

Таблица 4.9. Результаты итераций сжимающего оператора Гаусса – Зейделя с улучшением обусловленности

k	$[p_1](k)$	$[p_2](k)$	$[p_3](k)$
0	$[-10,\ 10]$	$[-10,\ 10]$	$[-10,\ 10]$
1	$[-4{,}97088,\ 5{,}24758]$	$[-3{,}61101,\ 4{,}13162]$	$[-3{,}45313,\ 7{,}36981]$
2	$[-2{,}82313,\ 3{,}09983]$	$[-2{,}28883,\ 2{,}80945]$	$[-0{,}818916,\ 4{,}73559]$
5	$[-1{,}26437,\ 1{,}54107]$	$[-0{,}939061,\ 1{,}45968]$	$[0{,}474056,\ 3{,}14881]$
10	$[-1{,}10986,\ 1{,}38656]$	$[-0{,}813738,\ 1{,}33436]$	$[0{,}573876,\ 2{,}98711]$
20	$[-1{,}10698,\ 1{,}38368]$	$[-0{,}811386,\ 1{,}33201]$	$[0{,}575741,\ 2{,}98409]$

Пример 4.12. *Если теперь* $[\mathbf{p}]$ *задается как в* Примере 4.4 *в следующем виде*:

$$[\mathbf{p}] = \begin{pmatrix} [-\infty,\ \infty] \\ [-\infty,\ \infty] \\ [-\infty,\ \infty] \end{pmatrix}, \tag{4.89}$$

то оператор $\mathcal{C}_{\mathrm{GSP}}$ *не сможет сжать* $[\mathbf{p}]$. *С другой стороны, заменой оператора* $\mathcal{C}_{\mathrm{GS}}$ *на оператор* $\mathcal{C}_{\mathrm{GE}}$ *на* Шаге 4 табл. 4.8 *можно получить сжимающий оператор* $\mathcal{C}_{\mathrm{GEP}}$, *который выполнит сжатие* $[\mathbf{p}]$ *за одну итерацию к виду*:

$$[\mathbf{p}] = \begin{pmatrix} [-1{,}10698,\ 1{,}38367] \\ [-0{,}785241,\ 1{,}33201] \\ [0{,}758351,\ 2{,}98409] \end{pmatrix}. \tag{4.90}$$

Этот результат несколько лучше результата, полученного оператором $\mathcal{C}_{\mathrm{GS}}$ *в* Примере 4.4. *Дальнейшего сжатия* $[\mathbf{p}]$ *нельзя получить итерационным применением оператора* $\mathcal{C}_{\mathrm{GE}}$. *В этом примере результаты, полученные оператором* $\mathcal{C}_{\mathrm{GEP}}$ *за одну итерацию, лучше, чем результаты, полученные оператором* $\mathcal{C}_{\mathrm{GSP}}$ *за* 20 *итераций* (см. табл. 4.9). *С другой стороны, во многих примерах оператор* $\mathcal{C}_{\mathrm{GEP}}$ *неспособен сжимать* $[\mathbf{p}]$, *в то время как оператор* $\mathcal{C}_{\mathrm{GSP}}$ *оказывается эффективным.* ■

4.3.3. Ньютоновский сжимающий оператор

Этот сжимающий оператор применяется к задаче выполнения ограничений

$$\mathcal{H} : (\mathbf{f}(\mathbf{x}) = \mathbf{0},\ \mathbf{x} \in [\mathbf{x}]), \tag{4.91}$$

при $n_{\mathbf{f}} = n_{\mathbf{x}}$ [Moore, 1979; Hansen, 1992a]. Он не является прямой интервализацией метода Ньютона с неподвижной точкой, но может быть ин-

терпретирован как улучшенная версия сжимающего оператора, описанного в параграфе 4.2.3. Он оказывается очень эффективным, когда $\mathbf{f}$ является гладкой на областях задачи (4.91) и когда эти области малы.

Пусть $\mathbf{x}_0$ — произвольный вектор из $[\mathbf{x}]$ (например, его центр), тогда для i-й компоненты функции $\mathbf{f}$ по теореме о среднем существует некоторое $\xi_i \in [\mathbf{x}]$, такое, что

$$f_i(\mathbf{x}_0) + \mathbf{J}_{f_i}(\xi_i)(\mathbf{x} - \mathbf{x}_0) = 0, \quad i = 1, \ldots, n_{\mathbf{f}}.$$

Несколько упрощая обозначения, можно написать

$$\begin{pmatrix} \mathbf{f}(\mathbf{x}) = \mathbf{0} \\ \mathbf{x} \in [\mathbf{x}] \end{pmatrix} \Rightarrow$$
$$\overset{\text{(по теореме о среднем)}}{\Rightarrow} \quad \begin{pmatrix} \mathbf{f}(\mathbf{x}_0) + \mathbf{J}_{\mathbf{f}}(\xi_1, \ldots, \xi_{n_{\mathbf{f}}})(\mathbf{x} - \mathbf{x}_0) = \mathbf{0} \\ \mathbf{x} \in [\mathbf{x}],\ \xi_i \in [\mathbf{x}], \quad i = 1, \ldots, n_{\mathbf{f}} \end{pmatrix} \tag{4.92}$$
$$\Rightarrow \quad \begin{pmatrix} \mathbf{A}\mathbf{p} + \mathbf{f}(\mathbf{x}_0) = \mathbf{0} \\ \mathbf{p} = \mathbf{x} - \mathbf{x}_0 \\ \mathbf{A} = \mathbf{J}_{\mathbf{f}}(\xi_1, \ldots, \xi_{n_{\mathbf{f}}}) \\ \mathbf{x} \in [\mathbf{x}],\ \xi_i \in [\mathbf{x}], \quad i = 1, \ldots, n_{\mathbf{f}} \end{pmatrix}$$

Это дает оператор сжатия, представленный в табл. 4.10 и обозначаемый как $\mathcal{C}_{\mathrm{N}}$ (т. е. Ньютоновский сжимающий оператор).

ЗАМЕЧАНИЕ 4.7. Порядок, в котором ограничения записываются и который был произволен в (4.92), оказывается важным для реализации оператора $\mathcal{C}_{\mathrm{N}}$. ■

ЗАМЕЧАНИЕ 4.8. С геометрической точки зрения, идея этого оператора состоит в заключении графика каждой координатной функции $f_i(\mathbf{x})$, заданной на параллелотопе $[\mathbf{x}]$, между двумя гиперплоскостями и решении связанных интервальных линейных систем. Можно видеть, как линеаризация на участках нелинейности преобразуется в неопределенность. ■

4.3.4. Параллельная линеаризация

В параграфе 4.3.3 число ограничений $n_{\mathbf{f}}$ предполагалось равным размерности $n_{\mathbf{x}}$ вектора x. Предположим теперь, что $n_{\mathbf{f}}$ и $n_{\mathbf{x}}$ могут быть различными. Описываемый ниже оператор сжатия обрабатывает все ограничения одновременно, используя подход на основе *параллельной линеаризации* [Jaulin, 2001b]. Идея состоит в том, чтобы на области $[\mathbf{x}]$ поместить график каждой координатной функции $f_i(\mathbf{x})$ между двумя параллельными гиперплоскостями (в противоположность интервальному ньютоновскому методу,

Таблица 4.10. Ньютоновский сжимающий оператор

Алгоритм $\mathcal{C}_{\mathrm{N}}$(вход : $\mathbf{f}$; (вход/выход : $[\mathbf{p}]$)
1 $\mathbf{x}_0 := \mathrm{mid}([\mathbf{x}])$;
2 $[\mathbf{A}] := [\mathbf{J}_{\mathbf{f}}]([\mathbf{x}])$;
3 $[\mathbf{p}] := [\mathbf{x}] - \mathbf{x}_0$;
4 $\mathcal{C}_{\mathrm{GSP}}(\mathbf{A}\mathbf{p} + \mathbf{f}(\mathbf{x}_0) = \mathbf{0},\ \mathbf{A} \in [\mathbf{A}],\ \mathbf{p} \in [\mathbf{p}])$;
5 $[\mathbf{x}] := [\mathbf{x}] \cap ([\mathbf{p}] + \mathbf{x}_0)$.

где эти гиперплоскости непараллельны). Функция $\mathbf{f}$ «захватывается» на $[\mathbf{x}]$ в вилку в соответствии с выражением

$$\mathbf{A}\mathbf{x} + \underline{\mathbf{b}} \leqslant \mathbf{f}(\mathbf{x}) \leqslant \mathbf{A}\mathbf{x} + \overline{\mathbf{b}}. \tag{4.93}$$

Этот захват в вилку опять может быть найден с использованием теоремы о среднем, с учетом того, что функция $\mathbf{f}$ дифференцируема. Предположим, что $\mathbf{x}_0$ есть некоторая точка из $[\mathbf{x}]$, например, его центр. Тогда существует такое $\boldsymbol{\xi} \in [\mathbf{x}]$, что

$$\begin{aligned} &\mathbf{f}(\mathbf{x}) = \mathbf{f}(\mathbf{x}_0) + \mathbf{J}_{\mathbf{f}}(\boldsymbol{\xi})(\mathbf{x} - \mathbf{x}_0) \\ \Leftrightarrow &\mathbf{f}(\mathbf{x}) = \mathbf{f}(\mathbf{x}_0) + \mathbf{J}_{\mathbf{f}}(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) + \mathbf{J}_{\mathbf{f}}(\boldsymbol{\xi})(\mathbf{x} - \mathbf{x}_0) - \mathbf{J}_{\mathbf{f}}(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) \\ \Leftrightarrow &\mathbf{f}(\mathbf{x}) = \mathbf{J}_{\mathbf{f}}(\mathbf{x}_0)\mathbf{x} + \mathbf{f}(\mathbf{x}_0) - \mathbf{J}_{\mathbf{f}}(\mathbf{x}_0)\mathbf{x}_0 + (\mathbf{J}_{\mathbf{f}}(\boldsymbol{\xi}) - \mathbf{J}_{\mathbf{f}}(\mathbf{x}_0))(\mathbf{x} - \mathbf{x}_0). \end{aligned} \tag{4.94}$$

В эквивалентном виде (4.94) можно записать как $\mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$, при

$$\mathbf{A} = \mathbf{J}_{\mathbf{f}}(\mathbf{x}_0) \tag{4.95}$$

и

$$\mathbf{b} = \mathbf{f}(\mathbf{x}_0) - \mathbf{J}_{\mathbf{f}}(\mathbf{x}_0)\mathbf{x}_0 + (\mathbf{J}_{\mathbf{f}}(\boldsymbol{\xi}) - \mathbf{J}_{\mathbf{f}}(\mathbf{x}_0))(\mathbf{x} - \mathbf{x}_0). \tag{4.96}$$

На основе (4.94) аппроксимация сверху задачи выполнения ограничений $\mathcal{H} : (\mathbf{f}(\mathbf{x}) = \mathbf{0}, \mathbf{x} \in [\mathbf{x}])$ записывается

$$\begin{pmatrix} \mathbf{A}\mathbf{x} + \mathbf{b} = 0 \\ \mathbf{A} = \mathbf{J}_{\mathbf{f}}(\mathbf{x}_0) \\ \mathbf{b} = \mathbf{f}(\mathbf{x}_0) - \mathbf{A}\mathbf{x}_0 + (\mathbf{J}_{\mathbf{f}}(\boldsymbol{\xi}) - \mathbf{A})(\mathbf{x} - \mathbf{x}_0) \\ \mathbf{x} \in [\mathbf{x}],\ \boldsymbol{\xi} \in [\mathbf{x}] \end{pmatrix}. \tag{4.97}$$

Результирующий оператор сжатия на основе метода параллельной линеаризации будем обозначать $\mathcal{C}_{||}$. Реализация оператора $\mathcal{C}_{||}$ показана в табл. 4.11. Оператор $\mathcal{C}_{\mathrm{LP}}$, называемый на Шаге 4 оператором $\mathcal{C}_{||}$, выполняет сжатие области для $\mathbf{x}$ использованием линейного программирования (см. параграф 4.2.5, Случай 2).

Таблица 4.11. Сжимающий оператор на основе параллельной линеаризации

Алгоритм $\mathcal{C}_{\|\|}$(вход : $\mathbf{f}$; (вход/выход : $[\mathbf{p}]$)
1 $\mathbf{x}_0 := \mathrm{mid}([\mathbf{x}])$;
2 $\mathbf{A} := \mathbf{J_f}(\mathbf{x}_0)$;
3 $[\mathbf{b}] := \mathbf{f}(\mathbf{x}_0) - \mathbf{A}\mathbf{x}_0 + ([\mathbf{J_f}]([\mathbf{x}]) - \mathbf{A})([\mathbf{x}] - \mathbf{x}_0)$;
4 $\mathcal{C}_{\mathrm{LP}}(\mathbf{Ax} - \mathbf{b} = \mathbf{0},\ \mathbf{x} \in [\mathbf{x}],\ \mathbf{b} \in [\mathbf{b}])$.

Чтобы количественно оценить качество захвата в линейную вилку (4.93), вычислим отношение $w([\mathbf{b}])/w([\mathbf{x}])$, где параллелотоп $[\mathbf{b}]$ найден на Шаге 3 работы оператора $\mathcal{C}_{||}$. Это отношение вычисляется по соотношению

$$\frac{w([\mathbf{b}])}{w([\mathbf{x}])} = \frac{w(([\mathbf{J_f}]([\mathbf{x}]) - \mathbf{J_f}(\mathbf{x}_0)) * ([\mathbf{x}] - \mathbf{x}_0))}{w([\mathbf{x}])}, \tag{4.98}$$

поэтому величина $w([\mathbf{b}])/w([\mathbf{x}])$ стремится к нулю вместе с величиной $w([\mathbf{x}])$, и это означает, что захват в вилку становится все точнее и точнее, когда $[\mathbf{x}]$ стягивается в точку. Рис. 4.2 иллюстрирует идею параллельной линеаризации $ax + |b| = x/4 + [1/3,\ 1]$ для функции f на интервале $[-2,\ 2]$.

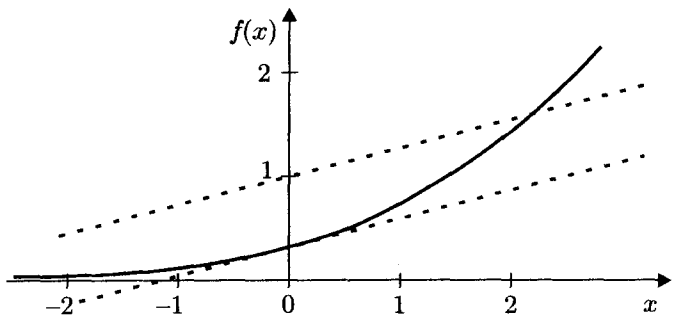


Рис. 4.2. Параллельная линеаризация в скалярном случае

4.3.5. Использование формальных преобразований

Алгебраические действия над заданными ограничениями делают возможным построить задачу выполнения ограничений $\mathcal{H}_1$, которая является аппроксимацией сверху задачи $\mathcal{H}$: $(\mathbf{f}(\mathbf{x}) = \mathbf{0}, \mathbf{x} \in [\mathbf{x}])$, т. е. такую, что $\mathcal{H} \Rightarrow \mathcal{H}_1$. За исключением полиномиального случая [Marti и Rueher, 1995; Benhamou и Granvillers, 1997], представляется, что не создан общий

формальный метод, который бы строил такие аппроксимации сверху, но есть богатая практика решения задачи $\mathcal{H}_1$ таким образом, что она включает несколько переменных для облегчения работы применением бисекции, если обнаруживается, что она необходима. Эта идея иллюстрируется следующим примером.

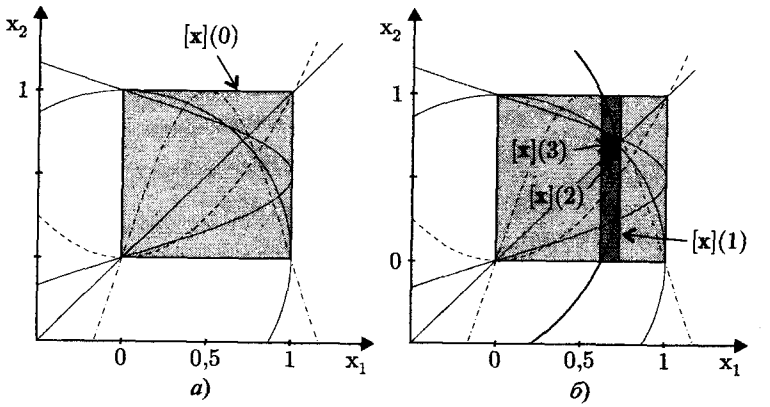


Рис. 4.3. Задача Примера 4.13 имеет пустое множество решения: *а*) операторы сжатия, такие как $\mathcal{C}_{||}$, $\mathcal{C}_{\mathrm{N}}$ или $\mathcal{C}_{\downarrow\uparrow}$ не могут сжать $[\mathbf{x}](0)$; *б*) формальное преобразование дает возможность обойти тупик

Пример 4.13. *Рассмотрим задачу выполнения ограничений*

$$\mathcal{H}: \begin{pmatrix} x_1 - x_2 = 0 \\ x_1^2 + x_2^2 - 1 = 0 \\ x_2 - \sin(\pi x_1) = 0 \\ x_1 - \sin(\pi x_2) = 0 \\ x_2 - x_1^2 = 0 \\ x_1 \in [0,\ 1],\ x_2 \in [0,\ 1] \end{pmatrix}. \tag{4.99}$$

Операторы сжатия, основанные на линеаризации, такие как $\mathcal{C}_{||}$ *или* $\mathcal{C}_{\mathrm{N}}$, *неэффективны для решения задачи* (4.99), *потому что области — слишком велики. Оператор* $\mathcal{C}_{\downarrow\uparrow}$ *также неэффективен, потому что каждое из пяти ограничений в* (4.99) *совместно с параллелотопом* $[\mathbf{x}](0) = [0,1] \times [0,1]$ (см. рис. 4.3, *а*). *Теперь, суммируя первые два ограничения, можно получить новое ограничение* $x_1^2 + x_2^2 - 1 + x_1 - x_2 = 0$. *Следова-*

тельно, задача

$$\mathcal{H}_1 : \begin{pmatrix} (x_1 + 1/2)^2 + (x_2 - 1/2)^2 - 3/2 = 0 \\ x_1 \in [0, 1],\ x_2 \in [0, 1] \end{pmatrix} \tag{4.100}$$

удовлетворяет условию $\mathcal{H} \Rightarrow \mathcal{H}_1$. *Метод вперед-назад может быть применен для сжатия* $[x_1]$ *и* $[x_2]$ *в* $\mathcal{H}_1$. *Элементарные ограничения, связанные с задачей* $\mathcal{H}_1$, *имеют вид*:

$$\begin{cases} a_1 = x_1 + 1/2, \\ a_2 = x_2 - 1/2, \\ a_3 = a_1^2, \\ a_4 = a_2^2, \\ a_3 + a_4 = 3/2. \end{cases}$$

Они используются для сжатия областей переменных x_1, x_2, a_1, a_2, a_3 *и* a_4, *как это подробно показывается ниже*:

$$\begin{array}{ll}
a_1 = x_1 + 1/2 \text{ и } x_1 \in [0, 1] & \Rightarrow a_1 \in [1/2, 3/2], \\
a_2 = x_2 - 1/2 \text{ и } x_2 \in [0, 1] & \Rightarrow a_2 \in [-1/2, 1/2], \\
a_3 = a_1^2 \text{ и } a_1 \in [1/2, 3/2] & \Rightarrow a_3 \in [1/4, 9/4], \\
a_4 = a_2^2 \text{ и } a_2 \in [-1/2, 1/2] & \Rightarrow a_4 \in [0, 1/4], \\
a_3 = 3/2 - a_4, a_4 \in [0, 1/4] \text{ и } a_3 \in [1/4, 9/4] & \Rightarrow a_3 \in [5/4, 3/2], \\
a_4 = 3/2 - a_3, a_3 \in [5/4, 3/2] \text{ и } a_4 \in [0, 1/4] & \Rightarrow a_4 \in [0, 1/4], \\
a_2^2 = a_4, a_2 \in [-1/2, 1/2] \text{ и } a_4 \in [0, 1/4] & \Rightarrow a_2 \in [-1/2, 1/2], \\
a_1^2 = a_3, a_1 \in [1/2, 3/2] \text{ и } a_3 \in [5/4, 3/2] & \Rightarrow a_1 \in \left[\sqrt{5/4}, \sqrt{3/2}\right], \\
x_2 = a_2 + 1/2 \text{ и } a_2 \in [-1/2, 1/2] & \Rightarrow x_2 \in [0, 1], \\
x_1 = a_1 - 1/2 \text{ и } a_1 \in \left[\sqrt{5/4}, \sqrt{3/2}\right] & \Rightarrow x_1 \in \left[\sqrt{5/4} - 1/2, \sqrt{3/2} - 1/2\right].
\end{array}$$

Дальнейшее сжатие не может быть выполнено. Результирующий сжатый параллелотоп имеет вид:

$$[\mathbf{x}](1) = \left[\sqrt{5/4} - 1/2, \sqrt{3/2} - 1/2\right] \times [0, 1].$$

Данное сжатие является оптимальным, как это показано параллелотопом $[\mathbf{x}](1)$, *отмеченным серой заливкой на* рис. 4.3, *б. Это имеет место потому, что переменные* x_1 *и* x_2 *появляются только один раз в ограничении задачи* $\mathcal{H}_1$. *Сжатая область* $[\mathbf{x}](1)$ *может быть преобразована обратно в задачу* $\mathcal{H}$ *для того, чтобы обойти упомянутый тупик. Первое ограничение в задаче* $\mathcal{H}$ *делает возможным сжать* $[\mathbf{x}](1)$ *до* $[\mathbf{x}](2)$, *а второе ограничение делает возможным сжать* $[\mathbf{x}](2)$ *до* $[\mathbf{x}](3)$ (см. рис. 4.3, *б*). *Несколько дополнительных итераций необходимы, чтобы выявить пустые интервалы.* ■

4.4. Взаимодействие между сжимающими операторами

4.4.1. Основная идея

Были представлены несколько сжимающих операторов решения задачи выполнения ограничений $\mathcal{H} : (\mathbf{f}(\mathbf{x}) = \mathbf{0},\ \mathbf{x} \in [\mathbf{x}])$. Ни один из них не может быть объявлен более универсальным, чем другие. Важным является то, что они дополняют друг друга. Хорошим является тот сжимающий оператор, который способен сжимать эту задачу, когда другие операторы неэффективны. Среди представленных ранее операторов сжатия, операторы $\mathcal{C}_{\text{GE}}$, $\mathcal{C}_{\text{N}}$, $\mathcal{C}_{\text{GSP}}$ и $\mathcal{C}_{||}$ эффективны только на параллелотопе $[\mathbf{x}]$ малых размеров, в то время как оператор $\mathcal{C}_{\downarrow\uparrow}$ более эффективен на параллелотопе $[\mathbf{x}]$ большого размера. Операторы $\mathcal{C}_{\text{GE}}$, $\mathcal{C}_{\text{GSP}}$ и $\mathcal{C}_{\text{LP}}$ рассматривают ограничения только вида $\mathbf{Ap} = \mathbf{b}$, в то время как операторы $\mathcal{C}_{||}$ и $\mathcal{C}_{\downarrow\uparrow}$ работают с нелинейными ограничениями более общего вида.

В этом параграфе мы сделаем так, чтобы на основе имеющихся операторов сжатия, взаимодействующих между собой, построить более эффективный сжимающий оператор, наследующий полезные качества своих предшественников. Любой новый оператор сжатия с желаемыми свойствами может добавляться в результирующий оператор.

Основная идея берется из *распространения интервального ограничения*, т. е. расширения на интервалы идеи *метода вперед-назад*, как это было сделано в [Waltz, 1975], что близко к методу релаксаций. Такое распространение было независимо предложено в [Cleary, 1987] и [Davis, 1987]. Идея состоит в том, чтобы сжимать области переменных данной задачи выполнения ограничений, используя последовательно все имеющиеся операторы сжатия. Порядок, в котором используются эти операторы сжатия, определяет *стратегию* [Montanary и Rossi, 1991].

Дадим сначала некоторые определения [Benhamou и Granvillers, 1997]. Сжимающий оператор $\mathcal{C}$, удовлетворяющий условиям

$$\begin{array}{ll} \forall [\mathbf{x}],\ \mathcal{C}([\mathbf{x}]) \subset [\mathbf{x}], & (\textit{сжатие}) \\ \forall [\mathbf{x}],\ [\mathbf{x}] \cap \mathbb{S} \subset \mathcal{C}([\mathbf{x}]), & (\textit{состоятельность}) \end{array} \tag{4.101}$$

где $\mathbb{S}$ — множество решения задачи $\mathcal{H}$.

Кроме того, сжимающий оператор является *монотонным*, если

$$[\mathbf{x}] \subset [\mathbf{y}] \Rightarrow \mathcal{C}([\mathbf{x}]) \subset \mathcal{C}([\mathbf{y}]). \tag{4.102}$$

Все сжимающие операторы, представленные в этом параграфе, являются монотонными [Granvillers, 1998].

Сжимающий оператор является *идемпотентным*, если суперпозиция операторов

$$\mathcal{C} \circ \mathcal{C}([\mathbf{x}]) \triangleq \mathcal{C}(\mathcal{C}([\mathbf{x}])) = \mathcal{C}([\mathbf{x}]). \tag{4.103}$$

Неподвижной точкой сжимающего оператора $\mathcal{C}$ является параллелотоп $[\mathbf{x}]$, удовлетворяющий условию

$$\mathcal{C}([\mathbf{x}]) = [\mathbf{x}]. \tag{4.104}$$

Если операторы $\mathcal{C}_1$ и $\mathcal{C}_2$ монотонны, то сжимающий оператор $\mathcal{C}_{1,2} \triangleq \mathcal{C}_1 \circ \mathcal{C}_2([\mathbf{x}])$ также монотонный. Однако, если даже операторы $\mathcal{C}_1$ и $\mathcal{C}_2$ идемпотентны, то сжимающий оператор $\mathcal{C}_{1,2}$ может не быть идемпотентным.

Совокупностью операторов сжатия назовем некоторый их набор $\mathcal{L}$, а *стратегией* — последовательность применения операторов, принадлежащих этому набору. Рассмотрим, например, совокупность $\mathcal{L}$, содержащую следующие четыре оператора:

$$\mathcal{L} = \{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \mathcal{C}_4\}.$$

Циклической стратегией, связанной с совокупностью $\mathcal{L}$, будем называть последовательность операторов

$$\mathcal{S} = \{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \mathcal{C}_4, \mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \mathcal{C}_4, \mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \mathcal{C}_4, \ldots\}. \tag{4.105}$$

Стратегия $\mathcal{S}$ является *полной*, если для любого $k \geqslant 1$ и любого сжимающего оператора $\mathcal{C}$ из данной совокупности существует $k_1 \geqslant k$, такое, что любой оператор $\mathcal{C}$ присутствует в ней k раз.

Сжимающий оператор $\mathcal{C}_\infty$, представленный в табл. 4.12, соответствует применению всех операторов совокупности $\mathcal{L}$ в соответствии со стратегией $\mathcal{S}$. Для совокупности сжимающих операторов и полной стратегии просто доказать, что оператор $\mathcal{C}_\infty$ является монотонным и идемпотентным.

Теорема 4.3. *Алгоритм $\mathcal{C}_\infty([\mathbf{x}])$ сходится к наибольшему параллелотопу $[\mathbf{z}]$, содержащемуся в параллелотопе $[\mathbf{x}]$, такому, что для $\forall \mathcal{C}$ из $\mathcal{L}$ имеет место равенство $\mathcal{C}([\mathbf{z}]) = [\mathbf{z}]$, когда все множества (или ограничения) являются замкнутыми.* ■

Таблица 4.12. Оператор, комбинирующий все операторы совокупности $\mathcal{L}$ в соответствии со стратегией $\mathcal{S}$

Алгоритм $\mathcal{C}_\infty$(вход/выход : $[\mathbf{x}]$)
1 $k = 0$; $[\mathbf{x}](0) = [\mathbf{x}]$;
2 повтор
3 $k = k + 1$;
4 выбрать сжимающий оператор $\mathcal{C}$ из совокупности $\mathcal{L}$ в соответствии со стратегией $\mathcal{S}$;
5 $[\mathbf{x}](k) := \mathcal{C}([\mathbf{x}](k-1))$;
6 до тех пор пока $[\mathbf{x}](k)$ не станет неподвижной точкой всех сжимающих операторов из совокупности $\mathcal{L}$;
7 $[\mathbf{x}] = [\mathbf{x}](k)$.

Доказательство. Обозначим через $[\mathbf{x}](k)$ параллелотоп на итерации k. Докажем сначала, что $\mathcal{C}_\infty([\mathbf{x}])$ содержит $[\mathbf{z}]$. Так как $[\mathbf{z}] \subset [\mathbf{x}](0)$ и поскольку все операторы сжатия в данной совокупности — монотонны, то

$$[\mathbf{z}] \subset [\mathbf{x}](k) \Rightarrow \mathcal{C}([\mathbf{z}]) \subset \mathcal{C}([\mathbf{x}](k)) \Rightarrow [\mathbf{z}] \subset [\mathbf{x}](k+1) \tag{4.106}$$

для любого оператора $\mathcal{C}$ из совокупности $\mathcal{L}$.

Докажем теперь, что $\mathcal{C}_\infty([\mathbf{x}]) = [\mathbf{z}]$. Поскольку применяемая стратегия полная, то для всех $\mathcal{C}$ из $\mathcal{L}$ суперпозиция $\mathcal{C}(\mathcal{C}_\infty([\mathbf{x}])) = \mathcal{C}_\infty([\mathbf{x}])$, т. е. $\mathcal{C}_\infty([\mathbf{x}])$ является неподвижной точкой для всех операторов сжатия из $\mathcal{L}$. Следовательно, $[\mathbf{z}] = \mathcal{C}_\infty([\mathbf{x}])$. ■

Этот результат установлен [Montanari и Rossi, 1991] для задач выполнения ограничений с конечными областями (см. также [Arsouze *et al.*, 2000]), где она была расширена на непрерывные области. Теорема показывает, что результат, даваемый оператором $\mathcal{C}_\infty$, не зависит от выбранной стратегии, с учетом того, что она является полной. В нашей реализации выбрана циклическая стратегия, поскольку она проста в реализации, так как не требует заранее составления некоторой динамической структуры, включающей использование указателей. Более эффективные стратегии можно найти в [Montanari и Rossi, 1991]. На практике, цикл в алгоритме табл. 4.12 останавливается, когда величины сжатий представляются достаточно малыми.

Проиллюстрируем теперь алгоритм табл. 4.12 примером, где все ограничения выбраны линейными, чтобы облегчить вычисления вручную. Для простоты, пусть совокупность содержит операторы сжатия, основанные на методе вперед-назад. Чтобы визуально показать сжатия и эффект захода

процесса в тупик, взята задача выполнения ограничений только с двумя переменными.

Пример 4.14. *Рассмотрим следующие две задачи, которые сжимаются оператором $\mathcal{C}_{\downarrow\uparrow}$ по отношению к своим двум группам ограничений.*

$$\mathcal{H}_1 : \begin{pmatrix} x_1 + x_2 = 0 \\ x_1 - 2x_2 = 0 \\ x_1 \in [-10, 10] \\ x_2 \in [-10, 10] \end{pmatrix} \quad \text{и} \quad \mathcal{H}_2 : \begin{pmatrix} x_1 + x_2 = 0 \\ x_1 - x_2 = 0 \\ x_1 \in [-10, 10] \\ x_2 \in [-10, 10] \end{pmatrix}. \tag{4.107}$$

Совокупность операторов содержит два сжимающих оператора, а именно, оператор $\mathcal{C}_{\downarrow\uparrow}$, применяемый к первому уравнению (оператор сжатия $\mathcal{C}_1$), и оператор $\mathcal{C}_{\downarrow\uparrow}$, применяемый ко второму уравнению (оператор сжатия $\mathcal{C}_2$). Выбрана циклическая стратегия. Для задачи $\mathcal{H}_1$ оператор $\mathcal{C}_\infty$ сходится к решению $\mathbf{x} = \mathbf{0}$ (рис. 4.4, а), а оператор $\mathcal{C}_\infty$ не может сжимать задачу $\mathcal{H}_2$. Причиной этого является то, что параллелотоп $[-10, 10] \times [-10, 10]$ совместен с каждым из ограничений независимо. Для визуального представления этого тупика рассмотрим набор ограничений, связанных с задачей $\mathcal{H}_2$:

$$\begin{aligned} \mathbb{E}_1 &= \{(x_1, x_2) \mid x_1 + x_2 = 0\}, \\ \mathbb{E}_2 &= \{(x_1, x_2) \mid x_1 - x_2 = 0\}. \end{aligned} \tag{4.108}$$

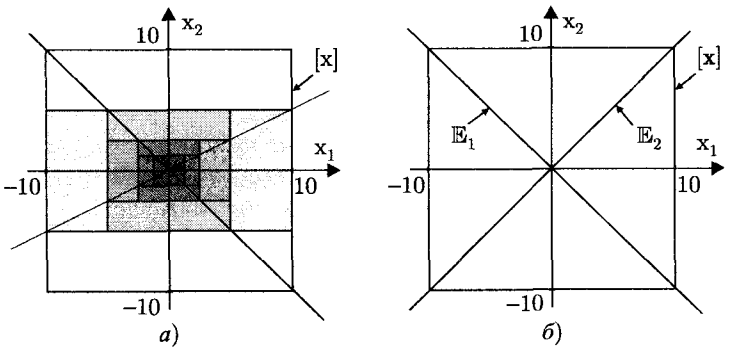


Рис. 4.4. Интерпретация работы оператора на основе метода вперед-назад: *а*) $\mathcal{C}_\infty$ сжимает $[\mathbf{x}]$ до точечного множества (синглтона) $\mathbf{0}$; *б*) сжимающий оператор в тупике, т. е. $\mathcal{C}_1([\mathbf{x}]) = [\mathbf{x}]$ и $\mathcal{C}_2([\mathbf{x}]) = [\mathbf{x}]$

Оператор $\mathcal{C}_1([\mathbf{x}])$ дает наименьший параллелотоп, который содержит параллелотоп $\mathbb{E}_1 \cap [\mathbf{x}]$. Но, поскольку $\mathbb{E}_1$ пересекает все поверхности параллелотопа $[\mathbf{x}]$ (см. рис. 4.4, *б*), *то этим наименьшим параллелотопом является сам параллелотоп $[\mathbf{x}]$ (т.е. $\mathcal{C}_1([\mathbf{x}]) = [\mathbf{x}]$). Подобная ситуация имеет место и с $\mathbb{E}_2$,* т. е. *$\mathcal{C}_2([\mathbf{x}]) = [\mathbf{x}]$. Как показано на* рис. 4.4, *а, тупик не может возникнуть в задаче $\mathcal{H}_1$. Заметим, что сжимающие операторы $\mathcal{C}_1$, $\mathcal{C}_2$ и $\mathcal{C}_\infty$ все являются идемпотентными в задачах $\mathcal{H}_1$ и $\mathcal{H}_2$. Суперпозиция $\mathcal{C}_1 \circ \mathcal{C}_2$ является идемпотентной в задаче $\mathcal{H}_2$, но не идемпотентна в задаче $\mathcal{H}_1$.* ■

ЗАМЕЧАНИЕ 4.9. Пример 4.14 подсказывает три идеи, являющиеся верными при $n_{\mathbf{x}} = 2$, но которые оказываются неверными при $n_{\mathbf{x}} \geqslant 3$. Первая — это то, что если $\mathbb{E}_1$ или $\mathbb{E}_2$ на рис. 4.4, *б* перемещается, то тупик исчезает и, следовательно, прекращение работы оператора $\mathcal{C}_{\downarrow\uparrow}$ нетипично. Вторая мысль — это то, что если оператор $\mathcal{C}_{\downarrow\uparrow}$ не работает в линейном случае, то центр параллелотопа $[\mathbf{x}]$ является решением задачи $\mathcal{H}$ (см. рис. 4.4, *б*). Последняя идея — это то, что если ограничения монотонны по всем переменным с одинаковым типом монотонности, то оператор $\mathcal{C}_{\downarrow\uparrow}$ не зайдет в тупик (см. рис. 4.5). Рисунок 4.6 дает контрпример этим трем идеям, показана ситуация с двумя линейными ограничениями с монотонностью одного типа с пустым множеством решения, в которой оператор $\mathcal{C}_{\downarrow\uparrow}$ не работает. Эти два ограничения представлены двумя параллельными плоскостями. Обе плоскости касаются всех поверхностей параллелотопа $[\mathbf{x}]$, и, следовательно, оператор $\mathcal{C}_{\downarrow\uparrow}$ не может сжать параллелотоп $[\mathbf{x}]$. ■

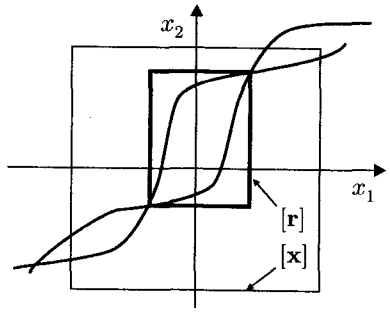


Рис. 4.5. При работе оператора $\mathcal{C}_{\downarrow\uparrow}$ параллелотоп $[\mathbf{x}]$ будет сходиться к параллелотопу $[\mathbf{r}]$, который является наименьшим совместным с двумя ограничениями

4.4.2. Сжимающие операторы и функции включения

Сжимающие операторы могут использовать функции включения. Эти функции необязательно минимальны, поэтому операторы сжатия могут

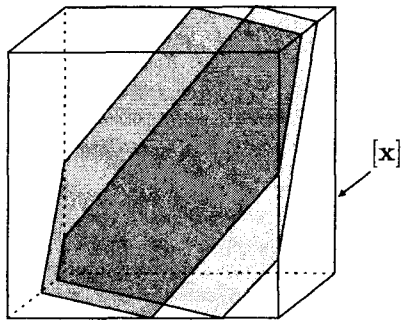


Рис. 4.6. Контрпример трем идеям из Замечания 4.9

быть применены для улучшения точности функций включения. (Вычислительная сложность результирующих операторов сжатия остается полиномиальной.) Эта идея является основной для проверки *совместности параллелотопов*, разработанной в контексте процедур метода вперед-назад [Benhamou et. al., 1999]. Использование сжимающих операторов для улучшения функций включения оказывается особенно полезным при работе с задачами высокой размерности (типичный случай, при размерности более десяти) и когда имеются многократные появления переменных в формальном выражении функции f.

Верхняя граница $\overline{y}$ определяется как

$$\overline{f} = \max_{\mathbf{x}\in[\mathbf{x}]} f(\mathbf{x}), \tag{4.109}$$

где функция $f : \mathbb{R}_n \Rightarrow \mathbb{R}$ может быть рассчитана по алгоритму UUB (UUB — upper upper bound, поиск верха верхней границы), представленного в табл. 4.13. Алгоритм UUB использует оператор $\mathcal{C}$, сжимающий задачу

$$\mathcal{H} : (f(\mathbf{x}) = y,\ \mathbf{x} \in [\mathbf{x}],\ y \in [\underline{y}, \overline{y}]), \tag{4.110}$$

где $[\underline{y}, \overline{y}]$ — интервал, гарантированно содержащий значение $\overline{f}$ и равный первоначально $[f]([\mathbf{x}])$. Сложность вычисления может быть выдержана на уровне полиномиальной, что обеспечивается полиномиальной сложностью оператора $\mathcal{C}$.

Интервал, рассчитанный на Шаге 2 алгоритма UUB, содержит $\overline{f}$, как определяется в (4.109). На Шаге 3 $\underline{y}$ необходимо является нижней границей для $\overline{f}$, но утверждение может быть сделано более эффективным при использовании алгоритма локальной минимизации, например такого, как точечный

Таблица 4.13. Расчет верхней границы для функции включения с сжимающим оператором

Алгоритм UUB(вход : $f, [\mathbf{x}]$; выход : $\overline{y}$])
1 повтор
2 $[\underline{y}, \overline{y}] := [f]([\mathbf{x}])$ // где $[f]$— классическая функция включения;
3 $\underline{y} := f(\mathrm{mid}([\mathbf{x}]))$;
4 $([\mathbf{x}], [y]) := \mathcal{C}(f([\mathbf{x}]) = y,\ \mathbf{x} \in [\mathbf{x}],\ y \in [\underline{y}, \overline{y}])$;
5 до того момента, когда улучшение по $\underline{y}$ и $\overline{y}$ станет достаточно малым.

ньютоновский метод. Оператор сжатия на Шаге 4 выбрасывает части параллелотопа $[\mathbf{x}]$, содержащие те $\mathbf{x}$, которые соответствуют $f(\mathbf{x}) < \underline{y}$.

Пример 4.15. *Найдем верхнюю границу функции* $f(x) = x^2 - x\frac{3}{2}$ *на интервале* $[x] = [0, 4]$. *Для естественной функции включения алгоритм* UUB *на* Шаге 2 *дает*

$$[f]([x]) = [x]^2 - \frac{3}{2}[x] = [0, 16] - \frac{3}{2}[0, 4] = [-6, 16]. \tag{4.111}$$

Плохой результат, полученный вследствие того, что переменная x *дважды появляется в выражении функции* f, *иллюстрируется разложением ограничения* $f(x) = x^2 - x\frac{3}{2}$ *на два элементарных ограничения* $a = x^2$ *и* $y = a - x\frac{3}{2}$. *В пространстве* (x, a), *т. е. в пространстве* (x, x^2), *связь* $y = -x\frac{3}{2} + a$ *соответствует ортогональной проекции точки* (x, a) *на прямую линию с вектором направления* $\mathbf{v} = (-\frac{3}{2}, 1)^{\mathrm{T}}$. *Как показано на* рис. 4.7,*а, интервал* $[f]([x]) = [-6, 16]$ *включает проекции пар* $x = 4$, $x^2 = 0$ *и* $x = 0$, $x^2 = 16$, *которые являются недопустимыми. Поэтому он — значительно больше интервала* $[f]([x]) = [-0{,}5625, 12]$. *Роль алгоритма* UUB *сводится к заключению неизвестной верхней границы* $\overline{f} = 12$ *для* $f([x])$ *в меньший интервал. На* Шаге 3 *алгоритм* UUB *вычисляет* $\underline{y} = f(\mathrm{mid}([x])) = f(2) = 4 - 3 = 1$. *На данном этапе наилучшая имеющаяся оценка для* $\overline{f}$ *— интервал* $[1, 16]$. *На* Шаге 4 алгоритм UUB *вызывает сжимающий оператор для задачи выполнения ограничений*

$$(x^2 - \frac{3}{2}x = y,\ x \in [0, 4],\ y \in [1, 16]). \tag{4.112}$$

Предположим для простоты, что доступен только один сжимающий оператор — $C_{\downarrow\uparrow}$. Сжатие области (4.112) *соответствует следующим операциям*:

$$
\begin{aligned}
&[x] = [0, 4],\\
&[a] = [x]^2 = [0, 16],\\
&[y] = ([a] - {}^3\!/_2[x]) \cap [1, 16] = [1, 16],\\
&[a] = ([y] + {}^3\!/_2[x]) \cap [a] = ([1, 16] + [0, 6]) \cap [0, 16] = [1, 16],\\
&[x] = \sqrt{[a]} \cap [0, 4] = [1, 4].
\end{aligned}
\tag{4.113}
$$

Эта последовательность операций иллюстрируется на рис. 4.7, *б. Параллелотоп* $[1, 4] \times [1, 16]$, *содержащий пару* (x, x^2), *отмечен серой заливкой. На второй итерации цикла улучшенная нижняя граница для* $\overline{f}$ *получается в виде* $\underline{y} = f(\mathrm{mid}([x])) = f(2{,}5) = 2{,}5$. *Наилучшая, известная теперь оценка для* $\overline{f}$ *равна* $[2{,}5, 16]$. Рис. 4.7, *в* и 4.7, *г показывают результаты второй и третьей итераций цикла. Процесс сходится к истинному значению* $\overline{f} = 12$. ■

Таблица 4.14. Алгоритм оценивания функции включения с оператором сжатия IFEC (IFEC — inclusion function evaluation with a contractor)

Алгоритм IFEC(вход : $f, [\mathbf{x}]$; выход : $[\underline{y}, \overline{y}]$)
1 $\underline{y} := -\mathrm{UUB}(-f, [\mathbf{x}])$;
2 $\overline{y} := \mathrm{UUB}(f, [\mathbf{x}])$.

Такой же алгоритм может быть использован для вычисления нижней границы $\underline{y}$ интервала $f([\mathbf{x}])$. Достаточно рассчитать $-\mathrm{UUB}(-f, [\mathbf{x}])$. Функция включения $[f]$ для f может быть, следовательно, получена выполнением алгоритма IFEC, представленного в табл. 4.14. Применение этого алгоритма для оценивания функции включения с использованием оператора сжатия сохраняет полиномиальную сложность вычислений данного оператора.

4.5. Сжимающие операторы на множествах

4.5.1. Определения

Оператор $\mathcal{C}_{\mathbb{S}_\mathbf{p}} := \mathbb{IR}^{n_\mathbf{P}} \to \mathbb{IR}^{n_\mathbf{P}}$ является сжимающим оператором на множестве $\mathbb{S}_\mathbf{p}$ из $\mathbb{IR}^{n_\mathbf{P}}$, если он удовлетворяет условиям:

$$
\forall [\mathbf{p}] \in \mathbb{IR}^{n_\mathbf{P}},\ \begin{cases} \mathcal{C}_{\mathbb{S}_\mathbf{p}}([\mathbf{p}]) \subset [\mathbf{p}] & \text{(сжатие)},\\ \mathcal{C}_{\mathbb{S}_\mathbf{p}}([\mathbf{p}]) \cap \mathbb{S}_\mathbf{p} = [\mathbf{p}] \cap \mathbb{S}_\mathbf{p} & \text{(состоятельность)}, \end{cases}
\tag{4.114}
$$

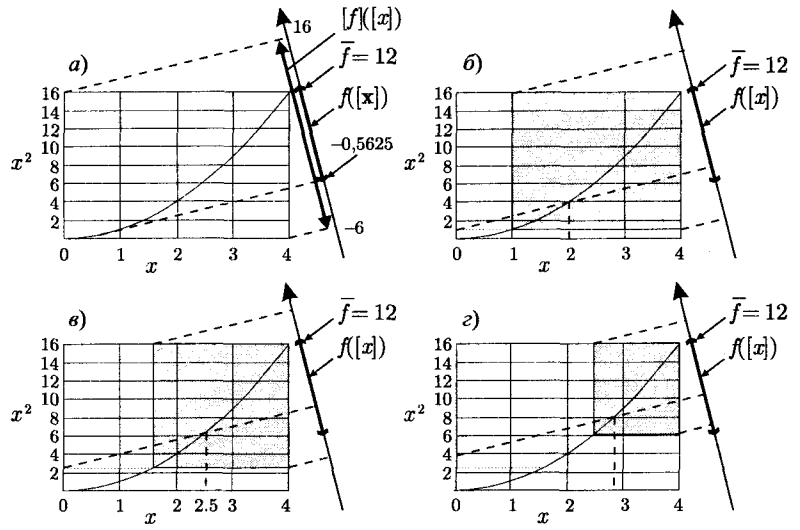


Рис. 4.7. Процедура сжатия при вычислении верхней границы интервала $f([x])$; так как масштабы по двум осям — различны, то направления ортогональных проекций в действительности не соответствуют направлениям, отмеченным штриховыми линиями, которые следует понимать условно

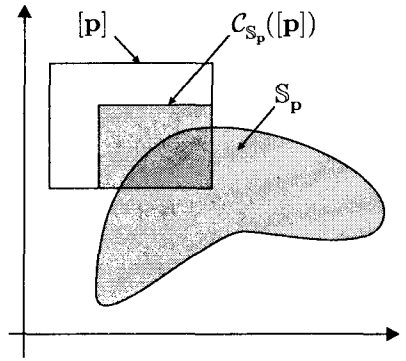


Рис. 4.8. Сжимающий оператор на множестве

как показано на рис. 4.8. В случаях, когда понятия сжимающих операторов на множествах и для задач выполнения ограничений, определяющих эти множества, могут рассматриваться как эквивалентные, в последующих главах понятие сжимающих операторов на множествах будет использоваться предпочтительнее. Они упрощают представление алгоритмов и позволяют избежать использования терминологии таких задач.

Свойства сжимающих операторов на множествах представлены в табл. 4.15.

Таблица 4.15. Свойства сжимающих операторов на множествах

$\mathcal{C}_{\mathbb{S}_\mathbf{p}}$ монотонен тогда и только тогда, когда	$[\mathbf{p}] \subset [\mathbf{q}] \Rightarrow \mathcal{C}_{\mathbb{S}_\mathbf{p}}([\mathbf{p}]) \subset \mathcal{C}_{\mathbb{S}_\mathbf{p}}([\mathbf{q}])$
$\mathcal{C}_{\mathbb{S}_\mathbf{p}}$ минимален тогда и только тогда, когда	$\forall [\mathbf{p}] \in \mathbb{IR}^{n_\mathbf{p}},\ \mathcal{C}_{\mathbb{S}_\mathbf{p}}([\mathbf{p}]) = [[\mathbf{p}] \cap \mathbb{S}_\mathbf{p}]$
$\mathcal{C}_{\mathbb{S}_\mathbf{p}}$ точен тогда и только тогда, когда	$\forall \mathbf{p} \in \mathbb{R}^{n_\mathbf{p}},\ \mathcal{C}_{\mathbb{S}_\mathbf{p}}(\mathbf{p}) = \{\mathbf{p}\} \cap \mathbb{S}_\mathbf{p}$
$\mathcal{C}_{\mathbb{S}_\mathbf{p}}$ идемпотентен тогда и только тогда, когда	$\forall [\mathbf{p}] \in \mathbb{IR}^{n_\mathbf{p}}$, $\mathcal{C}_{\mathbb{S}_\mathbf{p}}(\mathcal{C}_{\mathbb{S}_\mathbf{p}}([\mathbf{p}])) = \mathcal{C}_{\mathbb{S}_\mathbf{p}}([\mathbf{p}])$
$\mathcal{C}_{\mathbb{S}_\mathbf{p}}$ сжимает сильнее, чем оператор $\mathcal{C}'_{\mathbb{S}_\mathbf{p}}$ тогда и только тогда, когда	$\forall [\mathbf{p}] \in \mathbb{IR}^{n_\mathbf{p}}, \mathcal{C}_{\mathbb{S}_\mathbf{p}}([\mathbf{p}]) \subset \mathcal{C}'_{\mathbb{S}_\mathbf{p}}([\mathbf{p}])$

Пусть $\mathcal{C}_{\mathbb{S}^1_\mathbf{p}}$ и $\mathcal{C}_{\mathbb{S}^2_\mathbf{p}}$ — два монотонных сжимающих оператора на множествах $\mathbb{S}^1_\mathbf{p}$ и $\mathbb{S}^2_\mathbf{p}$, и определим действия с ними как

$$\mathcal{C}_{\mathbb{S}^1_\mathbf{p}} \cap \mathcal{C}_{\mathbb{S}^2_\mathbf{p}}([\mathbf{p}]) \triangleq \mathcal{C}_{\mathbb{S}^1_\mathbf{p}}([\mathbf{p}]) \cap \mathcal{C}_{\mathbb{S}^2_\mathbf{p}}([\mathbf{p}]), \tag{4.115}$$

$$\mathcal{C}_{\mathbb{S}^1_\mathbf{p}} \sqcup \mathcal{C}_{\mathbb{S}^2_\mathbf{p}}([\mathbf{p}]) \triangleq \mathcal{C}_{\mathbb{S}^1_\mathbf{p}}([\mathbf{p}]) \sqcup \mathcal{C}_{\mathbb{S}^2_\mathbf{p}}([\mathbf{p}]). \tag{4.116}$$

Тривиально показать, что верны следующие свойства:

1) из $\mathbb{S}^1_\mathbf{p} \subset \mathbb{S}^2_\mathbf{p}$ следует, что оператор $\mathcal{C}_{\mathbb{S}^2_\mathbf{p}}$, является также сжимающим оператором на множестве $\mathbb{S}^1_\mathbf{p}$,
2) оператор $\mathcal{C}_{\mathbb{S}^1_\mathbf{p}} \cap \mathcal{C}_{\mathbb{S}^2_\mathbf{p}}$ является сжимающим на множестве $\mathbb{S}^1_\mathbf{p} \cap \mathbb{S}^2_\mathbf{p}$,
3) оператор $\mathcal{C}_{\mathbb{S}^1_\mathbf{p}} \sqcup \mathcal{C}_{\mathbb{S}^2_\mathbf{p}}$ является сжимающим на множестве $\mathbb{S}^1_\mathbf{p} \cap \mathbb{S}^2_\mathbf{p}$. (4.117)

Свойство 1 будет использоваться для сжатия областей в задачах оптимизации, свойство 3 оказывается полезным для разработки сжимающих операторов в задачах, содержащих операцию дизъюнкции ограничений (т. е. использующих булевский оператор «ИЛИ»).

Пример 4.16. *Рассмотрим операцию проверки включения* $[t_{\mathbb{S}_\mathbf{p}}]$ *для множества* $\mathbb{S}_\mathbf{p}$. *Сжимающий оператор* $\mathcal{C}_{\mathbb{S}_\mathbf{p}}$ *на множестве* $\mathbb{S}_\mathbf{p}$ *задается как*

$$\begin{aligned} &\mathcal{C}_{\mathbb{S}_\mathbf{p}}([\mathbf{p}]) = \varnothing, \text{ если } [t_{\mathbb{S}_\mathbf{p}}]([\mathbf{p}]) = 0, \\ &\mathcal{C}_{\mathbb{S}_\mathbf{p}}([\mathbf{p}]) = [\mathbf{p}], \text{ в противном случае.} \end{aligned} \tag{4.118}$$

Этот сжимающий оператор является точным тогда и только тогда, когда проверка $[t_{\mathbb{S}_\mathbf{p}}]$ *является точной.* ■

4.5.2. Множества, определяемые ограничениями в виде равенств и неравенств

Сжимающий оператор $\mathcal{C}_\infty$, построенный в параграфе 4.4 для задачи выполнения ограничений вида $\mathcal{H} : (\mathbf{f}(\mathbf{x}) = \mathbf{0}, \mathbf{x} \in [\mathbf{x}])$, может быть использован для построения эффективных сжимающих операторов на более общих классах множеств. Рассмотрим, например, множество $\mathbb{S}_\mathbf{p}$, задаваемое ограничениями в виде равенств и неравенств:

$$\mathbb{S}_\mathbf{p} = \{\mathbf{p} \in \mathbb{R}^{n_\mathbf{p}} \mid \mathbf{g}(\mathbf{p}) \leqslant \mathbf{0}, \mathbf{h}(\mathbf{p}) = \mathbf{0}\}, \tag{4.119}$$

где $\mathbf{g}$ и $\mathbf{h}$ — нелинейные векторно-значные функции, и это неравенство понимается в покомпонентном смысле. Положим

$$\mathbf{x} = \begin{pmatrix} \mathbf{p} \\ \mathbf{v} \end{pmatrix},\ [\mathbf{x}] = [\mathbf{p}] \times [\mathbf{v}] \ \text{ и } \ \mathbf{f}(\mathbf{x}) = \begin{pmatrix} \mathbf{g}(\mathbf{p}) + \mathbf{v}) \\ \mathbf{h}(\mathbf{p}) \end{pmatrix}, \tag{4.120}$$

где $\mathbf{v}$ — вектор вспомогательных переменных с областью значений $[\mathbf{v}] = [0, \infty[\times \ldots \times [0, \infty[$. Поскольку

$$\begin{pmatrix} \mathbf{f}(\mathbf{x}) = \mathbf{0} \\ \mathbf{x} \in [\mathbf{x}] \end{pmatrix} \Leftrightarrow \begin{pmatrix} \mathbf{g}(\mathbf{p}) + \mathbf{v} = \mathbf{0} \\ \mathbf{h}(\mathbf{p}) = \mathbf{0} \\ \mathbf{p} \in [\mathbf{p}] \\ \mathbf{v} \in [0, \infty[\times \ldots \times [0, \infty[\end{pmatrix} \Leftrightarrow \begin{pmatrix} \mathbf{g}(\mathbf{p}) \leqslant \mathbf{0} \\ \mathbf{h}(\mathbf{p}) = \mathbf{0} \\ \mathbf{p} \in [\mathbf{p}] \end{pmatrix}, \tag{4.121}$$

то некоторый сжимающий оператор $\mathcal{C}$ для задачи вида $\mathcal{H}$: $(\mathbf{f}(\mathbf{x}) = \mathbf{0}, \mathbf{x} \in [\mathbf{x}])$ может быть применен как сжимающий оператор на множестве $\mathbb{S}_\mathbf{p}$ с использованием алгоритма, описанного в табл. 4.16. Шаг 1 определяет задачу, связанную с множеством $\mathbb{S}_\mathbf{p}$, на Шаге 2 сжимается область рассматриваемого параллелотопа $[\mathbf{x}]$, а на Шаге 3 вычисляется проекция параллелотопа $[\mathbf{x}]$ на $\mathbf{p}$-*пространство*.

Таблица 4.16. Сжимающий оператор на множестве, задаваемом ограничениями в виде равенств и неравенств

Алгоритм $\mathcal{C}_{\mathbb{S}_\mathbf{p}}$ (вход : $\mathbf{g}, \mathbf{h}$; вход/выход : $[\mathbf{p}]$)
1 $\mathbf{x} := (\mathbf{p}, \mathbf{v})$; $[\mathbf{x}] := [\mathbf{p}] \times [0, \infty[^{\times n_\mathbf{g}}$; $\mathbf{f}(\mathbf{x}) := (\mathbf{g}(\mathbf{p}) + \mathbf{v},\ \mathbf{h}(\mathbf{p}))$;
2 $[\mathbf{x}] := \mathcal{C}([\mathbf{x}])$;
3 $[\mathbf{p}] := \mathrm{proj}_{\mathbb{R}^{n_\mathbf{p}}}[\mathbf{x}]$. // проекция $[\mathbf{x}]$ на $\mathbf{p}$ – пространство

4.5.3. Улучшение операторов сжатия при использовании локального поиска

Рассмотрим множество $\mathbb{S}_\mathbf{p}$ ненулевого размера, сжимающий оператор $\mathcal{C}_{\mathbb{S}_\mathbf{p}}$ на множестве $\mathbb{S}_\mathbf{p}$ и параллелотоп $[\mathbf{p}]$, который подвергается сжатию, как показано на рис. 4.9, *а*. Некоторые допустимые точки отмечены жирными точками, а $[\mathbf{r}]$ обозначает наименьший параллелотоп, их содержащий. Единственными частями параллелотопа $[\mathbf{p}]$, которые могут быть отброшены *при сжатии, являются части, содержащиеся в* $[\mathbf{p}]$ *и вне* $[\mathbf{r}]$. Пусть $[\mathbf{q}]$ — некоторый параллелотоп с поверхностью, общей с параллелотопом $[\mathbf{p}]$ и касающейся параллелотопа $[\mathbf{r}]$. Сжатие $[\mathbf{q}]$ может быть распространено на $[\mathbf{p}]$ как показано на рис. 4.9, *б*. Этот прием можно применить и в операторе $\mathcal{C}_{\mathbb{S}_\mathbf{p}}$, чтобы сделать его более эффективным. Кроме того, тот факт, что $[\mathbf{p}]$ и $[\mathbf{r}]$ почти равны, можно использовать в качестве критерия остановки работы этого сжимающего оператора. Чтобы достичь эффективности, необходимо максимально возможно «раздуть» параллелотоп $[\mathbf{r}]$; (подробности см. в параграфе 5.4 (с. 148)).

4.6. Выводы

В настоящей главе дано важное понятие сжимающего оператора, применяемого для уменьшения размеров искомого параллелотопа без потери решений рассматриваемой задачи. Сжимающие операторы являются основными компонентами методов решения, которые будут рассмотрены в следующей главе. Как показано на рис. 4.10, методы решения используют

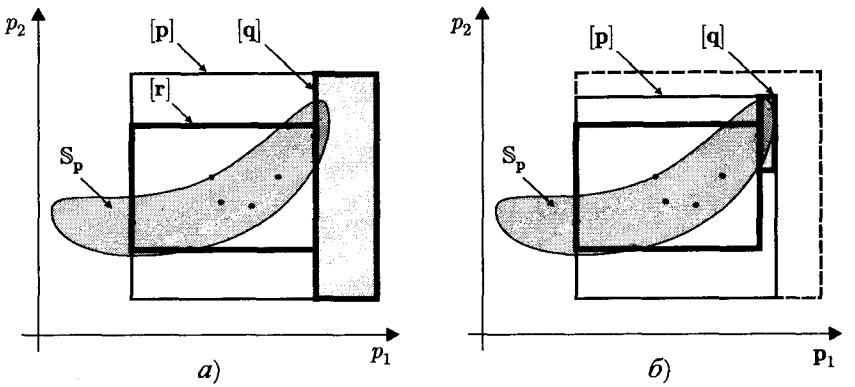


Рис. 4.9. Допустимые точки делают возможным повысить эффективность сжимающих операторов

функции включения и интервальные вычисления только через сжимающие операторы. Заметим, что, хотя многие сжимающие операторы основаны на функциях включения, некоторые сжимающие операторы используют технические приемы других типов. Например, это имеет место в том случае, когда сжимающие операторы строятся в связи с распространением ограничения *на непрерывных областях*.

Понятие сжимающего оператора обобщает понятие проверки включения, введенного в Главе 2, в том смысле, что проверка включения, применяемая к параллелотопу $[\mathbf{x}]$, может рассматриваться как специальный сжимающий оператор, который дает на выходе либо само множество $[\mathbf{x}]$, либо пустое множество. Как показано в параграфе 4.4.2, сжимающие операторы могут также быть полезными для улучшения точности функций включения. Это иллюстрируется возвратной стрелкой на блок-схеме, (см. рис. 4.10).

От сжимающих операторов требуется сложность вычислений порядка полиномиальной по времени и по объему, и, таким образом, не позволяется применять бисекцию областей. Как результат, они могут приводить к тупикам, *что показано в Примере 4.14. Бисекция все же будет являться средством* для выхода из таких тупиков. Идея состоит в разбиении параллелотопа $[\mathbf{x}]$ на два подпараллелотопа, и в применении к каждому из них сжимающего оператора. Когда размерность параллелотопа $[\mathbf{x}]$ велика, этот прием будет последним средством, так как бисекция по одному направлению часто сопровождается бисекцией по другим направлениям, и сложность расчетов возрастает экспоненциально. Например, при размерности исходного про-

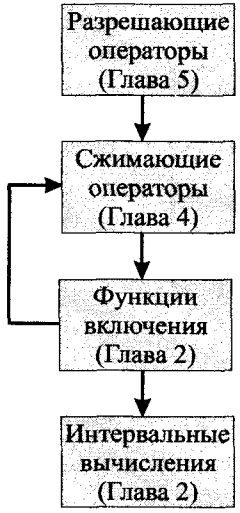


Рис. 4.10. Взаимодействие операторов, функций включения и интервальных вычислений: разрешающие операторы вызывают операторы сжатия, операторы сжатия используют функции включения, последние опираются на интервальные вычисления; точность функций включения может быть улучшена использованием операторов сжатия

стого параллелотопа равной 20, его бисекция по каждому из направлений дает более миллиона подпараллелотопов.

Недавние результаты показывают, что если ввести ограничение на число компонент x, по которым разрешается бисекция, то становится возможным сохранить полиномиальную сложность вычислений сжимающего оператора. Соответствующие методы базируются, например, на 3-B-совместности [Lhomme и Rueher, 1997] или на box-совместности [Benhamou *et. al.*, 1999]. Результат, который выглядит еще более обещающим, использует алгоритм, основанный на (3–2)-совместности [Sam-Haroud, 1995; Lottaz, 2000], который обеспечивает полиномиальную сложность сжимающего оператора.

Это является оптимальным для громадного класса операторов в задачах выполнения ограничений, в том смысле, что он строит наименьший параллелотоп, содержащий множество решений. Оказывается достаточным, что задача выполнения ограничений содержит ограничения, являющиеся в большинстве случаев тернарными (этого можно достичь декомпозицией задачи выполнения ограничений на задачу с элементарными ограничениями), и эти ограничения удовлетворяют некоторым построчно-выпуклым условиям. К сожалению, даже если такой сжимающий оператор является полиномиально сложным, это потребует вычисления покрытий размерности 5, что становится чрезвычайно трудным выполнить на современных компьютерах.

Сжимающие операторы, изученные в настоящей главе, будут важными составляющими методов решения, рассматриваемых в следующей главе.

Глава 5

Разрешающие операторы

5.1. Введение

В Главе 4 были введены сжимающие операторы, которые позволяли заключать в параллелотоп компактное множество $\mathbb{S}$, заданное нелинейными уравнениями или неравенствами. Хотя такой результат и является гарантированным, точность, с которой оценено множество $\mathbb{S}$, неуправляема. С другой стороны, бисекция (т. е. метод дробления интервалов пополам) позволяет регулировать точность, но приводит к экспоненциальной сложности вычислений. Следовательно, в попытке избавиться от проклятья размерностей нужно максимально избегать бисекции в случае, когда число переменных велико. Вот почему, по нашему мнению, при большом числе переменных к бисекции следует прибегать *только как к последнему средству*, когда все доступные операторы сжатия не работают. При этом приходится принимать решение о том, на областях каких переменных надо вводить бисекцию.

Во всех разрешающих операторах, предложенных в настоящей главе, исследуемый параллелотоп разбивается на объединение параллелотопов (*покрытие*). Такое покрытие в общем случае строится самим разрешающим оператором. На каждом параллелотопе покрытия вводятся процедуры операторов сжатия, проверок включения и локальной оптимизации. Все эти процедуры имеют полиномиальную сложность вычислений.

Результаты работы операторов сжатия зависят только от результатов, полученных на каждом из параллелотопов покрытия. Точность работы оператора сжатия регулируется коэффициентами, определяющими, например, ширину ε самых маленьких параллелотопов в данном покрытии, или точностью локализации глобального оптимума. В заданной задаче *результирующее множество* разрешающего оператора — это множество, в котором накоплены (объединены) параллелотопы с шириной меньшей чем ε при ε, стремящемся к нулю. Время вычислений в разрешающем операторе быстро увеличивается с ростом размерности вектора переменных и размера указанного результирующего множества.

Для иллюстрации методики, которой мы придерживаемся для получения эффективных разрешающих операторов, будет рассмотрено несколько

задач. Параграф 5.2 посвящен решению системы нелинейных уравнений, где число уравнений равно числу переменных. Описание множества, ограниченного системой нелинейных неравенств, дается в параграфе 5.3 с помощью двух покрытий снизу и сверху (внутреннего и внешнего). Параграф 5.4 связан с задачей нахождения наименьшего параллелотопа, содержащего множество, определенное системой нелинейных неравенств. В параграфе 5.5 рассматривается минимизация целевой функции при ограничениях, заданных в виде равенств и неравенств. Этот подход в параграфе 5.6 применяется к сложной задаче минимаксной оптимизации. Параграф 5.7 показывает метод описания множеств уровня целевой функции.

5.2. Решение квадратных систем нелинейных уравнений

Рассмотрим n переменных, связанных n уравнениями:

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) = 0, \\ \qquad\vdots \qquad\qquad \vdots \\ f_n(x_1, x_2, \ldots, x_n) = 0, \end{cases} \tag{5.1}$$

или, в эквивалентной векторной форме

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}. \tag{5.2}$$

Задача, которую необходимо решить, состоит в описании множества $\mathbb{S}_\mathbf{x}$ всех векторов $\mathbf{x}$, удовлетворяющих уравнению (5.2) и принадлежащих некоторому (возможно, очень большому) искомому параллелотопу $[\mathbf{x}]$. Рекурсивный алгоритм, представленный в табл. 5.1, вычисляет покрытие, которое содержит множество $\mathbb{S}_\mathbf{x}$. Назовем этот алгоритм SIVIAX, так как пространство поиска соответствует всему вектору $\mathbf{x}$, в то время как в параграфе 5.3 алгоритм SIVIAP будет работать с подвектором $\mathbf{p}$ вектора $\mathbf{x}$. Список $\mathcal{L}$ вначале засылается как пустой, а ε — некоторое малое положительное вещественное число. Объединение всех параллелотопов из списка $\mathcal{L}$, рассчитанное алгоритмом SIVIAX, содержит множество $\mathbb{S}_\mathbf{x}$. Оператор $\mathcal{C}_{\mathbb{S}_\mathbf{x}}$, используемый на Шаге 1, является сжимающим оператором на $\mathbb{S}_\mathbf{x}$, т. е., он удовлетворяет условию $\mathcal{C}_{\mathbb{S}_\mathbf{x}}([\mathbf{x}]) \cap \mathbb{S}_\mathbf{x} = [\mathbf{x}] \cap \mathbb{S}_\mathbf{x}$ (см. параграф 4.5). В Главе 4 было предложено несколько таких сжимающих операторов.

Замечание 5.1. В некоторых приложениях оказывается полезным проверить не существует ли некоторое единственное решение уравнения $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ в заданном

Таблица 5.1. Алгоритм SIVIAX для решения системы нелинейных уравнений

Алгоритм SIVIAX(вход : $[\mathbf{x}], \mathcal{C}_{\mathbb{S}_x}, \varepsilon$; вход/выход : $\mathcal{L}$)
1 $[\mathbf{x}] := \mathcal{C}_{\mathbb{S}_x}([\mathbf{x}])$;
2 если $([\mathbf{x}] = \varnothing)$, то возврат;
3 если $w([\mathbf{x}]) < \varepsilon$, то
4 $\quad \mathcal{L} := \mathcal{L} \cup \{[\mathbf{x}]\}$; возврат;
5 бисекция $[\mathbf{x}]$ на $[\mathbf{x}_1]$ и $[\mathbf{x}_2]$;
6 SIVIAX$([\mathbf{x}_1], \mathcal{C}_{\mathbb{S}_x}, \varepsilon, \mathcal{L})$; SIVIAX$([\mathbf{x}_2], \mathcal{C}_{\mathbb{S}_x}, \varepsilon, \mathcal{L})$.

параллелотопе $[\mathbf{x}]$ из списка $\mathcal{L}$. Если множества $\mathcal{C}_{\mathrm{N}}([\mathbf{x}])$, $\mathcal{C}_{\mathrm{NP}}([\mathbf{x}])$ или $\mathcal{C}_{\mathrm{K}}([\mathbf{x}])$ лежат строго внутри параллелотопа $[\mathbf{x}]$, то существует некоторое единственное решение уравнения $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ в параллелотопе $[\mathbf{x}]$ [Hansen, 1992б]. ■

Чтобы выполнить бисекцию параллелотопа $[\mathbf{x}]$ на два подпараллелотопа на Шаге 5, можно разрезать его по центру перпендикулярно одной из его сторон максимальной длины. Однако, как мы увидим, в случае, если данная задача плохо обусловлена, такая бисекция может оказаться неэффективной, и необходимо будет найти более подходящий критерий для выбора направления, по которому эта бисекция должна выполняться [Ratschеck и Rokne, 1995; Ratz и Csendes, 1995].

Определим i-й *сегмент симметрии* и i-ю (*гипер*) *плоскость симметрии* параллелотопа $[\mathbf{x}]$ следующим образом:

$$\mathrm{segm}_i([\mathbf{x}]) \triangleq m_1 \times \ldots \times m_{i-1} \times [x_i] \times m_{i+1} \times \ldots \times m_n,$$
$$\mathrm{plane}_i([\mathbf{x}]) \triangleq [x_1] \times \ldots \times [x_{i-1}] \times m_i \times [x_{i+1}] \times \ldots \times [x_n],$$

где $m_k = \mathrm{mid}([\mathbf{x}_k])$. Заметим, что $\mathrm{segm}_i([\mathbf{x}])$ и $\mathrm{plane}_i([\mathbf{x}])$ ортогональны. Эти определения иллюстрируются на рис. 5.1, где $n = 3$.

Рисунки 5.2–5.4 показывают, что эффективность бисекции может сильно зависеть от выбора плоскости, вдоль которой эта бисекция выполняется. На этих рисунках функция включения $[\mathbf{f}]$ функции $\mathbf{f}$ является минимальной, но этого не требуется. Рис. 5.2 иллюстрирует ситуацию, когда функция $\mathbf{f}$ плохо обусловлена и где параллелотоп $[\mathbf{x}]$ предполагается достаточно малым, чтобы позволить выполнить линеаризацию поведения функции $\mathbf{f}$ на $[\mathbf{x}]$. Сегменты симметрии параллелотопа $[\mathbf{x}]$, как и их образы, обозначены тонкими линиями. Бисекция вдоль плоскости $\mathrm{plane}_1([\mathbf{x}])$ (рис. 5.3) слабо улучшает описание поведения функции $\mathbf{f}$ на $[\mathbf{x}]$, в противоположность варианту выполнения бисекции вдоль плоскости $\mathrm{plane}_2([\mathbf{x}])$ (рис. 5.4), которая

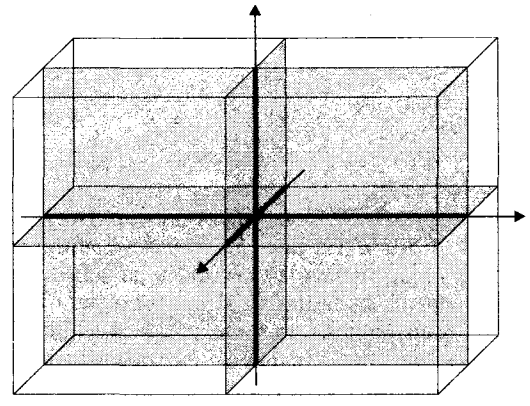


Рис. 5.1. Трехмерный куб, его три плоскости симметрии и три сегмента симметрии

оказывается значительно эффективнее. Представляется весьма естественным выполнять бисекцию параллелотопа $[\mathbf{x}]$ вдоль плоскости симметрии, ортогональной сегменту симметрии, вдоль которого функция $\mathbf{f}$ наиболее чувствительна к изменению аргумента, выбором номера индекса i, по которому имеем максимум

$$\mu_1(i) = \max_{j \in \{1, \ldots, n\}} w(f_i(\mathrm{segm}_i([\mathbf{x}]))). \tag{5.3}$$

Этот критерий может быть аппроксимирован следующим выражением:

$$\mu_2(i) = \max_{j \in \{1, \ldots, n\}} w([x_i]) \left| \frac{\partial f_i}{\partial x_i}(\mathrm{mid}([\mathbf{x}])) \right|, \tag{5.4}$$

которое вычисляется значительно проще. Если *эффективность* бисекции параллелотопа $[\mathbf{x}]$ на параллелотопы $[\mathbf{x}_1]$ и $[\mathbf{x}_2]$ количественно оценивается как

$$\frac{\mathrm{vol}([\mathbf{f}]([\mathbf{x}]))}{\mathrm{vol}([\mathbf{f}]([\mathbf{x}_1]) \cup [\mathbf{f}]([\mathbf{x}_2]))}, \tag{5.5}$$

то бисекция, показанная на рис. 5.3, обладает половинной эффективностью по отношению к бисекции, показанной на рис. 5.4.

Если индекс i направления бисекции выбран, то может оказаться более интересным преобразовать плоскость бисекции усечением вдоль гиперплоскости

$$\begin{aligned}\text{plane}_i(\alpha, [\mathbf{x}]) = &[x_1] \times \ldots \times [x_{i-1}] \\ &\times (\alpha \underline{x}_i + (1-\alpha)\overline{x}_i) \times [x_{i+1}] \times \ldots \times [x_n],\end{aligned} \tag{5.6}$$

где $\alpha \in]0,1[$, причем при $\alpha = 0{,}5$ получаем плоскость симметрии. Это иллюстрируется на рис. 5.5–5.7. На рис. 5.5 тонкими линиями показаны четыре горизонтальных и вертикальных сегмента параллелотопа $[\mathbf{x}]$, а также их образы, полученные по функции $\mathbf{f}$. В соответствии с критерием (5.4) бисекция должна быть выполнена по индексу $i = 2$. При $\alpha = 0{,}5$ получаем ситуацию, показанную на рис. 5.6. Бисекция, представленная на рис. 5.7 и соответствующая значению коэффициента $\alpha = 0{,}2$, приводит к большей эффективности в смысле критерия (5.5). Подходящий выбор величины коэффициента α в общем случае снижает число бисекций, выполняемых алгоритмом SIVIAX, но улучшение может быть значительным только в случае, когда задача была очень плохо обусловлена. Для простоты изложения, в данной книге мы будем рассматривать только бисекции вдоль плоскостей симметрии.

5.3. Описание свойств множеств, определяемых неравенствами

Рассмотрим систему нелинейных неравенств

$$\begin{cases} g_1(p_1, p_2, \ldots, p_{n_\mathbf{p}}) \in [y_1], \\ \quad\vdots \qquad\qquad \vdots \quad \vdots \\ g_{n_\mathbf{g}}(p_1, p_2, \ldots, p_{n_\mathbf{p}}) \in [y_{n_\mathbf{g}}], \end{cases} \tag{5.7}$$

или в векторной форме

$$\mathbf{g}(\mathbf{p}) \in [\mathbf{y}], \tag{5.8}$$

где предполагается, что $\mathbf{p}$ принадлежит априорному множеству поиска $[\mathbf{p}]$. Задача, которую предстоит решить в данном параграфе, состоит в нахождении множества

$$\mathbb{S}_\mathbf{p} \triangleq \{\mathbf{p} \in [\mathbf{p}] \mid \mathbf{g}(\mathbf{p}) \in [\mathbf{y}]\} = \mathbf{g}^{-1}([\mathbf{y}]) \cap [\mathbf{p}]. \tag{5.9}$$

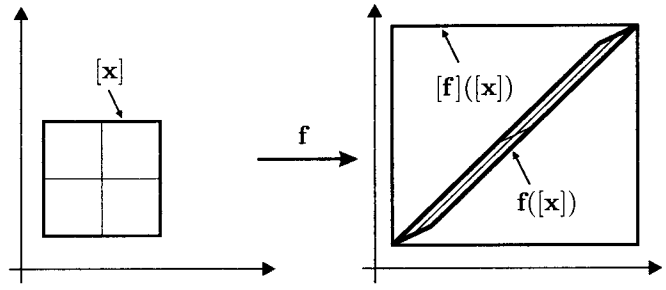


Рис. 5.2. Ситуация, когда функция $\mathbf{f}$ плохо обусловлена

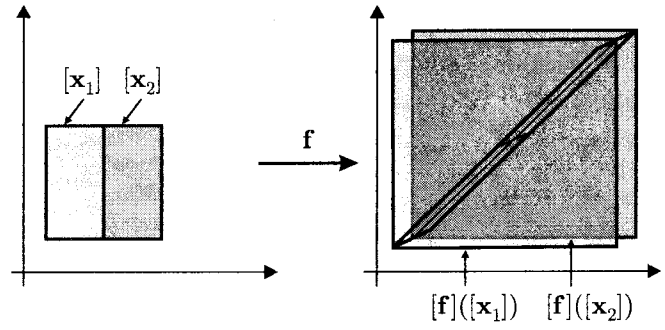


Рис. 5.3. Неэффективная бисекция

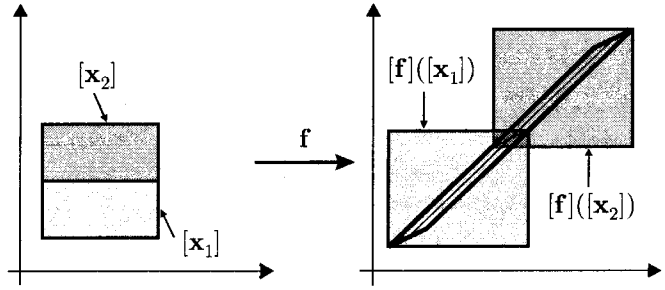


Рис. 5.4. Эффективная бисекция

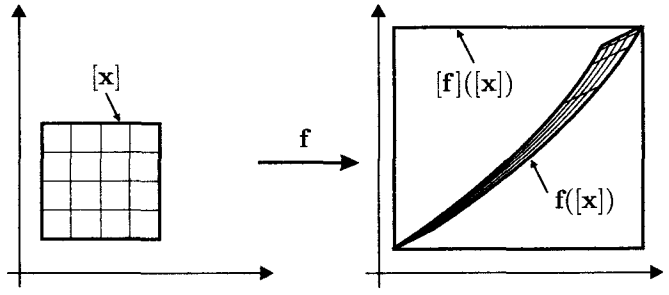


Рис. 5.5. Ситуация, когда функция $\mathbf{f}$ плохо обусловлена

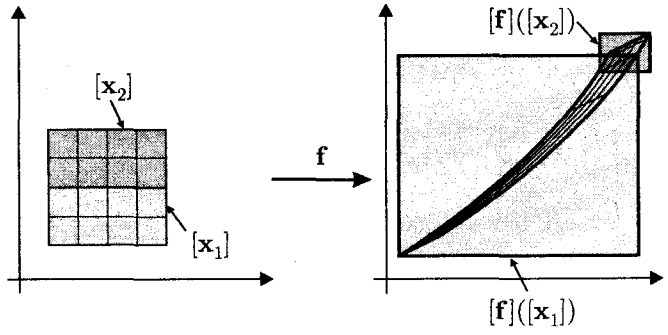


Рис. 5.6. Бисекция по центру ($\alpha = 0{,}5$)

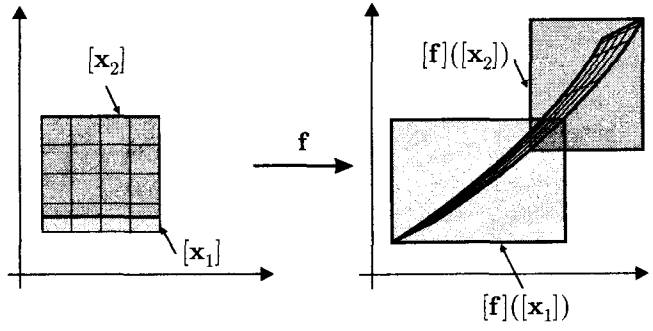


Рис. 5.7. Более эффективная бисекция ($\alpha = 0{,}2$)

Обозначим символом $\mathbf{x}$ вектор, получаемый присоединением $\mathbf{y}$ к $\mathbf{p}$, а символом $[\mathbf{x}]$ — параллелотоп $[\mathbf{p}] \times [\mathbf{y}]$. В некоторых приложениях интересным также оказывается описать множества $\mathbb{S}_\mathbf{x}$ и $\mathbb{S}_\mathbf{y}$, определяемые выражениями

$$\mathbb{S}_\mathbf{x} \triangleq \left\{ \mathbf{x} = \begin{pmatrix} \mathbf{p} \\ \mathbf{y} \end{pmatrix} \mid \mathbf{y} = \mathbf{g}(\mathbf{p}),\ \mathbf{y} \in [\mathbf{y}],\ \mathbf{p} \in [\mathbf{p}] \right\}, \tag{5.10}$$

$$\mathbb{S}_\mathbf{y} \triangleq \{ \mathbf{g}(\mathbf{p}) \mid \mathbf{p} \in [\mathbf{p}],\ \mathbf{g}(\mathbf{p}) \in [\mathbf{y}] \} = \mathbf{g}([\mathbf{p}]) \cap [\mathbf{y}]. \tag{5.11}$$

Заметим, что множество $\mathbb{S}_\mathbf{p}$ является ортогональной проекций множества $\mathbb{S}_\mathbf{x}$ на $\mathbf{p}$-пространство, а множество $\mathbb{S}_\mathbf{y}$ является ортогональной проекций множества $\mathbb{S}_\mathbf{x}$ на $\mathbf{y}$-пространство, как показано на рис. 5.8.

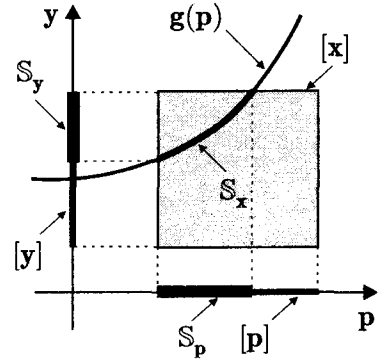


Рис. 5.8. Допустимые множества $\mathbb{S}_\mathbf{x}$, $\mathbb{S}_\mathbf{y}$ и $\mathbb{S}_\mathbf{p}$

Аппроксимации сверху множеств $\mathbb{S}_\mathbf{x}$, $\mathbb{S}_\mathbf{y}$ и $\mathbb{S}_\mathbf{p}$ и аппроксимация снизу (изнутри) множества $\mathbb{S}_\mathbf{p}$ могут быть получены с некоторой произвольной точностью с помощью рекурсивного алгоритма SIVIAPY, представленного в табл. 5.2.

В отличие от алгоритма SIVIAX, описанного в параграфе 5.2, где рассматривался единственный тип параллелотопа, алгоритм SIVIAPY различает $[\mathbf{p}]$ и $[\mathbf{y}]$, чтобы дать возможность описать множества $\mathbb{S}_\mathbf{p}$, $\mathbb{S}_\mathbf{y}$ и $\mathbb{S}_\mathbf{x}$.

Совокупность $\mathcal{L}$ инициализируется как пустой список. После завершения работы алгоритма SIVIAPY, список $\mathcal{L}$ можно записать как

$$\mathcal{L} = \{ [\mathbf{x}_1], [\mathbf{x}_2], \ldots, [\mathbf{x}_{\overline{k}}] \}, \tag{5.12}$$

или, в эквивалентной форме:

$$\mathcal{L} = \{ ([\mathbf{p}_1], [\mathbf{y}_1]), ([\mathbf{p}_2], [\mathbf{y}_2]), \ldots, ([\mathbf{p}_{\overline{k}}], [\mathbf{y}_{\overline{k}}]) \}. \tag{5.13}$$

Таблица 5.2. Алгоритм SIVIAPY

Алгоритм SIVIAPY(вход : $[\mathbf{x}], \mathcal{C}_{\mathbb{S}_\mathbf{x}}, \varepsilon$; вход/выход : $\mathcal{L}$)
1 $[\mathbf{x}] := \mathcal{C}_{\mathbb{S}_\mathbf{x}}([\mathbf{x}])$;
2 если $[\mathbf{x}] = \varnothing$, то возврат;
3 $([\mathbf{p}], [\mathbf{y}]) := [\mathbf{x}]$;
4 если $(w([\mathbf{x}]) < \varepsilon)$, то
5 $\quad \mathcal{L} := \mathcal{L} \cup \{[\mathbf{x}]\}$; возврат;
6 выполняется бисекция $[\mathbf{p}]$ на $[\mathbf{p}_1]$ и $[\mathbf{p}_2]$;
7 $[\mathbf{x}_1] := ([\mathbf{p}_1], [\mathbf{y}])$; $[\mathbf{x}_2] := ([\mathbf{p}_2], [\mathbf{y}])$;
8 SIVIAPY$([\mathbf{x}_1], \mathcal{C}_{\mathbb{S}_\mathbf{x}}, \varepsilon, \mathcal{L})$; SIVIAPY$([\mathbf{x}_2], \mathcal{C}_{\mathbb{S}_\mathbf{x}}, \varepsilon, \mathcal{L})$.

На Шаге 1 оператор $\mathcal{C}_{\mathbb{S}_\mathbf{x}}$ является сжимающим для множества $\mathbb{S}_\mathbf{x}$ (см. параграф 4.5, с. 130). По списку $\mathcal{L}$, полученному алгоритмом SIVIAPY, аппроксимации сверху $\overline{\mathbb{S}}_\mathbf{p}$, $\overline{\mathbb{S}}_\mathbf{y}$ и $\overline{\mathbb{S}}_\mathbf{x}$, соответственно, для множеств $\mathbb{S}_\mathbf{p}$, $\mathbb{S}_\mathbf{y}$ и $\mathbb{S}_\mathbf{x}$ могут быть получены по соотношениям

$$\overline{\mathbb{S}}_\mathbf{p} = \bigcup_{k=1,\ldots,\overline{k}} [\mathbf{p}_k], \quad \overline{\mathbb{S}}_\mathbf{y} = \bigcup_{k=1,\ldots,\overline{k}} [\mathbf{y}_k], \quad \overline{\mathbb{S}}_\mathbf{x} = \bigcup_{k=1,\ldots,\overline{k}} [\mathbf{x}_k], \tag{5.14}$$

а аппроксимация снизу множества $\mathbb{S}_\mathbf{p}$ находится как

$$\underline{\mathbb{S}}_\mathbf{p} = \bigcup_{k=1,\ldots,\overline{k}} \{[\mathbf{p}_k] \mid [\mathbf{g}]([\mathbf{p}_k]) \subset [\mathbf{y}]\}, \tag{5.15}$$

где $[\mathbf{g}]$ является функцией включения для $\mathbf{g}$.

На Шаге 8 для выполнения бисекции параллелотопа $[\mathbf{p}]$ можно рассматривать различные стратегии. Когда задача хорошо обусловлена, то, для простоты, бисекция выполняется вдоль *главной плоскости симметрии* параллелотопа $[\mathbf{p}]$, т. е. вдоль плоскости симметрии, ортогональной одному из ребер максимальной длины. В противном случае, бисекция выполняется перпендикулярно направлению с индексом i, по которому максимальна величина

$$\max_{j \in \{1,\ldots,n_\mathbf{g}\}} w([p_i]) \left| \frac{\partial g_j}{\partial p_i}(\mathrm{mid}([\mathbf{p}])) \right|. \tag{5.16}$$

Покрытие $\overline{\mathbb{S}}_\mathbf{p}$, выработанное алгоритмом SIVIAPY в $\mathbf{p}$-пространстве, накапливает множество $\mathbb{S}_\mathbf{p}$, которое по своей сути имеет размерность, равную $n_\mathbf{p}$. Это означает, что когда размерность вектора $\mathbf{p}$ велика (типично — более четырех) и когда необходима высокая точность описания множества $\mathbb{S}_\mathbf{p}$, никакая ЭВМ не будет в состоянии выполнить алгоритм SIVIAPY за разумное время. Если нужно знать только множество $\mathbb{S}_\mathbf{p}$, а множества $\mathbb{S}_\mathbf{y}$ и $\mathbb{S}_\mathbf{x}$ знать не надо, то размерность накапливающего покрытия может быть снижена до $n_\mathbf{p} - 1$. Будет достаточным запоминать текущий параллелотоп $[\mathbf{p}]$ в $\underline{\mathbb{S}}_\mathbf{p}$, когда выполняется условие $[\mathbf{g}]([\mathbf{p}_k]) \subset [\mathbf{y}]$, и удалять его из списка параллелотопов, которые еще предстоит подвергать бисекции. Соответствующий рекурсивный алгоритм SIVIAP, представленный в табл. 5.3, подобен алгоритму SIVIA из параграфа 3.4.1 (стр. 81). Основное различие состоит в том, что алгоритм SIVIAP использует сжимающие операторы для получения множества $\mathbb{S}_\mathbf{p}$. Два покрытия $\underline{\mathbb{S}}_\mathbf{p}$ и $\overline{\mathbb{S}}_\mathbf{p}$ инициализируются как пустые множества.

На Шаге 1 получения множества $\mathbb{S}_\mathbf{p}$ алгоритм SIVIAP использует сжимающий оператор $\mathcal{C}_{\mathbb{S}_\mathbf{p}}$. Мы будем предполагать, что в качестве $\mathcal{C}_{\mathbb{S}_\mathbf{p}}$ используется оператор

$$\mathcal{C}^0_{\mathbb{S}_\mathbf{p}} := \begin{cases} \mathbb{IR}^{n_\mathbf{p}} \to \mathbb{IR}^{n_\mathbf{p}}, \\ [\mathbf{p}] \mapsto \begin{cases} \varnothing, \text{ если } [\mathbf{g}]([\mathbf{p}]) \cap [\mathbf{y}] = \varnothing, \\ [\mathbf{p}], \quad \text{в противном случае,} \end{cases} \end{cases} \tag{5.17}$$

или какой-нибудь более эффективный сжимающий оператор, который может, например, содержать оператор $\mathcal{C}^0_{\mathbb{S}_\mathbf{p}}$ в своей памяти (см. параграф 4.4, с. 123). Если оператор $\mathcal{C}_{\mathbb{S}_\mathbf{p}}$ сжимает параллелотоп $[\mathbf{p}]$ до пустого множества $\varnothing$, то $[\mathbf{p}]$ исчезает на Шаге 2. Но если на Шаге 3 и Шаге 4 подтверждается, что параллелотоп $[\mathbf{p}]$ находится внутри множества $\mathbb{S}_\mathbf{p}$, то он запоминается в аппроксимациях $\underline{\mathbb{S}}_\mathbf{p}$ и $\overline{\mathbb{S}}_\mathbf{p}$. Шаги 5–8 подобны шагам алгоритма SIVIAX (см. табл. 5.1). После завершения работы алгоритма SIVIAP, имеем

$$\underline{\mathbb{S}}_\mathbf{p} \subset \mathbb{S}_\mathbf{p} \subset \overline{\mathbb{S}}_\mathbf{p}. \tag{5.18}$$

Пример 5.1. *Рассмотрим снова задачу* Примера 3.2 (стр. 84), *состоящую в нахождении множества векторов* $\mathbf{p}$, *удовлетворяющих ограничениям*

$$\begin{cases} \exp(p_1) + \exp(p_2) \in [10,\ 11], \\ \exp(2p_1) + \exp(2p_2) \in [62,\ 72]. \end{cases} \tag{5.19}$$

Для $[\mathbf{p}] = [0,\ 4] \times [0,\ 4]$ $\varepsilon = 0{,}001$ *алгоритм* SIVIAP *вырабатывает покрытие, подобное представленному на* рис. 3.9 (стр. 85) *за* 3,8 *секунд на компьютере* Pentium 133. *С той же величиной* ε, *алгоритм* SIVIA, *как показано в* Главе 3, *затрачивал* 6 *секунд. Выигрыш в вычислительных затратах, даваемый алгоритмом* SIVIA, *возрастает с увеличением размерности* $n_\mathbf{p}$, *но снижается, когда размер множества* $\mathbb{S}_\mathbf{p}$ *увеличивается.* ■

Таблица 5.3. Алгоритм SIVIAP

Алгоритм SIVIAP(вход : $[\mathbf{p}], \mathcal{C}_{\mathbb{S}_\mathbf{p}}, \mathbf{g}, [\mathbf{y}], \varepsilon$; вход/выход : $\underline{\mathbb{S}}_\mathbf{p}, \overline{\mathbb{S}}_\mathbf{p}$)
1 $[\mathbf{p}] := \mathcal{C}_{\mathbb{S}_\mathbf{p}}([\mathbf{p}])$;
2 если $([\mathbf{p}] = \varnothing)$, то возврат;
3 если $[\mathbf{g}]([\mathbf{p}]) \subset [\mathbf{y}]$
4 $\quad \underline{\mathbb{S}}_\mathbf{p} := \underline{\mathbb{S}}_\mathbf{p} \cup [\mathbf{p}]$; $\overline{\mathbb{S}}_\mathbf{p} := \overline{\mathbb{S}}_\mathbf{p} \cup [\mathbf{p}]$; возврат;
5 если $(w([\mathbf{p}]) < \varepsilon)$, то
6 $\quad \overline{\mathbb{S}}_\mathbf{p} := \overline{\mathbb{S}}_\mathbf{p} \cup [\mathbf{p}]$; возврат;
7 выполняется бисекция $[\mathbf{p}]$ на $[\mathbf{p}_1]$ и $[\mathbf{p}_2]$;
8 SIVIAP$([\mathbf{p}_1], \mathcal{C}_{\mathbb{S}_\mathbf{p}}, \mathbf{g}, [\mathbf{y}], \varepsilon, \underline{\mathbb{S}}_\mathbf{p}, \overline{\mathbb{S}}_\mathbf{p})$; SIVIAP$([\mathbf{p}_2], \mathcal{C}_{\mathbb{S}_\mathbf{p}}, \mathbf{g}, [\mathbf{y}], \varepsilon, \underline{\mathbb{S}}_\mathbf{p}, \overline{\mathbb{S}}_\mathbf{p})$.

В следующем параграфе рассматривается задача нахождения наименьшего параллелотопа, содержащего множество $\mathbb{S}_\mathbf{p}$.

5.4. Интервальная оболочка множества, задаваемого неравенствами

Описание некоторого (полного) компактного множества $\mathbb{S}_\mathbf{p}$ может оказаться весьма трудоемким, когда размерность $n_\mathbf{p}$ вектора $\mathbf{p}$ высока и когда множество $\mathbb{S}_\mathbf{p}$ велико, так как покрытие всеми параллелотопами, выработанными по алгоритмам SIVIA или SIVIAP, сгущается (накапливается) на границе множества $\mathbb{S}_\mathbf{p}$. В надежде на меньшие вычислительные затраты, если это требуется, рассмотрим теперь задачу нахождения *интервальной оболочки* $[\mathbb{S}_\mathbf{p}]$ множества $\mathbb{S}_\mathbf{p}$ (наименьшего параллелотопа, содержащего это множество) вместо нахождения детального его описания. Это упрощенное описание является важным во многих практических задачах, таких как оценивание параметров, где интервальные компоненты интервальной оболочки соответствуют интервалам неопределенности параметров (см. Главу 6).

5.4.1. Первый подход

Первый подход к решению данной задачи (в контексте нелинейных ограничений) состоит в ее декомпозиции на $2n_\mathbf{p}$ задач оптимизации

$$\min_{\mathbf{p} \in \mathbb{S}_\mathbf{p}} p_i, \ \max_{\mathbf{p} \in \mathbb{S}_\mathbf{p}} p_i, \ i = 1, \ldots, n_\mathbf{p}. \tag{5.20}$$

Оптимизация может быть выполнена на основе обобщенного геометрического (сигномиального) программирования [Milanese и Vicino, 1991]

или интервального анализа [Jaulin, 1994]. Для каждой задачи оптимизации из (5.20) покрытие, выработанное интервальными методами при глобальной оптимизации (как описывается в параграфе 5.5), накапливается на границе $\partial\mathbb{S}_\mathbf{p} \cap \partial[\mathbb{S}_\mathbf{p}]$, т. е. на той части границы множества $\mathbb{S}_\mathbf{p}$, которая принадлежит границе интервальной оболочки этого множества. Это — радикальное упрощение по сравнению с идеей алгоритма SIVIAP, который накапливал покрытие на границе множества $\mathbb{S}_\mathbf{p}$ (см. рис. 5.9).

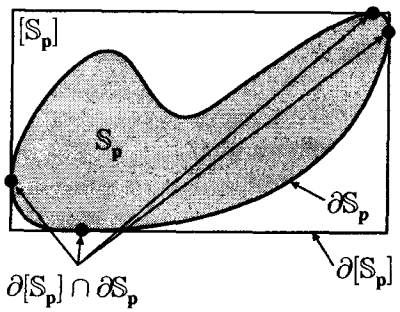


Рис. 5.9. Часть границы множества $\mathbb{S}_\mathbf{p}$, принадлежащая также границе множества $[\mathbb{S}_\mathbf{p}]$

Пример 5.2. *Рассмотрим задачу* [Jaulin, 1994] *описания множества*

$$\mathbb{S}_\mathbf{p} = \{(p_1, p_2) \in [0,\ 5]^2 \ \mid \ \forall t \in [0,\ 1], \ | t^2 + 2t + 1 - p_1 e^{p_2 t} \ | \ \leqslant 1\}.$$

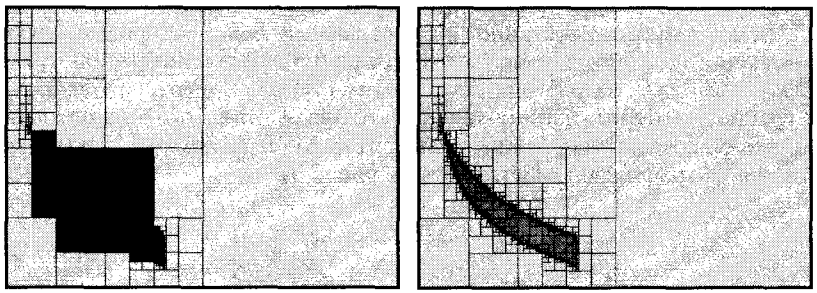
Покрытия, полученные при выполнении четырех задач оптимизации по (5.20), *представлены на* рис. 5.10, *а. Сравните с* рис. 5.10, *а, представляющим покрытие, выработанное алгоритмом* SIVIAP *при решении того же примера.* ■

5.4.2. Второй подход

Второй подход, основанный на методах интервального анализа, состоит в использовании алгоритма HULL [Jaulin, 2000a], который заключает множество $[\mathbb{S}_\mathbf{p}]$ в два параллелотопа $[\mathbf{p}_{\text{in}}]$ и $[\mathbf{p}_{\text{out}}]$ следующим образом:

$$[\mathbf{p}_{\text{in}}] \subset [\mathbb{S}_\mathbf{p}] \subset [\mathbf{p}_{\text{out}}]. \tag{5.21}$$

Делаются только предположения, что используемый оператор сжатия годится для работы с множеством $\mathbb{S}_\mathbf{p}$, т. е. можно проверить, принадлежит



а) *б)*

Рис. 5.10. Покрытия, выработанные в процессе решения Примера 5.2: *а*) при оценивании интервальной оболочки; *б*) при работе алгоритма SIVIAP; число бисекций в *а*) значительно меньше, чем в случае *б*)

ли некоторая заданная точка $\mathbf{p}$ этому множеству, и что имеется некоторый параллелотоп $[\mathbf{p}]$ (пусть даже очень большой), содержащий $\mathbb{S}_{\mathbf{p}}$. Вместо решения $2n_{\mathbf{p}}$ оптимизационных задач, алгоритм HULL вырабатывает две последовательности параллелотопов $[\mathbf{p}_{in}](k)$ и $[\mathbf{p}_{out}](k)$ и последовательность покрытий $\mathbb{L}(k)$, которые удовлетворяют условиям:

$$\begin{cases} [\mathbf{p}_{\text{in}}](k) \subset [\mathbb{S}_{\mathbf{p}}], \\ \mathbb{S}_{\mathbf{p}} \subset \mathbb{L}(k) \cup [\mathbf{p}_{\text{out}}](k), \\ [\mathbf{p}_{\text{in}}](k) \subset [\mathbf{p}_{\text{out}}](k). \end{cases} \tag{5.22}$$

Идея алгоритма HULL иллюстрируется на рис. 5.11. Как указывалось в Замечании 3.1 (с. 75), покрытие $\mathbb{L}(k)$ будет обозначаться как $\mathcal{L}(k)$, когда оно рассматривается в качестве списка параллелотопов.

Алгоритм HULL чистит $\mathcal{L}(k)$ путем максимально возможного увеличения аппроксимации снизу $[\mathbf{p}_{\text{in}}](k)$ и максимально возможного уменьшения аппроксимации сверху $[\mathbf{p}_{\text{out}}](k)$ при выполнении трех условий (5.22). Основные преобразования, используемые для решения данной задачи, описываются ниже. После каждого преобразования номер k увеличивается на 1. Для простоты обозначений, номер k в записях $[\mathbf{p}_{\text{in}}]$, $[\mathbf{p}_{\text{out}}]$ и $\mathcal{L}$ опускается.

1. Раздутие нижней аппроксимации. Если каким-нибудь методом локального поиска найдено, что некоторая точка $\widetilde{\mathbf{p}} \in \mathbb{S}_{\mathbf{p}}$ лежит вне параллелотопа $[\mathbf{p}_{\text{in}}](k)$ (этот случай положения точки отмечен жирным кружочком на рис. 5.11,*а*), то $[\mathbf{p}_{\text{in}}](k) := [\mathbf{p}_{\text{in}}](k) \sqcup \{\widetilde{\mathbf{p}}\}$ и $[\mathbf{p}_{\text{out}}](k) := [\mathbf{p}_{\text{out}}](k) \sqcup \{\widetilde{\mathbf{p}}\}$.

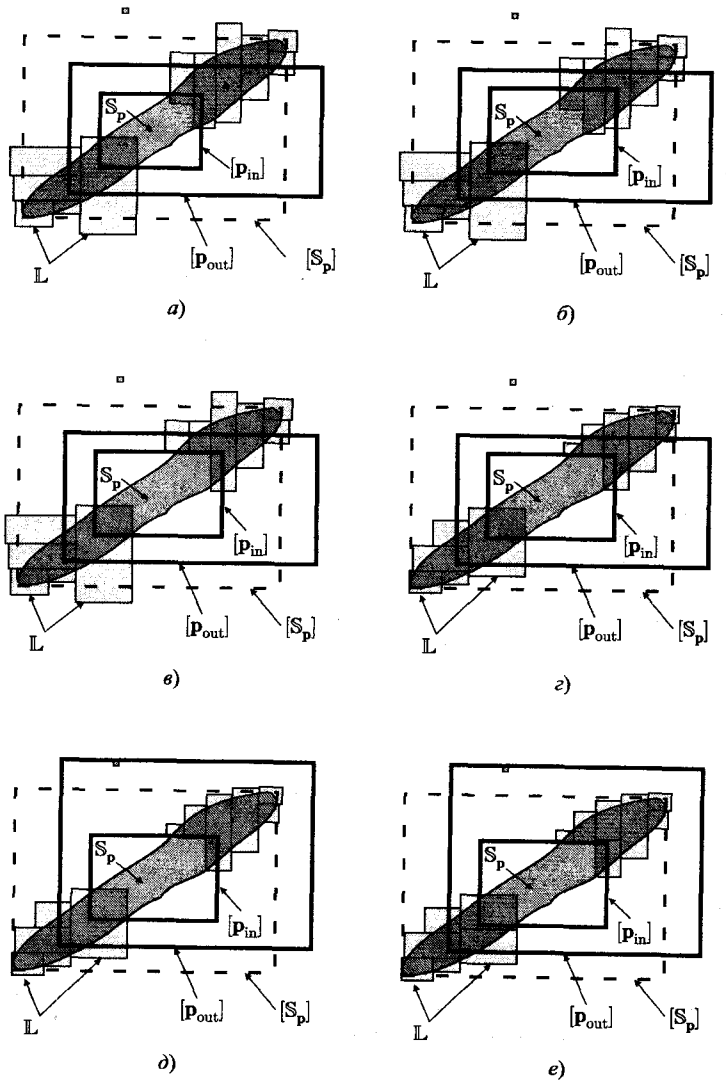


Рис. 5.11. Идея алгоритма HULL: *а*) исходная конфигурация; *б*) раздутие внутренней аппроксимации; *в*) отсечение; *г*) сжатие; *д*) раздутие внешней аппроксимации; *е*) выполнение бисекции

Напомним, что оператор $\sqcup$ объединения интервалов вычисляет наименьший параллелотоп, содержащий объединение его аргументов, т. е. $\mathbb{A} \sqcup \mathbb{B} = [\mathbb{A} \sqcup \mathbb{B}]$. Раздутие аппроксимации снизу иллюстрируется рис. 5.11, *б*.

2. Отсечение. Для любого заданного параллелотопа $[\mathbf{p}]$ из списка $\mathcal{L}$ множество $[\mathbf{p}] := [[\mathbf{p}] \setminus [\mathbf{p}_{\text{in}}]]$, где

$$[\mathbf{p}] \setminus [\mathbf{p}_{\text{in}}] \triangleq \{\mathbf{p} \in [\mathbf{p}] \mid \mathbf{p} \notin [\mathbf{p}_{\text{in}}]\}. \tag{5.23}$$

Если, например, $[\mathbf{p}] \subset [\mathbf{p}_{\text{in}}]$, то $[\mathbf{p}] \setminus [\mathbf{p}_{\text{in}}] = \varnothing$, и при отсечении $[\mathbf{p}]$ удаляется из $\mathcal{L}$. Заметим, что такое отсечение неэффективно, когда $[\mathbf{p}]$ содержит угол параллелотопа $[\mathbf{p}_{\text{in}}]$, поскольку тогда параллелотоп $[[\mathbf{p}] \setminus [\mathbf{p}_{\text{in}}]]$ равен параллелотопу $[\mathbf{p}]$. На рис. 5.11, *в* показано, что отсечение эффективно только для двух параллелотопов из списка $\mathcal{L}$.

3. Сжатие. Для любого заданного параллелотопа $[\mathbf{p}]$ из списка $\mathcal{L}$, множество $[\mathbf{p}] := \mathcal{C}_{\mathbb{S}_\mathbf{p}}([\mathbf{p}])$, где $\mathcal{C}_{\mathbb{S}_\mathbf{p}}$ — некоторый приемлемый для $\mathbb{S}_\mathbf{p}$ сжимающий оператор (см. рис. 5.11, *г*).

4. Раздутие верхней аппроксимации. Если ширина некоторого параллелотопа $[\mathbf{p}]$ из списка $\mathcal{L}$ меньше, чем ε, или если только очень маленькая часть $[\mathbf{p}]$ лежит вне $[\mathbf{p}_{\text{out}}]$ (т. е. расстояние $h^0_\infty([\mathbf{p}], [\mathbf{p}_{\text{out}}]) < \varepsilon$), то параллелотоп $[\mathbf{p}]$ удаляется из $\mathcal{L}$ и множества $[\mathbf{p}_{\text{out}}] := [\mathbf{p}_{\text{out}}] \sqcup [\mathbf{p}]$. Так, небольшой параллелотоп в верхней части всех фрагментов рис. 5.11 (отмечен серой заливкой) представляется слишком малым, чтобы рассматривать на нем бисекцию; таким образом, параллелотоп $[\mathbf{p}_{\text{out}}]$ раздувается, чтобы захватить указанный небольшой параллелотоп, даже если $[\mathbf{p}]$ и не пересекается с множеством $\mathbb{S}_\mathbf{p}$.

5. Бисекция. Если ширина некоторого параллелотопа $[\mathbf{p}]$ из списка $\mathcal{L}$ больше ε, то он подвергается бисекции на два подпараллелотопа $[\mathbf{p}_1]$ и $[\mathbf{p}_2]$. В списке $\mathcal{L}$ данный параллелотоп заменяется на эти два подпараллелотопа. Поскольку бисекция увеличивает сложность алгоритма HULL экспоненциально по отношению к размерности $n_\mathbf{p}$, то она должна использоваться лишь как крайнее средство. Это поясняется на рис. 5.11, *е*.

Тривиально показать, что после каждого из указанных преобразований все три условия (5.22) остаются выполненными. Для простоты, мы выбрали порядок работы со списком $\mathcal{L}$ *«первым взят — первым обработан»*, даже если есть и другие, более подходящие способы. Алгоритм HULL описан в табл. 5.4.

Следующие свойства имеют место [Jaulin, 2000a]:

$$\exists \overline{k} > 0 \text{ такое, что } \mathbb{L}(\overline{k}) = \varnothing, \tag{5.24}$$

$$[\mathbf{p}_{\text{in}}](\overline{k}) \subset [\mathbb{S}_\mathbf{p}] \subset [\mathbf{p}_{\text{out}}](\overline{k}), \tag{5.25}$$

где $\overline{k}$ — номер итерации k после завершения работы алгоритма HULL. Это означает, что алгоритм HULL заканчивает работу и обеспечивает гарантированное двустороннее оценивание множества $[\mathbb{S}_\mathbf{p}]$. Однако надо еще выполнить анализ сходимости $[\mathbf{p}_{\text{in}}](\overline{k})$ и $[\mathbf{p}_{\text{out}}](\overline{k})$ к $[\mathbb{S}_\mathbf{p}]$, когда ε стремится к нулю.

Пример 5.3. *Для задачи* Примера 5.1 *алгоритм* HULL *на компьютере* Pentium 133 *находит наименьший параллелотоп, содержащий множество* $\mathbb{S}_\mathbf{p}$, *за* 0,055 *секунды с точностью в шесть разрядов.* Рис. 5.12 *иллюстрирует технику решения.*

На первой итерации, на Шагах 4, 5 и 6 *алгоритму* HULL *удается обнаружить только самую левую компоненту множества* $\mathbb{S}_\mathbf{p}$. *Таким образом, становится возможным раздуть* $[\mathbf{p}_{in}](\overline{k})$ *и* $[\mathbf{p}_{out}](\overline{k})$ *так, что они становятся почти равными интервальной оболочке левой компоненты множества* $\mathbb{S}_\mathbf{p}$ (см. рис. 5.12, *а*).

Локальный поиск на Шаге 4 *выполняется по алгоритму* CROSS [Jaulin, 2000a], *представленному в*, табл. 6.3. *На* Шаге 7 *параллелотоп* $[\mathbf{p}]$ *не может быть отсечен, но на* Шаге 8 *оператор* $\mathcal{C}_{\mathbb{S}_\mathbf{p}}$ *сжимает параллелотоп* $[\mathbf{p}]$ *прямо до параллелотопа — решения* $[\mathbb{S}_\mathbf{p}]$. *На этом этапе алгоритм* HULL *еще не может заключить, что параллелотоп* $[\mathbf{p}]$ *равен* $[\mathbb{S}_\mathbf{p}]$. *Вот почему далее параллелотоп* $[\mathbf{p}]$ *подвергается бисекции на два подпараллелотопа. Список* $\mathcal{L}$ *теперь содержит два параллелотопа, представленные на* рис. 5.12, *б. Заметим, что условия* (5.22) *выполняются.*

Далее, алгоритм HULL *берет левый подпараллелотоп, не может выполнить на нем локальный поиск и сжимает его на* Шаге 8 *до тех пор, пока он не становится почти равным параллелотопам* $[\mathbf{p}_{\text{in}}]$ *и* $[\mathbf{p}_{\text{out}}]$. *Поскольку на* Шаге 10 *расстояние* $h^0_\infty([\mathbf{p}], [\mathbf{p}_{\text{out}}]) < \varepsilon$, *то паралелотоп* $[\mathbf{p}_{\text{out}}]$ *несколько раздувается.*

Далее, алгоритм HULL *переходит к* Шагу 2. *Опять условия* (5.22) *выполняются. Теперь алгоритм* HULL *берет последний (правый) подпараллелотоп из списка* $\mathcal{L}$ *и успешно обнаруживает саму правую компоненту множества* $\mathbb{S}_\mathbf{p}$. *Параллелотопы* $[\mathbf{p}_{\text{in}}]$ *и* $[\mathbf{p}_{\text{out}}]$ *раздуваются и становятся почти равными множеству* $[\mathbb{S}_\mathbf{p}]$. *На* Шаге 11 *параллелотоп* $[\mathbf{p}_{\text{out}}]$ *несколько раздувается на основе параллелотопа* $[\mathbf{p}]$, *и алгоритм* HULL *завершает свою работу, так как список* $\mathcal{L}$ *опустел.* ■

ЗАМЕЧАНИЕ 5.2. Когда множество $\mathbb{S}_\mathbf{p}$ слишком узко и вытянуто, может оказаться очень трудным найти допустимые точки, которые позволят запустить раздутие внутренней аппроксимации. Как результат, алгоритм HULL может выполнять бисекцию параллелотопов, которые находятся внутри множества $[\mathbb{S}_\mathbf{p}]$. Чтобы избежать этого явления, которое замедляет работу алгоритма, можно использовать главное свойство подхода, описанного в параграфе 5.4.1, в котором бисекция выполняется только на параллелотопах, частично лежащих вне множества $[\mathbb{S}_\mathbf{p}]$. Таким образом,

Таблица 5.4. Алгоритм построения интервальной оболочки множества, определяемого системой нелинейных неравенств

Алгоритм HULL(вход : $[\mathbf{p}], \mathcal{C}_{\mathbb{S}_\mathbf{p}}, \varepsilon$; выход : $[\mathbf{p}_{\text{in}}]$, $[\mathbf{p}_{\text{out}}]$)

```
1  [p_in] := ∅; [p_out] := ∅; L := [p];
2  повтор
3     заключить первый параллелотоп списка L в параллелотоп [p];
4     с помощью локального поиска, инициализированного
          от средней точки mid([p]), данного паралелотопа,
          искать допустимые точки p̃ вне параллелотопа [p_in]
5  для каждой точки p̃,
6     [p_in] := [p_in] ⊔ {p̃}; [p_out] := [p_out] ⊔ {p̃}; (раздутие
          внутренней аппроксимации);
7  [p] := [[p]\[p_in]]; (отсечение)
8  если [p] ≠ ∅; [p] := C_{S_p}([p]); (сжатие);
9  если [p] ≠ ∅
10    если (w([p]) < ε) или (h^0_∞([p], [p_out]) < ε)
11          [p_out] := [p_out] ⊔ [p]; раздутие внешней аппроксимации;
12    иначе
13 выполняется бисекция [p] и полученные подпараллелотопы
          заносятся в конец списка L;
14 до тех пор, пока список не станет пустым L = ∅.
```

на каждой итерации алгоритм HULL должен вычислять наименьший параллелотоп $[\mathbf{p}_{\text{ext}}]$, который содержит параллелотоп $[\mathbf{p}_{\text{out}}]$ и параллелотопы списка $\mathcal{L}$. Тогда алгоритм будет выполнять бисекцию только параллелотопов из $\mathcal{L}$, которые касаются границы параллелотопа $[\mathbf{p}_{\text{ext}}]$. ■

5.5. Глобальная оптимизация

Задача, которая теперь будет рассмотрена, состоит в минимизации целевой функции $c(\mathbf{p})$ на компактном множестве $\mathbb{S}_\mathbf{p}^\infty \subset \mathbb{R}^{n_\mathbf{p}}$:

$$\min_{\mathbf{p}\in\mathbb{S}_\mathbf{p}^\infty} c(\mathbf{p}). \tag{5.26}$$

При минимизации без ограничений множество $\mathbb{S}_\mathbf{p}^\infty$ будет браться в виде максимально большого возможного параллелотопа $[\mathbf{p}]$ из $\mathbb{R}^{n_\mathbf{p}}$. При минимизации с ограничениями, описание множества $\mathbb{S}_\mathbf{p}^\infty$ будет включать ограничения в виде равенств или неравенств. Например, $\mathbb{S}_\mathbf{p}^\infty$ может задаваться как

$$\mathbb{S}_\mathbf{p}^\infty \triangleq \{\mathbf{p} \in \mathbb{R}^{n_\mathbf{p}} \mid \mathbf{h}(\mathbf{p}) \leqslant \mathbf{0} \text{ и } \mathbf{p} \in [\mathbf{p}]\}. \tag{5.27}$$

Глобальный минимум будет обозначаться как $\widehat{c}$, а множество всех соответствующих точек глобального минимума как $\widehat{\mathbb{S}}_\mathbf{p}$. Всегда можно преобразовать задачу максимизации в задачу минимизации, пример, умножением $c(\mathbf{p})$ на -1. Наиболее эффективные алгоритмы оптимизации на основе интервального анализа совместно используют как операторы сжатия для $\widehat{\mathbb{S}}_\mathbf{p}$, так и классический локальный поиск по ветвящимся алгоритмам. Прежде чем описывать ветвящийся алгоритм, выполняющий минимизацию, мы поясним, как строится сжимающий оператор для множества $\widehat{\mathbb{S}}_\mathbf{p}$.

Аппарат, разработанный в Главе 4, не может быть прямо применен для построения сжимающего оператора $\mathcal{C}_{\widehat{\mathbb{S}}_\mathbf{p}}$ для множества $\widehat{\mathbb{S}}_\mathbf{p}$, так как множество всех точек глобального минимума обычно не описывается нелинейными равенствами или неравенствами. Теперь, если $\overline{c}$ — некоторое ограничение сверху на глобальный минимум $\widehat{c}$ (полученное, например, локальной

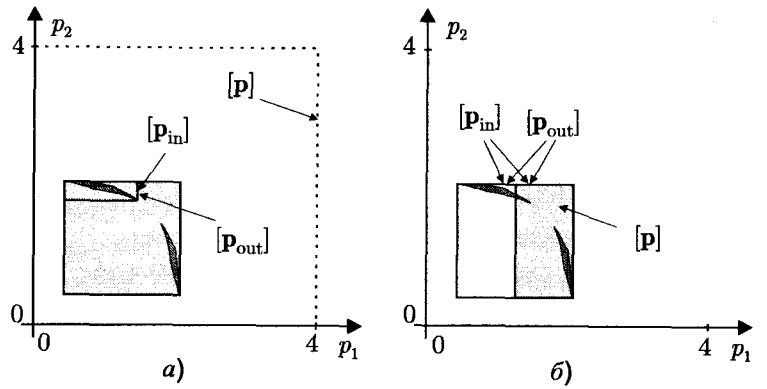


Рис. 5.12. Работа алгоритма HULL: *а*) параллелотоп $[\mathbf{p}]$ сжат до параллелотопа, отмеченного серой заливкой; локальный поиск находит только левую компоненту множества $\mathbb{S}_\mathbf{p}$; *б*) после бисекции алгоритм HULL способен найти правую компоненту множества $\mathbb{S}_\mathbf{p}$; когда HULL завершает работу, параллелотопы $[\mathbf{p}_{\text{in}}]$ и $[\mathbf{p}_{\text{out}}]$ неразличимы

минимизацией), то

$$\widehat{\mathbb{S}}_{\mathbf{p}} \subset \mathbb{S}_{\mathbf{p}}(\overline{c}), \tag{5.28}$$

где

$$\mathbb{S}_{\mathbf{p}}(\overline{c}) \triangleq \mathbb{P}(\overline{c}) \cap \mathbb{S}_{\mathbf{p}}^{\infty}, \tag{5.29}$$

при

$$\mathbb{P}(\overline{c}) \triangleq \{\mathbf{p} \in \mathbb{R}^{n_{\mathbf{p}}} \mid c(\mathbf{p}) \leqslant \overline{c}\}. \tag{5.30}$$

Таким образом, сжимающий оператор для множества $\mathbb{S}_{\mathbf{p}}(\overline{c})$, определяемый выражением

$$\mathcal{C}_{\mathbb{S}_{\mathbf{p}}(\overline{c})} = \mathcal{C}_{\mathbb{S}_{\mathbf{p}}^{\infty}} \cup \mathcal{C}_{\mathbb{P}(\overline{c})}, \tag{5.31}$$

также является сжимающим оператором для множества $\widehat{\mathbb{S}}_{\mathbf{p}}$, см. (4.115) (с. 132) и (4.117) (с. 132). Этот оператор в общем случае не является точным, и его эффективность сильно зависит от величины $\overline{c}$. Заметим, что если $\widetilde{c}$ — наименьшее значение верхней границы $\overline{c}$ такое, что $\mathbb{S}_{\mathbf{p}}(\overline{c}) \neq \varnothing$, то $\mathbb{S}_{\mathbf{p}}(\overline{c}) = \mathbb{S}_{\mathbf{p}}(\widehat{c}) = \widehat{\mathbb{S}}_{\mathbf{p}}$.

Замечание 5.3. Для повышения эффективности сжимающего оператора на множестве $\widehat{\mathbb{S}}_{\mathbf{p}}$ можно использовать дополнительную информацию. Если, например, целевая функция c, подлежащая минимизации, дважды дифференцируема по вектору $\mathbf{p}$, и минимизация выполняется в отсутствии ограничений, то множество

$$\mathbb{O} \triangleq \left\{\mathbf{p} \in \mathbb{R}^{n_{\mathbf{p}}} \mid \frac{dc}{d\mathbf{p}}(\mathbf{p}) = \mathbf{0},\ \frac{d^2c}{d\mathbf{p}^2}(\mathbf{p}) \geqslant \mathbf{0}\right\} \tag{5.32}$$

содержит $\widehat{\mathbb{S}}_{\mathbf{p}}$. Более эффективным сжимающим оператором для множества $\widehat{\mathbb{S}}_{\mathbf{p}}$ является оператор

$$\mathcal{C}_{\widehat{\mathbb{S}}_{\mathbf{p}}} \triangleq \mathcal{C}_{\mathbb{S}_{\mathbf{p}}(\overline{c})} \cap \mathcal{C}_{\mathbb{O}}. \tag{5.33}$$

■

В табл. 5.5 представлен ветвящийся алгоритм OPTIMIZE, который выполняет минимизацию. Здесь $\mathcal{Q}$ — рабочий список параллелотопов, упорядоченных по убыванию значения связанной с ними нижней границы целевой функции. Алгоритм OPTIMIZE заполняет список $\mathcal{L}$ параллелотопов. По завершению работы алгоритма множество $\mathbb{L}$, связанное с этим списком, содержит все точки глобального минимума целевой функции $c(\cdot)$ на множестве $\mathbb{S}_{\mathbf{p}}^{\infty}$ и интервал содержит глобальный минимум $\widehat{c}$. Здесь $[c(\cdot)]$ является функцией включения для функции $c(\cdot)$. Некоторый параллелотоп

из списка $\mathcal{Q}$, выбранный на Шаге 3, связан с наименьшей нижней границей целевой функции, которая соответствует выбору наиболее перспективного параллелотопа.

На Шаге 4 некоторая возможная процедура GODOWN локальной минимизации используется для снижения верхней границы $\overline{c}$. Вещественное число $\varepsilon > 0$ задает допуск на ширину, относительно которого параллелотопы из списка $\mathcal{Q}$ с шириной, меньшей допуска, не подвергаются бисекции. Интервал $[\widehat{c}]$, ограничивающий глобальный минимум, находится на Шагах 15 и 16 расчетом интервальной оценки значений функции c на всех параллелотопах списка $\mathcal{L}$.

В качестве иллюстрации рассмотрим ситуацию, представленную на рис. 5.13, *а*. Для $\widehat{c}$ возможна некоторая верхняя граница c_1, но сжимающий оператор $\mathcal{C}_{\mathbb{S}_{\mathbf{p}}(c_1)}$ оставляет $[\mathbf{p}]$ без изменения. Тогда локальный поиск может дать меньшее значение c_2 нижней границы для $\widehat{c}$ (рис. 5.13, *б*). Сжимающий оператор $\mathcal{C}_{\mathbb{S}_{\mathbf{p}}(c_2)}$ теперь может быть использован для более эффективного сжатия $[\mathbf{p}]$, как показано на рис. 5.13, *в*. Поскольку $\mathcal{C}_{\mathbb{S}_{\mathbf{p}}(c_2)}$ является также оператором на $\mathbb{S}_{\mathbf{p}}$, то ни одно из глобальных решений задачи минимизации не может быть потеряно.

Таблица 5.5. Алгоритм надежного выполнения минимизации

Алгоритм OPTIMIZE(вход : $[\mathbf{p}], c(\cdot), \mathcal{C}_{\widehat{\mathbb{S}}_{\mathbf{p}}}, \varepsilon$; выход : $[\widehat{c}]$, $\mathcal{L}$)	
1	$\mathcal{Q} := \{([\mathbf{p}], \infty)\}$; $\overline{c} := \infty$; $[\widehat{c}] := \varnothing$; $\mathcal{L} := \varnothing$;
2	повтор
3	захватить первый параллелотоп списка $\mathcal{Q}$ в параллелотоп $[\mathbf{p}]$;
4	$\overline{c} :=$ GODOWN(mid($[\mathbf{p}]$), $c(\cdot)$);
5	из списка $\mathcal{Q}$ удалить все пары $([\mathbf{p}_i]), c_i)$, для которых $c_i > \overline{c}$;
6	$[\mathbf{p}] := \mathcal{C}_{\widehat{\mathbb{S}}_{\mathbf{p}}}([\mathbf{p}])$;
7	если $[\mathbf{p}] \neq \varnothing$, то
8	если $(w([\mathbf{p}]) < \varepsilon)$, то
9	записать $([\mathbf{p}],\ \mathrm{lb}([c]([\mathbf{p}])))$ в список $\mathcal{L}$;
10	иначе
11	выполняется бисекция $[\mathbf{p}]$ на подпараллелотопы $[\mathbf{p}_1]$ и $[\mathbf{p}_2]$;
12	записать $([\mathbf{p}_1],\ \mathrm{lb}([c]([\mathbf{p}_1])))$ и $([\mathbf{p}_2],\ \mathrm{lb}([c]([\mathbf{p}_2])))$ в список $\mathcal{Q}$;
13	до тех пор, пока список не станет пустым $\mathcal{Q} = \varnothing$;
14	из списка $\mathcal{L}$ удалить все пары $([\mathbf{p}_i]), c_i)$, для которых $c_i > \overline{c}$;
15	для всех $[\mathbf{p}]$ из $\mathcal{L}$, $[\widehat{c}] := [\widehat{c}] \cup [c]([\mathbf{p}])$;
16	$[\widehat{c}] := [\widehat{c}] \cap]-\infty, \overline{c}]$.

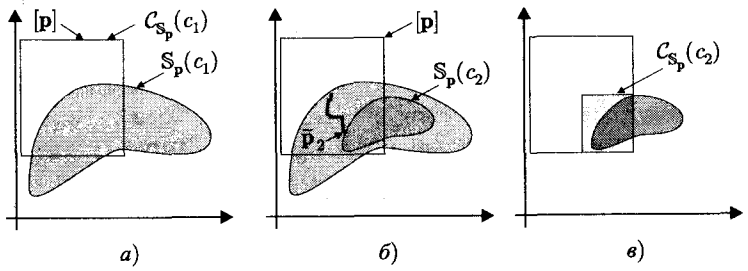


Рис. 5.13. Работа алгоритма OPTIMIZE: *а*) исходная конфигурация; *б*) локальная минимизация; в) сжатие

ЗАМЕЧАНИЕ 5.4. Эксперименты [Ratschek и Rokne, 1995; Ratz и Csendes, 1995] показали, что эффективный выбор бисекции состоит в разрезании параллелотопа вдоль $\text{plane}_i([\mathbf{p}])$ плоскости симметрии, такой, что

$$w\left(\left[\frac{\partial c}{\partial p_i}\right]([\mathbf{p}])\right) * w([p_i]) \geqslant w\left(\left[\frac{\partial c}{\partial p_j}\right]([\mathbf{p}])\right) * w([p_j]), \quad \forall j \in \{1, \ldots, \dim \mathbf{p}\}.$$

■

5.5.1. Алгоритм Мура–Скелбо

Упрощенная версия алгоритма OPTIMIZE была предложена в [Skelboe, 1974] и улучшена в [Moore, 1976]. Окончательный вариант алгоритма не использует каких-либо сжимающих операторов и не выполняет какой-либо локальный поиск. Следующие два примера [Walster *et al.*, 1985; Moore и Ratschek, 1988; Jansson и Knüppel, 1995] иллюстрируют эффективность алгоритма Мура–Скелбо и влияние размерности задачи.

Пример 5.4. *Функция Бранина*

$$c(\mathbf{p}) = (p_2 - \frac{5,1}{4\pi^2}p_1^2 + \frac{5}{\pi} + \frac{5}{\pi}p_1 - 6)^2 + 10(1 - \frac{1}{8\pi})\cos(p_1) + 10 \quad (5.34)$$

допускает три точки глобального минимума на параллелотопе $[\mathbf{p}] = [-5,\ 10] \times [0,\ 15]$, *а именно*

$$\widehat{\mathbf{p}}_1 = (-\pi,\ 12,275)^{\mathrm{T}},\ \widehat{\mathbf{p}}_2 = (\pi,\ 2,275)^{\mathrm{T}},\ \widehat{\mathbf{p}}_3 = (3\pi,\ 2,475)^{\mathrm{T}}. \quad (5.35)$$

Соответствующее значение глобального минимума равно $\widehat{c} \simeq 0,397887$. *Для* $\varepsilon = 10^{-5}$ *после* 422 *бисекций, выполненных за* 0,4 *секунды на компьютере* Pentium 90, *алгоритм Мура–Скелбо нашел* 18 *параллелотопов, объединение которых содержит все точки глобального минимума; этот минимум* $\widehat{c}$ *также был рассчитан с точностью до* 10 *разрядов. Эти* 18 *параллелотопов решения можно разложить на три группы. Интервальные оболочки этих групп имеют следующий вид*:

$$\begin{aligned}
[\widehat{\mathbf{p}}_1] &= [-3,141594,\ -3,141586] \times [12,274982,\ 12,275012],\\
[\widehat{\mathbf{p}}_2] &= [3,141591,\ 3,141599] \times [2,274982,\ 2,275014],\\
[\widehat{\mathbf{p}}_3] &= [9,424769,\ 9,424785] \times [2,474977,\ 2,475021].
\end{aligned} \quad (5.36)$$

■

Таблица 5.6. Время счета и число параллелотопов решения в зависимости от размерности n

n	1	...	5	6	7	8	9
Время счета, секунды	0,05	...	0,71	1,43	2,97	6,87	17,02
Число параллелотопов решения	1	...	66	147	294	547	955

Пример 5.5. *Рассмотрим семейство функций Леви–13*

$$c_n(\mathbf{p}) = \sin^2(3\pi p_1) + \sum_{i=1}^{n-1}(p_i - 1)^2(1 + \sin^2(3\pi p_{i+1}))+ \\ + (p_n - 1)^2(1 + \sin^2(2\pi p_n)), \quad (5.37)$$

при $n \geqslant 1$. *Начальный параллелотоп поиска равен* $[-10,\ 10]^{\times n}$, *если* $n \leqslant 4$, *и* $[-5,\ 5]^{\times n}$, *если* $n > 4$. *Каждая функция* c_n *допускает глобальный минимум* $\widehat{c} = 0$ *и только одну точку* $\widehat{\mathbf{p}}$ *глобального минимума при значениях всех переменных, равных единице. Количество точек локальных минимумов функций* c_n *растет экспоненциально от* n (900 *для* $n = 2$ *и* 10^5 *для* $n = 5$). *Результаты работы алгоритма Мура–Скелбо для шести значений* n *при* $\varepsilon = 10^{-5}$ *приведены в* табл. 5.6. *Времена счета указаны для компьютера* Pentium 90. ■

5.5.2. Алгоритм Хансена

Другим вариантом алгоритма, близким к OPTIMIZE, является алгоритм Хансена. Этот параграф описывает некоторые используемые операторы сжатия [Hansen и Segupta, 1980; Hansen, 1992б]. Все эти операторы могут

быть вложены в рамки Главы 4. Заметим, что рассматриваемый алгоритм не использует каких-либо операторов сжатия (например, такого оператора, как $\mathcal{C}_{\downarrow\uparrow}$), основанных на методе вперед-назад. Будут рассмотрены некоторые специфические детали алгоритма; за подробностями читатель может обратиться к работе [Hansen, 1992б], целиком посвященной данному вопросу.

Операторы сжатия верхней границы. Предположим, что известна некоторая верхняя граница $\bar{c}$ глобального минимума $\tilde{c}$. Сжатие некоторого параллелотопа при ограничении $c(\mathbf{p}) \leqslant \bar{c}$ приводит к сжатию следующей задачи выполнения ограничений:

$$\mathcal{H} : (c(\mathbf{p}) = z,\ \mathbf{p} \in [\mathbf{p}],\ z \in]-\infty,\ \bar{c}]). \tag{5.38}$$

Если функция $c(\cdot)$ дифференцируема, то аппроксимация сверху задачи $\mathcal{H}$ (5.38) (см. параграф 4.3, с. 113) задается выражениями

$$\mathcal{H}_1 : \begin{pmatrix} c(\mathbf{m}) + \sum_{i=1}^{n} (p_i - m_i) g_i(\boldsymbol{\xi}) = z \\ \mathbf{p} \in [\mathbf{p}],\ \boldsymbol{\xi} \in [\mathbf{p}],\ z \in]-\infty,\ \bar{c}] \end{pmatrix}, \tag{5.39}$$

где $g_i(\boldsymbol{\xi})$ — i-я компонента градиента функции c в точке $\boldsymbol{\xi}$ и $\mathbf{m} = \operatorname{mid}([\mathbf{p}])$. Разрешающий оператор для задачи $\mathcal{H}_1$ задается выражением

$$p_k = m_k + \frac{1}{g_i(\boldsymbol{\xi})} \left(z - c(\mathbf{m}) - \sum_{i=1, i \neq k}^{n} (p_i - m_i) g_i(\boldsymbol{\xi}) \right), \tag{5.40}$$

которое используется Хансеном для сжатия области переменной p_k. Другая аппроксимация сверху задачи (5.38) основана на разложении функции c в ряд Тейлора второго порядка в виде:

$$\mathcal{H}_2 : \begin{pmatrix} c(\mathbf{m}) + (\mathbf{p} - \mathbf{m})^{\mathrm{T}} \mathbf{g}(\mathbf{m}) + \dfrac{(\mathbf{p} - \mathbf{m})^{\mathrm{T}} \mathbf{H}(\boldsymbol{\xi})(\mathbf{p} - \mathbf{m})}{2} = z \\ \mathbf{p} \in [\mathbf{p}],\ \boldsymbol{\xi} \in [\mathbf{p}],\ z \in]\infty,\ \bar{c}] \end{pmatrix}, \tag{5.41}$$

где $\mathbf{m} = \operatorname{mid}([\mathbf{p}])$, $\mathbf{g}$ — вектор градиента функции c, а $\mathbf{H}(\boldsymbol{\xi})$ — ее матрица Гессе в точке $\boldsymbol{\xi}$. Первая строка (5.41) эквивалентна выражению

$$\begin{aligned} & c(\mathbf{m}) + (p_k - m_k) g_k(\mathbf{m}) \\ & \quad + \sum_{i=1, i \neq k}^{n} (p_i - m_i) g_i(\mathbf{m}) + \tfrac{1}{2} (p_k - m_k)^2 h_{kk}(\boldsymbol{\xi}) \\ & \quad + \tfrac{1}{2} (p_k - m_k) \sum_{i=1, i \neq k}^{n} (p_i - m_i) h_{ik}(\boldsymbol{\xi}) \\ & \quad + \tfrac{1}{2} \sum_{i=1, i \neq k}^{n} (p_i - m_i) \sum_{j=1, j \neq k}^{i} (p_j - m_j) h_{ij}(\boldsymbol{\xi}) = z. \end{aligned} \tag{5.42}$$

При этом задача $\mathcal{H}_2$ может быть переписана как

$$\mathcal{H}_2 : \begin{pmatrix} 1 : \alpha_k + \beta_k t_k + \gamma_k t_k^2 = 0 \\ 2 : t_k = p_k - m_k \\ 3 : \alpha_k = -z + c(\mathbf{m}) + \sum_{i=1, i \neq k}^{n} (p_i - m_i) g_i(\mathbf{m}) \\ \qquad + \tfrac{1}{2} \sum_{i=1, i \neq k}^{n} (p_i - m_i) \sum_{j=1, j \neq k}^{i} (p_j - m_j) h_{ij}(\boldsymbol{\xi}) \\ 4 : \beta_k = g_k(\mathbf{m}) + \tfrac{1}{2} \sum_{i=1, i \neq k}^{n} (p_i - m_i) h_{ik}(\boldsymbol{\xi}) \\ 5 : \gamma_k = \tfrac{1}{2} h_{kk}(\boldsymbol{\xi}) \\ 6 : \mathbf{p} \in [\mathbf{p}],\ \boldsymbol{\xi} \in [\mathbf{p}],\ z \in]-\infty,\ \bar{c}] \end{pmatrix}. \tag{5.43}$$

Области для α_k, β_k и γ_k легко получаются из ограничений 3–6 в (5.43). Ограничение 1 может быть использовано для получения области для t_k. Поскольку это ограничение является квадратичным и содержит только t_k, то можно разработать специальный алгоритм для получения наименьшей области для t_k, с ним совместной [Hansen, 1992б]. Далее, ограничение 2 может быть использовано для сжатия $[p_k]$.

Вогнутость и градиентные операторы сжатия. Если в задаче минимизации ограничения отсутствуют и если функция c дифференцируема, то все точки $\widehat{\mathbf{p}}$ глобального максимума должны удовлетворять уравнению $\mathbf{g}(\widehat{\mathbf{p}}) = \mathbf{0}$, где $\mathbf{g}(\mathbf{p})$ — значение градиента функции c в точке $\mathbf{p}$. Кроме того, если функция c дважды дифференцируема, то (в точке глобального минимума) она должна быть выпуклой по всем направлениям, включая направления осей пространства параметров. Следовательно, ограничения

$$g_1(\mathbf{p}) = 0, \ldots, g_n(\mathbf{p}) = 0, \tag{5.44}$$

$$\frac{\partial^2 c}{\partial p_1^2}(\mathbf{p}) \geqslant 0, \ \ldots, \ \frac{\partial^2 c}{\partial p_n^2}(\mathbf{p}) \geqslant 0, \tag{5.45}$$

можно использовать для сжатия текущего параллелотопа $[\mathbf{p}]$ на Шаге 6 алгоритма OPTIMIZE для выполнения безусловной минимизации. При максимизации знак неравенств в (5.45) должен быть сменен на противоположный.

Критерий остановки. На Шаге 8 алгоритма OPTIMIZE не выполняется бисекция параллелотопов, удовлетворяющих условию $w([\mathbf{p}]) < \varepsilon$. В версии Хансена алгоритма OPTIMIZE не подвергаются бисекции параллелотопы, которые удовлетворяют следующим двум условиям:

$$w([\mathbf{p}]) < \varepsilon_{\mathbf{p}} \ \text{ и } \ w([c]([\mathbf{p}])) < \varepsilon_{\mathbf{c}}. \tag{5.46}$$

Коэффициенты точности $\varepsilon_{\mathbf{p}}$ и $\varepsilon_{\mathbf{c}}$ выбираются пользователем.

Условие единственности. Предположим снова, что на задачу минимизации ограничения не наложены. Пусть $[\mathbf{p}]'$ — параллелотоп, полученный после одной итерации ньютоновского сжимающего оператора, примененного к параллелотопу $[\mathbf{p}]$ с учетом ограничения на градиент $\mathbf{g}(\mathbf{p}) = \mathbf{0}$. Если $[\mathbf{p}]'$ целиком содержится в $[\mathbf{p}]$, то существует по крайней мере одна точка глобального минимума функции c на параллелотопе $[\mathbf{p}]$ [Ratschek и Rokne, 1995; Wolfe, 1996]. Следовательно, когда на выходе алгоритма OPTIMIZE получается только один параллелотоп, условие единственности подтверждает, что точка глобального минимума — единственна. С другой стороны, если на выходе получаем несколько параллелотопов, то возможно, что некоторые из них вообще не содержат точек глобального минимума, даже если каждый их них удовлетворяет условию единственности. Для иллюстрации эффективности алгоритма Хансена рассмотрим те же примеры, что и при анализе алгоритма Мура–Скелбо.

Пример 5.6. *Рассмотрим снова функцию Бранина* (5.34) Примера 5.4. *Напомним, что она допускает три точки глобального минимума*:

$$\widehat{\mathbf{p}}_1 = (-\pi,\ 12{,}275),\ \widehat{\mathbf{p}}_2 = (\pi,\ 2{,}275),\ \widehat{\mathbf{p}}_3 = (3\pi,\ 2{,}475)$$

на параллелотопе $[\mathbf{p}] = [-5,\ 10] \times [0,\ 15]$, *и соответствующее значение глобального минимума равно* $\widehat{c} \simeq 0{,}397887$. *При* $\varepsilon_{\mathbf{p}} = \varepsilon_{\mathbf{c}} = 10^{-5}$ *после* 32 *итераций, выполненных за* 0,05 *секунды на компьютере* Pentium 90, *алгоритм Хансена нашел* 3 *параллелотопа, ширина каждого из которых меньше, чем* 10^{-7}. *Точка глобального минимума получена с точностью* 10^{-10}. ■

Пример 5.7. *Рассмотрим снова семейство функций Леви–13* Примера 5.5. *Для* $n \leqslant 50$ *и* $\varepsilon_{\mathbf{p}} = \varepsilon_{\mathbf{c}} = 10^{-5}$ *алгоритм Хансена дает результаты, представленные в* табл. 5.7. *Сравнивая* табл. 5.6 *и* 5.7, *видим, что использование сжимающих операторов дает возможность работать с задачами бóльшей размерности. Заметим, что в этом примере алгоритм Хансена всегда выдает единственный параллелотоп, который удовлетворяет условию единственности, что доказывает существование одной и только одной точки глобального минимума.* ■

Сжимающий оператор Фритца–Джона. Предположим теперь, что ограничения, которые необходимо выполнить на оптимальных точках, представлены в виде неравенств $\mathbf{h}(\mathbf{p}) \leqslant \mathbf{0}$. Хансен предлагает использовать сжимающий оператор, основанный на условиях Фритца–Джона. По аналогии с более известными условиями Куна–Такера [Pardalos и Rosen, 1987], условия Фритца–Джона дают необходимые условия того, чтобы вектор $\widehat{\mathbf{p}}$

Таблица 5.7. Время счета и число параллелотопов решения в зависимости от размерности n

n	1	...	5	...	20	50
Время счета, секунды	0,05	...	0,33	...	12,5	401
Число параллелотопов решения	1	...	1	...	1	1

был решением задачи оптимизации с ограничениями, что показывается следующей теоремой.

Теорема 5.1. *Если вектор* $\widehat{\mathbf{p}} \in [\mathbf{p}] \subset \mathbb{R}^n$ *является точкой локального минимума функции* c *при ограничениях* $h_i(\widehat{\mathbf{p}}) \leqslant 0$, $i \in \{1, \ldots, m\}$, *то существуют* $m+1$ *вещественных коэффициента* $u_0, u_1, \ldots, u_m$, *таких, что*

$$\begin{aligned}
&u_0 \frac{\partial c}{\partial p_i}(\widehat{\mathbf{p}}) + u_1 \frac{\partial h_1}{\partial p_i}(\widehat{\mathbf{p}}) + \ldots + u_m \frac{\partial h_m}{\partial p_i}(\widehat{\mathbf{p}}) = 0, && i = 1, \ldots, n,\\
&u_i h_i(\widehat{\mathbf{p}}) = 0, && i = 1, \ldots, m,\\
&u_i \geqslant 0, && i = 1, \ldots, m,\\
&u_0 + u_1 + \ldots + u_m = 1.
\end{aligned} \tag{5.47}$$

Коэффициенты $u_0, u_1, \ldots, u_m$ *называются множителями Лагранжа.* ■

В (5.47) $n + m + 1$ уравнений содержат $n + m + 1$ неизвестных (n для компонент вектора $\mathbf{p}$ и $m + 1$ для множителей Лагранжа). Поскольку все множители u_i должны быть положительными, а их сумма равна 1, то область возможных значений каждого множителя может быть задана как $[0, 1]$. Вторая строка в (5.7) дает, что если $h_i(\widehat{\mathbf{p}}) < 0$ (это означает, что ограничение, связанное с h_i, не задействовано в точке $\widehat{\mathbf{p}}$), то $u_i = 0$. Уравнения (5.47) можно записать в более компактной форме $\mathbf{f}^{\mathrm{FJ}}(\mathbf{t}) = \mathbf{0}$, где

$$\mathbf{f}^{\mathrm{FJ}}(\mathbf{t}) = \begin{pmatrix}
u_0 + u_1 + \ldots + u_m - 1\\
u_0 \dfrac{\partial c}{\partial p_1}(\mathbf{p}) + u_1 \dfrac{\partial h_1}{\partial p_1}(\mathbf{p}) + \ldots + u_m \dfrac{\partial h_m}{\partial p_1}(\mathbf{p})\\
\vdots\\
u_0 \dfrac{\partial c}{\partial p_n}(\mathbf{p}) + u_1 \dfrac{\partial h_1}{\partial p_n}(\mathbf{p}) + \ldots + u_m \dfrac{\partial h_m}{\partial p_n}(\mathbf{p})\\
u_1 h_1(\mathbf{p})\\
\vdots\\
u_m h_m(\mathbf{p})
\end{pmatrix} \tag{5.48}$$

и

$$\mathbf{t} = \begin{pmatrix} \mathbf{p} \\ \mathbf{u} \end{pmatrix}, \tag{5.49}$$

где $\mathbf{u} = (u_0, u_1, \ldots, u_m)^{\mathrm{T}}$. Функция $\mathbf{f}^{\mathrm{FJ}}$ называется *функцией Фритца–Джона*. Сжимающий оператор для уравнения $\mathbf{f}^{\mathrm{FJ}}(\mathbf{t}) = \mathbf{0}$ может быть использован для сжатия параллелотопа $[\mathbf{p}]$ без потери решений задачи минимизации с ограничениями. Проиллюстрируем теперь на следующих двух примерах эффективность сжимающего оператора Фритца–Джона, включающего в себя алгоритм Хансена.

Пример 5.8. *Рассмотрим задачу минимизации функции*

$$c(\mathbf{p}) = 0{,}1(p_1^2 + p_2^2) \tag{5.50}$$

на параллелотопе $[\mathbf{p}] = [-1, 1]^{\times 2}$ *при ограничении*

$$2\sin(2\pi p_2) - \sin(4\pi p_1) \leqslant 0. \tag{5.51}$$

Известно, что в этой задаче имеется 24 *точки локальных минимумов и только одна точка глобального минимума* $c(\widehat{\mathbf{p}}) = \widehat{c} = 0$ [Ratschek и Rokne, 1988]. *При* $\varepsilon_{\mathbf{p}} = \varepsilon_{\mathbf{c}} = 10^{-5}$ *после* 25 *итераций, выполненных за* 0,11 *секунды на компьютере* Pentium 90, *алгоритм Хансена нашел единственный параллелотоп, приблизительно равный*

$$[\mathbf{p}] = \begin{pmatrix} [-0{,}0000002,\ 0{,}000004] \\ [-0{,}0000002,\ 0{,}000004] \end{pmatrix}, \tag{5.52}$$

который содержит точку глобального минимума, а сам глобальный минимум $\widehat{c}$ *находится в интервале* $[0, 10^{-10}]$. ■

Пример 5.9. *В* [Hansen, 1992б] *иллюстрируются возможности алгоритма Хансена примерно на* 30 *тестовых примерах, но только один из них имел ограничения на вектор* $\mathbf{p}$, *именно, задача минимизации функции*

$$c(\mathbf{p}) = 12p_1^2 - 6{,}3p_1^4 + p_1^6 + 6p_1p_2 + p_2^2 \tag{5.53}$$

на параллелотопе $[\mathbf{p}] = [-2, 4]^{\times 2}$ *при ограничениях*

$$\begin{cases} 1 - 16p_1^2 - 25p_2^2 \leqslant 0, \\ 13p_1^3 - 145p_1 + 85p_2 - 400 \leqslant 0, \\ p_1p_2 - 4 \leqslant 0. \end{cases} \tag{5.54}$$

Известно, что в этой задаче есть две точки глобального минимума:

$$\widehat{\mathbf{p}}(1) \simeq (-0{,}066042,\ 0{,}192895)^{\mathrm{T}} \tag{5.55}$$

и

$$\widehat{\mathbf{p}}(2) \simeq (0{,}066042,\ -0{,}192895)^{\mathrm{T}}, \tag{5.56}$$

в которых значение минимума составляет $\widehat{c} \simeq 0{,}199035$. *При* $\varepsilon_{\mathbf{p}} = \varepsilon_{\mathbf{c}} = 10^{-5}$ *после* 44 *итераций, выполненных за* 0,22 *секунды на компьютере* Pentium 90, *алгоритм Хансена вырабатывает два параллелотопа, которые включают эти две точки глобального минимума с точностью, равной* 10^{-10}. *Глобальный минимум* $\widehat{c}$ *найден с точностью* 10^{-9}. *Для двух параллелотопов, найденных в пространстве параметров, два последних ограничения в* (5.54) *не задействованы, так как* $\widehat{u}_2 = \widehat{u}_3 = 0$. *Два других множителя Лагранжа равны* $\widehat{u}_0 \simeq 0{,}834087$ *и* $\widehat{u}_1 \simeq 0{,}165913$. ■

5.5.3. Использование метода вперед-назад

По литературе видно, что алгоритм OPTIMIZE используется с операторами сжатия, основанными на методе вперед-назад (распространении интервальных ограничений), такими как $\mathcal{C}_{\downarrow\uparrow}$, описанный в Главе 4 (см., например, [Zhou, 1996] и [van Hentenryk *et al.*, 1997]). Однако представляется странным, что сжимающие операторы, основанные на методе вперед-назад, не комбинировались с операторами, примененными Хансеном. Когда задача оптимизации нелинейна и целевая функция недифференцируема, то сжимающие операторы, основанные на линейных аппроксимациях (т. е. как все операторы, описанные в Главе 4, за исключением оператора $\mathcal{C}_{\downarrow\uparrow}$), являются в общем случае неэффективными, и только операторы, основанные на методе вперед-назад, оказываются способными сжимать большие области. Это иллюстрируется следующим примером [Jaulin, 2001б].

Пример 5.10. *Рассмотрим минимизацию функции*

$$c(\mathbf{p}) = \max_{k \in \{1, \ldots 10\}} |g(\mathbf{p}, k)|, \tag{5.57}$$

где

$$g(\mathbf{p}, k) = p_1 \exp(\frac{p_2}{4}k^2) + p_3 \exp(\frac{p_4}{4}k^2) - 20\exp(-0{,}2k^2) + \\ + 10\exp(-0{,}05k^2) + 0{,}1\sin(\frac{k^2}{4}), \tag{5.58}$$

и точка $\mathbf{p} = (p_1, p_2, p_3)^{\mathrm{T}}$ *принадлежит параллелотопу*

$$[\mathbf{p}] = [-60,\ 60] \times [-1,\ 0] \times [-60,\ 60] \times [-1,\ 0]. \tag{5.59}$$

Перестановка p_1 с p_3 и p_2 с p_4 не изменяет функцию $c(\mathbf{p})$. Таким образом, множество решения оказывается симметричным относительно плоскости ($p_1 - p_3 = 0$, $p_2 - p_4 = 0$). Эта задача плохо обусловлена и недифференцируема, и классические точечные локальные методы встречаются с трудностями при поиске даже точек локальных минимумов [Jaulin, 2001б]. *Кроме того, они могут не почувствовать, что задача имеет две точки глобального минимума. При подключении одного сжимающего оператора $\mathcal{C}_{\downarrow\uparrow}$, связанного с ограничением ($c(\mathbf{p}) \leqslant \overline{c}$), алгоритм* OPTIMIZE *на компьютере* Pentium 90 *за* 1,7 *секунды после* 109 *бисекций находит, что глобальный минимум лежит в интервале* [0,0653, 0,657]. *Список $\mathcal{L}$, выданный алгоритмом* OPTIMIZE, *содержит* 44 *маленьких параллелотопа, и каждый из них таков, что его образ по целевой функции включается в интервал* [0,0653, 0,0657]. *Эти параллелотопы можно разделить на две группы, связанные с каждой точкой глобального минимума.* ■

5.6. Минимаксная оптимизация

Рассмотрим теперь сложную задачу [Du, 1995] нахождения интервальной оценки вещественного числа c_n, определенного следующей последовательностью задач оптимизации:

$$
\begin{array}{ll}
c_1(\mathbf{p}_2, \ldots, \mathbf{p}_n) & = \max_{\mathbf{p}_1 \in [\mathbf{p}_1]} c_0(\mathbf{p}_1, \ldots, \mathbf{p}_n), \\
& \text{при условии } (\mathbf{p}_1, \ldots, \mathbf{p}_n) \in \mathbb{S}(1), \\
c_2(\mathbf{p}_3, \ldots, \mathbf{p}_n) & = \min_{\mathbf{p}_2 \in [\mathbf{p}_2]} c_1(\mathbf{p}_2, \ldots, \mathbf{p}_n), \\
& \text{при условии } (\mathbf{p}_2, \ldots, \mathbf{p}_n) \in \mathbb{S}(2), \\
\vdots & \vdots \qquad\qquad \vdots \\
c_i(\mathbf{p}_{i+1}, \ldots, \mathbf{p}_n) & = (-1)^i \min_{\mathbf{p}_i \in [\mathbf{p}_i]} (-1)^i c_{i-1}(\mathbf{p}_i, \ldots, \mathbf{p}_n), \\
& \text{при условии } (\mathbf{p}_i, \ldots, \mathbf{p}_n) \in \mathbb{S}(i), \\
\vdots & \vdots \qquad\qquad \vdots \\
c_{n-1}(\mathbf{p}_n) & = (-1)^{n-1} \min_{\mathbf{p}_{n-1} \in [\mathbf{p}_{n-1}]} (-1)^{n-1} c_{n-2}(\mathbf{p}_{n-1}, \mathbf{p}_n), \\
& \text{при условии } (\mathbf{p}_{n-1}, \mathbf{p}_n)) \in \mathbb{S}(n-1), \\
c_n & = (-1)^n \min_{\mathbf{p}_n \in [\mathbf{p}_n]} (-1)^n c_{n-1}(\mathbf{p}_n), \\
& \text{при условии } \mathbf{p}_n) \in \mathbb{S}(n),
\end{array}
\tag{5.60}
$$

где $c_0(\mathbf{p}_1, \ldots, \mathbf{p}_n)$ – известная функция от n векторов $\mathbf{p}_1, \ldots, \mathbf{p}_n$. Множество $\mathbb{S}(i)$ связано с ограничениями i-й задачи оптимизации.

При i четном $(-1)^i = 1$ имеем

$$
\begin{array}{l}
c_i(\mathbf{p}_{i+1}, \ldots, \mathbf{p}_n) = \min_{\mathbf{p}_i \in [\mathbf{p}_i]} c_{i-1}(\mathbf{p}_i, \ldots, \mathbf{p}_n), \\
\qquad\qquad \text{при условии } (\mathbf{p}_i, \ldots, \mathbf{p}_n) \in \mathbb{S}(i).
\end{array}
\tag{5.61}
$$

При i нечетном $(-1)^i = -1$ имеем

$$
\begin{array}{l}
c_i(\mathbf{p}_{i+1}, \ldots, \mathbf{p}_n) = \max_{\mathbf{p}_i \in [\mathbf{p}_i]} c_{i-1}(\mathbf{p}_i, \ldots, \mathbf{p}_n), \\
\qquad\qquad \text{при условии } (\mathbf{p}_i, \ldots, \mathbf{p}_n) \in \mathbb{S}(i).
\end{array}
\tag{5.62}
$$

Примером такой задачи минимизации является задача гарантированного оценивания величины c_3:

$$
\begin{array}{lccccc}
c_3 = & \min & \max & \min & p_1(p_2 + p_3). & \\
& p_3 \in [-1, 2] & p_2 \in [-1, 1] & p_1 \in [0, 10] & & \\
& \sin(p_3) \geqslant 0 & p_3^2 + p_2 \geqslant 2 & p_1^2 + p_2 p_3 \geqslant 1 & & \\
& & & & \overbrace{\qquad\qquad}^{c_0(p_1, p_2, p_3)} & \\
& & & \overbrace{\qquad\qquad\qquad}^{c_1(p_2, p_3)} & & \\
& & \overbrace{\qquad\qquad\qquad\qquad\qquad}^{c_2(p_3)} & & &
\end{array}
$$

Для простоты будем предполагать, что $c_i(\mathbf{p}_{i+1}, \ldots, \mathbf{p}_n)$ существует для всех $i \in \{0, \ldots, n-1\}$ и для всех $\mathbf{p}_{i+1}, \ldots, \mathbf{p}_n$. Это предположение, удовлетворяемое во всех прикладных задачах, рассматриваемых в данной книге, влечет, что для любых векторов $(\mathbf{p}_{i+1}, \ldots, \mathbf{p}_n)$ существует по крайней мере один такой вектор $\mathbf{p}_i$, что $(\mathbf{p}_i, \ldots, \mathbf{p}_n) \in \mathbb{S}(i)$.

Хотя многие прикладные задачи могут быть приведены к этой форме, но, как показано в Главе 8, минимаксная задача привлекала слабое внимание специалистов в области интервального анализа [Zuhe *et al.*, 1990; Didrit, 1997; Wolfe, 1999; Jaulin, 2001б]. Параграф 5.6.1 посвящен случаю без ограничений и показывает, как получается сходящаяся функция включения для c_{i-1}. Случай с ограничениями рассматривается в параграфе 5.6.2. Параграф 5.6.3 посвящен связанным между собой задачам работы с кванторами.

5.6.1. Случай отсутствия ограничений

Предположим, что $\mathbb{S}(i) \in \mathbb{R}^{\dim \mathbb{S}(i)}$ для $i = 0, \ldots, n-1$. Временно возьмем i четным, так, что

$$
c_i(\mathbf{p}_{i+1}, \ldots, \mathbf{p}_n) = \min_{\mathbf{p}_i \in [\mathbf{p}_i]} c_{i-1}(\mathbf{p}_i, \ldots, \mathbf{p}_n). \tag{5.63}
$$

Поясним теперь, как получать функцию включения для $c_i(\mathbf{p}_{i+1}, \ldots, \mathbf{p}_n)$ на основе функции включения для $c_{i-1}(\mathbf{p}_i, \ldots, \mathbf{p}_n)$. Принимая во внимание, что имеется некоторая функция включения для $c_0(\mathbf{p}_1, \ldots, \mathbf{p}_n)$, мы получим некоторую функцию включения для каждой из функций c_i, в том числе c_n, как определено в (5.60).

Обозначим $(\mathbf{p}_{i+1}, \ldots, \mathbf{p}_n)$ как $\mathbf{q}_{i+1}$. При этом уравнение (5.63) принимает вид:

$$c_i(\mathbf{q}_{i+1}) = \min_{\mathbf{p}_i \in [\mathbf{p}_i]} c_{i-1}(\mathbf{p}_i, \mathbf{q}_{i+1}). \tag{5.64}$$

Предлагаемый метод основан на следующей теореме.

Теорема 5.2. *Если* $[c_{i-1}]([\mathbf{p}_i], [\mathbf{q}_{i+1}])$ *является сходящейся функцией включения для* $c_{i-1}(\mathbf{p}_i, \mathbf{q}_{i+1})$, *и если* $[\mathbf{p}_i](k), k \in \{1, \ldots, \overline{k}\}$ *— некоторое разбиение* $[\mathbf{p}_i]$, *такое, что*

$$\forall k \in \{1, \ldots, \overline{k}\}, \quad w([\mathbf{p}_i](k)) \leqslant w([\mathbf{q}_{i+1}]), \tag{5.65}$$

то

$$c_i([\mathbf{q}_{i+1}]) \triangleq \min_{k \in \{1, \ldots, \overline{k}\}} [c_{i-1}]([\mathbf{p}_i](k), [\mathbf{q}_{i+1}]) \tag{5.66}$$

является сходящейся функцией включения для $c_i(\mathbf{q}_{i+1})$. ■

Доказательство. Первая часть доказательства устанавливает, что интервал $[c_i]$, определяемый по (5.66), является функцией включения для функции c_i. Во второй части доказывается, что $[c_i]$ сходится. Определим

$$c_i^{(k)}(\mathbf{q}_{i+1}) \triangleq \min_{\mathbf{p}_i \in [\mathbf{p}_i](k)} c_{i-1}(\mathbf{p}_i, \mathbf{q}_{i+1}). \tag{5.67}$$

Из (5.63) с учетом того, что параллелотопы $[\mathbf{p}_i](k)$ образуют некоторое разбиение параллелотопа $[\mathbf{p}_i]$, имеем

$$c_i(\mathbf{q}_{i+1}) = \min_{k \in \{1, \ldots, \overline{k}\}} c_i^{(k)}(\mathbf{q}_{i+1}). \tag{5.68}$$

Теперь, по (5.67) оценим интервал

$$c_i^k(\mathbf{q}_{i+1}) \in c_{i-1}([\mathbf{p}_i](k), \mathbf{q}_{i+1}), \tag{5.69}$$

который является подмножеством множества $[c_{i-1}]([\mathbf{p}_i](k), [\mathbf{q}_{i+1}])$, если $\mathbf{q}_{i+1} \in [\mathbf{q}_{i+1}]$. Следовательно, из (5.68) имеем

$$c_i([\mathbf{q}_{i+1}]) \subset \min_{k \in \{1, \ldots, \overline{k}\}} [c_{i-1}]([\mathbf{p}_i](k), [\mathbf{q}_{i+1}]). \tag{5.70}$$

Для доказательства сходимости $[c_i]$ предположим, что ширина параллелотопа $[\mathbf{q}_{i+1}]$ сходится к нулю. Из (5.65) тогда следует, что ширина всех параллелотопов $[\mathbf{p}_i](k)$ также сходится к нулю, и, следовательно, это происходит и с шириной интервала $[c_{i-1}]([\mathbf{p}_i](k), [\mathbf{q}_{i+1}])$ для всех $k \in \{1, \ldots, \overline{k}\}$. Таким образом, ширина интервала $[c_i]([\mathbf{q}_{i+1}])$ сходится к нулю. ■

Для некоторого заданного точечного вектора $\mathbf{q}_{i+1}$ рис. 5.14, *а* показывает интерпретацию $c_i(\mathbf{q}_{i+1})$. На рис. 5.14, *б* показана интервальная функция $c_{i-1}(\mathbf{p}_i, [\mathbf{q}_{i+1}])$. Функция включения (5.66), полученная разбиением параллелотопа $[\mathbf{p}_i]$, показана на рис. 5.14, *в*.

ЗАМЕЧАНИЕ 5.5. Оператор минимизации min в (5.66) должен интерпретироваться как интервальное расширение оператора min на вещественные числа. Например,

$$\begin{aligned} &\min\{[1,\ 6], [3,\ 5], [0,\ 10], [-1,\ 7], [9,\ 9]\} \\ &\quad = [\min\{1,\ 3,\ 0,\ -1,\ 9\},\ \min\{6,\ 5,\ 10,\ 7,\ 9\}] = [-1,\ 5]. \end{aligned} \tag{5.71}$$

■

ЗАМЕЧАНИЕ 5.6. Тривиально расширить Теорему 5.2 так, чтобы она работала и в случае, когда i — нечетное число. Если имеется сходящаяся функция включения для функции c_0, то рекурсивное применение Теоремы 5.2 дает функции включения для всех функций c_i. ■

Алгоритм, приведенный а табл. 5.8, взят из работы [Didrit, 1997] и является рекурсивной реализацией функции включения для $c(\mathbf{q}_{i+1})$, основанной на Теореме 5.2. Предполагается, что известна некоторая функция включения для c_0, и описание алгоритма излагается сперва для четных i. Положительный коэффициент ε характеризует ширину минимального параллелотопа, который еще будет подвергаться бисекции для обновления разбиения параллелотопа $[\mathbf{p}_i]$, так, чтобы это разбиение удовлетворяло условию (5.65). Поскольку i полагается четным, то верхняя граница $\overline{c}_i$ для $c_i([\mathbf{q}_{i+1}])$ задается следующим образом:

$$\min_{k \in \{1, \ldots, \overline{k}\}} (\mathrm{ub}([c_{i-1}]([\mathbf{p}_i](k),\ [\mathbf{q}_{i+1}]))), \tag{5.72}$$

см. рис. 5.14, *в*. Здесь символом $\mathrm{ub}([c](\cdot))$ обозначается верхняя граница интервала значений функции $c(\cdot)$.

Эта верхняя граница используется для отбрасывания всех параллелотопов $[\mathbf{p}_i](k)$, которые удовлетворяют условию

$$\mathrm{lb}([c_{i-1}]([\mathbf{p}_i](k),\ [\mathbf{q}_{i+1}])) > \overline{c}_i \tag{5.73}$$

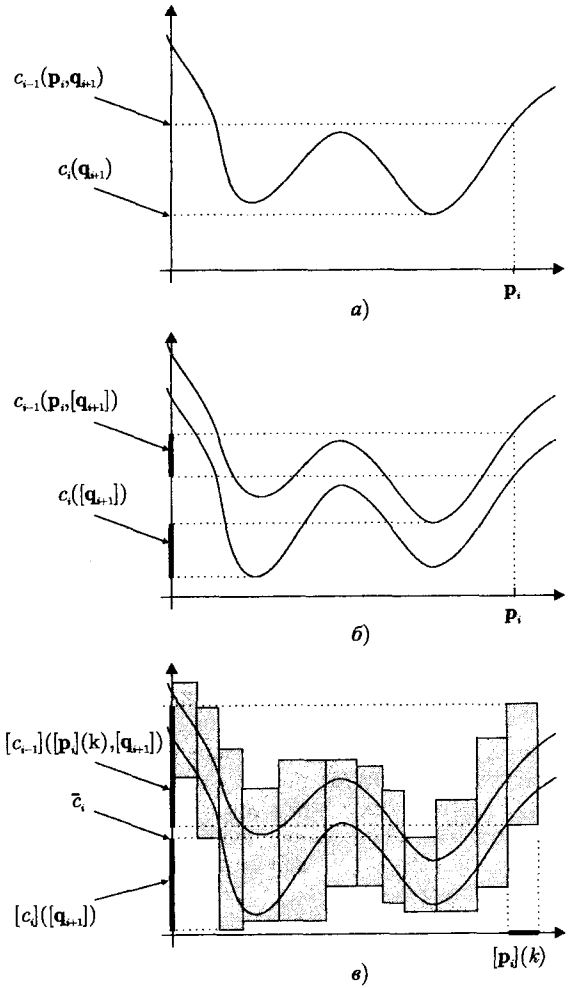


Рис. 5.14. Включение функции $c_i([\mathbf{q}_{i+1}])$ в минимаксном алгоритме (при отсутствии ограничений); $\mathbf{q}_{i+1}$ сохраняется для $(\mathbf{p}_{i+1},\ldots,\mathbf{p}_n)$: *а)* определение c_i по c_{i-1} для точечных аргументов; *б)* оценивание c_i по c_{i-1}, когда все аргументы — параллелотопы, за исключением $\mathbf{p}_i$; *в)* интервальное включение c_i и c_{i-1}, когда все их аргументы — параллелотопы

(см. Шаг 7). Здесь символом $\mathrm{lb}([c](\cdot))$ обозначается нижняя граница интервала значений функции $c(\cdot)$.

На рис. 5.14, *в* самый левый и самый правый параллелотопы удовлетворяют этому условию и, следовательно, могут быть исключены из рассмотрения. Этого, в принципе, можно было бы и не делать, но проверка условия (5.73) оказывается полезным упрощением для исключения параллелотопов без их бисекции до ширины ε, когда это возможно. При первом вызове, алгоритм рассчитывает $[c_n](\varepsilon)$, а на самом последнем вызове, (при рекурсивных вычислениях) оценивает

$$[c_0]([\mathbf{p}_1],[\mathbf{q}_2]) = [c_0]([\mathbf{p}_1],\ldots,[\mathbf{p}_n]). \tag{5.74}$$

ЗАМЕЧАНИЕ 5.7. В табл. 5.8 параллеллотоп $[\mathbf{p}_i]$ сохраняется для текущего паралелотопа $[\mathbf{p}_i](k)$ этого разбиения. ■

ЗАМЕЧАНИЕ 5.8. В простой версии алгоритма сжимающие операторы не используются, но их нужно включать для повышения эффективности. ■

Следующие строки должны заменять соответствующие строки табл. 5.8 для случая нечетных i:

$3'$	$\underline{c}_i := -\infty;\ \overline{c}_i := -\infty$	
	$\ldots$	
$7'$	если $(\mathrm{ub}([r]) \geqslant \underline{c}_i)$, то	
$8'$	$\underline{c}_i = \max\{\mathrm{lb}([r]),\ \underline{c}_i\};$	
$9'$	если $(\omega([\mathbf{p}_i]) < \varepsilon)$, то $\overline{c}_i = \max\{\overline{c}_i,\ \mathrm{ub}([r])\};$	

Здесь символами lb и ub обозначены, соответственно, нижняя и верхняя границы интервала $[r]$.

При этом алгоритм MINIMAX, рассчитывающий гарантированное включение c_n, просто выглядит как

Алгоритм MINIMAX(вход : $c_0,[\mathbf{p}_1],\ldots,[\mathbf{p}_n],\varepsilon$; выход : $[c_n]$)	
1	$[c_n] := [c_n](\varepsilon).$

5.6.2. Случай наличия ограничений

Предположим опять, что i — четное, поэтому

$$c_i(\mathbf{p}_{i+1},\ldots,\mathbf{p}_n) = \min_{\mathbf{p}_i\in[\mathbf{p}_i]} c_{i-1}(\mathbf{p}_i,\ldots,\mathbf{p}_n), \quad \text{при условии } (\mathbf{p}_i,\ldots,\mathbf{p}_n)\in\mathbb{S}(i). \tag{5.75}$$

Таблица 5.8. Функция включения для минимаксной задачи без ограничений

Алгоритм $[c_i]$(вход: $[\mathbf{p}_{i+1}], \ldots, [\mathbf{p}_n], \varepsilon$; выход: $[c_i]$) (i полагается четным)

```
1  если i = 0, то возврат [c_0]([p_1], ..., [p_n]);
2  Q := {[p_i]};
3  c_i := ∞;  c̄_i := ∞;
4  выполнить
5     захватить первый элемент из Q в параллелотоп [p_i];
6     рассчитать [r] := [c_{i-1}]([p_i], ..., [p_n], w([p_i]));
7     если (lb([r]) ⩽ c̄_i), то
8  c̄_i := min(ub([r]), c̄_i);
9  если (w([p_i]) < ε) то c_i := min(c_i, lb[r]));
10 иначе выполнить бисекцию параллелотопа [p_i]
   и занести результирующие параллелотопы в конец списка Q;
11 выполнять до тех пор, пока список Q ≠ ∅;
12 [c_i] := [c_i, c̄_i].
```

Снова обозначим $(\mathbf{p}_i, \ldots, \mathbf{p}_n)$ как $\mathbf{q}_{i+1}$. Уравнение (5.75) тогда принимает вид:

$$c_i(\mathbf{q}_{i+1}) = \min_{\mathbf{p}_i \in [\mathbf{p}_i]} c_{i-1}(\mathbf{p}_i, \mathbf{q}_{i+1}), \quad \text{при условии } (\mathbf{p}_i, \ldots, \mathbf{q}_{i+1}) \in \mathbb{S}(i). \tag{5.76}$$

В случае задачи с ограничениями Теорема 5.2. заменяется на следующую теорему, которая тоже используется для получения функций включения для каждой функции c_i, в том числе и c_n.

Теорема 5.3. *Предположим, что i — четное, и*

- *если $[c_{i-1}]([\mathbf{p}_i], [\mathbf{q}_{i+1}])$ является функцией включения для $c_{i-1}(\mathbf{p}_i, \mathbf{q}_{i+1})$,*
- *если $\{[\mathbf{p}_i](k), k \in \{1, \ldots, \overline{k}\}\}$ является разбиением параллелотопа $[\mathbf{p}_i]$, и*
- *если $[t_{\mathbb{S}(i)}]$ является проверкой включения для множества $\mathbb{S}(i)$,*

(т. е. $[t_{\mathbb{S}(i)}]([\mathbf{p}_i], [\mathbf{q}_{i+1}]) = 1 \Rightarrow ([\mathbf{p}_i], [\mathbf{q}_{i+1}]) \subset \mathbb{S}(i)$ *и* $[t_{\mathbb{S}(i)}]([\mathbf{p}_i], [\mathbf{q}_{i+1}]) = 0 \Rightarrow ([\mathbf{p}_i], [\mathbf{q}_{i+1}]) \cap \mathbb{S}(i) = \varnothing$), *то нижняя граница для $c_i([\mathbf{q}_{i+1}])$ равна*

$$\underline{c}_i([\mathbf{q}_{i+1}]) = \min_{\substack{k \in \{1, \ldots \overline{k}\} \\ [t_{\mathbb{S}(i)}]([\mathbf{p}_i], [\mathbf{q}_{i+1}]) \neq 0}} \mathrm{lb}([c_{i-1}]([\mathbf{p}_i](k), [\mathbf{q}_{i+1}])), \tag{5.77}$$

а верхняя граница для $c_i([\mathbf{q}_{i+1}])$ равна

$$\overline{c}_i([\mathbf{q}_{i+1}]) = \min_{\substack{k \in \{1, \ldots \overline{k}\} \\ [t_{\mathbb{S}(i)}]([\mathbf{m}_i](k), [\mathbf{q}_{i+1}]) = 1}} \mathrm{ub}([c_{i-1}]([\mathbf{p}_i](k), [\mathbf{q}_{i+1}])), \tag{5.78}$$

где $\mathbf{m}_i = \mathrm{mid}([\mathbf{p}_i](k))$. ■

Доказательство. Доказательство утверждения для нижней границы тривиально, и далее приводится только доказательство утверждения для верней границы. Если $[t_{\mathbb{S}(i)}]([\mathbf{m}_i](k), [\mathbf{q}_{i+1}]) = 1$, то отсюда следует, что для любого $\mathbf{q}_{i+1} \in [\mathbf{q}_{i+1}]$, $(\mathbf{m}_i(k), [\mathbf{q}_{i+1}]) \in \mathbb{S}(i)$. Тогда, в соответствии с (5.75)

$$\forall \mathbf{q}_{i+1} \in [\mathbf{q}_{i+1}],\ c_i(\mathbf{q}_{i+1}) \leqslant c_{i-1}(\mathbf{m}_i(k), \mathbf{q}_{i+1}). \tag{5.79}$$

Верхняя граница $\overline{c}_i(k)$ для $[c_{i-1}]([\mathbf{p}_i](k), [\mathbf{q}_{i+1}])$ является также верхней границей для $[c_{i-1}](\mathbf{m}_i(k), [\mathbf{q}_{i+1}])$ и, следовательно, верхней границей для $c_i([\mathbf{q}_{i+1}])$. Наименьшая из этих $\overline{k}$ верхних границ, даваемых (5.78), является также верхней границей для $c_i([\mathbf{q}_{i+1}])$. ■

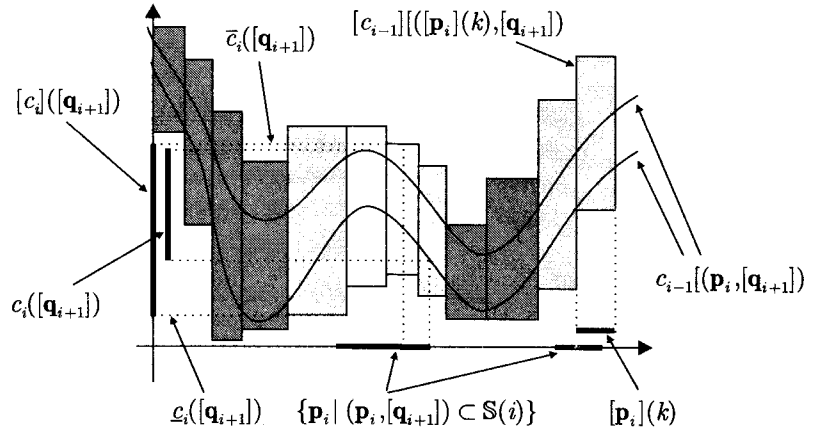


Рис. 5.15. Иллюстрация к Теореме 5.3, применяемой к минимаксной оптимизации с ограничениями

Рис. 5.15 иллюстрирует эту теорему. Праллелотопы, отмеченные темно-серой заливкой, соответствуют $[\mathbf{p}_i](k)$, таким, что $[t_{\mathbb{S}(i)}]([\mathbf{p}_i](k), [\mathbf{q}_{i+1}]) = 0$. Эти параллелотопы не учитываются при поиске границ $\underline{c}_i([\mathbf{q}_{i+1}])$

и $\overline{c}_i([\mathbf{q}_{i+1}])$. Параллелотопы, отмеченные светло-серой заливкой, связаны с параллелотопами $[\mathbf{p}_i](k)$, для которых $[t_{\mathbb{S}(i)}]([\mathbf{p}_i](k),\ [\mathbf{q}_{i+1}]) = [0,\ 1]$. Они должны рассматриваться при вычислении нижних границ $\underline{c}_i([\mathbf{q}_{i+1}])$. Параллелотопы, отмеченные белым, удовлетворяют условию $[t_{\mathbb{S}(i)}](\mathrm{mid}([\mathbf{p}_i](k)), [\mathbf{q}_{i+1}]) = 1$. Эти параллелотопы учитываются при вычислении границ $\underline{c}_i([\mathbf{q}_{i+1}])$ и $\overline{c}_i([\mathbf{q}_{i+1}])$.

Табл. 5.9 показывает алгоритм, реализующий функцию включения в соответствии с Теоремой 5.3. Алгоритм требует наличия начальной функции включения для функции c_0 и процедуры проверки принадлежности множествам $\mathbb{S}(i)$. Если обнаружено, что набор параллелотопов $([\mathbf{p}_i], \ldots, [\mathbf{p}_n])$ лежит вне множества $\mathbb{S}(i)$, то текущий параллелотоп $[\mathbf{p}_i]$ удаляется (см. Шаг 6). Текущая верхняя граница $\overline{c}_i$ обновляется на Шаге 10, чтобы позволить найти оценку (5.78) и возможную оценку параллелотопов (см. Шаг 8). Текущая нижняя граница $\underline{c}_i$ обновляется на Шаге 11, чтобы позволить найти оценку (5.77). Напомним, что в данном описании алгоритма предполагается, что i — четное. Подстройка алгоритма для работы с нечетными i — тривиальна. Заметим, что Замечания 5.7 и 5.8 здесь также имеют силу. Алгоритм MINIMAX, рассчитывающий гарантированное включение c_n, тот же, что и в случае задачи без ограничений.

Таблица 5.9. Функция включения для минимаксной задачи с ограничениями

Алгоритм $[c_i]$(вход: $[\mathbf{p}_{i+1}], \ldots, [\mathbf{p}_n], \varepsilon$; выход: $[c_i]$) (для i четного)
1 если $i = 0$, то возврат $[c_0]([\mathbf{p}_1], \ldots, [\mathbf{p}_n])$;
2 $\mathcal{Q} := \{[\mathbf{p}_i]\}$;
3 $\underline{c}_i := \infty$; $\overline{c}_i := \infty$;
4 выполнить
5 захватить первый элемент из $\mathcal{Q}$ в параллелотоп $[\mathbf{p}_i]$;
6 если $[t_{\mathbb{S}(i)}]([\mathbf{p}_i], \ldots, [\mathbf{p}_n]) \neq 0$, то
7 рассчитать $[r] := [c_{i-1}]([\mathbf{p}_i], \ldots [\mathbf{p}_n],\ w([\mathbf{p}_i]))$;
8 если $(\mathrm{lb}([r]) \leqslant \overline{c}_i)$, то
9 если $[t_{\mathbb{S}(i)}](\mathrm{mid}([\mathbf{p}_i]), [\mathbf{p}_{i+1}], \ldots, [\mathbf{p}_n]) = 1$, то
10 $\overline{c}_i := \min(\mathrm{ub}([r]), \overline{c}_i)$;
11 если $(w([\mathbf{p}_i]) < \varepsilon)$, то $\underline{c}_i := \min(\underline{c}_i, \mathrm{lb}[r]))$;
12 иначе выполнить бисекцию параллелотопа $[\mathbf{p}_i]$ и занести результирующие параллелотопы в конец списка $\mathcal{Q}$;
13 выполнять до тех пор, пока список $\mathcal{Q} \neq \varnothing$;
14 $[c_i] := [\underline{c}_i, \overline{c}_i]$.

Здесь символами lb и ub обозначены, соответственно, нижняя и верхняя границы интервала $[r]$.

5.6.3. Работа с кванторами

В этом параграфе рассматриваются задачи с неравенствами, использующими квантор существования $\exists$ и квантор общности $\forall$. Обзоры доступных методов можно найти в [Mishra, 1993; Caviness и Jonson, 1998; Dorato, 2000], а по их применениям — в [Ioakiminidis, 1999; El Kahoi и Weber, 2000]. Этот класс задач очень важен, в частности, в теории управления [Jaulin и Walter, 1996; Liska и Stainberg, 1996; Stainberg и Liska, 1996; Dorado *et. al.*, 1997; Hong *et. al.*, 1997; Jirstrand, 1997; Neubacher, 1997]. Некоторые примеры, связанные с робастным управлением, будут рассмотрены в Главе 7. Интервальный анализ дает весьма перспективный аппарат для решения таких проблем [Jaulin и Walter, 1996; Benhamou и Goualard, 2000; Ratschan, 2000a; 2000б]. В настоящем параграфе показывается, что задачи, в которых используются кванторы $\exists$ и $\forall$, тесно связаны с задачами на минимакс и что алгоритм MINIMAX может быть использован для их решения. Например, доказательство того, что

$$\forall p_3 \in [1,\ 3],\ \exists p_2 \in [1,\ 2],\ \forall p_1 \in [0,\ 1],\ p_1 + p_2 p_3 \leqslant 1, \tag{5.80}$$

сводится к доказательству того, что

$$\max_{p_3 \in [1,\ 3]} \min_{p_2 \in [1,\ 2]} \max_{p_1 \in [0,\ 1]} p_1 + p_2 p_3 \leqslant 1. \tag{5.81}$$

В более общем виде, доказательство того, что

$$\begin{pmatrix} \forall \mathbf{p}_3 \in [\mathbf{p}_3],\ \mathbf{p}_3 \in \mathbb{S}(3) \\ \exists \mathbf{p}_2 \in [\mathbf{p}_2],\ (\mathbf{p}_2, \mathbf{p}_3) \in \mathbb{S}(2) \\ \forall \mathbf{p}_1 \in [\mathbf{p}_1], (\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3) \in \mathbb{S}(1) \end{pmatrix} \text{ такие, что } c_0(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3) \geqslant 0, \tag{5.82}$$

может быть сведено к доказательству того, что

$$\min_{\substack{\mathbf{p}_3 \in [\mathbf{p}_3] \\ \mathbf{p}_3 \in \mathbb{S}(3)}} \max_{\substack{\mathbf{p}_2 \in [\mathbf{p}_2] \\ (\mathbf{p}_2, \mathbf{p}_3) \in \mathbb{S}(2)}} \min_{\substack{\mathbf{p}_1 \in [\mathbf{p}_1] \\ (\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3) \in \mathbb{S}(1)}} c_0(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3) \geqslant 0. \tag{5.83}$$

Таким образом, алгоритм MIMIMAX может быть использован для доказательства (или опровержения) (5.82). Разумеется, данный алгоритм необходимо несколько модифицировать, если текущее включение глобального оптимума c_3 имеет положительную нижнюю границу (или отрицательную верхнюю границу).

Рассмотрим теперь задачу описания множества $\mathbb{S}$ неравенствами, содержащими кванторы $\forall$ и $\exists$. Для простоты мы будем полагать, что

$$\mathbb{S} = \{\mathbf{p}_3 \in [\mathbf{p}_3](0) \mid (\forall \mathbf{p}_2 \in [\mathbf{p}_2],\ \exists \mathbf{p}_1 \in [\mathbf{p}_1]) \mid g(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3) \geqslant 0\}, \quad (5.84)$$

но будут также рассмотрены и множества более общего вида. Множество $\mathbb{S}$ может быть описано в следующей эквивалентной форме:

$$\mathbb{S} = \left\{ \mathbf{p}_3 \in [\mathbf{p}_3](0) \mid \min_{\mathbf{p}_2 \in [\mathbf{p}_2]} \max_{\mathbf{p}_1 \in [\mathbf{p}_1]} g(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3) \geqslant 0 \right\}. \quad (5.85)$$

Некоторый подпараллелотоп $[\mathbf{p_3}]$ параллелотопа $[\mathbf{p_3}](0)$ лежит внутри множества $\mathbb{S}$, если

$$\min_{\mathbf{p}_3 \in [\mathbf{p}_3]} \min_{\mathbf{p}_2 \in [\mathbf{p}_2]} \max_{\mathbf{p}_1 \in [\mathbf{p}_1]} g(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3) \geqslant 0, \quad (5.86)$$

и вне множества $\mathbb{S}$, если

$$\max_{\mathbf{p}_3 \in [\mathbf{p}_3]} \min_{\mathbf{p}_2 \in [\mathbf{p}_2]} \max_{\mathbf{p}_1 \in [\mathbf{p}_1]} g(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3) < 0. \quad (5.87)$$

Определим интервальную функцию

$$[h]([\mathbf{p}_3]) = \left[\min_{\mathbf{p}_3 \in [\mathbf{p}_3]} \min_{\mathbf{p}_2 \in [\mathbf{p}_2]} \max_{\mathbf{p}_1 \in [\mathbf{p}_1]} g(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3), \right. \\ \left. \max_{\mathbf{p}_3 \in [\mathbf{p}_3]} \min_{\mathbf{p}_2 \in [\mathbf{p}_2]} \max_{\mathbf{p}_1 \in [\mathbf{p}_1]} g(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3) \right]. \quad (5.88)$$

Включение $[h_{\mathbf{e}}]([\mathbf{p}_3])$ интервала $[h]([\mathbf{p}_3])$ может быть получено запуском алгоритма MINIMAX для нижней и верхней границ. Параметр точности ε в алгоритме MINIMAX можно взять равным $\varepsilon = w([\mathbf{p}_3])$, как это предложено в работе [Jaulin и Walter, 1996]. Проверка включения для множества $\mathbb{S}$ выполняется как

$$\begin{cases} [t]([\mathbf{p}_3]) = 1, & \text{если } \mathrm{lb}([h_{\mathbf{e}}]([\mathbf{p}_3])) \geqslant 0, \\ [t]([\mathbf{p}_3]) = 0, & \text{если } \mathrm{ub}([h_{\mathbf{e}}]([\mathbf{p}_3])) < 0, \\ [t]([\mathbf{p}_3]) = [0,1], & \text{в остальных случаях.} \end{cases} \quad (5.89)$$

Алгоритм SIVIA (см. табл. 3.2) теперь может быть использован, чтобы получить аппроксимацию снизу (изнутри) и аппроксимацию сверху множества $\mathbb{S}$ с помощью покрытий.

Возможным применением алгоритма является описание проекции некоторого множества, определенного нелинейными неравенствами, например, проекции трехмерного шара

$$\mathbb{O} = \{(x, y, z) \in \mathbb{R}^3 \mid x^2 + y^2 + z^2 \leqslant 1\} \quad (5.90)$$

на плоскость (x, y), задаваемую как

$$\mathbb{S} = \{(x, y) \in \mathbb{R}^2 \mid \exists z,\ x^2 + y^2 + z^2 \leqslant 1\}. \quad (5.91)$$

Здесь, на выходе алгоритма результирующее множество является множеством всех точек из $\mathbb{R}^3$, которые проектируются внутрь границы $\partial\mathbb{S}$ множества $\mathbb{S}$. Это множество имеет размерность, равную размерности границы $\partial\mathbb{S}$, т. е. — единице. С другой стороны, если описание множества $\mathbb{S}$ выполнено запуском программы SIVIA для описания множества $\mathbb{O}$ и далее проектированием выходных параллелотопов на плоскость (x, y), то результирующее множество должно быть границей множества $\mathbb{O}$ (размерность которого равна двум вместо единицы как у множества $\partial\mathbb{S}$), поэтому сложность вычислений будет выше. Кроме того, этот подход не обеспечит построения внутренней аппроксимации множества $\mathbb{S}$, в отличие от предлагаемого нами подхода.

5.7. Множества уровня целевой функции

Рассмотрим целевую функцию $c := \mathbb{R}^n \to \mathbb{R},\ \mathbf{p} \mapsto c(\mathbf{p})$, для которой функция включения $[c]$ предполагается известной. Задача, к которой мы теперь обратимся, состоит в описании m *множеств уровня* $\mathbb{L}_1, \ldots, \mathbb{L}_m$, связанных с m заданными значениями $c_1, \ldots, c_m$ целевой функции, которые определены как

$$\mathbb{L}_i \triangleq \{\mathbf{p} \in [\mathbf{p}] \mid c(\mathbf{p}) = c_i\},\ i = 1, \ldots, m, \quad (5.92)$$

где $[\mathbf{p}]$ — интересующий нас параллелотоп. Табл. 5.10 представляет алгоритм ISOCRIT [Didrit *et al.*, 1997], реализующий такое описание. По завершению работы алгоритма ISOCRIT, объединение параллелотопов, занесенное в список $\mathbb{L}$, содержит m множеств уровня $\mathbb{L}$.

Структура алгоритма ISOCRIT подобна структуре алгоритма SIVIA, описанного в Главе 3. На Шаге 4 вычисляется наименьший интервал, содержащий пересечение $[c]([\mathbf{p}])$ с $(c_1, \ldots, c_m)$. Сжатие выполняется на Шаге 5, который может привести к пустому множеству, если уровня c_i нет в $[c]([\mathbf{p}])$.

В таком случае параллелотоп $[\mathbf{p}]$ не пересекается с каким-нибудь множеством уровня и исключается. На Шагах 7 и 8, перед занесением параллелотопа $[\mathbf{p}]$ в список $\mathcal{L}$, необходимо также, чтобы интервал $[h]$, рассчитанный на Шаге 4, был вырожденным. Когда такое условие появляется, то $[\mathbf{p}]$ является маленьким параллелотопом, который пересекается, самое большое, с одним из множеств уровня. Значение целевой функции, связанное с этим множеством уровня, записывается с $[\mathbf{p}]$ в список $\mathcal{L}$ для дальнейшей обработки.

Таблица 5.10. Алгоритм построения множеств уровня

Алгоритм ISOCRIT(вход : $c(\cdot), c_1, \ldots, c_m, [\mathbf{p}], \varepsilon$; выход : $\mathcal{L}$)

```
1  Q := {[p]}; L := ∅;
2  выполнить
3      захватить первый параллелотоп из Q в параллелотоп [p];
4      [h] := [[c]([p]) ∩ {c_1, ..., c_m}];
5      сжать [p] при ограничении [c]([p]) ∈ [h];
6      если [h] ≠ ∅,
7          если (w([p]) < ε) и если [h] — точка, то
8              записать пару ([p], [h]) в список L;
9          иначе
10             выполнить бисекцию [p] и записать
               результирующие параллелотопы в список Q;
11 выполнять до тех пор, пока список Q ≠ ∅.
```

Пример 5.11. *Для функции Бранина из* Примера 5.4 (с. 158) *и* Примера 5.6 (с. 162), *для* $c_1 = 1, c_2 = 10$, $c_3 = 50, c_4 = 150$ *и* $\varepsilon = 0{,}2$ *алгоритм* ISOCRIT *дает покрытие, представленное на* рис. 5.16 *после* 6951 *итераций при времени счета* 0,8 *секунд на компьютере* Pentium 90. *Рисунок показывает существование по крайне мере трех точек локального минимума.* ■

5.8. Выводы

В настоящей главе рассмотрены разрешающие операторы, дающие гарантированные решения трудных нелинейных задач. Понятия проверки включения и множества операторов сжатия позволили нам сосредоточить внимание больше на пространстве поиска, чем на технических средствах,

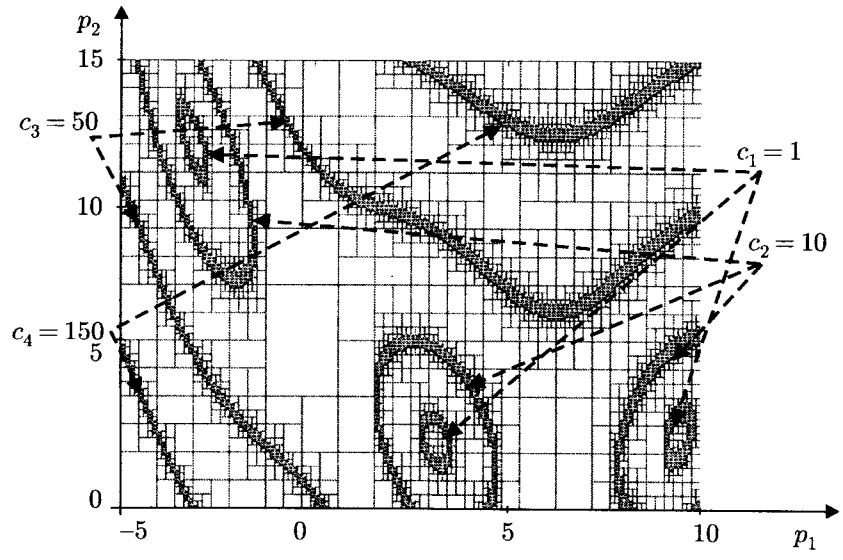


Рис. 5.16. Покрытие, полученное алгоритмом ISOCRIT для функции Бранина

которые необходимо использовать (интервальный анализ, анализ совместности и задача выполнения ограничений).

Интервальный анализ представлен в этих разрешающих операторах через понятия сжимающего оператора и проверки включения. Вот почему в литературе они часто называются *интервальными разрешающими операторами*.

Эта глава завершает Раздел II, посвященный интервальным методам. В главе даны алгоритмы решения задач, являющихся сутью инженерной практики, таких как поиск всех решений систем уравнений и неравенств, и оптимизация целевых функций различных типов при различных ограничениях.

Эти методы будут применяться в Разделе III к типовым инженерным задачам. Глава 6 посвящена оцениванию искомых неизвестных величин по экспериментальным данным в условиях неопределенности. Глава 7 посвящена управлению систем с неопределенностями. Глава 8 посвящена задачам из робототехники. Все усилия будут направлены на то, чтобы сделать рассматриваемые задачи понятными для читателей с целью собственных приложений.

Раздел III

Приложения

Глава 6

Оценивание

6.1. Введение

Рассмотрим некоторое множество $\mathcal{X}$ вещественных чисел $x_1, \ldots, x_n$, составляющих вектор $\mathbf{x}$. Для простоты положим, что $n = \dim \mathbf{x}$ — конечно. Переменная $\mathcal{X}$ может представлять, например:

- моменты времени, в которые происходит заданное событие (или значения, принимаемые некоторой независимой переменной);
- значения некоторого физического параметра, такого как скорость некоторой химической реакции;
- значения, принимаемые интересующей нас величиной в заданные моменты времени.

Каждая из компонент вектора $\mathbf{x}$ будет полагаться частично или полностью неизвестной, и целью оценивания является использование всех доступных сведений для получения более точной информации о численных значениях всех или некоторых из его компонент (см., например, [Walter и Pronzato, 1997]). Когда эти величины являются константами, часто говорят о *идентификации параметров*.

Предположим, что между этими переменными могут быть заданы некоторые соотношения, основанные на физических законах и гипотезах относительно рассматриваемой системы. Примерами таких соотношений являются выражения $x_1 \leqslant 3x_2$ или $x_3 = \sin(x_1 + x_2)$. Эти соотношения определяют некоторое подмножество $\mathbb{M}$ из $\mathbb{R}_n$, называемое *подмножеством допустимого множества*, которое содержит все векторы $\mathbf{x}$, удовлетворяющие всем этим соотношениям. (*Прим. перев.*: в отечественной литературе принят термин *информационное множество*.) Символ $\mathbb{M}$ выбран как напоминание того факта, что это ограниченное множество является *моделью* реальности. Заметим, что если $z(t)$ — интересующая нас величина, зависящая от времени, то эта функция z не рассматривается как переменная, но ее значения во все интересующие моменты времени образуют набор переменных. Так,

значения $z(1), z(5)$ и $z(10)$ могут являться переменными. Заметим также, что в $\mathcal{X}$ не нужно включать любую величину, которая может предполагаться известной точно.

ЗАМЕЧАНИЕ 6.1. Связи типа дифференциальных уравнений, вида $\ddot{y}(t)+3\dot{y}(t)-$ $-\sin(t)=0$ не могут быть использованы в такой форме, поскольку число n переменных берется конечным. В этой главе рассматриваются только такие системы непрерывного времени, для которых, следовательно, можно найти явные решения. ■

ЗАМЕЧАНИЕ 6.2. В подходе, рассматриваемом в этой главе, нет необходимости делать различия между входными переменными, которые известны и более-менее управляемы, и выходными переменными, которые отслеживают, как система реагирует на входные сигналы и возмущения. Таким образом, этот подход хорошо подходит для моделирования поведения, как развито в [Willems, 1986a; 1986б], где такого различия не делается. Когда это удобно, такое различение, разумеется, остается возможным. ■

Пример 6.1. *Рассмотрим систему с входом $u(t)$ и выходом $y(t)$, где t — время. Предположим, что*

$$\forall t \in \mathbb{R}^+, \quad y(t) = u(t) + \exp(-pt), \tag{6.1}$$

где p — некоторый неизвестный скалярный параметр. Предположим, что накоплено два измерения $y(t_1)$ и $y(t_2)$ выхода в моменты t_1 и t_2. Множество всех переменных, которое нас интересуют, может быть записано

$$\mathcal{X} = \{t_1, t_2, u_1, u_2, p, y_1, y_2\}, \tag{6.2}$$

где u_1, u_2, y_1 и y_2 представляют $u(t_1), u(t_2), y(t_1)$ и $y(t_2)$. Допустимым множеством является множество $\mathbb{M}$ всех значений $t_1, t_2, u_1, u_2, p, y_1, y_2$, таких, что $y_1 = u_1 + e^{-pt_1}$ и $y_2 = u_2 + e^{-pt_2}$. ■

Независимо от соотношений, определяющих множество $\mathbb{M}$, предположим, что доступна некоторая информация о значениях $\mathbf{x}$, которые этот вектор может принимать. Эта информация может приходить из измерений и по априорным сведениям о достоверности этих значений. Будут рассматриваться два способа описания этой информации.

Первый способ — описание с помощью некоторой вещественной функции $\check{c}$ в $\mathbf{x}$-пространстве, которая будет называться *априорной штрафной функцией*. По договоренности, значение функции $\check{c}(\mathbf{x})$ уменьшается, когда допустимость значения $\mathbf{x}$ возрастает. (*Прим. перев.*: принятые в отечественной литературе понятия *штрафной функции* или *функции невязки* эквивалентны термину «*априорная функция допустимости*», примененному авторами.) Априорная штрафная функция может быть получена методом максимума правдоподобия, основанным на статистической информации о природе измерительного шума, но могут подразумеваться и многие другие методы (например, методы нечеткой логики).

Второй метод описания априорной информации — использование множества $\check{\mathbb{X}}$ из $\mathbb{R}^n$, содержащего все допустимые значения вектора $\mathbf{x}$. Оно будет называться *априорным допустимым множеством*. Это множество может быть получено из технических паспортных данных измерительных датчиков; эти сведения обычно содержат информацию о максимальной ошибке измерения, паспортизованной для любого заданного диапазона работы.

Пример 6.2. *Рассмотрим снова* Пример 6.1 *и предположим, что получены приближенные значения всех переменных, описываемых как*

$$\check{x} = (\check{t}_1, \check{t}_2, \check{u}_1, \check{u}_2, \check{p}, \check{y}_1, \check{y}_2). \tag{6.3}$$

Возможная априорная штрафная функция имеет вид:

$$\begin{aligned}\check{c}(t_1, t_2, u_1, u_2, p, y_1, y_2) = {} & w_{t_1}(t_1 - \check{t}_1)^2 + w_{t_2}(t_2 - \check{t}_2)^2 + \\ & + w_{u_1}(u_1 - \check{u}_1)^2 + w_{u_2}(u_2 - \check{u}_2)^2 + \\ & + w_p(p - \check{p})^2 + \\ & + w_{y_1}(y_1 - \check{y}_1)^2 + w_{y_2}(y_2 - \check{y}_2)^2,\end{aligned} \tag{6.4}$$

где w_{x_i} — некоторый положительный коэффициент, выражающий степень доверия, связанную с приближенной величиной $\check{x}_i$. Когда значения $\check{x}_i$ недоступны, можно полагать $w_{x_i} = 0$. Заметим, что минимум функции $\check{c}$ достигается в точке $\check{x} = (\check{t}_1, \check{t}_2, \check{u}_1, \check{u}_2, \check{p}, \check{y}_1, \check{y}_2)$ и соответствующее значение функции $\check{c}$ равно нулю. В качестве альтернативы, можно принять решение описывать априорное знание о $\mathbf{x}$ априорным допустимым множеством $\check{\mathbb{X}}$, содержащим $\check{x}$, и достаточно большим, чтобы включать все значения $\mathbf{x}$, представляющиеся приемлемыми. Здесь априорное допустимое множество может быть определено как

$$\begin{aligned}\check{\mathbb{X}} &= \{\mathbf{x} = (t_1, t_2, u_1, u_2, p, y_1, y_2) \mid t_1 \in [\check{t}_1], \ldots, y_2 \in [\check{y}_2]\} \\ &= [\check{t}_1] \times [\check{t}_2] \times [\check{u}_1] \times [\check{u}_2] \times [\check{p}] \times [\check{y}_1] \times [\check{y}_2],\end{aligned} \tag{6.5}$$

где $[\check{t}_1], \ldots, [\check{y}_2]$ — априорные интервалы, по предположению содержащие истинные значения соответствующих переменных. (Прим. перев.: в отечественной литературе относительно (6.5) принят термин *множество неопределенности замера*.) ■

Оценивание будет рассматриваться как действие, учитывающее допустимое множество $\mathbb{M}$ для получения более точной информации о $\mathbf{x}$. Будут рассмотрены два подхода. Первый, описываемый в параграфе 6.2, основан на штрафной функции $\check{c}(\mathbf{x})$. Далее находится апостериорное множество-оценка $\widehat{\mathbb{X}}_{\mathrm{c}}$, как множество всех точек глобального минимума этой априорной функции на допустимом множестве $\mathbb{M}$:

$$\widehat{\mathbb{X}}_{\mathrm{c}} = \arg\min_{\mathbf{x}\in\mathbb{M}} \check{c}(\mathbf{x}). \tag{6.6}$$

Второй подход, рассматриваемый в параграфах 6.3 и 6.4, нацелен на описание *апостериорного допустимого множества* $\widehat{\mathbb{X}}_{\mathrm{s}}$, определяемого следующим образом:

$$\widehat{\mathbb{X}}_{\mathrm{s}} = \check{\mathbb{X}} \cap \mathbb{M}. \tag{6.7}$$

В обоих подходах должна выполняться проекция апостериорных оценок на пространство интересующих нас переменных.

ЗАМЕЧАНИЕ 6.3. Здесь термин «априорное» означает — до учета допустимого множества, в то время как в математической статистике обычно этот термин означает — до принятия в расчет данной информации. ■

Пример 6.3. *Рассмотрим снова* Пример 6.2. *В первом подходе апостериорная оценка имеет вид*:

$$\widehat{\mathbb{X}}_{\mathrm{c}} = \begin{array}{l} \arg\min \\ y_1 = u_1\exp(-pt_1) \\ y_2 = u_2\exp(-pt_2) \end{array} \check{c}(t_1, t_2, u_1, u_2, p, y_1, y_2). \tag{6.8}$$

Эта оценка содержит наилучшие значения $\mathbf{x}$ *(в смысле функции* $\check{c}$*), возможные в* $\mathbb{M}$. *Во втором подходе апостериорное допустимое множество может быть*

$$\begin{aligned} \widehat{\mathbb{X}}_{\mathrm{s}} = \{\mathbf{x} = (t_1, t_2, u_1, u_2, p, y_1, y_2) \mid \check{c}(\mathbf{x}) \leqslant \delta, \\ y_1 = u_1\exp(-pt_1) \ \text{ и } \ y_2 = u_2\exp(-pt_2)\}, \end{aligned} \tag{6.9}$$

где δ *— некоторое наперед заданное положительное вещественное число. Если интерес представляет только значение параметра* p, *то нужно спроектировать* $\widehat{\mathbb{X}}_c$ *или* $\widehat{\mathbb{X}}_s$ *на пространство параметров.* ■

6.2. Оценивание параметров с помощью оптимизации

Предположим, что множество $\mathcal{X}$ всех переменных может быть разбито на два множества $\mathcal{Z}$ и $\mathcal{Y}$, где $\mathcal{Z} = \{z_1, \ldots, z_{n_z}\}$ и $\mathcal{Y} = \{y_1, \ldots, y_{n_y}\}$

таковы, что существует некоторая функция $\boldsymbol{\phi} : \mathbb{R}^{n_z} \to \mathbb{R}^{n_y}$, для которой имеет место следующее тождество:

$$(z_1, \ldots, z_{n_z}, y_1, \ldots, y_{n_y})^{\mathrm{T}} \in \mathbb{M} \Leftrightarrow (y_1, \ldots, y_{n_y})^{\mathrm{T}} = \boldsymbol{\phi}(z_1, \ldots, z_{n_z}). \tag{6.10}$$

Упрощая обозначения, можно записать в более краткой форме

$$(\mathbf{z}, \mathbf{y}) \in \mathbb{M} \Leftrightarrow (\mathbf{y}) = \boldsymbol{\phi}(\mathbf{z}). \tag{6.11}$$

Определим апостериорную штрафную функцию

$$\hat{c}(\mathbf{z}) \triangleq \check{c}(\mathbf{z}, \boldsymbol{\phi}(\mathbf{z})). \tag{6.12}$$

Решение (6.6) сводится к задаче безусловной (в отсутствие ограничений) минимизации функции $\hat{c}(\mathbf{z})$. За исключением важных специальных случаев, таких, когда апостериорная штрафная функция $\hat{c}$ является квадратичной по $\mathbf{z}$, функция $\hat{c}(\mathbf{z})$ не является выпуклой, и обычные локальные итерационные алгоритмы нелинейного программирования могут «залипать» на локальных минимумах. Методы гарантированного поиска глобального минимума, такие, как OPTIMIZE (см. табл. 5.5, с. 157), должны, следовательно, быть предпочтительными всюду, где они применимы.

Пример 6.4. *Рассмотрим снова* Пример 6.1 *и положим* $\mathcal{Z} = \{t_1, t_2, u_1, u_2, p\}$ *и* $\mathcal{Y} = \{y_1, y_2\}$, *поэтому*

$$\boldsymbol{\phi}(\mathbf{z}) = \begin{pmatrix} u_1\exp(-pt_1) \\ u_2\exp(-pt_2) \end{pmatrix}. \tag{6.13}$$

Решение задачи (6.8) *сводится к нахождению множества всех точек глобального минимума функции*

$$\hat{c}(\mathbf{z}) = \check{c}(t_1, t_2, u_1, u_2, p, u_1\exp(-pt_1), u_2\exp(-pt_2)). \tag{6.14}$$

Заметим, что если t_1, t_2, u_1 *и* u_2 *полагались бы известными точно, они не должны были появиться в составе вектора* $\mathbf{z}$, *и оптимизация выполнялась бы только относительно параметра* p. *Ситуация, рассматриваемая здесь, является более общей, поскольку величины, взятые в качестве входных и выходных сигналов, а также моменты времени, на которых сигналы измерены, могут иметь неопределенности и могут также нуждаться в оценивании.* ■

В дальнейшем априорная штрафная функция $\check{c}(\mathbf{z})$ будет характеризовать расстояние между вектором $\mathbf{x}$ и некоторым заданным вектором $\check{\mathbf{x}}$

априорной оценки для переменных. Взвешенная L_2-норма

$$\check{c}(\mathbf{x}) = \| \mathbf{x} - \check{\mathbf{x}} \|_2^2 = \sum_{i=1}^{n} w_{x_i}(x_i - \check{x}_i)^2 \tag{6.15}$$

будет использоваться в параграфе 6.2.1, а взвешенная L_∞-норма

$$\check{c}(\mathbf{x}) = \| \mathbf{x} - \check{\mathbf{x}} \|_\infty = \max_{i=1,\ldots,n} w_{x_i}|x_i - \check{x}_i| \tag{6.16}$$

будет использоваться в параграфе 6.2.2. В обоих случаях w_{x_i} — предварительно заданный положительный весовой коэффициент, выражающий степень доверия к априорной оценке $\check{x}_i$. Эти две нормы могут приводить к совершенно разным апостериорным оценкам $\hat{\mathbf{x}}$, как показано на рис. 6.1. Геометрически оценка $\hat{\mathbf{x}}$ есть проекция оценки $\check{\mathbf{x}}$ на допустимое множество $\mathbb{M}$, как это определено для применяемой нормы.

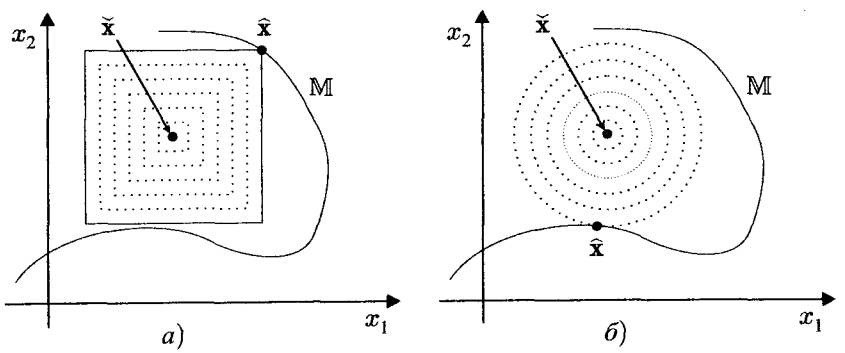


Рис. 6.1. Апостериорная оценка $\hat{\mathbf{x}}$ очень чувствительна к применяемой норме: *а*) L_∞-норма; *б*) L_2-норма

ЗАМЕЧАНИЕ 6.4. Если заранее ничего неизвестно о переменных x_i, то весовые коэффициенты w_{x_i} могут быть взяты равными нулю, и в этом случае (6.15) и (6.16) не являются нормами. ■

6.2.1. Оценивание параметров методом наименьших квадратов при моделировании структур

Структурные модели широко используются в биологии и фармакологии для изучения метаболизмов и поведения лекарств [Jaquez, 1972]. Они также находят применение в экологии и химической промышленности [Happel, 1986]. Структурная модель состоит из конечного числа однородных резервуаров, называемых ячейками и представляемых кружками, которые могут обмениваться веществом, что символизируется стрелками. Эволюция количества вещества в каждой из ячеек описывается системой, составленной из обыкновенных дифференциальных уравнений первого порядка, и обычно предполагается, что эти уравнения линейны и стационарны, и что поток вещества, покидающего ячейку i, пропорционален количеству q этого вещества в этой ячейке. Уравнения, описывающие поведение модели ячейки, получаются из уравнений сохранения в форме уравнения состояния. Как это делалось в [Kieffer и Walter, 1998], рассмотрим, например, систему, представленную на рис. 6.2.

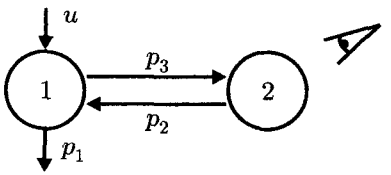


Рис. 6.2. Модель с двумя ячейками

Эволюция вектора $\mathbf{q} = (q_1, q_2)^{\mathrm{T}}$ количества вещества в двух ячейках описывается линейным стационарным уравнением состояния

$$\begin{cases} \dot{q}_1 = -(p_3 + p_1)q_1 + p_2 q_2 + u, \\ \dot{q}_2 = p_3 q_1 - p_2 q_2. \end{cases} \tag{6.17}$$

Возьмем систему при нулевых начальных условиях $(\mathbf{q}(0_-) = \mathbf{0})$ и предположим, что на вход Ячейки 1 поступает функция Дирака, $u(t) = \delta(t)$, поэтому $q_1(0_+) = 1$ и $q_2(0_+) = 0$. Предположим также, что содержимое Ячейки 2 наблюдается на 16 моментах времени, согласно условию

$$y_i = q_2(t_i), \ i = 1, \ldots, 16. \tag{6.18}$$

Тривиально показать, что

$$y_i = \alpha(\exp(-\lambda_1 t_1) - \exp(-\lambda_2 t_i)), \quad i = 1, \ldots, 16, \tag{6.19}$$

где

$$\alpha = \frac{p_3}{\sqrt{(p_1 - p_2 + p_3)^2 + 4p_2 p_3}}, \tag{6.20}$$

$$\lambda_1 = \frac{p_1 + p_2 + p_3 - \sqrt{(p_1 - p_2 + p_3)^2 + 4p_2 p_3}}{2} \tag{6.21}$$

и

$$\lambda_2 = \frac{p_1 + p_2 + p_3 + \sqrt{(p_1 - p_2 + p_3)^2 + 4p_2p_3}}{2}. \tag{6.22}$$

Эти уравнения определяют допустимое множество $\mathbb{M}$. Далее, предположим, что моменты измерений t_i известны точно, и, следовательно, не нужно рассматривать их как переменные. Переменными в данной задаче являются компоненты вектора $\mathbf{p} = (p_1, p_2, p_3)^{\mathrm{T}}$, который играет роль вектора $\mathbf{z}$, и вектор $\mathbf{y} = (y_1, \ldots, y_{16})^{\mathrm{T}}$. Априорные значения $\check{y}_i$ переменных y_i получены в результате измерений, выполненных при изучении системы, и приведены в табл. 6.1.

Никакой априорной информации нет относительно вектора параметров $\mathbf{p}$, и замеры $\check{y}_i$ представляются одинаково надежными, поэтому априорная штрафная функция выбирается как

$$\check{c}(\mathbf{p}, \mathbf{y}) = \sum_{i=1}^{16} (\check{y}_i - y_i)^2, \tag{6.23}$$

которая не зависит от $\mathbf{p}$. Соответствующая задача минимизации имеет вид

$$\min_{(\mathbf{p},\mathbf{y}) \in \mathbb{M}} \sum_{i=1}^{16} (\check{y}_i - y_i)^2. \tag{6.24}$$

Теперь, выражение $(\mathbf{p}, \mathbf{y}) \in \mathbb{M}$ эквивалентно выражению

$$\forall i \in \{1, \ldots, 16\},\ y_i = \phi_i(\mathbf{p}), \tag{6.25}$$

где $\phi_i(\mathbf{p})$ рассчитывается согласно (6.19)–(6.22). Минимизация априорной штрафной функции, определенной по (6.23) при ограничении $(\mathbf{p}, \mathbf{y}) \in \mathbb{M}$, следовательно, приводит к минимизации без ограничений апостериорной штрафной функции

$$\hat{c}(\mathbf{p}) = \sum_{i=1}^{16} (\check{y}_i - \phi_i(\mathbf{p}))^2. \tag{6.26}$$

Для параллелотопа поиска в пространстве параметров, заданном в виде $[0{,}01, 2{,}0] \times [0{,}05, 3{,}0] \times [0{,}05, 3{,}0]$, и для параметров точности, равных 10^{-9} (*Прим. перев.*: т. е. допуски на ширину параллелотопа) ε_{p} и ε_{c}, равных 10^{-9}, алгоритм Хансена для задачи минимизации без ограничений (см. параграф 5.5.2, стр. 159) окружает две точки глобального минимума штрафной функции следующими двумя параллелотопами:

$$\begin{aligned}&[1{,}925402,\ 1{,}925404] \times [0{,}232717,\ 0{,}232719] \times [0{,}145075,\ 0{,}145077],\\ &[0{,}232717,\ 0{,}232719] \times [1{,}925402,\ 1{,}925404] \times [0{,}145075,\ 0{,}145077].\end{aligned} \tag{6.27}$$

Таблица 6.1. Экспериментальные данные

t_i	1	2	3	4	5	6	7	8
$\check{y}_i$	0,0532	0,0478	0,0410	0,0328	0,0323	0,0148	0,0216	0,0127

t_i	9	10	11	12	13	14	15	16
$\check{y}_i$	0,0099	0,0081	0,0065	0,0043	0,0013	0,0015	0,0060	0,0126

Заметим, что эти параллелотопы можно вывести один из другого, переставляя местами интервальные значения переменных p_1 и p_2, в то время как p_3 принимает одинаковое интервальное значение в обоих параллелотопах. Это согласуется с выводом исследования по идентифицируемости [Walter и Pronzato, 1997], который показывает, что система на рис. 6.2 является только локально идентифицируемой, и параметры p_1 и p_2 могут быть взаимно-заменяемыми без изменения связи вход-выход, в то время как параметр p_3 является единственно идентифицируемым по экспериментальным данным. Представляется важным подчеркнуть, что вывод по представленному оцениванию не основывался на таком априорном изучении идентифицируемости, который можно и не делать, когда используются такие глобальные методы, как здесь. Две структурные системы, соответствующие решениям (6.27), показаны на рис. 6.3.

Рис. 6.4 представляет данные $\check{y}(t_i)$ и полученный отклик $\hat{y}(t)$ для значений параметров, соответствующих глобальному минимуму.

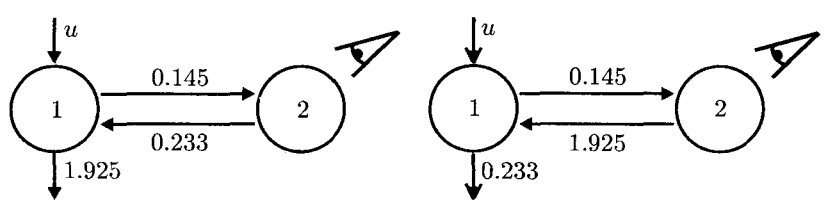


Рис. 6.3. Две существенно разные структурные модели с одинаковым наблюдаемым поведением

ЗАМЕЧАНИЕ 6.5. Когда оптимизация выполняется относительно вектора параметров $\mathbf{p}$, то алгоритму Хансена требуется примерно сутки вычислений на компьютере Pentium 233, чтобы получить этот ответ. А выполняя сперва оптимизацию относительно α, λ_1 и λ_2 и затем решая (6.20)–(6.22) для $\mathbf{p}$ по алгоритму SIVIAX (параграф 5.2, с. 139), можно сократить время вычислений примерно до одной минуты [Kieffer и Walter, 1998]. ■

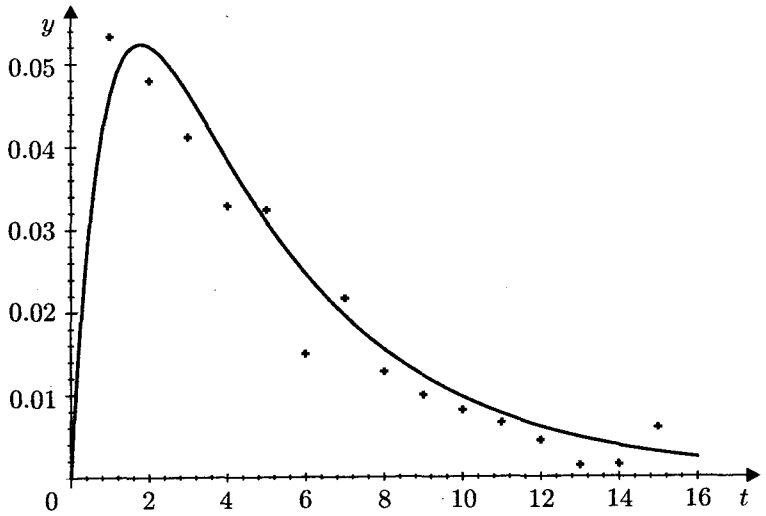


Рис. 6.4. Данные $\check{y}(t_i)$ (крестики) и выход $\hat{y}(t)$ (сплошная кривая) оценочной модели

6.2.2. Минимаксное оценивание параметров

Как и в параграфе 6.2.1, рассмотрим систему, для которой n_y замеров, $\check{y}_1, \ldots, \check{y}_{n_y}$, связанных с выходными переменными $y_1, \ldots, y_{n_y}$, получены в известные моменты времени t_i, $i = 1, \ldots, n_y$. Таким образом, замеры $\check{y}_i$ приближенно соответствуют неизвестным значениям переменных y_i, которые бы и составили выборку замеров в идеальных условиях (при отсутствии шума измерения). Предположим, что система зависит от неизвестного вектора параметров $\mathbf{p} \in \mathbb{R}^{n_p}$, который подлежит оцениванию. Переменными в этой задаче оценивания являются

$$\mathbf{x} = (p_1, \ldots, p_{n_p}, y_1, \ldots, y_{n_p})^{\mathrm{T}}. \tag{6.28}$$

Следуя (6.16), возьмем априорную штрафную функцию в виде:

$$\check{c}(\mathbf{p}, \mathbf{y}) = \check{c}(\mathbf{y}) = \|\check{\mathbf{y}} - \mathbf{y}\|_\infty = \max_{i \in \{1, \ldots, n_y\}} |\check{y}_i - y_i|. \tag{6.29}$$

Поскольку о векторе $\mathbf{p}$ ничего не известно, положим $w_{p_i} = 0$ и, следовательно, $\check{c}(\mathbf{p}, \mathbf{y})$ не зависит от $\mathbf{p}$ (см. Замечание 6.4).

Предположим, что соотношения, определяющие допустимое множество, имеют вид:

$$y_i = \phi(\mathbf{p}, t_i),\ i = 1, \ldots, n_y, \tag{6.30}$$

где $\phi(\mathbf{p}, t_i)$ — некоторая заранее заданная функция.

По (6.12), апостериорная штрафная функция, подлежащая минимизации, записывается

$$\hat{c}(\mathbf{p}) = \check{c}(\phi(\mathbf{p}, t_1), \ldots, \phi(\mathbf{p}, t_{n_y})) = \max_{i \in \{1, \ldots, n_y\}} |\check{y}_i - \phi(\mathbf{p}, t_i)|. \tag{6.31}$$

Оценивание $\mathbf{p}$ приводит к вычислению точек глобального минимума функции

$$\hat{c}(\mathbf{p}) = \max_{i=1, \ldots, n_y} |f_i(\mathbf{p})|, \tag{6.32}$$

где $f_i(\mathbf{p}) = \check{y}_i - \phi(\mathbf{p}, t_i)$. Полученная задача минимаксной оптимизации известна как *дискретная задача чебышевской аппроксимации*. Этот тип задач появляется в проблеме *объединения информации от датчиков* [McKendall, 1990; McKendall и Mintz, 1992] или в *теории принятия решений* [Berger, 1985], когда нужно минимизировать максимум вероятности неприемлемой ошибки или риска.

Поскольку $\hat{c}(\mathbf{p})$ не является всюду дифференцируемой, то традиционные градиентные методы обычно *очень неэффективны* и, кроме того, имеют известные недостатки, присущие локальным методам.

Большинство существующих методов также по сути являются локальными и основаны на итерационном применении техники линейного или квадратичного программирования. Интервальные разрешающие операторы были использованы в [Wolfe, 1999] для одномерной задачи ($\dim \mathbf{p} = 1$), в [Zuhe *et al.*, 1990] и [Jaulin, 2001б] для более общего случая. Можно применить алгоритм OPTIMIZE (см. табл. 5.5, с. 157), как это иллюстрируется следующим примером. (Алгоритм MINIMIX из параграфа 5.6 не подходит для рассматриваемой задачи максимизации, так как она выполняется при конечном числе моментов времени.)

Пример 6.5. *Предположим, что переменные* $p_1, \ldots, p_4, y_1, \ldots, y_{10}$ связаны соотношениями

$$y_i = p_1 \exp(p_2 t_i) + p_3 \exp(p_4 t_i), \qquad i = 1, \ldots, 10. \tag{6.33}$$

Переменные p_i *и* y_i *образуют векторы* $\mathbf{p}$ *и* $\mathbf{y}$. *Поскольку перестановки* p_1 *с* p_3 *и* p_2 *с* p_4 *не влияют на суть соотношений, то вектор* $\mathbf{p}$ *не является однозначно идентифицируемым. Любой метод надежного оценивания параметров должен, следовательно, привести к симметричным решениям*

с учетом того, что область поиска весьма велика и содержит все эти решения. Предположим, что данные табл. 6.2 *получены для известных моментов времени* t_i. *Эти данные показаны на* рис. 6.5. *Априорная штрафная функция задается следующим образом:*

$$\check{c}(\mathbf{p},\mathbf{y}) = \check{c}(\mathbf{y}) = \max_{i=1,\ldots,10} |y_i - \check{y}_i|. \tag{6.34}$$

Опять эта функция не зависит от $\mathbf{p}$, *так как об этом векторе нет никакой априорной информации. Допустимое множество имеет вид:*

$$\mathbb{M} = \{(\mathbf{p},\mathbf{y}) \,|\, y_i = p_1 \exp(p_2 t_i) + p_3 \exp(p_4 t_i), \quad i = 1, \ldots, 10\}, \tag{6.35}$$

а апостериорная штрафная функция

$$\hat{c}(\mathbf{p}) = \max_{i=1,\ldots,10} |\check{y}_i - \phi(\mathbf{p}, t_i)|. \tag{6.36}$$

Минимизация этой функции будет выполнена в Примере 6.7. ■

Таблица 6.2. Экспериментальные данные

t_i	0,25	1	2,25	4	6,25
$\check{y}_i$	6,9465	0,8902	−3,0562	−3,7537	−2,8262

t_i	9	12,25	16	20,25	25
$\check{y}_i$	−1,6660	−0,7961	−0,3086	−0,1330	−0,1218

Для нахождения оценки $\widehat{\mathbf{p}}$ можно применить алгоритм OPTIMIZE (см. табл. 5.5, с. 157), который минимизирует функцию $\hat{c}$, определенной выражением (6.32). На Шаге 6 этого алгоритма может быть использован сжимающий оператор $\mathcal{C}([\mathbf{p}])$, связанный с ограничением

$$\max_{i\in\{1,\ldots,n\}} = |f_i(\mathbf{p})| \leqslant \overline{c}, \tag{6.37}$$

где $f_i(\mathbf{p}) = \check{y}_i - \phi(\mathbf{p}, t_i)$ и $\overline{c}$ — верхняя граница функции $\hat{c}(\hat{\mathbf{p}})$. Оператор $\mathcal{C}([\mathbf{p}])$ может быть получен сжатием задачи выполнения ограничений

$$\mathcal{H}: \begin{pmatrix} f_1(\mathbf{p}) = c_1 \\ \vdots \\ f_{n_y}(\mathbf{p}) = c_{n_y} \\ p_1 \in [p_1], \ldots, p_{n_p} \in [p_{n_p}] \\ c_1 \in [-\overline{c}, \overline{c}], \ldots, c_{n_y} \in [-\overline{c}, \overline{c}] \end{pmatrix}. \tag{6.38}$$

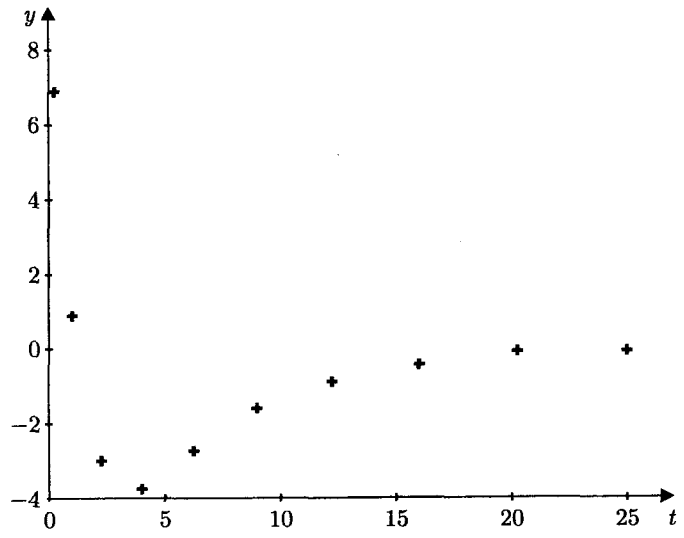


Рис. 6.5. Данные для примера минимаксного оценивания

В локальном поиске, необходимом на Шаге 4 алгоритма OPTIMIZE для понижения верхней границы $\overline{c}$, будет использоваться подход из [Jaulin, 2001б], который предполагает, что каждый параметр p_i появляется только один раз в выражении функции $f_j(\mathbf{p})$. Этот подход дает быстрые вычисления, не нуждается в расчете градиентов или субградиентов штрафной функции, прост в применении и иллюстрирует возможности интервальных методов при выполнении локальной оптимизации.

Для заданного $\mathbf{p}$, заданного номера i оси пространства параметров и заданной верхней границы $\overline{c}$ функции $\hat{c}(\widehat{\mathbf{p}})$ (если нет улучшенного значения границы $\overline{c}$, то в качестве нее можно взять $\overline{c} = c(\mathbf{p})$) определим следующие множества:

$$\begin{aligned} \mathbb{L}_i(\mathbf{p}) &\triangleq \{\mathbf{q} \in \mathbb{R}^{n_p} \,|\, \forall \ell \in \{1, \ldots, i-1, i+1, \ldots, n_p\},\ p_\ell = q_\ell\}, \\ \mathbb{S}_k(\overline{c}) &\triangleq \{\mathbf{p} \in \mathbb{R}^{n_p} \,|\, f_k(\mathbf{p}| \leqslant \overline{c}\}, \\ \mathbb{S}(\overline{c}) &\triangleq \bigcap\nolimits_{k=1}^{n_y} \mathbb{S}_k(\overline{c}) = \{\mathbf{p} \in \mathbb{R}^{n_p} \,|\, \check{c}(\mathbf{p}) \leqslant \overline{c}\}, \\ \mathbb{Q}_i(\mathbf{p},\overline{c}) &\triangleq \mathbb{L}_i(\mathbf{p}) \cap \mathbb{S}(\overline{c}) = (\mathbb{L}_i(\mathbf{p}) \cap \mathbb{S}_1(\overline{c})) \cap \ldots \cap (\mathbb{L}_i(\mathbf{p}) \cap \mathbb{S}_{n_y}(\overline{c})). \end{aligned} \tag{6.39}$$

Множества $\mathbb{L}_i(\mathbf{p}), \mathbb{S}(\bar{c})$ и $\mathbb{Q}_i(\mathbf{p}, \bar{c})$ показаны на рис. 6.6 для $\dim \mathbf{p} = 2$ и $i = 1$. Любая точка внутри множества $\mathbb{Q}_i(\mathbf{p}, \bar{c})$ удовлетворяет неравенству $\hat{c}(\mathbf{q}) \leqslant \bar{c})$ и, следовательно, может быть использована для понижения верхней границы $\bar{c}$ по направлению i.

Алгоритм CROSS, представленный в табл. 6.3, использует преимущество идеи понижения верхней границы $\bar{c}$ функции $\hat{c}(\widehat{\mathbf{p}})$. Малое положительное число κ используется для остановки процедуры, когда улучшение не является достаточно существенным. Множество $\mathbb{Q}_i(\mathbf{p}, \bar{c})$, рассчитываемое на Шаге 5, в общем случае является отрезком или конечным объединением отрезков на рассматриваемой оси. На Шаге 7 вектор $\mathbf{q}$ обычно берется равным центру наибольшего сегмента из множества $\mathbb{Q}_i(\mathbf{p}, \bar{c})$. С ситуацией, когда $\mathbb{Q}_i(\mathbf{p}, \bar{c}) = \varnothing$ (на Шаге 6), можно встретиться только когда $\hat{c}(\mathbf{p}) \geqslant \bar{c}$, т. е. когда цикл процедуры выполняется впервые. Если улучшение верхней границы представляется существенным ($\bar{c} - \widetilde{c} > \kappa$), то цикл заново запускается со значения $\widetilde{\mathbf{q}}$.

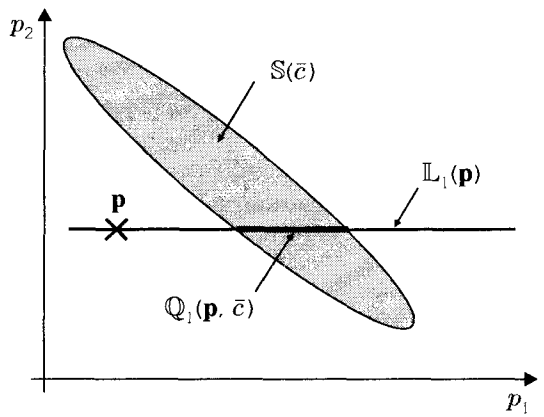


Рис. 6.6. Локальный поиск из точки $\mathbf{p}$ по оси параметра p_1

Работа этого алгоритма теперь поясняется на очень простой двумерной задаче.

Пример 6.6. *Задача минимизации состоит в поиске наименьшего круга $\mathcal{D}$, содержащего n точек $A_1, \ldots, A_n$ из $\mathbb{R}^2$. Центр круговой области решения является точкой минимума штрафной функции*

$$c(p_1, p_2) = \max_{j \in \{1, \ldots, n\}} \left\{ \sqrt{(p_1 - x_{A_j})^2 + (p_2 - y_{A_j})^2} \right\}, \qquad (6.40)$$

Таблица 6.3. Локальный алгоритм понижения верхней границы оптимального значения штрафной функции

Алгоритм CROSS(вход : $c, \mathbf{p}, \kappa$; вход/выход : $\bar{c}$)	
1	$\widetilde{c} := \bar{c}$; $\widetilde{\mathbf{q}} := \mathbf{p}$;
2	выполнить
3	$\bar{c} := \widetilde{c}$; $\mathbf{p} := \widetilde{\mathbf{q}}$;
4	для всех $i \in \{1, \ldots, n_{\mathrm{p}}\}$
5	$\mathbb{Q}_i(\mathbf{p}, \bar{c}) := \mathbb{L}_i(\mathbf{p}) \cap \mathbb{S}(\bar{c})$;
6	если $\mathbb{Q}_i(\mathbf{p}, \bar{c}) \neq \varnothing$, то к следующему i;
7	выбрать $\mathbf{q}$ внутри $\mathbb{Q}_i(\mathbf{p}, \bar{c})$;
8	если $c(\mathbf{q}) \leqslant \widetilde{c}$, то $\{\widetilde{\mathbf{q}} := \mathbf{q},\ \widetilde{c} := c(\mathbf{q})\}$;
9	выполнять пока $\bar{c} - \widetilde{c} > \kappa$
	(допуск κ выбирается пользователем).

а радиус круга равен минимуму $\widehat{c}$. Если, например, $n = 3$ и заданы три точки $A_1(0, 4), A_2(0, -4), A_3(4, 0)$, то $\widehat{\mathbf{p}} = \mathbf{0}$ является точкой минимума, а сам минимум равен $\widehat{c} = 4$. Заметим, что функция c недифференцируема в точке $\widehat{\mathbf{p}}$. Рис. 6.7 *показывает множества уровня функции c. Алгоритм* CROSS *запускается в точке $\mathbf{p} = (2, 8)^{\mathrm{T}}$, и при этом $\bar{c} = 6$. Множества $\mathbb{S}_1(\bar{c}), \mathbb{S}_2(\bar{c})$ и $\mathbb{S}_3(\bar{c})$ являются кругами* (рис. 6.8, *отмечены серым цветом*). *Более темным цветом отмечено результирующее множество $\mathbb{S}(\bar{c})$.*

Цикл впервые выполняется при $i = 1$, и на Шаге 5 алгоритм CROSS *вычисляет множество $\mathbb{Q}_1(\mathbf{p}, \bar{c})$. Поскольку*

$$\mathbb{Q}_1(\mathbf{p}, \bar{c}) = (\mathbb{L}_1(\mathbf{p}) \cap \mathbb{S}_1(\bar{c})) \cap (\mathbb{L}_1(\mathbf{p}) \cap \mathbb{S}_2(\bar{c})) \cap (\mathbb{L}_1(\mathbf{p}) \cap \mathbb{S}_3(\bar{c})), \qquad (6.41)$$

то вычисление множества $\mathbb{Q}_1(\mathbf{p}, \bar{c})$ приводит к вычислению множества $\mathbb{L}_1(\mathbf{p}) \cap \mathbb{S}_j(\bar{c})$, $j = 1, 2, 3$. Эти операции могут быть выполнены автоматически решением задачи выполнения ограничений методом вперед-назад с переменными p_i и r при ограничении $|f_j(\mathbf{p})| - r = 0$ с областью $[0, \bar{c}]$ для величины r и множеством вещественных чисел $\mathbb{R}$ для параметра p_i. Например, для $j = 1$

$$\mathbb{S}_1(\bar{c}) = \{\mathbf{p} \in \mathbb{R}^2 \mid \sqrt{(p_1 - x_{A_1})^2 + (p_2 - y_{A_1})^2} \leqslant \bar{c}\} \qquad (6.42)$$

и

$$\mathbb{L}_1(\mathbf{p}) = \{\mathbf{q} \in \mathbb{R}^2 \mid q_2 = p_2\}. \qquad (6.43)$$

Так как $\mathbf{p} = (2, 8)^{\mathrm{T}}, \bar{c} = 6$ и точка A_1 имеет координаты $x_{A_1} = 0$ и $y_{A_1} = 4$, то

$$\mathbb{L}_1(\mathbf{p}) \cap \mathbb{S}_1(c) = \{\mathbf{p} \in \mathbb{R}^2 \mid \sqrt{p_1^2 + (p_2 - 4)^2} \leqslant 6,\ p_2 = 8\}. \qquad (6.44)$$

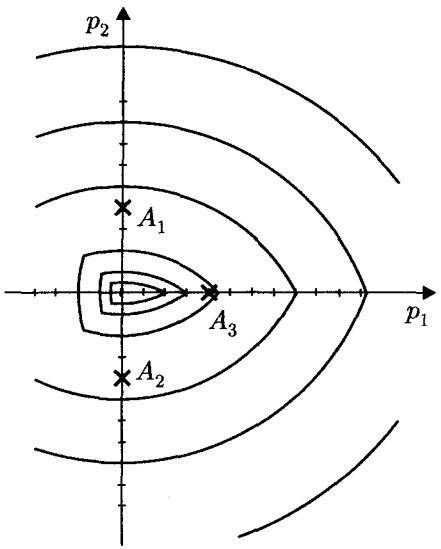


Рис. 6.7. Множества уровня штрафной функции с Примера 6.6

Теперь множество $\mathbb{L}_1(\mathbf{p}) \cap \mathbb{S}_1(\bar{c})$ *может быть вычислено следующим образом:*

$$\begin{aligned}
\mathbb{L}_1(\mathbf{p}) \cap \mathbb{S}_1(\bar{c}) &= \{\mathbf{p} \in \mathbb{R}^2 \mid \sqrt{p_1^2 + (8-4)^2} = r,\ r \in [0,\,6],\ p_2 = 8\} \\
&= \{\mathbf{p} \in \mathbb{R}^2 \mid p_1^2 = r^2 - 16,\ r \in [0,\,6],\ p_2 = 8\} \\
&= \{\mathbf{p} \in \mathbb{R}^2 \mid p_1 \in \mathrm{sqr}^{-1}(r^2 - 16),\ r \in [0,\,6],\ p_2 = 8\} \\
&= \{\mathbf{p} \in \mathbb{R}^2 \mid p_1 = \mathrm{sqr}^{-1}([0,\,6]^2 - 16),\ p_2 = 8\} \\
&= \{\mathbf{p} \in \mathbb{R}^2 \mid p_1 \in [-\sqrt{20},\,\sqrt{20}],\ p_2 = 8\} \\
&= [-\sqrt{20},\,\sqrt{20}] \times [8,\,8].
\end{aligned}$$

Здесь интервальная функция sqr^{-1} *является обратной к интервальной функции возведения в квадрат* sqr; *функцию* sqr^{-1} *не следует путать с классической функцией квадратного корня (именно,* $\mathrm{sqr}^{-1}(4) = \{2, 2\}$ *в то время как* $\sqrt{4} = 2$). *(Заметим, что* $\mathrm{sqr}^{-1}([4, 9]) = [-3, -2] \cup [2, 3]$; *таким образом, может придется работать с объединениями интервалов.) Эта же причина применительно к операции* $\mathbb{L}_1(\mathbf{p}) \cap \mathbb{S}_2(\bar{c})$ *приводит*

к пустому множеству. Следовательно, множество $\mathbb{Q}_1(\mathbf{p}, \bar{c}) = \varnothing$. *Таким образом, горизонтальное направление, соответствующее значению* $i = 1$, *исключается из рассмотрения, и алгоритм* CROSS *теперь исполняется по вертикальной оси при значении* $i = 2$. *Получаем множество*

$$\mathbb{Q}_2(\mathbf{p}, \bar{c}) = \mathbb{L}_2(\mathbf{p}) \cap \mathbb{S}(\bar{c}) = [2, 2] \times [-1{,}657, 1{,}657], \tag{6.45}$$

которое соответствует сегменту, отмеченному на рис. 6.8 *темно-серым цветом. Когда вышли из цикла и если центр множества* $\mathbb{Q}_2(\mathbf{p}, \bar{c})$ *выбирается в точке* $\tilde{\mathbf{q}}$, *то* $\tilde{\mathbf{q}} = (2, 0)^{\mathrm{T}}$ *и значение* $\bar{c} = \sqrt{20}$. *Алгоритм* CROSS *далее опять запускается из точки* $\mathbf{p} = \tilde{\mathbf{q}}$. ■

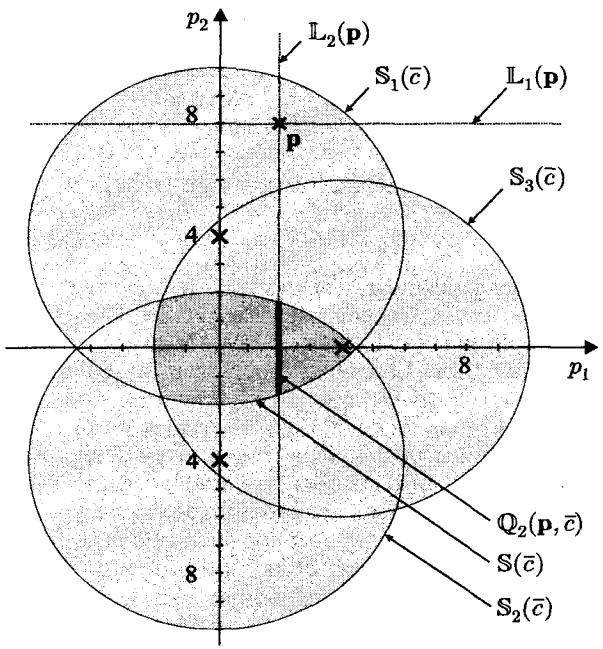


Рис. 6.8. Пример итерации алгоритма CROSS

Пример 6.7. *Рассмотрим снова* Пример 6.5, *где штрафная функция задается по* (6.36). *Когда начальный вектор параметров и исходные данные определены, то процедура* MINIMAX *окна* OPTIM *управления пакетом* MATLAB [Brayton *et al.*, 1979] *находит значение штрафной функции,*

равное 0,0657 *за* 5 *секунд на компьютере* Pentium 133. *Однако эта процедура является локальной, чувствительна к начальному вектору параметров (процедура может даже разойтись), не дает никакой гарантии результату (даже локально), часто останавливается вследствие плохой обусловленности и никогда не обнаруживает, что задача имеет два решения для вектора* $\mathbf{p}$. *В противоположность этому, подход, описанный здесь, способен решить эту минимаксную задачу глобально и эффективно. На том же самом компьютере при* $\varepsilon = 0{,}05$ *в алгоритме* OPTIMIZE *и* $\kappa = 0{,}001$ *в алгоритме* CROSS *и при параллелотопе поиска* $[-60, 60] \times [-1, 0] \times [-60, 60]$, *алгоритм* OPTIMIZE *после* 1,7 *секунд счета и* 109 *бисекций дает, что* $\widehat{c} \in [0{,}0653, 0{,}0657]$. *Результирующее покрытие* $\overline{\mathbb{S}}$ *(которое содержит все точки глобального минимума) содержит* 44 *параллелотопа и имеет две симметричные несвязные компоненты. Это показывает, что интервальные сжимающие операторы могут дополнять процедуры пакета* MATLAB, *даже с точки зрения затрат машинного времени.* ■

6.3. Гарантированное оценивание параметров

6.3.1. Введение

Подход к гарантированному оцениванию параметров заново привлек внимание исследователей в девяностых годах [Walter, 1990; Combettes, 1993; Deller *et al.*, 1993; Norton, 1994; Norton, 1995; Milanese *et al.*, 1996; Walter и Pronzato, 1997; см. также библиографию в данных работах]. Из причин такого интереса отметим только две. Первая, этот подход может работать с неизбежными структурными ошибками, вытекающими из того факта, что уравнения, использованные для описания исследуемой системы, в реальности всегда являются приближенными. Эти ошибки часто являются детерминированными и, следовательно, неадекватно описываются вероятностными переменными. Вторая, гарантированное оценивание параметров хорошо приспособлено к гарантированному описанию неопределенности параметров, что является предпосылкой к ряду методов в робастном управлении (см. Главу 7). В контексте оценивания при ограниченных ошибках интервальные методы были разработаны независимо в [Moore, 1992] и в [Jaulin и Walter, 1993б].

При гарантированном оценивании множество, которое предстоит описать, есть $\widehat{\mathbb{X}}_s = \check{\mathbb{X}} \cap \mathbb{M}$. Даже если некоторый алгоритм может, в принципе, быть найден для решения этой задачи, часто обнаруживается, что точное описание множества $\widehat{\mathbb{X}}_s$ оказывается слишком сложным, так как множество $\mathcal{X} = \{x_1, \ldots, x_n\}$ в общем случае содержит слишком много

элементов. Однако на практике часто можно перейти к разбиению вектора $\mathbf{x} = (x_1, \ldots, x_n)^T$ на три вектора $\mathbf{y}$, $\mathbf{p}$ и $\mathbf{t}$ так, что существует некоторая функция ϕ, такая, что

$$\mathbf{x} \in \mathbb{M} \Leftrightarrow (\mathbf{y}, \mathbf{p}, \mathbf{t}) \in \mathbb{M} \Leftrightarrow \mathbf{y} = \boldsymbol{\phi}(\mathbf{p}, \mathbf{t}). \tag{6.46}$$

Когда векторы $\mathbf{p}$ и $\mathbf{t}$ связаны при формировании вектора $\mathbf{z}$, то это соответствует соотношению (6.10) (с. 187). Разделение векторов $\mathbf{p}$ и $\mathbf{t}$ делает возможным различать параметры, подлежащие оцениванию (*вектор параметров* $\mathbf{p} = (p_1, \ldots, p_{n_p})^T$), от других величин с неопределенностью (от вектора значений $\mathbf{t} = (t_1, \ldots, t_{n_t})^T$, принимаемых *независимой переменной* — временем), введенных только для обеспечения оценивания вектора $\mathbf{p}$. На практике вектор $\mathbf{t}$ может соответствовать действительным моментам времени, в которых экпериментальные данные сняты, если эти моменты имеют неопределенность.

Вектор $\mathbf{y} = (y_1, \ldots, p_{n_y})^T$ (*вектор выхода*) содержит переменные, значения которых могут быть вычислены по множеству ограничения $\mathbb{M}$, если значения векторов $\mathbf{p}$ и $\mathbf{t}$ известны. *Описывающая функция* (или функция модели) есть некоторая функция, отображающая из пространства $\mathbb{R}^{n_p+n_t}$ в пространство $\mathbb{R}^{n_y}$ и вычисляющая вектор $\mathbf{y}$ по векторам $\mathbf{p}$ и $\mathbf{t}$. Поскольку интересующие переменные запоминаются в векторе $\mathbf{p}$, множество, которое надо описать, является проекцией $\widehat{\mathbb{P}}$ множества $\widehat{\mathbb{X}}_s$ на пространство параметров:

$$\widehat{\mathbb{P}} = \{\mathbf{p} \in \mathbb{R}^{n_p} \mid \exists \mathbf{t} \in \mathbb{R}^{n_t},\ \exists \mathbf{y} \in \mathbb{R}^{n_y},\ (\mathbf{y}, \mathbf{p}, \mathbf{t}) \in \widehat{\mathbb{X}}_s = \check{\mathbb{X}} \cup \mathbb{M}\}. \tag{6.47}$$

В силу (6.46) из (6.47) следует

$$\widehat{\mathbb{P}} = \{\mathbf{p} \in \mathbb{R}^{n_p} \mid \exists \mathbf{t} \in \mathbb{R}^{n_t},\ \mathbf{y} = \boldsymbol{\phi}(\mathbf{p}, \mathbf{t}) \text{ и } (\mathbf{y}, \mathbf{p}, \mathbf{t}) \in \check{\mathbb{X}}\} \tag{6.48}$$

$$= \{\mathbf{p} \in \mathbb{R}^{n_p} \mid \exists \mathbf{t} \in \mathbb{R}^{n_t},\ (\boldsymbol{\phi}(\mathbf{p}, \mathbf{t}), \mathbf{p}, \mathbf{t}) \in \check{\mathbb{X}}\}. \tag{6.49}$$

Алгоритм SIVIA может быть использован для нахождения множества $\widehat{\mathbb{P}}$, это обеспечивается тем, что проверка включения $[\tau_p]([\mathbf{p}])$ — доступна для проверки

$$\tau_p(\mathbf{p}) \triangleq (\exists \mathbf{t} \in \mathbb{R}^{n_y} \mid (\boldsymbol{\phi}(\mathbf{p}, \mathbf{t}), \mathbf{p}, \mathbf{t}) \in \check{\mathbb{X}}). \tag{6.50}$$

Пусть $[\tau_{ypt}]([\mathbf{y}], [\mathbf{p}], [\mathbf{t}])$ — некоторая проверка включения для проверки $\tau_{ypt} \triangleq ((\mathbf{y}, \mathbf{p}, \mathbf{t}) \in \check{\mathbb{X}})$. Проверка включения для проверки

$$\tau_{pt}(\mathbf{p}, \mathbf{t}) \triangleq ((\boldsymbol{\phi}(\mathbf{p}, \mathbf{t}), \mathbf{p}, \mathbf{t}) \in \check{\mathbb{X}}) \tag{6.51}$$

имеет вид:

$$[\tau_{\text{pt}}]([\mathbf{p}],[\mathbf{t}]) = [\tau_{\text{ypt}}]([\boldsymbol{\phi}]([\mathbf{p}],[\mathbf{t}]),[\mathbf{p}],[\mathbf{t}]), \tag{6.52}$$

где $[\boldsymbol{\phi}]([\mathbf{p}],[\mathbf{t}])$ — функция включения для функции $\boldsymbol{\phi}(\mathbf{p},\mathbf{t})$.

Проверка включения для $\tau_{\text{p}}(\mathbf{p})$ окончательно дается алгоритмом, приведенным в табл. 6.4. Начальный параллелотоп поиска $[\check{\mathbf{t}}]$ полагается достаточно большим, чтобы содержать проекцию множества $\check{\mathbb{X}}$ на подпространство $\mathbf{t}$.

С одной стороны, алгоритм пытается разбить параллелотоп $[\check{\mathbf{t}}]$ на подпараллелотопы $[\mathbf{t}]$, такие, что

$$(\boldsymbol{\phi}([\mathbf{p}],[\mathbf{t}]),[\mathbf{p}],[\mathbf{t}]) \cap \check{\mathbb{X}} = \varnothing \tag{6.53}$$

для всех таких подпараллелотопов. Все остальные, подлежащие исследованию подпараллелотопы из $[\check{\mathbf{t}}]$, запоминаются в очереди $\mathcal{Q}$. Если алгоритм успешно работает, то это значит, что $[\mathbf{p}] \cap \mathbb{P} = \varnothing$, и на Шаге 3 он выдает значение 0.

С другой стороны, алгоритм пытается найти один вектор $\mathbf{t}$, такой, что

$$(\boldsymbol{\phi}([\mathbf{p}],\mathbf{t}),[\mathbf{p}],\mathbf{t}) \subset \check{\mathbb{X}}. \tag{6.54}$$

Когда такой вектор $\mathbf{t}$ найден, это значит, что $[\mathbf{p}] \subset \mathbb{P}$, и на Шаге 5 алгоритм выдает значение 1.

Проверка на Шаге 5 введена, чтобы избежать бесконечного расщепления параллелотопа $[\check{\mathbf{t}}]$ и чтобы установить некоторую связь с правилами разбиения параллелотопа $[\mathbf{p}]$, которых алгоритм SIVIA придерживается. Когда алгоритм не может сделать заключение, то на Шаге 3 выдается интервал $[0,1]$.

На практике размерность n_{t} вектора $\mathbf{t}$ моментов времени высока по сравнению с размерностью n_{p} вектора $\mathbf{p}$ параметров, и бисекции, выполняемые на Шаге 8, экспоненциально по n_{t} увеличивают сложность алгоритма. Теперь рассмотрим ситуацию, когда можно избежать данные бисекции. Предположим, что априорное допустимое множество есть параллелотоп, записываемый:

$$\check{\mathbb{X}} \triangleq [\check{y}_1] \times \ldots \times [\check{y}_{n_{\text{y}}}] \times [\check{p}_1] \times \ldots \times [\check{p}_{n_{\text{p}}}] \times [\check{t}_1] \times \ldots \times [\check{t}_{n_{\text{t}}}]. \tag{6.55}$$

Предположим также, что t_i соответствуют моментам времени, в которые получены замеры y_i, $i = 1, \ldots n_{\text{y}}$. Как результат, n_{t} равно n_{y} и i-я компонента соотношения $\mathbf{y} = \boldsymbol{\phi}(\mathbf{p},\mathbf{t})$ может быть записана как $y_i = \phi_i(\mathbf{p},t_i)$. Это означает, что величина y_i не зависит от моментов времени, в которые были

Таблица 6.4. Проверка включения при анализе допустимости параллелотопа $[\mathbf{p}]$

Алгоритм $[\tau_{\text{p}}]$(вход : $\tau_{\text{pt}}, [\mathbf{p}], [\check{\mathbf{t}}]$; выход : $[\tau_{\text{p}}]$)
1 $\mathcal{Q} := \{[\check{\mathbf{t}}]\}$; $[\tau_{\text{p}}] := 0$;
2 повтор
3 если $\mathcal{Q} = \varnothing$, то возврат;
4 захватить первый параллелотоп из $\mathcal{Q}$ в параллелотоп $[\mathbf{t}]$;
5 если $([\tau_{\text{pt}}]([\mathbf{p}], \text{mid}([\mathbf{t}])) = 1)$, то $[\tau_{\text{p}}] := 1$; возврат;
6 если $([\tau_{\text{pt}}]([\mathbf{p}], [\mathbf{t}]) \neq 0)$, то
7 если $w([\mathbf{t}]) \leqslant w([\mathbf{p}])$, $[\tau_{\text{p}}] := [0,1]$;
8 иначе выполнить бисекцию $[\mathbf{t}]$ и записать результирующие параллелотопы в конец списка $\mathcal{Q}$;
9 выполнять всегда.

получены другие замеры, что является весьма обычной ситуацией. Проверка $\tau_{\text{p}}(\mathbf{p})$, определенная по (6.50), эквивалента следующим действиям:

$$\begin{pmatrix} \exists t_1 \in [\check{t}_1], \ldots, \exists t_{n_{\text{y}}} \in [\check{t}_{n_{\text{y}}}], \text{ такие что} \\ p_1 \in [\check{p}_1](0), \ldots, p_{n_{\text{p}}} \in [\check{p}_{n_{\text{p}}}](0), \\ \phi_1(\mathbf{p}, t_1) \in [\check{y}_1], \ldots, \phi_{n_{\text{y}}}(\mathbf{p}, t_{n_{\text{y}}}) \in [\check{y}_{n_{\text{y}}}] \end{pmatrix}, \tag{6.56}$$

т. е. действиям

$$\begin{pmatrix} p_1 \in [\check{p}_1](0), \ldots, p_{n_{\text{p}}} \in [\check{p}_n](0) \\ \exists t_1 \in [\check{t}_1] \,|\, \phi_1(\mathbf{p}, t_1) \in [\check{y}_1] \\ \vdots \\ \exists t_{n_{\text{y}}} \in [\check{t}_{n_{\text{y}}}] \,|\, \phi_{n_{\text{y}}}(\mathbf{p}, t_{n_{\text{y}}}) \in [\check{y}_{n_{\text{y}}}] \end{pmatrix}. \tag{6.57}$$

Пусть $[\sigma_i]([\mathbf{p}])$ — проверка включения для $\sigma_i(\mathbf{p}) \triangleq (p_i \in [\hat{p}_i])$, и пусть $[\eta_i]([\mathbf{p}])$ — проверка включения для проверки $\eta_i(\mathbf{p}) \triangleq (\exists t_i \in [t_i] \,|\, \phi_i([\mathbf{p}], t_i) \in [\hat{y}])$. Тогда проверка включения для $\tau_{\text{p}}(\mathbf{p})$ задается

$$[\tau_{\text{p}}]([\mathbf{p}]) = [\sigma_1]([\mathbf{p}]) \wedge \ldots \wedge [\sigma_{n_{\text{p}}}]([\mathbf{p}]) \wedge [\eta_1]([\mathbf{p}]) \wedge \ldots \wedge [\eta_{n_{\text{y}}}]([\mathbf{p}]). \tag{6.58}$$

Теперь проверка включения $[\eta_i]([\mathbf{p}])$, реализуемая в строках табл. 6.4, требует только одномерных бисекций вдоль оси t_i. Оценивание интервала $[\tau_{\text{p}}](\mathbf{p})$, следовательно, требует n_{y} поисков в одномерных пространствах, вместо одного поиска в n_{y}-размерном пространстве, это — принципиальное упрощение.

Весьма упрощенный случай будет рассмотрен в следующем параграфе, где вектор $\mathbf{t}$ предполагается точно известным, и, таким образом, его не надо включать в список рассматриваемых переменных. Мы вернемся к неопределенным замерам независимых переменных в параграфе 6.3.4.

6.3.2. Случай известных независимых переменных

Предположим, что вектор $\mathbf{t}$ не нужно включать в список переменных задачи оценивания, например, вследствие пренебрежительно малых ошибок в значении момента времени, в который производится измерение. Уравнение (6.49) упрощается к виду:

$$\widehat{\mathbb{P}} = \{\mathbf{p} \in \mathbb{R}^{n_\mathbf{p}} \mid (\boldsymbol{\phi}(\mathbf{p}), \mathbf{p}) \in \check{\mathbb{X}}\}. \tag{6.59}$$

Предположим также, что $\check{\mathbb{X}}$ является параллелотопом, и это означает, что априорная информация по каждой переменной не зависит от априорной информации по другим переменным, т. е.

$$\check{\mathbb{X}} \triangleq [\check{y}_1] \times \ldots \times [\check{y}_{n_\mathbf{y}}] \times [\check{p}_1] \times \ldots \times [\check{p}_{n_\mathbf{p}}] = [\check{\mathbf{y}}] \times [\check{\mathbf{p}}]. \tag{6.60}$$

Тогда

$$\begin{aligned} (\boldsymbol{\phi}(\mathbf{p}), \mathbf{p}) \in \check{\mathbb{X}} &\Leftrightarrow \boldsymbol{\phi}(\mathbf{p}) \in [\check{\mathbf{y}}] \text{ и } \mathbf{p} \in [\check{\mathbf{p}}] \\ &\Leftrightarrow \mathbf{p} \in \boldsymbol{\phi}^{-1}([\check{\mathbf{y}}]) \text{ и } \mathbf{p} \in [\check{\mathbf{p}}] \\ &\Leftrightarrow \mathbf{p} \in [\check{\mathbf{p}}] \cap \boldsymbol{\phi}^{-1}([\check{\mathbf{y}}]). \end{aligned} \tag{6.61}$$

Таким образом,

$$\widehat{\mathbb{P}} = [\check{\mathbf{p}}] \cap \boldsymbol{\phi}^{-1}([\check{\mathbf{y}}]) \tag{6.62}$$

и построение множества $\widehat{\mathbb{P}}$ является задачей на обращение множества, которая может быть решена с использованием алгоритма SIVIA. Заметим, что для построения множества $\widehat{\mathbb{P}}$ доступны несколько специальных методов, когда функция $\boldsymbol{\phi}(\mathbf{p})$ линейна. В этом случае множество $\widehat{\mathbb{P}}$ является многогранником, который может быть построен точно [Walter и Pietrac-Lahanier, 1989], или заключено в некоторое множество упрощенной формы, например, эллипсоид [Fogel и Huang, 1992] или параллелотоп [Milanese и Belforte, 1982; Belforte *et al.*, 1990]. Когда функция $\boldsymbol{\phi}(\mathbf{p})$ нелинейна, имеется весьма мало методов, приводящих к гарантированным результатам. Один из них основан на обобщенном геометрическом программировании [Milanese и Vicino, 1991]. Другой подход, основанный на интервальном анализе и подобный алгоритму SIVIA, был независимо разработан в [Moore, 1992].

Пример 6.8. *В качестве иллюстрационного примера будет рассмотрена двухпараметрическая задача, которая позволяет графически отображать получаемые покрытия. Этот пример взят из работы* [Jaulin и Walter, 1993a] *и является упрощенным вариантом задачи, рассмотренной в* [Milanese и Vicino, 1991]. *Пример связан с электрохимией и его можно найти в* [Braems *et al.*, 2001]. *Множество* $\widehat{\mathbb{P}}$, *которое нужно оценить, содержит все значения вектора* $\mathbf{p}$ *параметров, такие, что график функции*

$$f(\mathbf{p}, t) = 20\exp(-p_1 t) - 8\exp(-p_2 t) \tag{6.63}$$

проходит через множества неопределенности всех замеров — вертикальные отрезки (рис. 6.9). *Численные значения соответствующих множеств неопределенности приведены в* табл. 6.5.

Таблица 6.5. Моменты замеров и соответствующие множества неопределенности

i	t_i	$[\check{y}_i]$
1	0,75	[2,7; 12,1]
2	1,5	[1,04; 7,14]
3	2,25	[−0,13; 3,61]
4	3	[−0,95; 1,15]
5	6	[−4,85; −0,29]
6	9	[−5,06; −0,36]
7	13	[−4,1; −0,04]
8	17	[−3,16; 0,3]
9	21	[−2,5; 0,51]
10	25	[−2; 0,67]

В этом модельном примере интервальные данные (Прим. перев: т. е. множества неопределенности замеров) рассчитывались прибавлением интервала несмещенной ошибки с радиусами $\rho_i = 0{,}5|y_i| + 1$ *к каждой* i*-й компоненте вектора замеров*

$$\check{\mathbf{y}} = (7{,}39, 4{,}09, 1{,}74, 0{,}097, -2{,}57, -2{,}71, -2{,}07, -1{,}44, -0{,}98, -0{,}66)^{\mathrm{T}}, \tag{6.64}$$

для $i = 1, \ldots, 10$. *Апостериорное множество допустимых значений параметров (допустимое множество) получается по* (6.62), *где область поиска параллелотопа* $[\hat{\mathbf{p}}]$ *бралась как* $[-0{,}1, 1{,}5]^{\times 2}$. *Координатные функции для функции* $\boldsymbol{\phi}$ *задавались как*

$$\phi_i(\mathbf{p}) = 20\exp(-p_1 t_i) - 8\exp(-p_2 t_i), \quad i = 1, \ldots, 10. \tag{6.65}$$

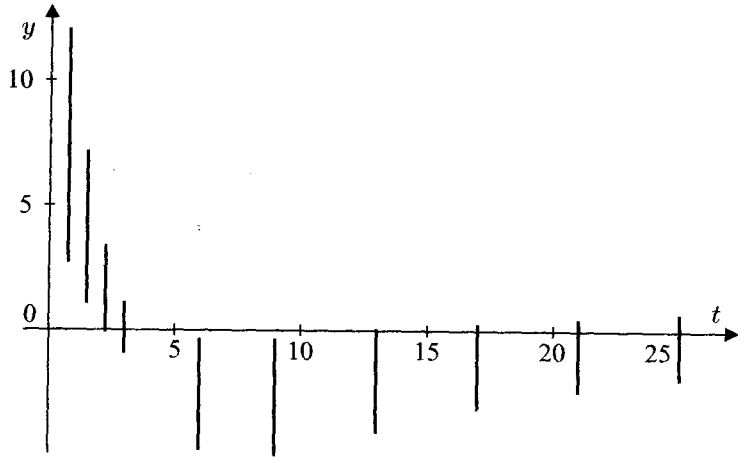


Рис. 6.9. Экспериментальные данные Примера 6.8 и множества (интервалы) неопределенности замеров

Менее чем за 4 *секунды на компьютере* Pentium 133 *алгоритм* SIVIA *рассчитывает покрытие, показанное на* рис. 6.10, *аппроксимирующее сверху и снизу апостериорное допустимое множество параметров* $\mathbf{p}$. ■

6.3.3. Защита от выбросов

Разработанный подход до сих пор опирался на гипотезу о том, что априорное допустимое множество $\check{\mathbb{X}}$ содержало истинные значения переменных. Такая гипотеза, к сожалению, не всегда реалистична. Предположим снова, что множество $\check{\mathbb{X}}$ есть параллелотоп, заданный (6.55). Даже если каждая априорная интервальная компонента множества $\check{\mathbb{X}}$ получена из разумной модели ошибок измерения, например, по паспортным данным измерительного устройства (датчика), на практике измерения ведут себя не так, как предполагалось. Например,

- некоторые ограничения из множества $\mathbb{M}$ не всегда верны;
- датчик может ломаться во время измерений;
- могут иногда возникать ситуации, когда ошибки измерений становятся плохими.

Переменная x_i, истинное значение которой не принадлежит априорному интервалу (*Прим. перев.*: т. е. множеству неопределенности замера), называется *выбросом* (промахом). Задача определения местоположения робота (см. Главу 8) дает пояснения, когда такие выбросы являются неизбежными в большей или меньшей степени. Присутствие выбросов может быть обнаружено после завершения процесса оценивания, если выявляется, что апостериорное допустимое множество становится пустым. К сожалению, процесс может и не обнаружить выбросы. Чтобы защититься от того факта, что априорное множество $\check{\mathbb{X}}$ может не содержать истинных значений некоторых переменных из вектора $\mathbf{x}$, можно увеличить (выполнить релаксацию) множества $\check{\mathbb{X}}$. Для этого рассмотрим *функцию релаксации* $\lambda : \mathbb{R}^n \to [0,1]$, такую, что $\lambda = 1$ тогда и только тогда, когда $\mathbf{x} \in \check{\mathbb{X}}$. *Априорное допустимое множество с релаксацией*

$$\check{\mathbb{X}}_\alpha \triangleq \{\mathbf{x} \in \mathbb{R}^n \mid \lambda(\mathbf{x}) \geqslant \alpha\} = \lambda^{-1}([\alpha, 1]) \tag{6.66}$$

при $\alpha \in [0,1]$ содержит множество $\check{\mathbb{X}}$. Кроме того, $\check{\mathbb{X}}_1 = \check{\mathbb{X}}$ и $\check{\mathbb{X}}_0 = \mathbb{R}^n$.

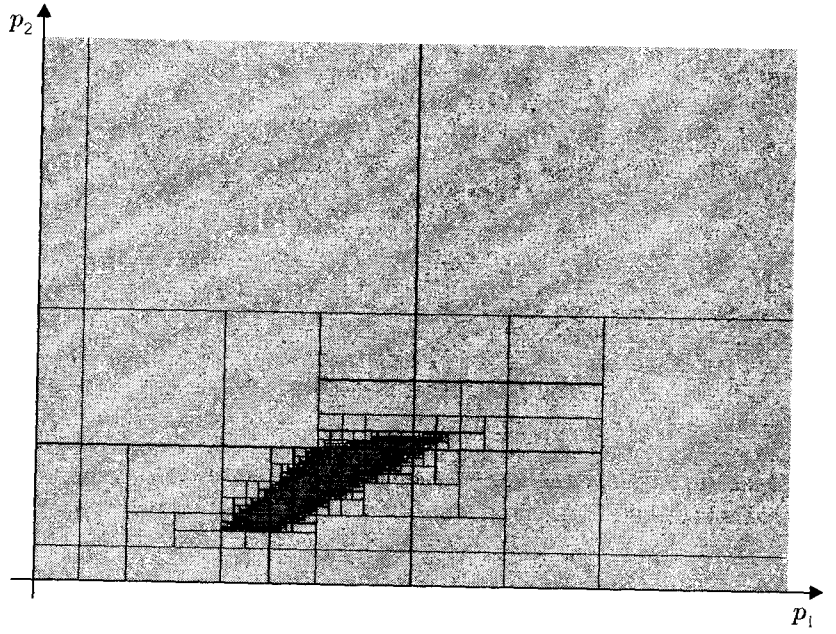


Рис. 6.10. Покрытие, выработанное алгоритмом SIVIA для Примера 6.8, для аппроксимации сверху и снизу апостериорного множества $\widehat{\mathbb{P}}$ параметров; внешнее ограничение (область поиска) соответствует параллелотопу $[-0{,}1, 1{,}5]^{\times 2}$

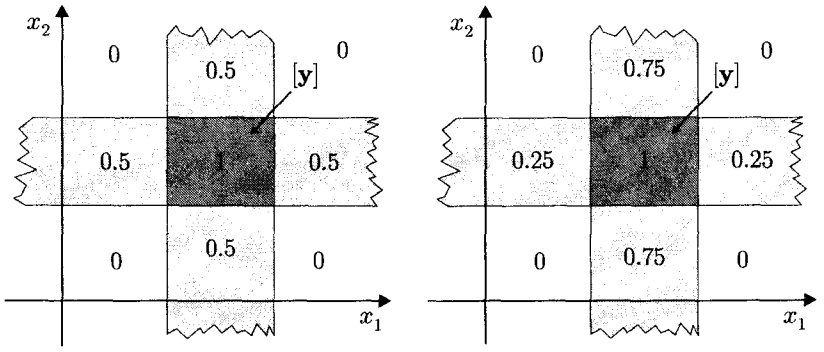


Рис. 6.11. Функции релаксации

Пример 6.9. *Определим характеристическую функцию* $\pi_{[a,b]} : \mathbb{R} \to \mathbb{R}$ *интервала* $[a, b]$ *как*

$$\pi_{[a,b]}(x) = \begin{cases} 1, & \text{если } x \in [a,b], \\ 0, & \text{если } x \notin [a,b]. \end{cases} \tag{6.67}$$

Функция

$$\lambda_1(\mathbf{x}) = \frac{\pi_{[y_1]}(x_1) + \ldots + \pi_{[y_n]}(x_n)}{n} \tag{6.68}$$

является функцией релаксации для параллелотопа $[\mathbf{y}] = [y_1] \times \ldots \times [y]$. *Если величины* $w_i, i \in \{1, \ldots, n\}$, *являются положительными весовыми коэффициентами, то функция*

$$\lambda_w(\mathbf{x}) = \frac{\omega_1 \pi_{[y_1]}(x_1) + \ldots + \omega_n \pi_{[y_n]}(x_n)}{\omega_1 + \ldots + \omega_n} \tag{6.69}$$

также является функцией релаксации для параллелотопа $[\mathbf{y}] = [y_1] \times \ldots \times [y_n]$. Рис. 6.11 *иллюстрирует функции релаксации* $(\pi_{[y_1]}(x_1) + \pi_{[y_2]}(x_2))/2$ *(слева) и* $(3\pi_{[y_1]}(x_1) + \pi_{[y_2]}(x_2))/4$ *(справа).* ■

Разрешение q замерам из n замеров x_i быть вне их априорных допустимых интервалов приводит к увеличению априорного допустимого параллелотопа $\check{\mathbb{X}}$ при выборе

$$\lambda(\mathbf{x}) = \frac{\pi_{[\check{x}_1]}(x_1) + \ldots + \pi_{[\check{x}_n]}(x_n)}{n} \quad \text{и} \quad \alpha = 1 - \frac{q}{n}, \tag{6.70}$$

где $[\check{x}_i]$ — интервальные компоненты параллелотопа $\check{\mathbb{X}}$. Проверка включения для параллелотопа $\check{\mathbb{X}}_\alpha$ может быть выполнена через интервальное оценивание функции $\lambda(\mathbf{x})$, а описание параллелотопа $\check{\mathbb{X}}_\alpha$ может быть выполнено алгоритмом SIVIA. Для описания *релаксированного апостериорного допустимого множества* параметров, определяемого как

$$\widehat{\mathbb{P}}_\alpha = \{\mathbf{p} \in \mathbb{R}^{n_\mathrm{p}} \mid \exists \mathbf{t} \in \mathbb{R}^{n_\mathrm{t}},\ (\boldsymbol{\phi}(\mathbf{p}, \mathbf{t}), \mathbf{p}, \mathbf{t}) \in \check{\mathbb{X}}_\alpha\} \tag{6.71}$$

(см. (6.49)), алгоритм SIVIA также может быть использован с учетом того, что проверка включения доступна для проверки

$$\tau_\mathrm{p}^\alpha \triangleq (\exists \mathbf{t} \in \mathbb{R}^{n_\mathrm{t}} \mid (\boldsymbol{\phi}(\mathbf{p}, \mathbf{t}), \mathbf{p}, \mathbf{t}) \in \check{\mathbb{X}}_\alpha). \tag{6.72}$$

Такая проверка задается алгоритмом из табл. 6.4, где проверка $\tau_\mathrm{pt}(\mathbf{p}, \mathbf{t})$ заменяется проверкой $\tau_\mathrm{pt}^\alpha \triangleq ((\boldsymbol{\phi}(\mathbf{p}, \mathbf{t}), \mathbf{p}, \mathbf{t}) \in \check{\mathbb{X}}_\alpha)$.

Чтобы проиллюстрировать этот подход, предположим, что величины, взятые в качестве независимых переменных, определены и известны, поэтому компоненты вектора $\mathbf{t}$ не являются переменными задачи оценивания. Априорная допустимая область переменных тогда — параллелотоп $\check{\mathbb{X}} = [\check{\mathbf{y}}] \times [\check{\mathbf{p}}]$. При этом, апостериорное допустимое множество описывается как

$$\widehat{\mathbb{P}} = \{\mathbf{p} \in \mathbb{R}^{n_\mathrm{p}} \mid (\boldsymbol{\phi}(\mathbf{p}, \mathbf{t}), \mathbf{p}, \mathbf{t}) \in [\check{\mathbf{y}}] \times [\check{\mathbf{p}}]\}. \tag{6.73}$$

Разрешим теперь замерам y_i в количестве до q из общего числа n_y быть вне априорных допустимых интервалов. Это приводит к замещению параллелотопа $[\check{\mathbf{y}}]$ в (6.73) на множество

$$\check{\mathbb{Y}}_q \triangleq \{\mathbf{y} \in \mathbb{R}^{n_\mathrm{p}} \mid \pi_{[\check{y}_1]}(y_1) + \ldots + \pi_{\check{y}_{n_\mathrm{y}}}(y_{n_\mathrm{y}}) \leqslant n_\mathrm{y} - q\} \tag{6.74}$$

и $\widehat{\mathbb{P}}$ на

$$\widehat{\mathbb{P}}_q = \{\mathbf{p} \in [\check{\mathbf{p}}] \mid \boldsymbol{\phi}(\mathbf{p}) \in \check{\mathbb{Y}}_q\} = [\check{\mathbf{p}}] \cap \boldsymbol{\phi}^{-1}(\check{\mathbb{Y}}_q). \tag{6.75}$$

Алгоритм SIVIA может быть применен для описания множества $\widehat{\mathbb{P}}_q$ для любого предписанного целого числа q из интервала $[0, n_\mathrm{y}]$.

Пример 6.10. *Рассмотрим снова* Пример 6.8, *но предположим, что теперь вектор* $\check{\mathbf{y}}$, *содержащий все доступные данные, имеет вид*:

$$\check{\mathbf{y}} = (7{,}39,\ 0,\ 1{,}74,\ 0{,}097,\ -2{,}57,\ -2{,}71,\ -2{,}07,\ 0,\ -0{,}98,\ -0{,}66)^\mathrm{T}, \tag{6.76}$$

что иллюстрирует ситуацию, в которой имеется два выброса ($\check{y}(2) = 0$ *вместо* 4,09 *и* $\check{y}(8) = 0$ *вместо* $-1{,}44$). *Соответствующие множества неопределенности представлены на* рис. 6.12. *Алгоритм* SIVIA *вырабатывает покрытия, представленные на* рис. 6.13 *для значений* $q = 0$ (см. рис. 6.13, *а*), $q = 1$ (см. рис. 6.13, *б*) и $q = 2$ (см. рис. 6.13, *в*); *покрытия*

связаны с множествами $\widehat{\mathbb{P}}_0$, $\widehat{\mathbb{P}}_1$ *и* $\widehat{\mathbb{P}}_2$, *соответственно. Требуемая точность была* $\varepsilon = 0{,}005$ *и априорный параллелотоп поиска был* $[\mathbf{p}] = [-0{,}1,\ 1{,}5] \times [-0{,}1,\ 1{,}5]$. *Для указанной выборки* (6.76) *с двумя выбросами множество* $\widehat{\mathbb{P}}_0$ *оказывается пустым, и это доказывает, что данные содержат по крайней мере один выброс. Тот факт, что множество* $\widehat{\mathbb{P}}_1$ *непусто, должен служить предупреждением, что выбросы могут быть и не обнаружены. Если взглянуть на аппроксимацию сверху апостериорного допустимого множества, которое можно было бы получить при отсутствии выбросов (множество* $\widehat{\mathbb{P}}$, см. рис. 6.10), *то должны были бы придти к переоценке действительного числа выбросов. Поскольку реально есть два выброса, то множество* $\widehat{\mathbb{P}}_2$ *дает такую аппроксимацию сверху и содержит множество* $\widehat{\mathbb{P}}$. *Множество* $\widehat{\mathbb{P}}_2$ *является несвязным, поскольку есть две различных стратегии исключения данных по двум интервалам, чтобы сделать совместными остающиеся восемь переменных.* ■

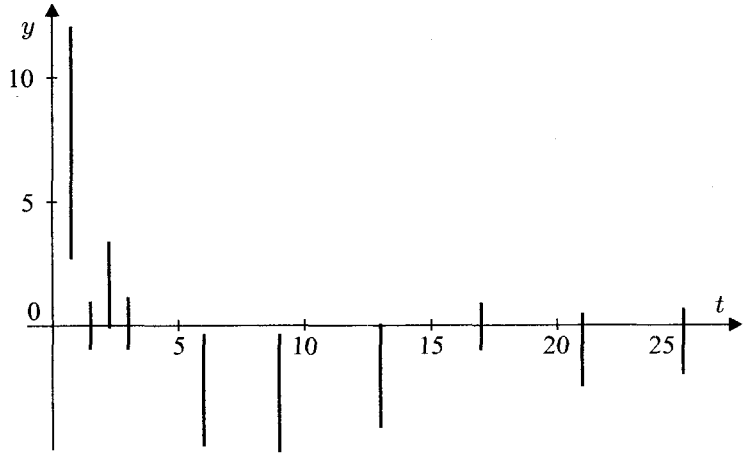


Рис. 6.12. Множества (интервалы) неопределенности замеров при двух выбросах, замеры № 2 и № 8

Размер множества $\widehat{\mathbb{P}}_q$ возрастает с ростом числа q, и, разумеется, нужен некоторый компромисс между степенью защиты от выбросов и размером результирующего множества оценки параметров. Возможная стратегия [Jaulin *et al.*, 1996] состоит в том, чтобы начать оценивание, задав $q = 0$, и далее увеличивать q на единицу до тех пор, пока множество $\widehat{\mathbb{P}}_q$ остается

пустым. Такой выбор числа q, соответствующий гарантированной реализации алгоритма OMNE (Outlier Minimal Number Estimator - алгоритм оценки минимального числа выбросов, рассмотренный в [Lahanier *et al.*, 1987]), в рассматриваемом Примере 6.10 приводит к остановке после вычисления множества $\widehat{\mathbb{P}}_1$.

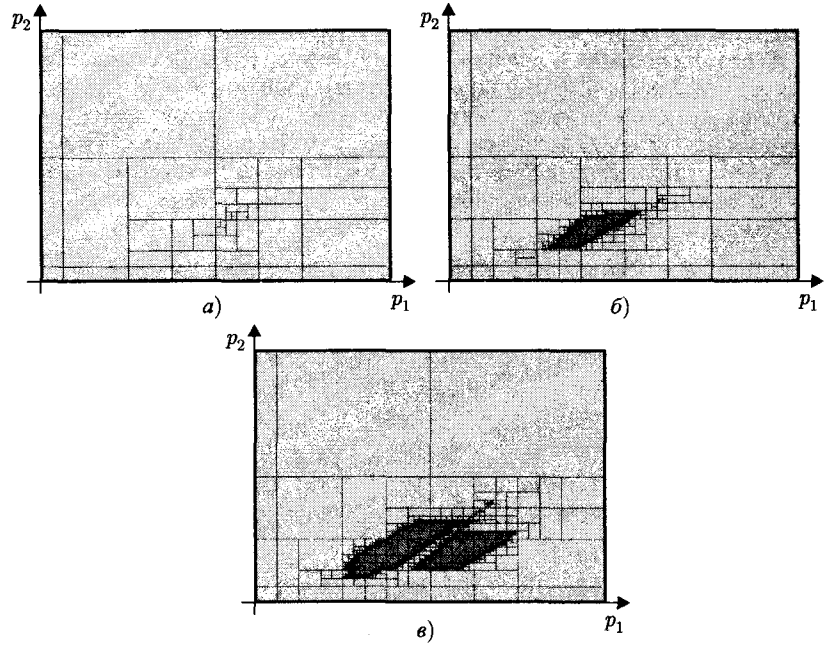


Рис. 6.13. Покрытия, выработанные алгоритмом SIVIA в Примере 6.10: *а*) в предположении об отсутствии выбросов; *б*) допускается один выброс; *в*) допускается наличие до двух выбросов; внешние рамки соответствуют параллелотопу поиска $[-0{,}1,\ 1{,}5] \times [-0{,}1,\ 1{,}5]$ на плоскости параметров

6.3.4. Случай неопределенности независимых переменных

В литературе, посвященной оцениванию параметров, мало внимания уделено случаю с неопределенностью в независимых переменных. Если координатные компоненты функции $\phi(\mathbf{p}, \mathbf{t})$ — билинейны по $\mathbf{t}$ и $\mathbf{p}$, и если

априорные допустимые области $[\check{\mathbf{t}}]$ и $[\check{\mathbf{p}}]$ для $\mathbf{t}$ и $\mathbf{p}$ — параллелотопы, то задача оценивания апостериорного допустимого множества $\widehat{\mathbb{P}}$ может быть решена точно методом, описанном в [Cerone, 1991; 1996]. Может быть также рассчитана эллипсоидальная аппроксимация сверху этого множества [Norton, 1987; Clémenti и Gentil, 1990; Pronzato и Walter, 1994; Norton, 1996; Veres и Norton, 1996].

Мы рассмотрим более общий случай, когда функция $\phi(\mathbf{p}, \mathbf{t})$ может быть нелинейна по $\mathbf{p}$ и $\mathbf{t}$. Самые лучшие, гарантирующие и точные результаты такого оценивания в нелинейном случае впервые были представлены в [Jaulin и Walter, 1999]. Оценивание параметров в нелинейных задачах с такими *ошибками в переменных* рассматривались долгое время в связи с задачами на метод наименьших квадратов еще давно [Schwnetlick и Tiller, 1985], но результаты там были получены локальными методами, и, таким образом, не являются гарантирующими.

Таблица 6.6. Априорные допустимые интервалы для Примера 6.11

i	$\check{t}_i$	$[\check{t}_i]$	$[\check{y}_i]$
1	0,75	[−0,25, 1,75]	[2,7, 12,1]
2	1,5	[0,5, 2,5]	[1,04, 7,14]
3	2,25	[1,25, 3,25]	[−0,13, 3,61]
4	3	[2, 4]	[−0,95, 1,15]
5	6	[5, 7]	[−4,85, −0,29]
6	9	[8, 10]	[−5,06, 0,36]
7	13	[12, 14]	[−4,1, 0,04]
8	17	[16, 18]	[3,16, 0,3]
9	21	[20, 22]	[−2,5, 0,51]
10	25	[24, 26]	[−2, 0,67]

Пример 6.11. *Рассмотрим снова* Пример 6.5 (с. 193). *Предположим теперь, что значения независимых переменных имеют неопределенность. Априорный интервал (множество неопределенности) $[\check{t}_i]$ каждого значения получено добавлением интервала $[-1, 1]$ к моменту $\check{t}_i$ измерения. Полученные интервалы неопределенности $[\check{t}_i]$, $i = 1, \ldots, 10$, приведены в* табл. 6.6 *и показаны на* рис 6.14. *Множества неопределенности каждой пары «замер — момент времени» отмечены параллелотопами серого цвета. Множество $\widehat{\mathbb{P}}$, которое необходимо оценить, содержит все значения вектора параметров $\mathbf{p} = (p_1, p_2)^{\mathrm{T}}$, такие, что график функции*

$$f(\mathbf{p}, t) = 20\exp(-p_1 t) - 8\exp(-p_2 t) \tag{6.77}$$

проходит через все эти десять параллелотопов, (см. рис. 6.14). *Искомое множество описывается*

$$\widehat{\mathbb{P}} = \{\mathbf{p} \in \mathbb{R}^{n_p} \mid \exists \mathbf{t} \in \mathbb{R}^{n_t}, (\boldsymbol{\phi}(\mathbf{p}, \mathbf{t}), \mathbf{p}, \mathbf{t}) \in \check{\mathbb{X}}\}, \tag{6.78}$$

где

$$\phi_i(\mathbf{p}, t) = \phi_i(\mathbf{p}, t_i) = 20\exp(-p_1 t_i) - 8\exp(-p_2 t_i) \tag{6.79}$$

и

$$\check{\mathbb{X}} = [\check{y}_i] \times \ldots \times [\check{y}_{10}] \times [\check{p}_1] \times [\check{p}_2] \times [\check{t}_1] \times \ldots \times [\check{t}_{10}]. \tag{6.80}$$

Априорный параллелотоп параметров брался в виде

$$\check{\mathbf{p}} = [\check{p}_1] \times [\check{p}_2] = [0, 1{,}2] \times [0, 0{,}5]. \tag{6.81}$$

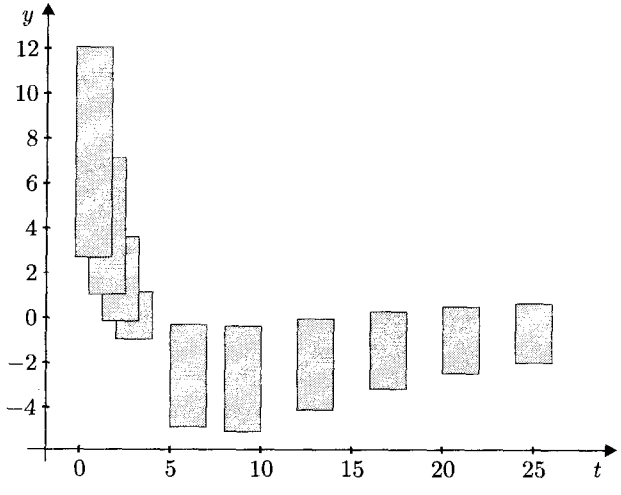


Рис. 6.14. Экспериментальные данные с неопределенностью в замерах и моментах времени

Алгоритм SIVIA *может быть применен для оценивания множества $\widehat{\mathbb{P}}$, с учетом того, что проверка включения доступна для*

$$\exists \mathbf{t} \in \mathbb{R}^{10} \mid (\boldsymbol{\phi}(\mathbf{p}, \mathbf{t}), \mathbf{p}, \mathbf{t}) \in \check{\mathbb{X}}. \tag{6.82}$$

Эта проверка может быть переписана следующим образом:

$$\exists (t_1, \ldots, t_10) \in \mathbb{R}^{10},\ (\phi_1(\mathbf{p}, t_1), \ldots, \phi_{10}(\mathbf{p}, t_{10}), p_1, p_2, t_1, \ldots, t_{10}) \in [\check{y}_1] \times \ldots \times [\check{y}_{10}] \times [\check{p}_1] \times [\check{p}_2] \times [\check{t}_1] \times \ldots \times [\check{t}_{10}], \quad (6.83)$$

или в эквивалентном виде:

$$\exists \begin{pmatrix} t_1 \\ \vdots \\ t_{10} \end{pmatrix} \in \begin{pmatrix} [\check{t}_1] \\ \vdots \\ [\check{t}_{10}] \end{pmatrix} \mid \begin{pmatrix} \phi_1(\mathbf{p}, t_1) \\ \vdots \\ \phi_{10}(\mathbf{p}, t_{10}) \end{pmatrix} \in \begin{pmatrix} [\check{y}_1] \\ \vdots \\ [\check{y}_{10}] \end{pmatrix} \quad (6.84)$$

$$\text{и } \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} \in \begin{pmatrix} [\check{p}_1] \\ [\check{p}_2] \end{pmatrix},$$

т. е. как

$$(\mathbf{p} \in [\check{\mathbf{p}}]) \wedge \eta_1(\mathbf{p}) \wedge \ldots \wedge \eta_{n_y}(\mathbf{p}), \quad (6.85)$$

где $\eta_i(\mathbf{p})$ *— проверка*

$$\eta_i(\mathbf{p}) = (\exists t_1 \in [\check{t}_i] \mid \phi_i(\mathbf{p}, t_i) \in [\check{y}_i]). \quad (6.86)$$

Проверка включения для $\eta_i(\mathbf{p})$ *была далее получена по алгоритму* табл. 6.4. *Для* $\varepsilon = 0{,}01$ *алгоритм* SIVIA *за* 38 *секунд на компьютере* Pentium 133 *вырабатывает покрытие, показанное на* рис. 6.15 [Jaulin *и* Walter, 1999].

Параллелотопы, отмеченные темно-серым цветом, составляют аппроксимацию снизу (изнутри) множества $\widehat{\mathbb{P}}$, *а параллелотопы, отмеченные серым цветом, имеют пустое пересечение с множеством* $\widehat{\mathbb{P}}$. *Для параллелотопов, отмеченных черным, нельзя сделать однозначного вывода.* ■

6.3.5. Вычисление интервальной оболочки апостериорного допустимого множества

В параграфах 6.3.2–6.3.4 оценивание параметров иллюстрировалось на модифицированной задаче, рассмотренной в [Milanese и Vicino, 1991]. Исходной задачей там было вычисление интервальной оболочки $[\widehat{\mathbb{P}}]$ апостериорного множества $\widehat{\mathbb{P}}$ параметров. Чрезвычайно интересным становится вопрос: нельзя ли вычислить более точную аппроксимацию сверху апостериорного допустимого множества, или же это окажется слишком трудоемким и бесполезным. Чтобы пояснить такое сравнение, рассмотрим здесь задачу из [Milanese и Vicino, 1991] для иллюстрации нашего подхода.

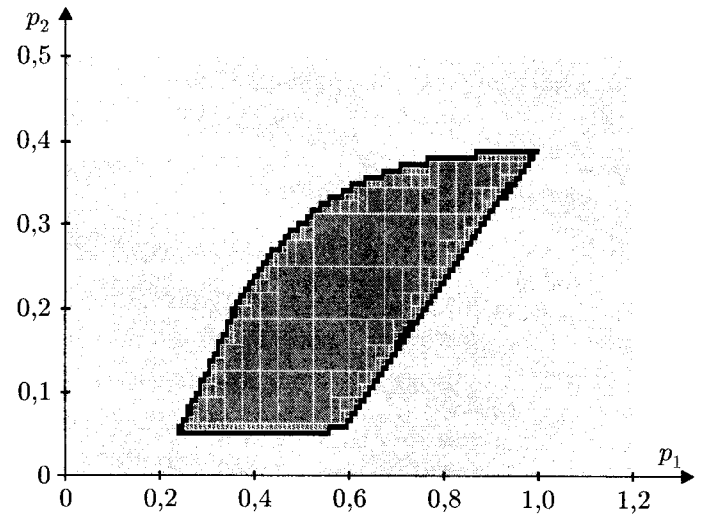


Рис. 6.15. Покрытие, выработанное алгоритмом SIVIA, для оценивания апостериорного допустимого множества $\widehat{\mathbb{P}}$ параметров в Примере 6.11

Пример 6.12. *Рассмотрим ту же ситуацию, что и в* Примере 6.8, *но с числом параметров* $n_{\mathbf{p}} = 4$ *вместо* 2, *а допустимое множество* $\mathbb{M}$ *определяется уравнениями*

$$y_i = p_1 \exp(-p_2 t_i) + p_3 \exp(-p_4 t_i), \quad i = 1, \ldots, 10. \quad (6.87)$$

Априорный допустимый параллелотоп параметров описывается $[\check{\mathbf{p}}] = [2, 60] \times [0, 1] \times [-30, -1] \times [0, 0{,}5]$. *Как и в* Примере 6.8, *выходные данные задаются по* (6.64), *а известные моменты измерений задаются* табл. 6.5, *только множество неопределенности каждого* i*-го замера определяется с максимальной ошибкой* $\rho_i = 0{,}05|y_i| + 0{,}1$ (*вместо* $\rho_i = 0{,}5|y_i| + 1$, *как в* Примере 6.8). *Апостериорное допустимое множество определяется как*

$$\widehat{\mathbb{P}} = \{\mathbf{p} \in \mathbb{R}^4 \mid (\boldsymbol{\phi}(\mathbf{p}), \mathbf{p}) \in [\check{\mathbf{y}}] \times [\check{\mathbf{p}}]\} = \boldsymbol{\phi}^{-1}([\check{\mathbf{y}}]) \cap [\check{\mathbf{p}}]. \quad (6.88)$$

Теперь, множество $\boldsymbol{\phi}^{-1}([\check{\mathbf{y}}])$ *может быть определено следующими неравенствами*:

$$p_1 \exp(-p_2 t_i) + p_3 \exp(-p_4 t_i) \in [\check{y}_i], \quad i = 1, \ldots, 10, \quad (6.89)$$

и, следовательно, алгоритм HULL, *описанный в* Главе 5, (см. табл. 5.4, с. 154), *может быть применен для нахождения параллелотопа* $[\widehat{\mathbb{P}}]$. *При* $\varepsilon = 0{,}001$ *поиск аппроксимаций* $[\widehat{\mathbb{P}}]$ *снизу и сверху занимает менее* 8 *секунд на компьютере* Pentium 133; *аппроксимации имеют вид:*

$$[\mathbf{p}_{\text{in}}] = [17{,}2,\ 26{,}79] \times [0{,}301,\ 0{,}49] \times [-16,\ -5{,}4] \times [0{,}0767,\ 0{,}1354],$$
$$[\mathbf{p}_{\text{out}}] = [17{,}05,\ 27] \times [0{,}298,\ 0{,}495] \times [-16{,}2,\ -5{,}34] \times [0{,}0763,\ 0{,}136].$$

В работе [Milanese и Vicino, 1991] *для решения этой задачи был предложен подход на основе обобщенного геометрического программирования, использующий алгоритм* Фалька [Falk, 1973]. *Этот подход дает менее точные результаты, чем описываемый здесь, и требует, по крайней мере, на порядок большего времени вычислений* [Jaulin, 2000a]. ■

6.4. Гарантированное оценивание состояний

6.4.1. Введение

В этом параграфе рассматривается гарантированное оценивание вектора состояния нелинейной дискретной по времени системы при наличии ограниченной ошибки. Читатели, незнакомые с понятием фазового состояния, могут заглянуть в параграф 7.2 (с. 239) для первого ознакомления с простым линейным случаем. Рассмотрим нелинейную дискретную систему, описываемую выражениями:

$$\begin{cases} x_1(k) = f_1(x_1(k-1),\ \ldots,\ x_{n_x}(k-1), k), \\ \quad\vdots \quad \vdots \quad \vdots \\ x_{n_x}(k) = f_{n_x}(x_1(k-1),\ \ldots,\ x_{n_x}(k-1), k), \\ y_1(k) = q_1(x_1(k),\ \ldots,\ x_{n_x}(k), k), \\ \quad\vdots \quad \vdots \quad \vdots \\ y_{n_y}(k) = g_{n_y}(x_1(k),\ \ldots,\ x_{n_x}(k), k), \end{cases} \tag{6.90}$$

где k — номер момента времени, принимающий значения от 1 до $\overline{k}$; $x_1(k),\ \ldots,\ x_{n_x}(k)$ — фазовые переменные (переменные состояния); $y_1(k),\ \ldots,\ y_{n_y}(k)$ — выходные переменные; f_i и g_i — известные функции, которые могут быть представлены конечными алгоритмами. Для простоты изложения входные переменные не указаны, но это не будет создавать

каких-либо конкретных концептуальных трудностей. В векторной форме (6.90) можно записать в более понятном виде:

$$\begin{cases} \mathbf{x}(k) = \mathbf{f}(\mathbf{x}(k-1), k), \\ \mathbf{y}(k) = \mathbf{g}(\mathbf{x}(k), k). \end{cases} \tag{6.91}$$

Полагается, что вектор выходных переменных $\mathbf{y}(k)$ представляет замер состояния системы, и задача, которую необходимо решить, состоит в оценивании состояния $\mathbf{x}(k)$ по имеющейся информации. Когда эти сведения обрабатываются в реальном времени, данные $\check{\mathbf{y}}(i), i > k$, которые поступят в систему после момента k, недоступны при оценивании состояния $\mathbf{x}(k)$. Таким образом, наиболее общим допущением является предположение, что при оценивании могут использоваться только накопленные предыдущие замеры. Но мы также рассматриваем случай апостериорной обработки информации (*Прим. перев.*: т. е. обработки пост-фактум, после накопления всей этой выборки замеров.) не в реальном времени, когда замеры, следующие за моментом k, используются для нахождения оценки $\mathbf{x}(k)$.

В линейном случае (т. е. когда $\mathbf{f}$ и $\mathbf{g}$ линейны) для оценивания состояния доступными являются многие подходы. Их можно классифицировать по тому, как они работают с неопределенностью. Некоторые из них очевидно не учитывают тот факт, что (6.91) является аппроксимацией реального процесса и что измерения искажены шумом. Это как раз имеет место, например, в наблюдателях Люэнбергера для оценивания состояния [Luenberger, 1966].

Другие алгоритмы оценивания, такие как широко используемый фильтр Калмана [Kalman, 1960; Sorenson, 1983], основаны на статистическом описании неопределенности и предполагают, что шум измерения и возмущения состояния являются реализациями случайных переменных с известными статистическими свойствами. Последняя группа методов соответствует оцениванию состояния [Shweppe, 1968; Witsenhausen, 1968; Bertsekas и Rhodes, 1971; Chernous'ko, 1994; Durien *et al.*, 1996; Maksarov и Norton, 1996; Milanese *et al.*, 1996; Kurzhanski и Valyi, 1997, и ссылки в этих работах]. Эти методы основываются на предположении, что все неопределенные переменные принадлежат известным компактным множествам, и на попытке построить некоторые простые множества, такие как эллипсоиды, параллелотопы с ребрами, ориентированными по осям декартовой системы, или параллелотопы общего положения, гарантированно содержащие все векторы состояния, совместные с указанным предположением.

В нелинейном случае методология оценивания развита значительно слабее и все еще активно разрабатывается даже в детерминированном случае [Kang и Krener, 1998]. Когда неопределенность учитывается очевидным образом, то это часто основывается в большинстве случаев на линеариза-

ции. Обычно используется обобщенный фильтр Калмана [Gelb, 1974], основанный на линеаризации системы (6.91) около фазовой траектории. Такая линеаризация по своей сути локальна и может не давать надежных оценок. Она также приводит к произвольной статистической интерпретации матриц ковариации, рассчитанных по весьма сомнительному алгоритму, поскольку преобразование статистических свойств входных возмущений нелинейной системой в значительной степени неизвестно. В этом параграфе будет рассмотрен новый подход к нелинейному оцениванию состояния, частично опирающийся на исследования [Kieffer *et al.*, 1998; 1999] и [Jaulin *et al.*, 2001б].

Множество всех переменных, участвующих в (6.90), представляется

$$\begin{aligned}&\{x_1(0), \ldots, x_{n_x}(0),\\ &x_1(1), \ldots, x_{n_x}(1), y_1(1), \ldots, y_{n_y}(1),\\ &\vdots\\ &x_1(\overline{k}), \ldots, x_{n_x}(\overline{k}), y_1(\overline{k}), \ldots, y_{n_y}(\overline{k})\}.\end{aligned} \tag{6.92}$$

Предположим, что известно, что компоненты вектора начального состояния $x_1(0), \ldots, x_{n_x}(0)$ лежат в некоторых априорных ограниченных интервалах $[\check{x}_1(0)], \ldots, [\check{x}_{n_x}(0)]$. Каждый интервал $[\check{x}_i(0)]$ представляет априорное знание по переменной $x_i(0)$ состояния и может быть произвольно большим в отсутствие информации. Пусть $\check{y}_i(k)$ — результат измерения наблюдаемой переменной $y_i(k)$. При ограниченной ошибке измерений, как это рассматривалось в Главе 3, измерения, выполненные в k-е моменты времени, дают априорные интервалы (множества — интервалы неопределенности) $[\check{y}_i(k)]$, содержащие по определению истинные значения наблюдаемых переменных $y_i(k)$.

Когда оценивание должно выполняться в реальном времени, то перед получением замера интервал $[\check{y}_i(k)]$ полагается равным всей числовой оси $[-\infty, \infty]$. Априорные области $[\check{x}_i(k)]$ переменных состояния $x_i(k)$, $i = 1, \ldots, n_x$, $k = 1, \ldots, \overline{k}$ равными числовой оси $[-\infty, \infty]$.

Вектор начальных состояний $x_1(0), \ldots, x_{n_x}(0)$ имеет особый статус, так как если эти значения известны, то значения всех других переменных могут быть рассчитаны с помощью (6.90). По этой причине они будут именоваться *исходными переменными*, а вектор $\mathbf{x}(0)$ — *исходным вектором*. Пусть $\mathbf{y}$ — вектор всех наблюдаемых переменных

$$\mathbf{y} = (y_1(1), \ldots, y_{n_y}(1), \ldots, y_1(\overline{k}), \ldots, y_{n_y}(\overline{k}))^{\mathrm{T}}, \tag{6.93}$$

а $\boldsymbol{\phi}$ — векторная функция, по которой вычисляются значения компонент вектора $\mathbf{y}$, когда (6.90) моделируется, начиная со значения $\mathbf{x}(0)$. Прежде всего

мы рассмотрим оценивание в нереальном времени (пост-фактум) множества $\hat{\mathbb{X}}(0)$ векторов начальных состояний $\mathbf{x}(0) = (x_1(0), \ldots, x_{n_x}(0)$, принадлежащих априорной области $[\check{\mathbf{x}}(0)]$ и совместных с априорной областью переменных $\mathbf{y}$, т. е. с

$$[\check{\mathbf{y}}] = [\check{y}_1(1)] \times \ldots \times [\check{y}_{n_y}(1)] \times \ldots \times [\check{y}_1(\overline{k})] \times \ldots \times [\check{y}_{n_y}(\overline{k})]. \tag{6.94}$$

Это множество задается выражением

$$\hat{\mathbb{X}}(0) = [\check{\mathbf{x}}(0)] \cap \boldsymbol{\phi}^{-1}([\check{\mathbf{y}}]). \tag{6.95}$$

Такое оценивание по сути является задачей на обращение множества, и она может быть решена алгоритмом SIVIA, как это показано в параграфе 6.4.2. В параграфе 6.4.3 эта методология расширяется на случай оценивания всех переменных (а не только вектора начальных состояний). Будет показано, что использование оператора сжатия, основанного на методе вперед-назад (двустороннее распространение ограничений), особенно хорошо подходит к случаю оценивания состояния. Наконец, в параграфе 6.4.4 рассматривается рекурсивный случай оценивания, когда могут использоваться только предыдущие замеры.

6.4.2. Оценивание начального состояния

Этот параграф иллюстрирует применение алгоритма SIVIA для оценивания множества векторов всех начальных состояний, совместных с априорными интервальными данными [Jaulin, 1994]. Рассмотрим нелинейное уравнение состояния

$$\begin{cases} x_1(k+1) = \cos(x_1(k)x_2(k)), \\ x_2(k+1) = 3x_1(k) - \sin(x_2(k)), \\ \quad y(k) = x_1^2(k) - x_2(k). \end{cases} \tag{6.96}$$

Вектор наблюдения содержит десять замеров

$$\begin{aligned}\check{\mathbf{y}} &= (y(0), \ldots, y(9))^{\mathrm{T}}\\ &= (3, -5, 0{,}6, 2{,}2, -3{,}8, -1{,}4, 0{,}4, -1{,}2, -1{,}8, 2{,}6)^{\mathrm{T}},\end{aligned} \tag{6.97}$$

выработанных при моделировании (6.96) для $k = 0, \ldots, 9$ при векторе начального состояния $\mathbf{x}^* = (2, 1)^{\mathrm{T}}$ и при добавлении реализации ошибки в виде случайного шума, ограниченного интервалом $[-0{,}5, 0{,}5]$, к истинным

значениям $y^*(k)$. Априорный допустимый параллелотоп $[\check{\mathbf{y}}]$ для вектора $\mathbf{y}$ был получен добавлением интервала ошибки $[-0{,}5, 0{,}5]$ к компонентам вектора наблюдения $\check{\mathbf{y}}$. Имеем

$$[\check{\mathbf{y}}] = [2{,}5,\ 3{,}5] \times [-5{,}5,\ -4{,}5] \times [0{,}1,\ 1{,}1] \times [1{,}7,\ 2{,}7] \times [-4{,}3,\ -3{,}3] \times [-1{,}9,\ -0{,}9] \times [-0{,}1,\ 0{,}9] \times [-1{,}7,\ -0{,}7] \times [-2{,}3,\ -1{,}3] \times [2{,}1,\ 3{,}1]. \quad (6.98)$$

Множество всех значений вектора $\mathbf{x}(0)$ *начальных состояний из параллелотопа* $[\check{\mathbf{x}}(0)]$, совместных с априорным параллелотопом $[\check{\mathbf{y}}]$, задается (6.95). При $\varepsilon = 0{,}01$ и $[\check{\mathbf{x}}(0)] = [-5, 5]^{\times 2}$. Оценка по алгоритму SIVIA, представленная на рис. 6.16, *а*, найдена на компьютере Pentium 133 менее, чем за 1 секунду. Увеличенный фрагмент вокруг вектора $\mathbf{x}^*(0)$ истинного начального состояния, показанный на рис. 6.16, *б*, получен при $\varepsilon = 0{,}0001$ и начальном параллелотопе $[\check{\mathbf{x}}(0)] = [1{,}98, 2{,}02] \times [0{,}98, 1{,}02]$.

6.4.3. Оценивание всех переменных состояния

Как отмечалось выше, если начальное состояние есть известный точечный вектор $\mathbf{x}(0)$, то точечные значения всех других переменных могут быть вычислены моделированием соотношений (6.90) по $\mathbf{x}(0)$. Однако подход к оцениванию множества $\widehat{\mathbb{X}}(0)$, описанный в параграфе 6.4.2, не дает каких-либо оценок границ других интересующих нас переменных. Такие оценки можно было бы получить на основе множества $\widehat{\mathbb{X}}(0)$ моделированием по (6.90), но эта задача еще более сложна, чем задача обращения множества, и ее сравнительное рассмотрение будет сделано в параграфе 6.4.4. А сейчас мы рассмотрим подход, концептуально более простой. Некоторая адаптация алгоритма SIVIA, описанного в табл. 6.7, прежде всего будет использована для получения аппроксимации $\overline{\mathbb{X}}(0)$ сверху множества $\widehat{\mathbb{X}}(0)$, состоящего из объединения параллелотопов, каждый их которых имеет ширину, меньшую, чем ε. Главное отличие данной адаптации от версии в Главе 3, (см. табл. 3.1, с. 82) состоит в том, что подпараллелотопы параллелотопа $[\mathbf{x}(0)]$, которые надо бы проверять на допустимость, тем не менее подвергаются бисекции, пока их размер не станет меньше заданного допуска точности ε. Как результат, по завершению работы данного алгоритма получено больше параллелотопов, чем необходимо для описания множества $\widehat{\mathbb{X}}(0)$ при заданной точности его оценивания. Но ожидается, что моделирование множества, начинающееся с этих маленьких параллелотопов, будет более точным, чем оценивание, выполненное по начальной версии алгоритма SIVIA. В алгоритме табл. 6.7 список $\mathcal{L}$ является списком параллелотопов в пространстве начальных состояний и инициализируется как пустой. Первое

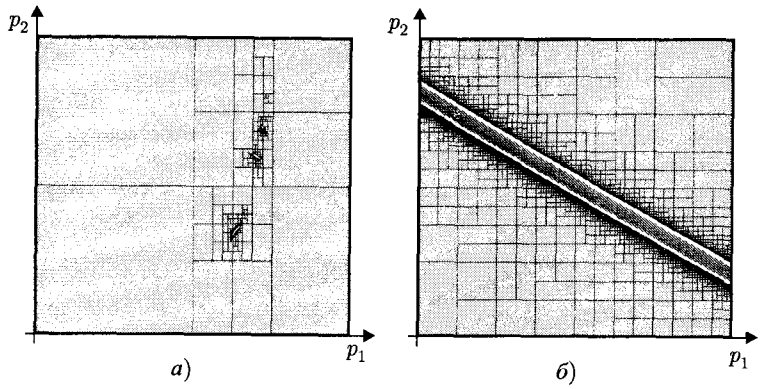


Рис. 6.16. Оценивание множества всех начальных позиций, совместных с нтервальными данными из $[-5, 5]^{\times 2}$: *а*) параллелотоп поиска $[-5, 5]$; *б*) параллелотоп поиска $[1{,}98, 2{,}02]$

обращение к алгоритму начинается с $[\mathbf{x}(0)] = [\check{\mathbf{x}}(0)]$, где $[\check{\mathbf{x}}(0)]$ – априорный параллелотоп вектора начальных состояний. Сжимающий оператор $\mathcal{C}_{\widehat{\mathbb{X}}(0)}$ – оператор для поиска решения $\widehat{\mathbb{X}}(0)$ (см. параграф 4.5, с. 130). По завершению работы алгоритма SIVIA, объединение всех параллелотопов списка $\mathcal{L}$ является аппроксимацией сверху $\overline{\mathbb{X}}(0)$ множества $\widehat{\mathbb{X}}(0) = \phi^{-1}([\check{\mathbf{y}}]) \cap [\check{\mathbf{x}}(0)]$.

Таблица 6.7. Алгоритм SIVIA для моделирования множества

Алгоритм SIVIA(вход : $[\mathbf{x}(0)], \mathcal{C}_{\widehat{\mathbb{X}}(0)}, \varepsilon$; вход/выход : $\mathcal{L}$)
1 $[\mathbf{x}(0)] := \mathcal{C}_{\widehat{\mathbb{X}}(0)}([\mathbf{x}(0)])$;
2 если $([\mathbf{x}(0)] = \varnothing)$, то возврат;
3 если $(w([\mathbf{x}(0)]) < \varepsilon)$, то $\mathcal{L} = \mathcal{L} \cup \{[\mathbf{x}(0)]\}$; возврат;
4 выполняется бисекция $[\mathbf{x}(0)]$ на $[\mathbf{x}(0)]_1$ и $[\mathbf{x}(0)]_2$;
5 SIVIA$([\mathbf{x}(0)]_1, \mathcal{C}_{\widehat{\mathbb{X}}(0)}, \varepsilon, \mathcal{L})$; SIVIA$([\mathbf{x}(0)]_2, \mathcal{C}_{\widehat{\mathbb{X}}(0)}, \varepsilon, \mathcal{L})$.

Алгоритм SETSIMULATION моделирования множества, описанный в табл. 6.8, может быть применен в дальнейшем. Он использует функции включения функций $\mathbf{f}$ и $\mathbf{g}$ для вычисления параллелотопов, содержащих последовательные значения состояния и выходных векторов для любого начального вектора в любом заданном параллелотопе из $\overline{\mathbb{X}}(0)$ (занесен-

ных в список $\mathcal{L}$). Алгоритму моделирования удается брать объединения параллелотопов для получения аппроксимаций сверху множества всех значений $\mathbf{x}(k)$ или $\mathbf{y}(k)$ для любых $\mathbf{x}(0)$ из $\widehat{\mathbb{X}}(0)$.

Таблица 6.8. Алгоритм моделирования уравнений состояния для всех параллелотопов списка $\mathcal{L}$, выработанного алгоритмом SIVIA по табл. 6.7

Алгоритм SETSIMULATION(вход : $\mathcal{L}$; выход : $[\mathbf{x}(1)], \ldots, [\mathbf{x}(\overline{k})], [\mathbf{y}(1)], \ldots, [\mathbf{y}(\overline{k})]$)
1 $[\mathbf{x}(1)] = \varnothing; \ldots; [\mathbf{x}(\overline{k}] = \varnothing;\ [\mathbf{y}(1)] = \varnothing; \ldots; [\mathbf{y}(\overline{k})] = \varnothing;$
2 для всех $[\mathbf{x}^{'}(0)] \in \mathcal{L}$
3 для $k = 1$ до $k = \overline{k}$
4 $[\mathbf{x}^{'}(k)] := [\mathbf{f}]([\mathbf{x}^{'}(k-1)]);$
5 $[\mathbf{x}(k)] := [\mathbf{x}(k)] \sqcup [\mathbf{x}^{'}(k)];$
6 $[\mathbf{y}^{'}(k)] := [\mathbf{g}]([\mathbf{x}^{'}(k)]);$
7 $[\mathbf{y}(k)] := [\mathbf{y}(k)] \sqcup [\mathbf{y}^{'}(k)].$

В качестве иллюстрации рассмотрим нелинейную систему дискретного времени

$$\begin{cases} \begin{pmatrix} x_1(k) \\ x_2(k) \end{pmatrix} = \begin{pmatrix} 0{,}1x_1(k-1) + x_2(k-1)\exp(x_1(k-1)) \\ x_1(k-1) + 0{,}1x_2^2(k-1) + \sin(k) \end{pmatrix}, \\ y(k) = x_2(k)/x_1(k), \end{cases} \tag{6.99}$$

при $k \in \{1, \ldots, 15\}$. Интервальные данные генерировались следующим образом. Сперва, от истинного значения $\mathbf{x}^*(0) = (-1, 0)^{\mathrm{T}}$ вектора начального состояния путем моделирования были рассчитаны истинные значения $\mathbf{x}^*(k)$ и $y^*(k)$, $k \in \{1, \ldots, 15\}$. Каждое истинное значение наблюдения $y^*(k)$ зашумлялось добавлением ошибки с равномерным распределением из интервала $[-e, e]$ для получения зашумленных данных $\check{y}(k)$. Наконец, множества (интервалы) неопределенности замеров $\check{y}(k)$ строились в виде $[\check{y}(k)] = [\check{y}(k) - e,\ \check{y}(k) + e]$. Таким образом, каждый интервал $[\check{y}(k)]$ гарантированно содержал неизвестное точное значение истинной величины $y^*(k)$. Задача, которую надо решить, состоит в следующем: *для заданной системы* (6.99) *интервальных данных* $[\check{y}(k)]$, *ограниченных интервалов* $[\check{x}_1(0)]$ *и* $[\check{x}_2(0)]$, *содержащих начальные состояния* $x_1(0)$ *и* $x_2(0)$, *вычислить* (*точные*) *интервальные оценки значений переменных* $x_1(k)$, $x_2(k)$ *и* $y(k)$, $k \in \{1, \ldots, 15\}$.

Сжимающим оператором для множества $\widehat{\mathbb{X}}(0)$, необходимым на Шаге 1 алгоритма табл. 6.7, может быть оператор $\mathcal{C}_{\downarrow\uparrow}$ (на основе метода вперед-

назад), описанный в параграфе 4.2.4 (с. 107), и примененный к задаче выполнения ограничений $\mathcal{H} : (\boldsymbol{\phi}(\mathbf{x}(0)) - \mathbf{y} = 0,\ \mathbf{x}(0) \in [\check{\mathbf{x}}(0)],\ \mathbf{y} \in [\check{\mathbf{y}}])$. Первая моделирующая функция $\boldsymbol{\phi}$ задается в табл. 6.9. Соотношения этой функции преобразуются в элементарные ограничения введением вспомогательных переменных во второй моделирующей функции, как показано в табл. 6.10. Результирующий сжимающий оператор показан в табл. 6.11.

Априорные области значений компонент вектора начальных состояний брались в следующем виде:

$$[\check{x}_1(0)] = [-1{,}2, -0{,}8],\quad [\check{x}_2(0)] = [-0{,}2, 0{,}2]. \tag{6.100}$$

Таблица 6.9. Моделирующая функция

Алгоритм $\boldsymbol{\phi}$(вход : $x_1(0), x_2(0)$; выход : $y(1), \ldots, y(\overline{k})$)
1 для $k := 1$ до $\overline{k}$;
2 $x_1(k) := 0{,}1 * x_1(k-1) + x_2(k-1) * \exp(x_1(k-1));$
3 $x_2(k) := x_1(k-1) + 0{,}1 * x_2^2(k-1) + \sin(k);$
4 $y(k) := x_2(k)/x_1(k).$

Таблица 6.10. Моделирующая функция со вспомогательными переменными

Алгоритм $\boldsymbol{\phi}$(вход : $x_1(0), x_2(0)$; выход : $y(1), \ldots, y(\overline{k})$)
1 для $k := 1$ до $\overline{k}$
2 $z_1(k) := \exp(x_1(k-1));$
3 $z_2(k) := x_2(k-1) * z_1(k);$
4 $x_1(k) := 0{,}1 * x_1(k-1) + z_2(k);$
5 $z_3(k) := 0{,}1 * \mathrm{sqr}(x_2(k-1));$
6 $z_4(k) := z_3(k) + \sin(k));$
7 $x_2(k) := x_1(k-1) + z_4(k);$
8 $y(k) := x_2(k)/x_1(k).$

В отсутствие шума (т. е. при $e = 0$) сжимающий оператор, приведенный в табл. 6.11, способен найти истинные значения всех переменных с точностью 8 разрядов за 0,1 секунду на компьютере Pentium 133. Обнаружено, что для получения этого результата *не нужно* выполнять бисекции. Параллелотопы, изображенные в левой части (при $e = 0$) рис. 6.17, получались после каждой итерации сжимающего оператора $\mathcal{C}_{\downarrow\uparrow}$. Подробности решения данного примера и использованной методологии можно найти в [Jaulin *et al.*, 2001a].

Для максимальной ошибки $e = 0{,}5$ (т. е., при наличии шума) размер множества $\widehat{\mathbb{X}}(0)$ уже не нулевой, и, следовательно, даже при идеальном сжимающем операторе бисекции необходимо выполнять (см. правую часть рис. 6.17). При $\varepsilon = 0{,}001$, время вычислений на компьютере Pentium 133 составляет 3 секунды. Априорные интервальные данные $[\tilde{y}(k)]$ показаны в левой части рис. 6.18, здесь же, в правой части рисунка, показаны сжатые интервалы $[\hat{y}(k)]$. Поскольку $x_1^*(5)$ и $x_1^*(13)$ почти равны нулю, и поскольку $y(k) = x_2(k)/x_1(k)$, никакого сжатия интервалов $[\tilde{y}(5)]$ и $[\tilde{y}(13)]$ не получено. Рис. 6.19 представляет сжатые области, полученные для переменных состояния $x_1(k)$ (слева) и $x_2(k)$ (справа) в зависимости от номера k.

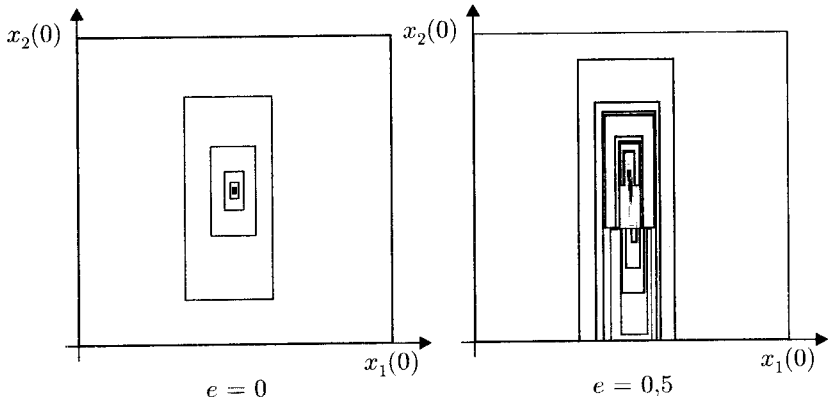


Рис. 6.17. Слева: сжатия, выработанные при отсутствии шума; справа: сжатия и бисекции, выполненные при наличии шума; обе наружные границы соответствуют прямоугольнику $[-1{,}2, -0{,}8] \times [-0{,}2, 0{,}2]$ в пространстве начальных состояний $(x_1(0), x_2(0))$

6.4.4. Гарантированное оценивание на основе метода вперед-назад

Когда множество $\mathcal{X}$ содержит много переменных, как, например, при оценивании по длинной выборке данных, необходимо избегать бисекции параллелотопов в прямом произведении областей всех переменных, чтобы избежать комбинаторного взрыва сложности вычислений. С этой целью можно рассмотреть два подхода.

Первый подход опирается на отбор как можно меньшего множества *исходных* переменных, таких, что величины всех остальных переменных могут быть вычислены единственным образом по знанию значений этих исходных переменных с использованием имеющихся ограничений. Это подход, которого мы придерживались в двух предыдущих параграфах, когда исходными переменными были $x_1(0), \ldots x_{n_x}(0)$. Исходные переменные запоминались в единственном *исходном векторе* ($\mathbf{x}(0)$ в параграфах 6.4.2 и 6.4.3), и поиск выполнялся в *исходном пространстве*, которому этот исходный вектор принадлежит, с использованием техники обращения множеств.

Таблица 6.11. Сжимающий оператор на основе метода вперед-назад для множества $\widehat{\mathbb{X}}(0)$ векторов начального состояния

Алгоритм $\mathcal{C}_{\widehat{\mathbb{X}}(0)}$(вход : $[\tilde{y}_1], \ldots, [\tilde{y}(\overline{k})]$; вход/выход : $[x_1(0)], [x_2(0)]$)

```
1  для k := 1 до k̄
2        [x1(k)] := [−∞, ∞]; [x2(k)] := [−∞, ∞];
3        [z1(k)] := [−∞, ∞]; [z2(k)] := [−∞, ∞];
4        [z3(k)] := [−∞, ∞]; [z4(k)] := [∞, ∞];
5  выполнять
6     для k := 1 до k̄                                   // вперед
7        [z1(k)] := [z1(k)] ∩ exp([x1(k − 1)]);
8        [z2(k)] := [z2(k)] ∩ ([x2(k − 1)] ∗ [z1(k)]);
9        [x1(k)] := [x1(k)] ∩ (0,1 ∗ [x1(k − 1)] + [z1(k)]);
10       [z3(k)] := [z3(k)] ∩ (0,1 ∗ sqr([x2(k − 1)]));
11       [z4(k)] := [z4(k)] ∩ ([z3(k)] + sin(k));
12       [x2(k)] := [x2(k)] ∩ ([x1(k − 1)] + [z4(k)]);
13       [y(k)] := [y(k)] ∩ ([x2(k)]/[x1(k)]);
14    для k := k̄ до 1                                   // обратно
15       [x2(k)] := [x2(k)] ∩ ([y(k)] ∗ [x1(k)]);
16       [x1(k)] := [x1(k)] ∩ ([x2(k)]/[y(k)]);
17       [x1(k − 1)] := [x1(k − 1)] ∩ ([x2(k)] − [z4(k)]);
18       [z4(k)] := [z4(k)] ∩ ([x2(k)] − [x1(k − 1)]);
19       [z3(k)] := [z3(k)] ∩ ([z4(k)] − sin(k));
20       [x2(k − 1)] := [x2(k − 1)] ∩ (0,1 ∗ sqr^{−1}([z3(k)]));
21       [x1(k − 1)] := [x1(k − 1)] ∩ (10 ∗ ([x1(k)] − [z2(k)]));
22       [z2(k)] := [z2(k)] ∩ ([x1(k)] − 0,1 ∗ [x1(k − 1)]);
23       [x2(k − 1)] := [x2(k − 1)] ∩ ([z2(k)]/[z1(k)]);
24       [z1(k)] := [z1(k)] ∩ ([z2(k)]/[x2(k − 1)]);
25       [x1(k − 1)] := [x1(k − 1)] ∩ log([z1(k)]);
26 пока сжатие существенно.
```

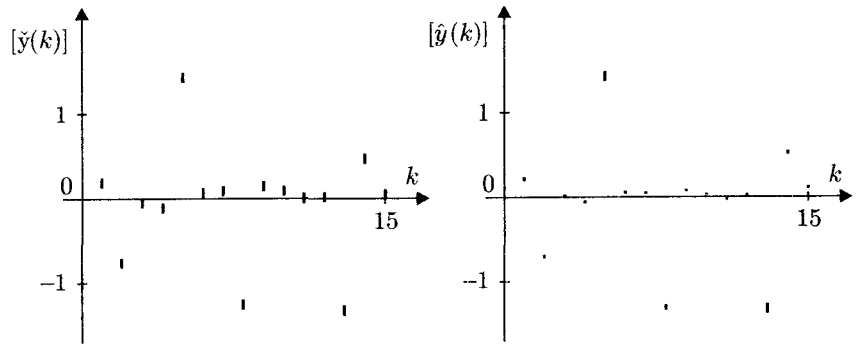


Рис. 6.18. Шум присутствует, $e = 0{,}5$; интервалы неопределенности замеров $\check{y}(k)$ (слева) и сжатые интервалы неопределенности $\hat{y}(k)$ (справа), содержащие истинные значения $y^*(k)$, при учете ограничений

Второй -- *кластерный подход* [Dechter и Dechter, 1987; Dechter и Pearl, 1989; Meiri *et al.*, 1990; Gyssens *et al.*, 1994; Seybold *et al.*, 1998], который будет рассмотрен теперь, разбивает множество $\mathcal{X}$ на группы переменных, чтобы сформировать векторы, когда это возможно. Ограничения на множество $\mathbb{M}$ далее преобразовывались в ограничения на эти векторы. Переменные из множества $\mathcal{X}$ должны быть сгруппированы таким образом, что граф ограничений является деревом (т. е. не содержит циклов). Проиллюстрируем сперва принцип этого подхода на простом примере.

Пример 6.13. *Рассмотрим множество переменных* $\mathcal{X} = \{x_1, ..., x_7\}$ *при ограничениях*

$$\begin{aligned} x_1 \exp(x_2) + x_3 \leqslant 2, \qquad & x_1 x_3 = 5, \\ x_2 - x_3 = 0, \qquad & x_2 - x_3\sqrt{x_4} = 7, \\ x_2 + x_3 x_5 \exp(x_2) = 1, \qquad & x_5 + \sin(x_6 x_7) = 0. \end{aligned} \tag{6.101}$$

Граф ограничений отображен на рис. 6.20, *а. Связь между каждыми двумя переменными* x_i *и* x_j *означает, что есть ограничение, включающее обе эти переменные. Этот граф не является деревом, потому что содержит циклы. Некоторый (не всегда эффективный) эвристический подход к группировке переменных состоит в том, чтобы преобразовать граф ограничений к виду дерева следующим образом.*

1. *Отобрать все циклы длины* ℓ *(сперва* ℓ *берется равным трем, но если циклы такой длины не найдены, то берем* $\ell = 4, 5, \dots$ *). В этом примере имеется четыре цикла длиной три, а именно* (x_1, x_2, x_3), (x_2, x_3, x_4), (x_2, x_3, x_5) *и* (x_5, x_6, x_7).

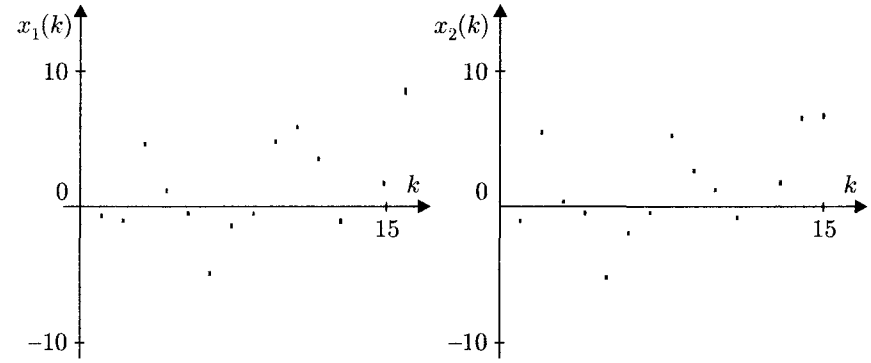


Рис. 6.19. Шум присутствует, $e = 0{,}5$; сжатые интервалы неопределенности переменных $x_1(k)$ (слева) и $x_2(k)$ (справа) в зависимости от номера k

2. *Сгруппировать переменные, ассоциированные с той дугой, которая принадлежит максимальному числу отобранных циклов, в единый вектор. Например, здесь такой дугой является связь* (x_2, x_3), *которая имеется в первых трех циклах, и переменные* x_2 *и* x_3 *группируются в вектор* $(x_2, x_3)^{\mathrm{T}}$. *Граф на* рис. 6.20, *б показывает такую группировку. Такие два шага рассматриваемого подхода повторяются до тех пор, пока граф не станет деревом, показанным на* рис. 6.20, *в.* ■

После группировки переменных так, что граф ограничений принимает вид дерева, мы получаем конечный набор $\mathcal{V} = \{\mathbf{v}_1, \dots, \mathbf{v}_n\}$ векторов переменных $\mathbf{v}_i$ размерностей $d_i \in \mathbb{N}$, и априорные области $\check{\mathbb{V}}_i \subset \mathbb{R}^{d_i}$, $i \in \{1, \dots, n\}$. Некоторые векторные переменные, скажем, $\mathbf{v}_i$ и $\mathbf{v}_j$, связаны *бинарными ограничениями* $\mathbb{C}_{i,j}$, которые могут рассматриваться как подмножества в $\mathbb{R}^{d_i} \times \mathbb{R}^{d_j}$. Могут также существовать и *одинарные ограничения* $\mathbb{C}_i$, выражающие связи между компонентами вектора $\mathbf{v}_i$. Априорная область $\check{\mathbb{V}}_i$ для вектора $\mathbf{v}_i$ получается тогда пересечением множества $\mathbb{C}_i$, которое может рассматриваться как подмножество $\mathbb{R}^{d_i}$, с прямым произведением априорных областей компонент вектора $\mathbf{v}_i$.

Пример 6.14. *Рассмотрим снова задачу* Примера 6.13. *Множество векторных переменных* $\mathcal{V}$ *имеет вид* $\{v_1, \mathbf{v}_2, v_3, v_4, \mathbf{v}_5\}$, *где* $v_1 = x_1$, $\mathbf{v}_2 = (x_2, x_3)^{\mathrm{T}}$, $v_3 = x_4$, $v_4 = x_5$, $\mathbf{v}_5 = (x_6, x_7)^{\mathrm{T}}$. *Есть только единственное*

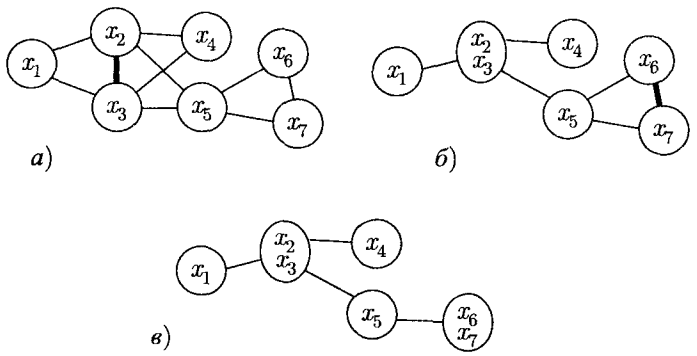


Рис. 6.20. Группировка переменных в кластеры для преобразования графа с циклами к виду дерева

одинарное ограничение, задаваемое как

$$\mathbb{C}_2 = \{\mathbf{v}_2 \in \mathbb{R}^2,\ x_2 - x_3 = 0\}. \tag{6.102}$$

Если априорные области для x_2 и x_3 обозначить как $\check{\mathbb{X}}_2$ и $\check{\mathbb{X}}_3$, соответственно, то априорная область для вектора $\mathbf{v}_2$ записывается

$$\check{\mathbb{V}}_2 = \{\mathbf{v}_2 \in \check{\mathbb{X}}_2 \times \check{\mathbb{X}}_3 \mid x_2 - x_3 = 0\}. \tag{6.103}$$

Бинарные ограничения между этими векторными переменными представляются

$$\begin{aligned}
\mathbb{C}_{1,2} &= \{(v_1, \mathbf{v}_2) \in \mathbb{R}^3 \mid x_1 \exp(x_2) + x_3 \leqslant 2 \text{ и } x_1 x_3 = 5\},\\
\mathbb{C}_{2,3} &= \{(\mathbf{v}_2, v_3) \in \mathbb{R}^3 \mid x_2 - x_3\sqrt{x_4} = 7\},\\
\mathbb{C}_{2,4} &= \{(\mathbf{v}_2, v_4) \in \mathbb{R}^3 \mid x_2 + x_3 x_5 \exp(x_2) = 1\},\\
\mathbb{C}_{4,5} &= \{(v_4, \mathbf{v}_5) \in \mathbb{R}^3 \mid x_5 + \sin(x_6 x_7) = 0\}. \quad \blacksquare
\end{aligned} \tag{6.104}$$

В свете оценивания состояния, одинарное ограничение можно обойти, и ограничение $\mathbb{C}_{i,j}$ в общем случае может быть сведено к виду:

$$\mathbb{C}_{i,j} = \{(\tilde{\mathbf{v}}_i, \tilde{\mathbf{v}}_j) \in \mathbb{R}^{d_i} \times \mathbb{R}^{d_j} \mid \tilde{\mathbf{v}}_j = \mathbf{f}_j(\tilde{\mathbf{v}}_i)\}. \tag{6.105}$$

Для упрощения обозначений мы будем ссылаться на это ограничение как $\mathbb{C}_{i,j} : \mathbf{v}_j = \mathbf{f}_j(\mathbf{v}_i)$.

Значение, принимаемое вектором $\mathbf{v}_i$, назовем *совместным* (с ограничениями), если

$$\begin{aligned}
&\exists(\tilde{\mathbf{v}}_1 \in \check{\mathbb{V}}_1, \ldots, \tilde{\mathbf{v}}_{i-1} \in \check{\mathbb{V}}_{i-1}, \tilde{\mathbf{v}}_{i+1} \in \check{\mathbb{V}}_{i+1}, \ldots, \tilde{\mathbf{v}}_n \in \check{\mathbb{V}}_n)\\
&\mid (\tilde{\mathbf{v}}_1, \ldots, \tilde{\mathbf{v}}_{i-1}, \mathbf{v}_i, \tilde{\mathbf{v}}_{i+1}, \ldots, \tilde{\mathbf{v}}_n) \quad \text{удовлетворяет всем ограничениям.}
\end{aligned} \tag{6.106}$$

Множество всех векторов $\mathbf{v}_i$ назовем *допустимой областью* переменной $\mathbf{v}_i$ и будем обозначать эту область как $\hat{\mathbb{V}}_i$. Рассмотрим две переменные $\mathbf{v}_i$ и $\mathbf{v}_j$, связанные ограничением $\mathbb{C}_{i,j}$. Определим *локальный оператор сжатия* области $\mathbb{V}_i$ относительно переменной $\mathbf{v}_j$ как

$$\rho_j(\mathbb{V}_i) = \{\tilde{\mathbf{v}}_i \in \mathbb{V}_i \mid \exists \tilde{\mathbf{v}}_j \in \mathbb{V}_j,\ (\tilde{\mathbf{v}}_i, \tilde{\mathbf{v}}_j) \in \mathbb{C}_{i,j}\}. \tag{6.107}$$

Рис. 6.21 поясняет это определение. Термин *локальный* показывает, что учитывается только одно ограничение. Заметим, что $\hat{\mathbb{V}}_i \subset \rho_j(\check{\mathbb{V}}_i) \subset \check{\mathbb{V}}_i$. Если имеем бинарное ограничение $\mathbb{C}_{i,j} : \mathbf{v}_j = \mathbf{f}_j(\mathbf{v}_i)$, то

$$\rho_j(\mathbb{V}_i) = \mathbb{V}_i \cap \mathbf{f}_j^{-1}(\mathbb{V}_j), \qquad \rho_i(\mathbb{V}_j) = \mathbb{V}_j \cap \mathbf{f}_j(\mathbb{V}_i). \tag{6.108}$$

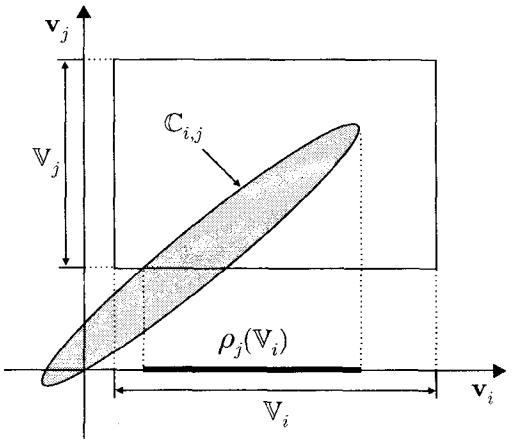


Рис. 6.21. Локальный оператор сжатия

Области совместности $\hat{\mathbb{V}}_i$ могут быть получены расширением вектора переменных [Jaulin *et al.*, 2001б] в *алгоритме метода вперед-назад*, предложенного в [Benhamou *et al.*, 1999б] для вещественных переменных из $\mathbb{R}$.

Сначала один узел дерева выбирается в качестве корня. Далее, дерево просматривается от его листьев до его корня. Это — *прямой этап метода вперед-назад*. Во время этого просмотра, области узлов сжимаются относительно порожденных ими узлов (отростков). Как результат, область, связанная с этим корнем, является допустимой областью. Наконец, дерево просматривается от его корня к листьям. Это — *обратный этап метода вперед-назад*. При этом втором просмотре области узлов сжимаются относительно порождающих их узлов (родителей). По завершению этих операций каждая область равна соответствующей допустимой области (т. е. для любого i $\mathbb{V}_i = \widehat{\mathbb{V}}_i$). Доказательство этого факта можно найти в [Jaulin *et al.*, 2001б]. Заметим, что ту же идею для дискретных областей можно найти в [Freuder, 1978], [Montanari и Rossi, 1991] и [Collin *et al.*, 1991].

Пример 6.15. *Рассмотрим дерево на* рис. 6.22. *Его корнем является вектор* $\mathbf{v}_1$, *а листья* — $\mathbf{v}_4$, $\mathbf{v}_5$, $\mathbf{v}_6$,$\mathbf{v}_7$, $\mathbf{v}_8$. *Прямой этап метода вперед-назад состоит в выполнении следующего набора операций* (*начиная с любого* i, $\mathbb{V}_i := \check{\mathbb{V}}_i$):

$$\begin{aligned} \mathbb{V}_2 &:= \mathbb{V}_2 \cap \rho_4(\mathbb{V}_2) \cap \rho_5(\mathbb{V}_2) \cap \rho_6(\mathbb{V}_2), \\ \mathbb{V}_3 &:= \mathbb{V}_3 \cap \rho_7(\mathbb{V}_3) \cap \rho_8(\mathbb{V}_3), \\ \mathbb{V}_1 &:= \mathbb{V}_1 \cap \rho_2(\mathbb{V}_1) \cap \rho_3(\mathbb{V}_1). \end{aligned} \tag{6.109}$$

На этом этапе $\mathbb{V}_1 = \widehat{\mathbb{V}}_1$. *Обратный этап метода вперед-назад состоит в вычислении*

$$\begin{aligned} \mathbb{V}_2 &:= \mathbb{V}_2 \cap \rho_1(\mathbb{V}_2), \\ \mathbb{V}_4 &:= \mathbb{V}_4 \cap \rho_2(\mathbb{V}_4), \\ \mathbb{V}_5 &:= \mathbb{V}_5 \cap \rho_2(\mathbb{V}_5), \\ \mathbb{V}_6 &:= \mathbb{V}_6 \cap \rho_2(\mathbb{V}_6), \\ \mathbb{V}_3 &:= \mathbb{V}_3 \cap \rho_1(\mathbb{V}_3), \\ \mathbb{V}_7 &:= \mathbb{V}_7 \cap \rho_3(\mathbb{V}_7), \\ \mathbb{V}_8 &:= \mathbb{V}_8 \cap \rho_3(\mathbb{V}_8). \end{aligned} \tag{6.110}$$

Теперь, для всех i $\mathbb{V}_i = \widehat{\mathbb{V}}_i$. ■

Применим эту методологию к оценке состояния дискретной по времени системы

$$\begin{cases} \mathbf{x}(k) = \mathbf{f}(\mathbf{x}(k-1)), \\ \mathbf{y}(k) = \mathbf{g}(\mathbf{x}(k)), \end{cases} \qquad k = 1, \ldots, \overline{k}, \tag{6.111}$$

где $\mathbf{x}(k) \in \mathbb{R}_n$ — вектор состояния, а $\mathbf{y}(k) \in \mathbb{R}^m$ — вектор наблюдения. Система (6.111) является специальным случаем системы (6.90), и с ней можно

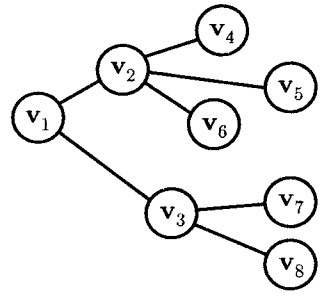


Рис. 6.22. Дерево графа ограничений Примера 6.15

работать в общем виде по тем же строкам. Функции $\mathbf{f}$ и $\mathbf{g}$ могут быть нелинейными. Рассмотрим два типа операторов оценивания, а именно *оператор последовательного оценивания*, когда оценка состояния в момент k может основываться только на замерах вектора наблюдения до момента k включительно, и *оператор пост-фактум оценивания*, когда для выполнения оценивания доступен вектор наблюдения на всем временном интервале наблюдения (вся выборка). Оценивание в реальном времени может использовать только операторы последовательного оценивания, а оценивание после проведения наблюдения (не в реальном времени) позволяет использовать операторы пост-фактум оценивания и использовать преимущество доступности всей выборки для оценивания состояния системы в любой момент времени.

Оператор последовательного оценивания. В момент с номером k имеем следующее множество всех векторных переменных, подлежащих рассмотрению:

$$\mathbf{V}_k = \{\mathbf{x}(0), \ldots, \mathbf{x}(k), \mathbf{y}(1), \ldots, \mathbf{y}(k)\}. \tag{6.112}$$

Соответствующее множество априорных областей записывается как

$$\mathbf{D}_k = \{\check{\mathbb{X}}(0), \ldots, \check{\mathbb{X}}(k), \check{\mathbb{Y}}(1), \ldots, \check{\mathbb{Y}}(k)\}. \tag{6.113}$$

Для простоты предположим, что нет никакой специальной априорной информации о значениях векторов $\mathbf{x}(1), \ldots, \mathbf{x}(k)$. Все множества $\check{\mathbb{X}}(1), \ldots, \check{\mathbb{X}}(k)$ полагаются совпадающими с $\mathbb{R}^n$. Накопленные замеры вектора наблюдения $\check{\mathbf{y}}(i)$, $0 \leqslant i \leqslant k$ используются для формирования априорных допустимых множеств (множеств неопределенности замеров) $\check{\mathbb{Y}}(i)$ на основе предположения о их достоверности. Допустимое множество связанных

с замерами описывается

$$\mathbf{C}_k = \{\mathbf{x}(\ell) = \mathbf{f}(\mathbf{x}(\ell-1));\ \ell = 1, \ldots, k\} \cup \{\mathbf{y}(\ell) = \mathbf{g}(\mathbf{x}(\ell));\ \ell = 1, \ldots, k\}. \quad (6.114)$$

Соответствующий граф ограничений представлен на рис. 6.23. Несмотря на присутствие стрелок, граф — неориентированный, и стрелки должны восприниматься только как направления, по которым действуют функции, участвующие в ограничениях.

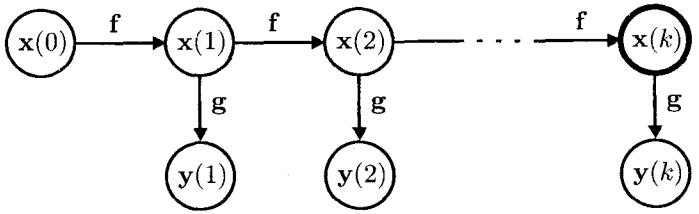


Рис. 6.23. Граф, связанный с системой (6.111); жирный кружок показывает выбранный корень $\mathbf{x}(k)$, но можно выбрать и любой другой корень

Прямой этап метода вперед-назад, примененный к графу рис. 6.23, рассчитывает область допустимости $\widehat{\mathbb{X}}(k)$ для $\mathbf{x}(k)$ с учетом того, что $\mathbf{x}(k)$ выбран в качестве корня дерева. Это приводит к оператору (CSE) последовательного оценивания состояния, представленному в табл. 6.12.

Таблица 6.12. Алгоритм последовательного оценивания состояния, основанный на применении прямого этапа метода вперед-назад

Алгоритм CSE(вход : $\check{\mathbb{X}}(0)$, $\check{\mathbb{Y}}(1)$, ..., $\check{\mathbb{Y}}(k)$; выход : $\widehat{\mathbb{X}}(k)$)
1 $\mathbb{X}(0) := \check{\mathbb{X}}(0)$;
2 для $\ell := 1$ до k
3 $\quad \mathbb{X}(\ell) := \mathbf{f}(\mathbb{X}(\ell-1)) \cap \mathbf{g}^{-1}(\check{\mathbb{Y}}(\ell))$;
4 $\widehat{\mathbb{X}}(k) := \mathbb{X}(k)$.

В момент $k+1$ измерение $\check{\mathbf{y}}(k+1)$ становится доступным, и алгоритм табл. 6.12 может снова использоваться для расчета множества $\widehat{\mathbb{X}}(k+1)$. Очевидно, что скорее всего надо применять рекурсивный оператор (RCSE) последовательной оценки состояния, приведенный в табл. 6.13 [Kieffer *et al.*, 1998; 1999], чтобы воспользоваться преимуществом, даваемым знанием результатов оценивания, уже полученных на момент k. Этот алгоритм выполняет оптимальное сжатие области значений вектора $\mathbf{x}(k)$ после измерения в момент k.

Таблица 6.13. Алгоритм рекурсивного последовательного оценивания состояния

Алгоритм RCSE(вход : $\check{\mathbb{X}}(0)$; выход : $\widehat{\mathbb{X}}(1)$, ..., $\widehat{\mathbb{X}}(\overline{k})$)
1 $\widehat{\mathbb{X}}(0) := \check{\mathbb{X}}(0)$; $k = 1$;
2 для $k := 1$ до $\overline{k}$
3 $\quad$ ожидание до момента прихода множества $\check{\mathbb{Y}}(k)$;
4 $\quad \widehat{\mathbb{X}}(k) := \mathbf{f}(\widehat{\mathbb{X}}(k-1)) \cap \mathbf{g}^{-1}(\check{\mathbb{Y}}(k))$.

Рис. 6.24 иллюстрирует принцип выполнения одной итерации алгоритма RCSE. В момент с номером $k-1$ известно, что состояние принадлежит множеству $\widehat{\mathbb{X}}(k-1)$. *Прогнозируемое множество* (множество прогноза) $\mathbf{f}(\widehat{\mathbb{X}}(k-1))$ при этом содержит все возможные значения состояния на момент с номером k. Когда наступает момент k и приходит новое измерение, появляющееся множество $\mathbf{g}^{-1}(\check{\mathbb{Y}}(k))$ содержит все значения вектора состояний, которые привели к измерению, принадлежащему множеству $\check{\mathbb{Y}}(k)$. Таким образом, в момент k состояние принадлежит уточненному множеству $\mathbf{f}(\widehat{\mathbb{X}}(k-1)) \cap \mathbf{g}^{-1}(\check{\mathbb{Y}}(k))$. При этом не требуется, чтобы $\widehat{\mathbb{X}}(k-1)$ или $\mathbf{f}(\widehat{\mathbb{X}}(k-1))$ состояли из одной связной компоненты, как мы увидим в параграфе 8.4.6 (с. 327), в задаче слежения за движущимся роботом.

Оператор пост-фактум оценивания. Предположим теперь, что $\overline{k}$ измерений вектора $\check{\mathbf{y}}(k)$, $k \in \{1, \ldots, \overline{k}\}$, и связанные с ними априорные допустимые области $\check{\mathbb{Y}}(k)$ известны на выходе системы. Применение метода вперед-назад делает возможным получить апостериорные допустимые области, которые совместны со всей информацией, накопленной за все время наблюдения. Таблица 6.14 описывает результирующий алгоритм (NCSE) пост-фактум оценивания, когда в качестве корня взят узел $\mathbf{x}(\overline{k})$.

ЗАМЕЧАНИЕ 6.6. Реализация рассмотренных операторов оценивания состояния требует некоторого способа представления множеств и выполнения локального сжимающего оператора ρ_i. Область $\mathbb{V}$ представляется некоторым покрытием, заключающим эту область. В рассматриваемых алгоритмах оценивания локальный сжимающий оператор ρ_i соответствует либо образу $\mathbf{f}(\mathbb{V})$ множества $\mathbb{V}$ для векторной функции $\mathbf{f}$, либо прообразу $\mathbf{f}^{-1}(\mathbb{W})$ множества $\mathbb{W}$ для векторной функции $\mathbf{f}$. Гарантированное включение для образа $\mathbf{f}(\mathbb{V})$ может быть рассчитано по алгоритму IMAGESP (см. Глава 3, табл. 3.3, с. 87), а гарантированное включение для об-

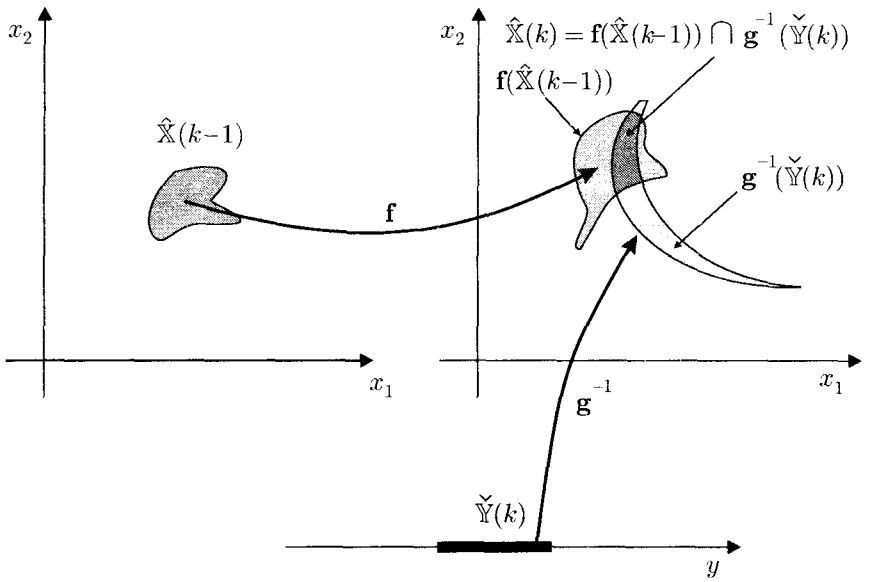


Рис. 6.24. Принцип выполнения одной итерации рекурсивного алгоритма последовательного оценивания состояния (RCSE)

Таблица 6.14. Алгоритм пост-фактум оценивания, основанный на методе вперед-назад

Алгоритм NCSE(вход : $\check{\mathbb{X}}(0)$, $\check{\mathbb{Y}}(1)$, ..., $\check{\mathbb{Y}}(\overline{k})$; выход : $\hat{\mathbb{X}}(0)$, ..., $\hat{\mathbb{X}}(\overline{k})$)	
1 $\hat{\mathbb{X}}(0) := \check{\mathbb{X}}(0)$;	
2 для $k := 1$ до $\overline{k}$	// вперед
3 $\quad \hat{\mathbb{X}}(\overline{k}) := \mathbf{f}(\hat{\mathbb{X}}(k-1)) \cap \mathbf{g}^{-1}(\check{\mathbb{Y}}(k))$;	
4 для $k := \overline{k}$ до 1	// назад
5 $\quad \hat{\mathbb{Y}}(k) := \mathbf{g}(\hat{\mathbb{X}}(k))$; $\hat{\mathbb{X}}(k-1) := \hat{\mathbb{X}}(k-1) \cap \mathbf{f}^{-1}(\hat{\mathbb{X}}(k))$.	

раза $\mathbf{f}^{-1}(\mathbb{W})$ может быть рассчитано алгоритмом SIVIA (см. Глава 3, табл. 3.1, с. 82). ■

Иллюстрация. Рассмотрим нелинейную систему

$$\begin{cases} \begin{pmatrix} x_1(k) \\ x_2(k) \end{pmatrix} = 3 \begin{pmatrix} \sin(x_1(k-1) + x_2(k-1)) \\ \cos(x_1(k-1) + x_2(k-1)) \end{pmatrix}, \\ y(k) = |x_1(k)|, \end{cases} \tag{6.115}$$

при $k \in \{1, \ldots, 10\}$. Для $\mathbf{x}^*(0) = (0,0)^{\mathrm{T}}$ истинные значения состояния $\mathbf{x}^*(k)$ и наблюдений $y^*(k)$, $k \in \{1, \ldots, 10\}$, были выработаны путем моделирования этой системы. Выходные данные $\check{y}(k)$ (множества неопределенности замеров) далее получались добавлением шума ограниченной величины к истинным значениям наблюдений $y^*(k)$

$$\check{y}(k) = y^*(k) + \xi(k), \tag{6.116}$$

где $\xi(k)$ — реализация случайной переменной, равномерно распределенной в интервале $[-0{,}1, 0{,}1]$. Априорная область k-го наблюдения (множество неопределенности замера) при этом имеет вид:

$$\check{\mathbb{Y}}(k) = \check{y}(k) + [-0{,}1, 0{,}1], \tag{6.117}$$

поэтому

$$y^*(k) \in \check{\mathbb{Y}}(k), \; k = 1, \ldots, 10. \tag{6.118}$$

Априорная область для вектора $\mathbf{x}(k)$ берется как $\check{\mathbb{X}}(k) = \mathbb{R}^2$. Апостериорные области для $\mathbf{x}(k)$, полученные оператором последовательного оценивания и оператором пост-фактум оценивания, представлены на рис. 6.25. Общее время вычислений для обоих операторов менее минуты на компьютере Pentium 133. Рамки на всех малых рисунках соответствуют квадрату $[-4, 4] \times [-4, 4]$. Первый малый рисунок весь отмечен серым цветом, это отражает тот очевидный факт, что оператор последовательного оценивания не может дать какую-либо уточненную информацию о начальном векторе $\mathbf{x}(0)$ ($\check{\mathbb{X}}(0) = \mathbb{R}^2$), в отличие от того, что дает оператор пост-фактум оценивания. Последние два малых рисунка для $k = 10$ — идентичны, потому что оба оператора обработали к этому моменту одинаковую информацию.

6.5. Выводы

Задачи оценивания включают неопределенные переменные, связанные через ограничения. Эти ограничения используются для уменьшения неопределенности по указанными переменным, и, таким образом, можно

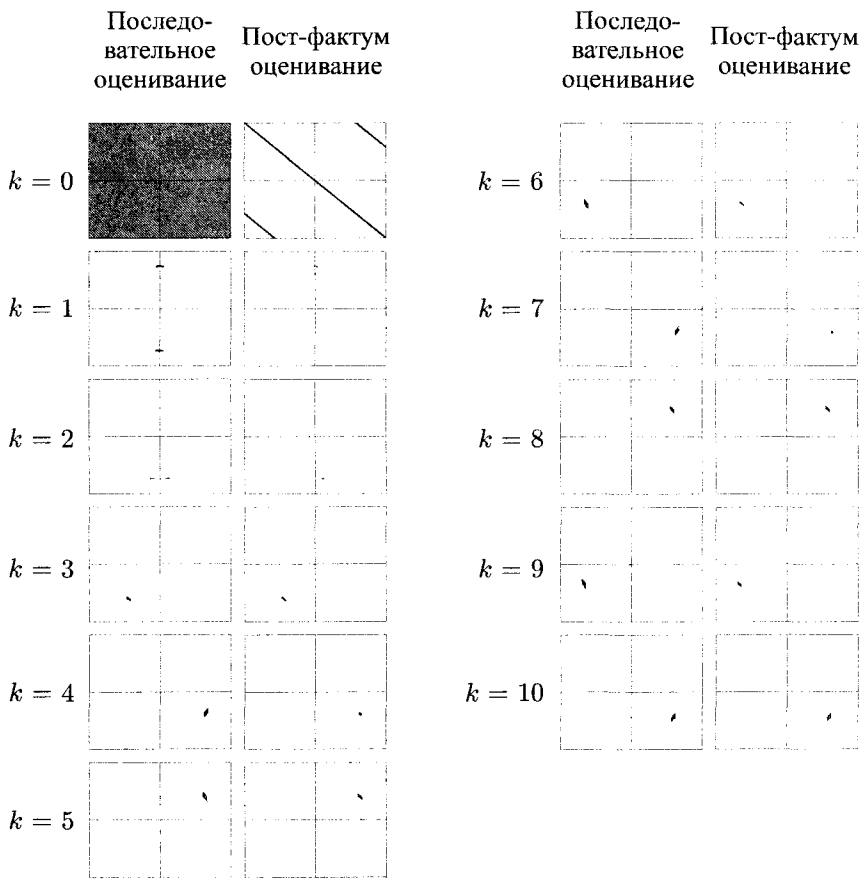


Рис. 6.25. Оценки, получаемые операторами последовательного оценивания и пост-фактум оценивания

воспользоваться тем преимуществом, которое этот аппарат дает при решении задач выполнения ограничений. В данной главе показано, как этот формализм и алгоритмы задач выполнения ограничений могут быть адаптированы к оцениванию. Предложена унифицированная схема для решения большого класса задач оценивания, и в этой связи показана эффективность интервальных методов решения из Главы 5.

Теперь становится возможным получать гарантированные оценки параметров и переменных состояния даже тогда, когда эти параметры не являются идентифицируемыми, а переменные состояния ненаблюдаемы. Очень просто учитываются нелинейные ограничения или ограничения, переменные по времени. Этот подход может быть расширен на случай, когда множество $\mathcal{X}$ содержит бесконечное число переменных, и некоторые из них могут быть целыми числами или булевскими переменными.

Метод наименьших квадратов оказывается неэффективным вследствие того факта, что штрафная функция, которую необходимо минимизировать, является суммой членов, зависящих от одних и тех же параметров, поэтому неизбежным становится многократное появление этих параметров, и оно приводит к весьма плохим функциям включения для штрафной функции. Это затрудняет исключение ненужных частей из области поиска. Гарантированное оценивание параметров и состояний оказывается более простым в реализации, и распространение интервальных ограничений приводит к тому, что удается решать задачи высокой размерности.

Глава 7

Робастное управление

7.1. Введение

Цель настоящей главы — показать применение техники интервальных вычислений, рассмотренной в Разделе II, для решения некоторых задач робастного управления. Здесь робастность понимается по отношению к неопределенности модели процесса, которым надо управлять. Рассмотрены задачи, простирающиеся от анализа свойств существующей неопределенной системы до проектирования регулятора управления процессом с неопределенностью.

Будут рассматриваться только системы, описываемые обыкновенными линейными дифференциальными уравнениями с постоянными по времени коэффициентами. Основная идея состоит в том, что проверка устойчивости есть поиск решения системы нелинейных неравенств. Управление системой, описываемой нелинейными дифференциальными или разностными уравнениями, является более сложным. Первый подход к управлению нелинейной системой дифференциальных уравнений на основе интервальных методов был предложен в [Jaulin и Walter, 1997].

Робастное управление является областью активных исследований уже много лет [Horowitz, 1963; Dorato *et al.*, 1993; Kwakernaak, 1993; Barmish, 1994; Boyd *et al.*, 1994; Francis и Khargonekar, 1995], и не представляется возможным описать детально состояние всех исследований в одной главе. Мы сосредоточим внимание на задачах робастной устойчивости и робастного управления для неопределенных систем, описываемых параметрическими моделями, в которых неизвестные параметры находятся в пределах известных ограничений. С этим типом неопределенности трудно работать методами таких подходов, как H_∞-анализ или μ-анализ, в то время как интервальный анализ оказывается особенно применимым, что показывается в приводимых примерах.

Мы максимально постарались сделать эту главу понятной читателю, который не является специалистом по управлению. Как результат, читатели, специалисты по управлению, найдут изложение некоторых вопросов чересчур упрощенным и могут возразить, что не все важные приемы описаны.

Мы чувствуем свою вину, но надеемся, что такие читатели все-таки найдут здесь интересный материал, который они легко распространят и адаптируют на другие задачи.

Глава организована следующим образом. В параграфе 7.2 напоминаются основные сведения из устойчивости линейных детерминированных моделей, а также понятие степени устойчивости. Неопределенность в параметрах модели вводится в параграфе 7.3. Рассматриваются различные типы параметрической зависимости и напоминаются некоторые результаты из имеющейся литературы, включая выдающуюся теорему Харитонова и теорему о реберном полиноме. Указана некоторая ограниченность этих результатов.

В параграфе 7.4 показано, как аппарат интервального анализа может быть применен для анализа устойчивости существующих неопределенных систем, когда приемы из параграфа 7.3 не могут быть использованы. Параграф 7.5 посвящен разработке регуляторов для процессов с неопределенностями.

7.2. Устойчивость детерминированных линейных систем

Предположим пока, что рассматриваемая система Σ не включает неопределенностей. При управлении системой принято различать *входные переменные*, которые используются для воздействия на систему, и *выходные переменные*, являющиеся сигналами наблюдения за реакцией системы Σ. Будем обозначать вектор входных переменных в момент t как $\mathbf{u}(t)$, а вектор выходных переменных — как $\mathbf{y}(t)$. Мы будем рассматривать только непрерывные по времени системы, но большинство рассматриваемых понятий может быть перенесено на случай дискретного времени. Когда система Σ линейна, стационарна и первоначально находится в состоянии покоя ($\mathbf{u} \equiv \mathbf{0}$ и $\mathbf{y} \equiv \mathbf{0}$ до момента времени $t = 0$), преобразование Лапласа выхода системы связано со входом как

$$\mathbf{y}(s) = \mathbf{G}(s)\mathbf{u}(s), \tag{7.1}$$

где s — переменная Лапласа, а $\mathbf{G}(s)$ — передаточная матрица системы Σ. Важным альтернативным представлением линейной стационарной системы Σ является представление в *пространстве состояний*

$$\Sigma : \begin{cases} \dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t), \\ \mathbf{y}(t) = \mathbf{x}(t), \end{cases} \tag{7.2}$$

где $\mathbf{x}$ — вектор состояния; $\dot{\mathbf{x}}$ — первая производная вектора состояния по времени; $\mathbf{A}$, $\mathbf{B}$ и $\mathbf{C}$ — матрица системы, матрица управления и наблюдения, соответственно. Состояние $\mathbf{x}(t)$ может рассматриваться как минимум

информации, которую необходимо знать в момент t, чтобы быть в состоянии рассчитать эволюцию системы в ответ на известные будущие входные воздействия. Будем предполагать, что размерность n вектора состояний $\mathbf{x}$ конечна, и всегда в этом случае система Σ будет описываться системой обыкновенных линейных дифференциальных уравнений.

Преобразование Лапласа для представления состояния (7.2) при нулевых начальных условиях имеет вид:

$$\begin{cases} s\mathbf{x}(s) = \mathbf{A}\mathbf{x}(s) + \mathbf{B}\mathbf{u}(s), \\ \mathbf{y}(s) = \mathbf{C}\mathbf{x}(s). \end{cases} \tag{7.3}$$

Исключая $\mathbf{x}(s)$ из (7.3), получаем $\mathbf{y}(s) = \mathbf{C}(s\mathbf{I}_n - \mathbf{A})^{-1}\mathbf{B}\mathbf{u}(s)$, где $\mathbf{I}_n$ – единичная матрица размерности $n \times n$, так что передаточная матрица от $\mathbf{u}(s)$ к $\mathbf{y}(s)$ представляется

$$\mathbf{G}(s) = \mathbf{C}(s\mathbf{I}_n - \mathbf{A})^{-1}\mathbf{B}. \tag{7.4}$$

Таким образом, компонентами матрицы $\mathbf{G}(s)$ являются рациональные функции от s. Не будут рассматриваться системы с запаздыванием или с частными производными, которые имеют бесконечномерный вектор состояния.

Говорим, что система Σ *асимптотически устойчива* тогда и только тогда, когда ее состояние $\mathbf{x}(t)$ сходится к $\mathbf{0}$ при t, стремящемуся к бесконечности и входном воздействии $\mathbf{u}(t) = \mathbf{0}$ для всех $t \geqslant 0$. Как следствие, вектор $\mathbf{y}(t)$ также сходится к $\mathbf{0}$. Асимптотическая устойчивость будет единственным типом устойчивости, рассматриваемым в данной книге, и всегда, когда мы будем говорить об устойчивой системе, нужно понимать, что подразумевается асимптотически устойчивая система. И наоборот, когда мы говорим о *неустойчивой* системе, мы имеем в виду, что система – не асимптотически устойчива. По очевидным соображениям безопасного функционирования, устойчивость является жизненным требованием для большинства управляемых систем, и остальная часть настоящего параграфа будет посвящена методам исследования системы Σ на устойчивость.

7.2.1. Характеристический полином

Когда для любого $t \geqslant 0$ входное воздействие $\mathbf{u}(t) = \mathbf{0}$, то фазовый вектор определяется формулой

$$\mathbf{x}(t) = \exp(\mathbf{A}t)\,\mathbf{x}(0). \tag{7.5}$$

Поскольку элементы матрицы $e^{\mathbf{A}t}$ являются линейными комбинациями выражений вида $\exp(\lambda_i t)$, где λ_i – собственные значения матрицы $\mathbf{A}$, то система Σ устойчива тогда и только тогда, когда все собственные значения имеют

строго отрицательные вещественные части, т. е. $\mathrm{Re}(\lambda_i) < 0$, $i = 1, \ldots, n$. Здесь собственные значения матрицы $\mathbf{A}$ являются также корнями ее *характеристического полинома*, определенного как

$$P(s) = \det(s\mathbf{I}_n - \mathbf{A}). \tag{7.6}$$

Этот полином может быть записан следующим образом:

$$P(s) = a_n s^n + a_{n-1}s^{n-1} + \ldots + a_1 s + a_0. \tag{7.7}$$

Кроме того, $P(s)$ соответствует знаменателям членов матрицы системы $\mathbf{G}(s)$, как это видно из (7.4), если нет сокращений корней между их числителями и знаменателями.

По аналогии, говорим, что полином $P(s)$ *устойчив* тогда и только тогда, когда все его корни имеют строго отрицательные вещественные части, и что корни системы Σ являются корнями ее характеристического полинома. Таким образом, система Σ устойчива тогда и только тогда, когда устойчив полином $P(s)$, именно, если все его корни находятся в левой части $\mathbb{C}^-$ комплексной плоскости. Необходимым условием устойчивости полинома $P(s)$ является то, что все его коэффициенты имеют одинаковый знак. Если коэффициент a_n при старшем члене в (7.7) равен 1, то это условие сводится к тому, что $a_i > 0$, $i = 0, \ldots, n-1$.

7.2.2. Критерий Рауса

Существуют эффективные методы проверки устойчивости заданного полинома. Критерий Рауса основан на построении *таблицы Рауса* (табл. 7.1). Как показано, коэффициенты a_i полинома запоминаются в первых двух строках таблицы. Эти строки окаймлены справа нулями. Остальные $n-1$ строк таблицы Рауса вычисляются следующим образом

$$\begin{array}{lll} b_1 = \dfrac{a_{n-1}a_{n-2} - a_n a_{n-3}}{a_{n-1}}, & b_2 = \dfrac{a_{n-1}a_{n-4} - a_n a_{n-5}}{a_{n-1}}, & \cdots \\ c_1 = \dfrac{b_1 a_{n-3} - a_{n-1}b_2}{b_1}, & c_2 = \dfrac{b_1 a_{n-5} - a_{n-1}b_3}{b_1}, & \cdots \\ \vdots & \vdots & \ddots \end{array} \tag{7.8}$$

У полинома $P(s)$ число корней с положительными вещественными частями равно числу перемен знака в первом столбце таблицы. Например, если первый столбец содержит последовательность чисел $1, 12, -4, 3, 2, -1$ в порядке их следования, то число смен знака равно трем и полином $P(s)$

Таблица 7.1. Таблица Рауса

a_n	a_{n-2}	a_{n-4}	...	0
a_{n-1}	a_{n-3}	a_{n-5}	...	0
b_1	b_2	b_3	...	0
c_1	c_2	c_3	...	0
...	...	...	...	0
g_1	0			
h_1				

имеет три корня с положительными вещественными частями. Предположим, что коэффициент a_n после нормировки равен единице, и определим *вектор Рауса* как вектор всех элементов первого столбца после удаления первого коэффициента:

$$\mathbf{r} = (a_{n-1}, b_1, \ldots, h_1)^{\mathrm{T}}. \tag{7.9}$$

Полином $P(s)$ устойчив тогда и только тогда, когда $\mathbf{r} > \mathbf{0}$, и неустойчив (т. е. не устойчив асимптотически) тогда и только тогда, когда у $\mathbf{r}$ существует некоторая компонента r_i, такая, что $r_i \leqslant 0$. Для анализа ситуаций, когда некоторые компоненты $\mathbf{r}$ являются нулями, можно обратиться к [Levine, 1996].

7.2.3. Степень устойчивости

Расположение корней полинома $P(s)$ дает больше информации, чем то, что является ли система Σ устойчивой или неустойчивой. Когда Σ устойчива, то вещественные части корней характеристического полинома связаны со скоростью, с которой состояние $\mathbf{x}$ сходится к $\mathbf{0}$ при отсутствии воздействий, а комплексная часть ответственна за колебательность этого процесса. Например, если полином имеет корни

$$(-0{,}2 - 3j,\ -0{,}2 + 3j,\ -0{,}5 - 7j,\ -0{,}5 + 7j,\ -1 - 3j,\ -1 + 3j), \tag{7.10}$$

то при отсутствии входных воздействий, все компоненты $y_i(t)$ вектора $\mathbf{y}(t)$ выхода системы Σ, задаваемой (7.1) или (7.2), имеют вид:

$$y_i(t) = a_1 \sin(3t + \phi_1)\exp(-0{,}2t) + \alpha_2 \sin(7t + \phi_2)\exp(-0{,}5t) + \\ + \alpha_3 \sin(3t + \phi_3)\exp(-t), \tag{7.11}$$

где коэффициенты $\alpha_1, \alpha_2, \alpha_3$, ϕ_1, ϕ_2 и ϕ_3 зависят от начальных условий $\mathbf{x}(0)$. Функция $y_i(t)$ показана на рис. 7.1 при $\alpha_1 = \alpha_2 = \alpha_3 = 1$

и $\phi_1 = \phi_2 = \phi_3 = 0$. Расположение корней прямо связано с временны́м поведением процессов на выходе системы. Для придания процессам нужного поведения естественно потребовать, чтобы эти корни лежали в предписанной области Γ комплексной плоскости. Это соответствует понятию Γ-*устойчивости*. В настоящей книге показывается, как интервальные методы могут эффективно доказывать Γ-устойчивость полинома без вычисления его корней. Например, доказательство того, что полином

$$P(s) = s^8 + 16s^7 + 112s^6 + 448s^5 + 1120s^4 + 1792s^3 + \\ + 1792s^2 + 1024s + 256 \tag{7.12}$$

является Γ-устойчивым, где Γ — круг радиуса 2 с центром в точке -3 комплексной плоскости, сводится к проверке того факта, что задача выполнения ограничений

$$\mathcal{H} : (P(s) = 0, |s + 3| > 2) \tag{7.13}$$

не имеет решения, что может быть сделано с помощью алгоритма SIVIAX, описанного на с. 140.

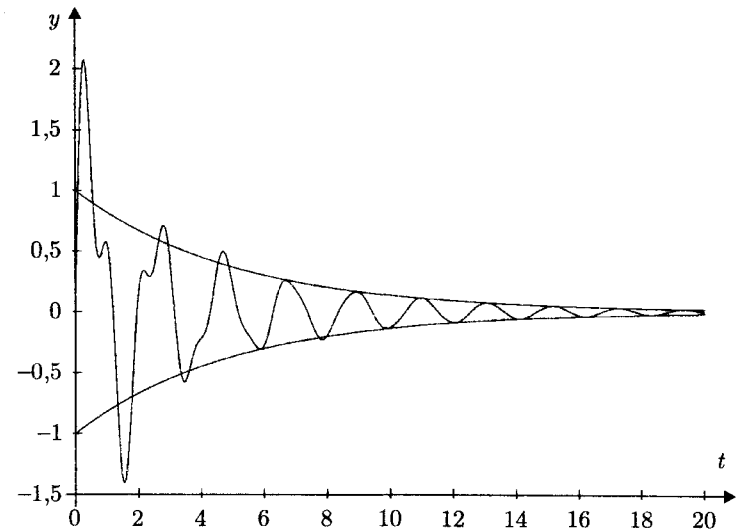


Рис. 7.1. Возможный вид процесса на выходе системы с корнями (7.10); асимптотические экспоненциальные огибающие кривые соответствуют уравнениям $\pm\exp(-0{,}2t)$

Иногда возможно свести Г-устойчивость к асимптотической устойчивости алгебраическим преобразованием [Sondergeld, 1983]. Это так, когда, например, Γ — часть Γ_δ комплексной плоскости, расположенной слева от вертикальной линии $\mathrm{Re}(s) = -\delta$. Говорим, что система Σ *δ-устойчива* тогда и только тогда, когда она Γ_δ-устойчива, т. е. все ее корни находятся в области Γ_δ. Например, если корни системы Σ заданы по (7.10), то система является 0,1-устойчивой, но 0,3-неустойчивой. Чтобы проверить δ-устойчивость полинома $P(s)$, достаточно проверить устойчивость полинома

$$\begin{aligned} Q_\delta(s) &= P(s-\delta) = (s-\delta)^n + a_{n-1}(s-\delta)^{n-1} + \ldots + a_1(s-\delta) + a_0 \\ &= s^n + b_{n-1}(\delta)s^{n-1} + \ldots + b_1(\delta)s + b_0(\delta), \end{aligned} \tag{7.14}$$

как устанавливается следующей теоремой.

Теорема 7.1. *Полином $P(s)$ δ-устойчив тогда и только тогда, когда устойчив полином $P(s-\delta)$.* ■

Доказательство. Предположение «$Q_\delta(s)$ устойчив» эквивалентно вложению

$$\forall s \in \mathbb{C},\ Q_\delta(s) = 0 \Rightarrow \mathrm{Re}(s) < 0, \tag{7.15}$$

т. е. вложению

$$\forall s \in \mathbb{C},\ P(s-\delta) = 0 \Rightarrow \mathrm{Re}(s) < 0, \tag{7.16}$$

которое эквивалентно вложению

$$\forall s \in \mathbb{C},\ P(s) = 0 \Rightarrow \mathrm{Re}(s+\delta) < 0. \tag{7.17}$$

Это означает, что все корни полинома $P(s)$ лежат в области Γ_δ. ■

Теперь можно проверять устойчивость полинома $Q_\delta(s)$, используя критерий Рауса. Определим δ-вектор Рауса $\mathbf{r}(\delta)$ как вектор Рауса, связанный с полиномом $Q_\delta(s)$. Тогда

$$P(s)\ \delta\text{-устойчив} \Leftrightarrow Q(s)\ \delta\text{-устойчив} \Leftrightarrow \mathbf{r}(\delta) > \mathbf{0}. \tag{7.18}$$

Степень устойчивости (или скорость затухания) полинома $P(s)$ определяется как

$$\delta_M \triangleq \sup_{\mathbf{r}(\delta)>\mathbf{0}} \delta = \max_{\mathbf{r}(\delta)\geqslant\mathbf{0}} \delta. \tag{7.19}$$

Аналогично, степень устойчивости системы Σ — такая же, как и у ее характеристического полинома.

Чем выше степень устойчивости системы Σ, тем быстрее ее состояние сходится к положению равновесия в отсутствии входных воздействий. Таким образом, степень устойчивости — важный фактор, который необходимо

учитывать при проектировании регулятора. Рис. 7.2 показывает смысл степени устойчивости δ_M для расположения корней (7.10); найденная здесь величина δ_M равна 0,2. Это означает, что асимптотически главной компонентой из процессов $y_i(t)$ является процесс $\alpha_1 \sin(3t + \phi_1)\exp(-0{,}2t)$ (см. рис. 7.1).

Часто, однако, уравнения, описывающие систему Σ, неточны, и хотелось бы учесть неопределенность ее модели при оценивании устойчивости системы, для того, чтобы сделать выводы о ее робастности по отношению к этой неопределенности. Методы учета неопределенности рассматриваются в следующем параграфе.

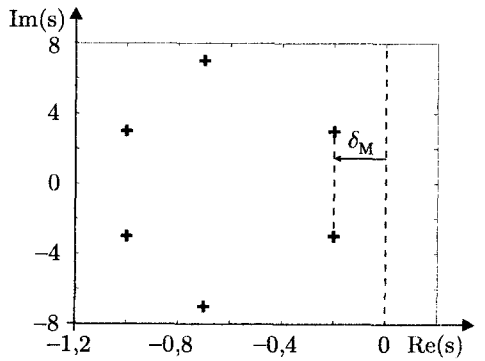


Рис. 7.2. Степень устойчивости δ_M; крестики соответствуют корням характеристического полинома

7.3. Основные проверки робастной устойчивости

Предположим теперь, что представление системы Σ в пространстве состояний зависит от стационарного вектора параметров размерности $n_\mathbf{p}$:

$$\Sigma(\mathbf{p}) : \begin{cases} \dot{\mathbf{x}}(t) = \mathbf{A}(\mathbf{p})\mathbf{x}(t) + \mathbf{B}(\mathbf{p})\mathbf{u}(t), \\ \mathbf{y}(t) = \mathbf{C}(\mathbf{p})\mathbf{x}(t), \end{cases} \tag{7.20}$$

где известно, что вектор $\mathbf{p}$ принадлежит параллелотопу $[\mathbf{p}]$. Пусть $\Sigma([\mathbf{p}])$ — множество всех систем $\Sigma(\mathbf{p})$, таких, у которых $\mathbf{p} \in [\mathbf{p}]$. Говорим, что система $\Sigma([\mathbf{p}])$ *робастно устойчива* тогда и только тогда, когда $\Sigma(\mathbf{p})$ — устойчива для любого $\mathbf{p} \in [\mathbf{p}]$. Доказательство робастной устойчивости системы $\Sigma(\mathbf{p})$

является фундаментальным вопросом теории робастного управления. Множество всех характеристических полиномов, связанных с $\Sigma(\mathbf{p})$, задается

$$P(s,[\mathbf{p}]) \triangleq \{a_n(\mathbf{p})s^n + a_{n-1}(\mathbf{p})s^{n-1} + \ldots + a_0(\mathbf{p}) \,|\, \mathbf{p} \in [\mathbf{p}]\}. \qquad (7.21)$$

Определим *коэффициентную функцию*

$$\mathbf{a}(\mathbf{p}) \triangleq (a_n(\mathbf{p}),\, a_{n-1}(\mathbf{p}),\, \ldots,\, a_1(\mathbf{p}),\, a_0(\mathbf{p}))^{\mathrm{T}} \qquad (7.22)$$

и *коэффициентное множество*

$$\mathbb{A} \triangleq \{\mathbf{a}(\mathbf{p}) \,|\, \mathbf{p} \in [\mathbf{p}]\} = \mathbf{a}([\mathbf{p}]). \qquad (7.23)$$

Полином $P(s,\mathbf{p})$ полностью описывается своей коэффициентной функцией $\mathbf{a}(\mathbf{p})$. Вот почему, для краткости, мы будем также использовать $\mathbf{a}(\mathbf{p})$, чтобы указывать на полином $P(s,\mathbf{p})$, а матрицу $\mathbb{A}$, чтобы указывать на соответствующее множество полиномов. Таким образом, по контексту символ $\mathbf{a}$ может быть некоторой коэффициентной функцией или полиномом, а матрица $\mathbb{A}$ — множеством коэффициентных функций или семейством полиномов.

Рассмотрим семейство $\mathbb{A}$ полиномов и предположим, что степень каждого из них равна n (таким образом, коэффициент a_n никогда не равен нулю). Говорим, что семейство $\mathbb{A}$ робастно устойчиво тогда и только тогда, когда все полиномы из $\mathbb{A}$ устойчивы, и оно *робастно неустойчиво* тогда и только тогда, когда все полиномы из этого семейства неустойчивы.

Рассматриваемая задача теперь состоит в проверке робастной устойчивости (или неустойчивости) различных типов множества $\mathbb{A}$.

• **Случай 1.** Коэффициентное множество $\mathbb{A}$ является параллелотопом

$$\mathbb{A} = [a_n] \times [a_{n-1}] \times \ldots \times [a_1] \times [a_0]. \qquad (7.24)$$

Соответствующее семейство полиномов классически называется *интервальным полиномом*. Оно может быть записано как

$$\mathbb{A} = [a_n]s^n + [a_{n-1}]s^{n-1} + \ldots + [a_1]s + [a_0]. \qquad (7.25)$$

Заметим, что когда множество $\mathbb{A}$ задается в форме (7.23), т. е. $\mathbb{A} = \mathbf{a}(\mathbf{p})$, то $\mathbf{a}$ может не быть тождественной функцией и может быть нелинейной по $\mathbf{p}$. Если, например, $\mathbf{a}$ — функция, отображающая из $\mathbb{R}^2$ в $\mathbb{R}^2$, определена выражением

$$\mathbf{a}(\mathbf{p}) = (\sin(p_1), \exp(p_2))^{\mathrm{T}}, \qquad (7.26)$$

то образ любого параллелотопа $\mathbf{p}$, даваемый функцией $\mathbf{a}$, является параллелотопом, и, следовательно, $\mathbb{A}$ является параллелотопом, или, что эквивалентно, интервальным полиномом.

• **Случай 2.** Коэффициентное множество $\mathbb{A}$ является многогранником, или, что эквивалентно, *афинным семейством*.

• **Случай 3.** Коэффициентное множество $\mathbb{A}$ есть образ параллелотопа $[\mathbf{p}]$, даваемый функцией $\mathbf{a}(\cdot)$ (вообще говоря, нелинейной). Будем говорить, что имеем дело с нелинейной зависимостью от параметров. Тогда $\mathbb{A}$ обозначает *нелинейную зависимость от параметров*, которая может быть записана в виде (7.23).

Пример 7.1. *Поскольку компоненты вектора параметров* $\mathbf{p}$ *появляются в коэффициентах независимо друг от друга, то семейство полиномов*

$$\mathbb{A} = \{(p_1 + \sin^2(p_2))s^2 + \exp(\sqrt{p_3})s + p_5 \ln(1+p_6) \,|\, \mathbf{p} \in [\mathbf{p}]\} \qquad (7.27)$$

есть интервальный полином и, следовательно, является многогранником. Здесь, хотя зависимость от параметров нелинейна, полином тем не менее является интервальным. Он может быть записан одинаково как

$$\mathbb{A} = ([p_1] + \sin^2([p_2]))s^2 + \exp\bigl(\sqrt{[p_3]}\bigr)s + [p_5]\ln(1+[p_6]), \qquad (7.28)$$

или как

$$\mathbb{A} = ([p_1] + \sin^2([p_2])) \times \exp\bigl(\sqrt{[p_3]}\bigr) \times [p_5]\ln(1+[p_6]). \qquad (7.29)$$

Семейство полиномов

$$\mathbb{A} = \{(\cos(p_1) + \sin^2(p_2))s^2 + 3\cos(p_1)s + p_3 + \sin^2(p_2) \,|\, \mathbf{p} \in [\mathbf{p}]\} \qquad (7.30)$$

не является интервальным полиномом, но есть многогранник. Например, если $[\mathbf{p}] = [0, \pi/2]^{\times 2}$, *то* $\cos([p_1]) = \sin^2([p_2]) = [0,1]$. *При этом свойство* $\mathbb{A}$ *может быть переписано в следующем виде*:

$$\mathbb{A} = \{(p'_1 + p'_2)s^2 + 3p'_1 s + p'_3 + p'_2 \,|\, \mathbf{p}' \in [0,\,1] \times [0,\,1] \times [0,\,\pi/2]\}. \qquad (7.31)$$

Вследствие зависимости между коэффициентами полинома, ему нельзя дать форму, подобную (7.28). ■

7.3.1. Интервальные полиномы

Теорема Харитонова [Харитонов, 1978] дает необходимые и достаточные условия робастной устойчивости интервальных полиномов (см. также [Bialas, 1983; Barmish, 1984], а также многочисленные публикации

80-х годов). Этот очень важный результат является началом подхода на основе экстремальных точек к проверке на робастную устойчивость систем с неопределенностями [Bartlett *et al.*, 1988].

Теорема 7.2. (Харитонов) *Интервальный полином*

$$[\mathbf{a}] = [\underline{a}_n, \overline{a}_n] \times \ldots \times [\underline{a}_1, \overline{a}_1] \times [\underline{a}_0, \overline{a}_0]$$

робастно устойчив тогда и только тогда, когда четыре полинома $K_1(s)$, $K_2(s)$, $K_3(s)$ и $K_4(s)$, задаваемые, соответственно, как

$$K_1(s) : \underline{a}_n s^n + \underline{a}_{n-1} s^{n-1} + \overline{a}_{n-2} s^{n-2} + \overline{a}_{n-3} s^{n-3} + \underline{a}_{n-4} s^{n-4} + \underline{a}_{n-5} s^{n-5} + \ldots,$$

$$K_2(s) : \overline{a}_n s^n + \underline{a}_{n-1} s^{n-1} + \underline{a}_{n-2} s^{n-2} + \overline{a}_{n-3} s^{n-3} + \overline{a}_{n-4} s^{n-4} + \underline{a}_{n-5} s^{n-5} + \ldots,$$

$$K_3(s) : \overline{a}_n s^n + \overline{a}_{n-1} s^{n-1} + \underline{a}_{n-2} s^{n-2} + \underline{a}_{n-3} s^{n-3} + \overline{a}_{n-4} s^{n-4} + \overline{a}_{n-5} s^{n-5} + \ldots,$$

$$K_4(s) : \underline{a}_n s^n + \overline{a}_{n-1} s^{n-1} + \overline{a}_{n-2} s^{n-2} + \underline{a}_{n-3} s^{n-3} + \underline{a}_{n-4} s^{n-4} + \overline{a}_{n-5} s^{n-5} + \ldots,$$

являются устойчивыми. ■

Таким образом, изучение робастной устойчивости несчетного множества полиномов сведено к проверке устойчивости максимум четырех из них, независимо от степени n полиномов. (Когда $n \leqslant 5$, оказывается возможным доказать робастную устойчивость даже с меньшими вычислениями [Anderson *et al.*, 1987].) Если семейство $\mathbb{A}$ не является интервальным полиномом, теорема Харитонова все еще может использоваться путем построения оболочки для $\mathbb{A}$ из интервальных полиномов $[\mathbf{a}]$. Если $[\mathbf{a}]$ робастно устойчиво, то $\mathbb{A}$ также робастно устойчиво, но указанное условие становится, разумеется, только достаточным. Это иллюстрируется следующим примером.

Пример 7.2. *Семейство полиномов*

$$\mathbb{A} = \{p_5 s^4 + (p_4 + \cos^2(p_3))s^3 + 2p_1 s^2 + p^2\sqrt{p_4}s + p_1 \mid p_1 \in [5, 7],\ p_2 \in [3, 4],\ p_3 \in [-\pi/4, \pi/4],\ p_4 \in [1, 2],\ p_5 \in [1, 2]\}$$

не является интервальным полиномом, но является подмножеством интервального полинома

$$[\mathbf{a}] = [1, 2]s^4 + [3/2, 3]s^3 + [10, 14]s^4 + [3, 4\sqrt{2}]s + [5, 7]. \qquad (7.32)$$

Полиномы Харитонова, связанные с $[\mathbf{a}]$, *имеют вид*:

$$\begin{aligned} K_1(s) &= s^4 + {}^3\!/_2 s^3 + 14s^2 + 4\sqrt{2}s + 5, \\ K_2(s) &= 2s^4 + {}^3\!/_2 s^3 + 10s^2 + 4\sqrt{2}s + 7, \\ K_3(s) &= 2s^4 + 3s^3 + 10s^2 + 3s + 7, \\ K_4(s) &= s^4 + 3s^3 + 14s^2 + 3s + 5. \end{aligned} \qquad (7.33)$$

Легко доказать, что каждый из них устойчив, например, по критерию Рауса. Следовательно, семейство $[\mathbf{a}]$ *является робастно устойчивым, и таковым является семейство* $\mathbb{A}$. *Предположим теперь* [Wei и Yedavali, 1989], *что*

$$p_1 \in [1{,}5, 4],\ p_2 = 1,\ p_3 = \pi/2,\ p_4 = 1,\ p_5 = 1. \qquad (7.34)$$

Тогда

$$[\mathbf{a}] = s^4 + s^3 + [3, 8]s^2 + s + [1{,}5,\ 4]. \qquad (7.35)$$

Соответствующий полином Харитонова $K_2(s) = s^4 + s^3 + 3s^2 + s + 4$ — *неустойчив, что влечет за собой робастную неустойчивость* $[\mathbf{a}]$. *Но отсюда не следует, что семейство* $\mathbb{A}$ *не является робастно устойчивым. И действительно, легко показать, что* $\mathbb{A}$ *является робастно устойчивым, например, путем формального построения таблицы Рауса в зависимости от* p_1. ■

На практике семейство $\mathbb{A}$ редко бывает интервальным полиномом. Например, если компоненты вектора параметров $\mathbf{p}$ присутствуют независимо в матрице системы $\mathbf{A}(\mathbf{p})$, то коэффициентная функция $\mathbf{a}(\mathbf{p})$ — мультилинейна, и коэффициентное множество $\mathbb{A}$ не является даже многогранником.

7.3.2. Афинное семейство полиномов

Основным результатом относительно робастной устойчивости афинного семейства является *теорема о реберном полиноме* [Bartlett *et al.*, 1988].

Теорема 7.3. (Реберная теорема) *Афинное семейство* $\mathbb{A}$ *является робастно устойчивым, если все его ребра робастно устойчивы.* ■

Напомним, что число ребер в параллелотопе увеличивается экспоненциально от его размерности. Например, параллелотоп размерности 10 имеет 5120 ребер. Следовательно, теорема о реберном полиноме может потребовать исследования робастной устойчивости многих реберных афинных семейств вида

$$P_{1,2}(s, \lambda) = \lambda P_1(s) + (1 - \lambda)P_2(s), \qquad \text{при } \lambda \in [0, 1]. \qquad (7.36)$$

Теперь уже эта задача не является легкой. Есть примеры, где две вершины $P_1(s)$ и $P_2(s)$ ребра являются устойчивыми, в то время как само ребро $P_{1,2}(s,\lambda)$ не является робастно устойчивым [Bialas и Garloff, 1985]. Робастная устойчивость ребра может быть установлена с использованием алгебраических условий [Bialas, 1985], основанных на критерии Гурвица. Другой подход состоит в работе с афинным семейством как со специальным случаем множественно-образных полиномов, как описывается в следующем параграфе.

7.3.3. Нелинейная зависимость от параметров

Предположим, что характеристический полином системы $\Sigma(\mathbf{p})$ имеет вид

$$P(s,\,\mathbf{p}) = s^n + a_{n-1}(\mathbf{p})s^{n-1} + \ldots + a_1(\mathbf{p})s + a_0(\mathbf{p}). \tag{7.37}$$

Вектор Рауса для $P(s,\mathbf{p})$ является функцией от $\mathbf{p}$, называемой *функцией Рауса* [Didrit, 1997], и обозначается как $\mathbf{r}(\mathbf{p})$. По критерию Рауса верно следующее тождество:

$$P(s,\mathbf{p}) \text{ является устойчивым } \Leftrightarrow \mathbf{r}(\mathbf{p}) > \mathbf{0}. \tag{7.38}$$

Таким образом, система $\Sigma([\mathbf{p}])$ робастно устойчива тогда и только тогда, когда $\mathbf{r}(\mathbf{p}) > \mathbf{0}$ для любого $\mathbf{p}$ из параллелотопа $[\mathbf{p}]$. Пусть $[\mathbf{r}]$ — некоторая функция включения для функции Рауса $\mathbf{r}$. Если все компоненты $[r_i]([\mathbf{p}])$ в $[\mathbf{r}]([\mathbf{p}])$ положительны (т. е. если их нижние границы строго положительны), то система $\Sigma([\mathbf{p}])$ робастно устойчива. С другой стороны, если существует некоторая компонента r_i в $\mathbf{r}$, такая, что $r_i([\mathbf{p}]) \leqslant 0$, то $\Sigma([\mathbf{p}])$ – робастно неустойчива (т. е. $\Sigma(\mathbf{p})$ не является асимптотически устойчивой для любого $\mathbf{p}$ из параллелотопа $[\mathbf{p}]$).

Эффективный способ доказать робастную устойчивость или неустойчивость системы $\Sigma([\mathbf{p}])$ состоит в использовании сжимающих операторов (см. Главу 4). Доказательство робастной неустойчивости сводится к доказательству того, что неравенство $\mathbf{r}(\mathbf{p}) > \mathbf{0}$ не имеет решений на параллелотопе $[\mathbf{p}]$. Это эквивалентно тому, если сказать, что множество решений задачи выполнения ограничений

$$\mathcal{H} \,:\, (\mathbf{r}(\mathbf{p}) - \mathbf{y} = \mathbf{0},\ \mathbf{p} \in [\mathbf{p}],\ \mathbf{y} \in [0,\infty[^{\times n}) \tag{7.39}$$

пусто.

Заметим, что подход на основе анализа ребер не в состоянии доказать робастную неустойчивость даже для интервальных полиномов. Таким образом, доказательство робастной устойчивости сводится к доказательству того, что

$$\begin{aligned}
&\forall \mathbf{p} \in [\mathbf{p}],\ \mathbf{r}(\mathbf{p}) > 0\\
\Leftrightarrow &\forall \mathbf{p} \in [\mathbf{p}],\ \forall i \in \{1,\ldots,n\},\ r_i(\mathbf{p}) > 0\\
\Leftrightarrow &\forall i \in \{1,\ldots,n\},\ \forall \mathbf{p} \in [\mathbf{p}],\ r_i(\mathbf{p}) > 0\\
\Leftrightarrow &\forall i \in \{1,\ldots,n\},\ \forall \mathbf{p} \in [\mathbf{p}], \neg(r_i \leqslant 0)\\
\Leftrightarrow &\forall i \in \{1,\ldots,n\},\ \neg(\exists \mathbf{p} \in [\mathbf{p}] \,|\, r_i(\mathbf{p}) \leqslant 0),
\end{aligned} \tag{7.40}$$

где символ $\neg A$ — эквивалентен высказыванию «A — ложно». Теперь, скажем, что высказывание $\neg(\exists \mathbf{p} \in [\mathbf{p}] \,|\, r_i(\mathbf{p}) \leqslant 0)$ эквивалентно тому, как если сказать, что задача выполнения ограничений

$$\mathcal{H}_i \,:\, (r_i(\mathbf{p}) + y = 0,\ \mathbf{p} \in [\mathbf{p}],\ y \in]0,\infty[) \tag{7.41}$$

имеет пустое множество решений. Следовательно, система $\Sigma([\mathbf{p}])$ робастно устойчива тогда и только тогда, когда все эти задачи $\mathcal{H}_i$ имеют пустые множества решения. Таким образом, все сжимающие операторы, представленные в Главе 4, могут быть использованы для проверки как робастной устойчивости, так и робастной неустойчивости. Заметим, однако, что проверка робастной устойчивости при таком подходе требует сжатия n этих задач, вместо одной для решения задачи робастной неустойчивости.

ЗАМЕЧАНИЕ 7.1. Рассмотренные здесь соображения по доказательству робастной устойчивости можно также использовать для проверки того факта, что некоторый заданный параллелотоп находится внутри множества $\mathbb{S}$, определенного системой нелинейных неравенств. Таким образом, сжимающие операторы можно было бы использовать в таком алгоритме, как SIVIA, чтобы не только проверить, что некоторый параллелотоп лежит вне множества $\mathbb{S}$, но также чтобы проверить, что параллелотоп лежит внутри множества $\mathbb{S}$. ■

ЗАМЕЧАНИЕ 7.2. Когда коэффициентная функция $\mathbf{a}(\mathbf{p})$ является полиномом от $\mathbf{p}$, то таковой является и функция $\mathbf{r}(\mathbf{p})$. Тогда можно использовать полиномы Бернштейна для вычисления аппроксимации сверху функции $\mathbf{r}([\mathbf{p}])$ и, таким образом, для проверки знаков компонент функции $\mathbf{r}([\mathbf{p}])$ [Vicino *et al.*, 1990; Milanese *et al.*, 1991; Garloff, 2000]. ■

7.3.4. Заключение

Сложность задачи проверки робастной устойчивости системы $\Sigma([\mathbf{p}])$ принципиально зависит от природы коэффициентного множества $\mathbb{A}$. Когда $\mathbb{A}$ — параллелотоп, то теорема Харитонова сводит эту задачу к проверке устойчивости не более четырех полиномов, какой бы ни была степень этих полиномов с неопределенностями. Когда $\mathbb{A}$ — многогранник, то теорема

о реберном полиноме сводит эту задачу к проверке устойчивости полинома на ребре, что может и не выявить требуемого. Когда $\mathbb{A}$ — множественно-образный полином общего вида, то задача становится крайне сложной [Nemirovskii, 1993; Poljak и Rohn, 1993; Blondel и Tsitsiklis, 1995]. Следующий параграф представляет численные методы, основанные на интервальном анализе, которые могут быть использованы даже в этом наиболее сложном случае. Кроме того, эти методы сделают возможным описывать ту часть параллелотопа $[\mathbf{p}]$, которая соответствует устойчивым системам, когда система $\Sigma([\mathbf{p}])$ не является ни робастно устойчивой, ни робастно неустойчивой, а также описывать линии уровня степени устойчивости.

7.4. Анализ робастной устойчивости

Основные способы проверки, рассмотренные в параграфе 7.3, будут теперь использованы для анализа робастной устойчивости системы $\Sigma([\mathbf{p}])$.

7.4.1. Области устойчивости

Область устойчивости $\mathbb{S}_\mathbf{p}$ полинома

$$P(s,\mathbf{p}) = s^n + a_{n-1}(\mathbf{p})s^{n-1} + \ldots + a_1(\mathbf{p})s + a_0(\boldsymbol{p}) \qquad (7.42)$$

есть множество всех значений векторов параметров $\mathbf{p}$, таких, что полином $P(s,\mathbf{p})$ — устойчив. Это множество можно определить как

$$\mathbb{S}_\mathbf{p} \triangleq \{\mathbf{p} \in \mathbb{R}^{n_\mathbf{p}} \,|\, \mathbf{r}(\mathbf{p}) > 0\} = (]0,+\infty[^{\times n}). \qquad (7.43)$$

Таким образом, построение множества $\mathbb{S}_\mathbf{p}$ можно выполнить в рамках задачи обращения множества и сделать это с помощью алгоритма SIVIA [Walter и Jaulin, 1994].

Пример 7.3. *Рассмотрим мультиафинный полином* [Ackermann *et al.*, 1990; Ackermann, 1992]

$$P(s,\mathbf{p}) = s^3 + (p_1+p_2+2)s^2 + (p_1+p_2+2)s + \\ + 2p_1p_2 + 6p_1 + 6p_2 + 2 + \sigma^2, \qquad (7.44)$$

где коэффициент σ соответствует радиусу неустойчивого круга с центром в точке $\mathbf{p}_\mathrm{c} = (1,1)^\mathrm{T}$ внутри области устойчивости. Построение таблицы Рауса приводит к следующей функции Рауса:

$$\mathbf{r}(\mathbf{p}) = \begin{pmatrix} p_1+p_2+2 \\ (p_1-1)^2 + (p_2-1)^2 - \sigma^2 \\ 2(p_1+3)(p_2+3) - 16 + \sigma^2 \end{pmatrix}. \qquad (7.45)$$

Заметим, что если $\sigma = 0$, а p_1 и p_2 — положительны, то полином $P(s,\mathbf{p})$ — всегда устойчив, за исключением точки $\mathbf{p}_\mathrm{c}$.

При $[\mathbf{p}] = [-3,7] \times [-3,7]$, $\varepsilon = 0{,}05$ и $\sigma = 0{,}5$, алгоритм SIVIA *рассчитал покрытие* (рис. 7.3) *за* 0,3 *секунды на компьютере* Pentium 90 [Didrit, 1997]. *Поскольку все компоненты функции включения*

$$[\mathbf{r}]([2,7],[2,7]) = \begin{pmatrix} [6,\,16] \\ [\frac{3}{4},\,\frac{287}{4}] \\ [\frac{135}{4},\,\frac{735}{4}] \end{pmatrix} \qquad (7.46)$$

положительны, то на единственной итерации показано, что на параллелотопе $[2,7] \times [2,7]$ полином (7.44) *устойчив* (см. рис. 7.3).

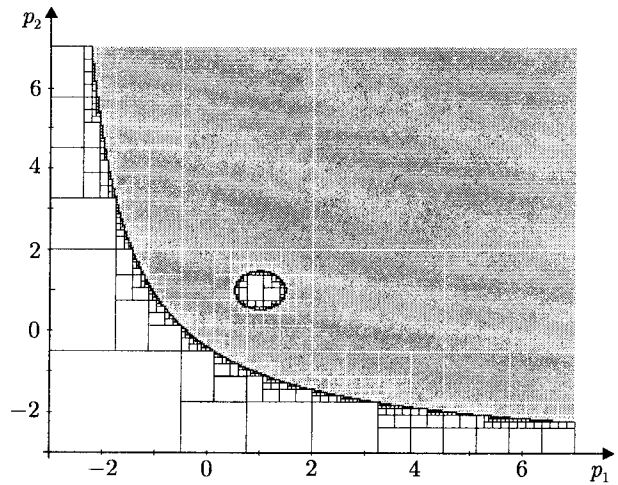


Рис. 7.3. Построение области устойчивости для Примера 7.3 при $\sigma = 0{,}5$; серые параллелотопы — области устойчивости; белые параллелотопы — области неустойчивости

С другой стороны, поскольку компоненты функции включения

$$[\mathbf{r}]([-3,-\tfrac{1}{2}],[-3,-\tfrac{1}{2}]) = \begin{pmatrix} [-4,\,1] \\ [\frac{17}{4},\,\frac{127}{4}] \\ [-\frac{63}{4},\,-\frac{13}{4}] \end{pmatrix}, \qquad (7.47)$$

то на единственной итерации показано, что на параллелотопе $[-3,-\frac{1}{2}]\times$ $\times[-3,-\frac{1}{2}]$ *полином* (7.44) *неустойчив.* Рис. 7.4 *получен при* $\sigma = 0$ *за* 0,2 *секунды. Алгоритм* SIVIA *не смог показать, что точка* $\mathbf{p} = (1,1)^{\mathrm{T}}$ *дает неустойчивость, но вокруг этой точки неустойчивости был построен маленький (неразличимый на рисунке) параллелотоп, по которому не дается никакого заключения об устойчивости.* ■

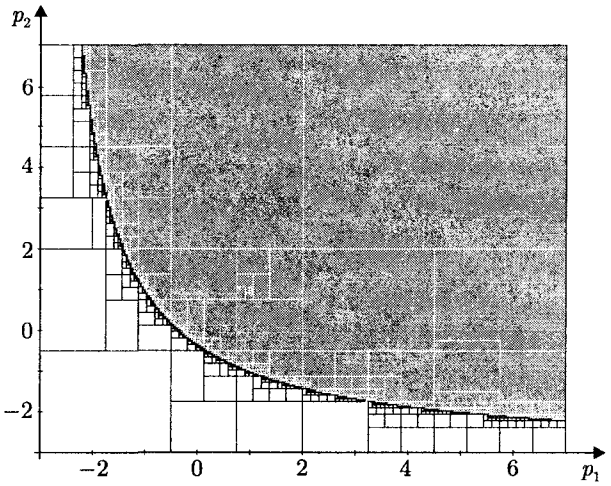


Рис. 7.4. Построение области устойчивости для Примера 7.3 при $\sigma = 0$; серые параллелотопы — области устойчивости; белые параллелотопы — области неустойчивости; точка неустойчивости $(1,1)$ окружена маленьким, неразличимым параллелотопом

Пример 7.3 был использован Аккерманом для иллюстрации пределов применения идеи изучения ребер: при $\sigma = 0{,}5$ и параллелотопе $[\mathbf{p}] = [2,2]\times[2,2]$, все ребра параллелотопа $[\mathbf{p}]$ дают устойчивость, но, тем не менее, сам параллелотоп содержит область неустойчивости. Этот пример решался в [Murdock *et al.*, 1991] на основе подхода, использующего алгоритм общего вида. Поскольку этот подход приводит к случайному поиску, его эффективность уменьшается с уменьшением σ, и не удается дать никакой гарантированности результата. В [Kiendl и Michalske, 1992] этот пример был изучен с использованием метода разбиения. Их подход оказывается состоятельным только в случае афинных семейств, что влечет за

собой пессимистичное изменение параметров модели с пагубными последствиями для качества результатов (см. рисунки в [Kiendl и Michalske, 1992]).

Следующий пример показывает, как алгоритм SIVIA может быть использован для построения *корневого годографа* $\mathcal{R}([\mathbf{p}])$ полинома с неопределенностью.

Пример 7.4. *Уже известно, что полином* (7.44) *с неопределенностью* $P(s,\mathbf{p})$ *из* Примера 7.3 *робастно устойчив при* $\sigma = 0$ *на параллелотопе* $\mathbf{p} \in \in [\mathbf{p}] = [2,7]\times[2,7]$. *Определим распределение* $\mathcal{R}([\mathbf{p}])$ *корней как множество всех корней полинома* $P(s,\mathbf{p})$ *при* $\mathbf{p}\in[\mathbf{p}]$, *т. е.*

$$\mathcal{R}([\mathbf{p}]) \triangleq \{s\in\mathbb{C} \mid P(s,\mathbf{p}) = 0,\ \mathbf{p}\in[\mathbf{p}]\}. \tag{7.48}$$

Таким образом, $\mathcal{R}([\mathbf{p}])$ *является проекцией на комплексную плоскость следующего множества*:

$$\{(s,\mathbf{p})\in\mathbb{C}\times[\mathbf{p}] \mid P(s,\mathbf{p}) = 0\}, \tag{7.49}$$

которое может быть построено алгоритмом SIVIA *из* параграфа 3.4.1 (с. 81). *Соответствующее покрытие, изображенное на* рис. 7.5, *пересекается с мнимой осью, поэтому устойчивость не доказана. Тем не менее это покрытие дает важную информацию о модели неопределенности. Например, показывается, что даже если модель робастно устойчива на параллелотопе* $[\mathbf{p}]$, *малые непараметрические возмущения могут сместить некоторые ее корни через мнимую ось.* ■

7.4.2. Степень устойчивости

Множества уровня. Когда система Σ зависит от вектора параметров $\mathbf{p}$, получаем δ-вектор Рауса, записываемый как $\mathbf{r}(\mathbf{p},\delta)$ (см. (7.18), с. 244). Степень устойчивости системы Σ определяется как

$$\delta_M(\mathbf{p}) \triangleq \sup_{\mathbf{r}(\mathbf{p},\delta)>\mathbf{0}} \delta = \sup_{\mathbf{r}(\mathbf{p},\delta)\geqslant\mathbf{0}} \delta. \tag{7.50}$$

Алгоритм ISOCRIT построения множеств уровня, описанный в Главе 5, (параграф 5.7, с. 177), теперь можно применить к функции $\delta_M(\mathbf{p})$. Напомним, что множеством уровня функции $\delta_M(\mathbf{p})$ при некотором значении уровня δ_1 является множество всех значений $\mathbf{p}$, в рассматриваемом пространстве параметров, таких, что $\delta_M(\mathbf{p}) = \delta_1$. Рассматриваемые уровни будем обозначать как $\delta_1, \delta_1, \ldots, \delta_m$. Предполагается, что коэффициентные функции $\mathbf{a}(\mathbf{p})$ и функция $\delta_M(\mathbf{p})$ — непрерывны.

Для своей работы алгоритм ISOCRIT требует некоторую функцию включения для функции $\delta_M(\mathbf{p})$, которую можно оценивать на любом параллелотопе $[\mathbf{p}]$ путем минимизации и максимизации значений $\delta_M(\mathbf{p})$ на этом

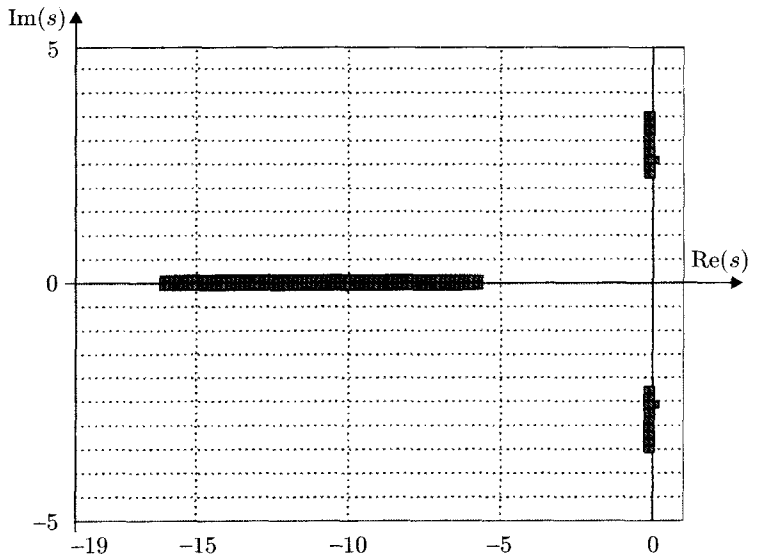


Рис. 7.5. Аппроксимация сверху множества расположения корней полинома $P(s, \mathbf{p})$ с неопределенностью из Примера 7.4

параллелотопе с помощью алгоритма OPTIMIZE, описанного в Главе 5 (табл. 5.5, с. 157).

Пример 7.5. *Для мультиафинного полинома* Примера 7.3 *при* $[\mathbf{p}] = [-3,7] \times [-3,7]$, $\delta_1 = 0{,}1$, $\delta_2 = 0$, $\varepsilon = 0{,}05$ *и* $\sigma = 0{,}5$ *алгоритм* ISOCRIT *рассчитал покрытие, представленное на* рис. 7.6, *за* 3,6 *секунд на компьютере* Pentium 90 [Didrit, 1997]. *Множество уровня, соответствующее уровню* $\delta_2 = 0$, *согласуется с результатом* Примера 7.3 (см. рис. 7.3). ■

Пример 7.6. *Рассмотрим теперь систему с неопределенностями* [Kokame и Mori, 1992]

$$\dot{\mathbf{x}} = \begin{pmatrix} 0 & 1 & -p_1 \\ 1 & 0 & -p_2 \\ p_1 & p_2 & 1 \end{pmatrix} \mathbf{x}, \quad p_1 \in [-7,\ 1{,}3], \quad p_2 \in [-1,\ 2{,}5]. \tag{7.51}$$

Ее характеристический полином имеет вид:

$$P(s, \mathbf{p}) = s^3 + s^2 + (p_1^2 + p_2^2 + 1)s + 1. \tag{7.52}$$

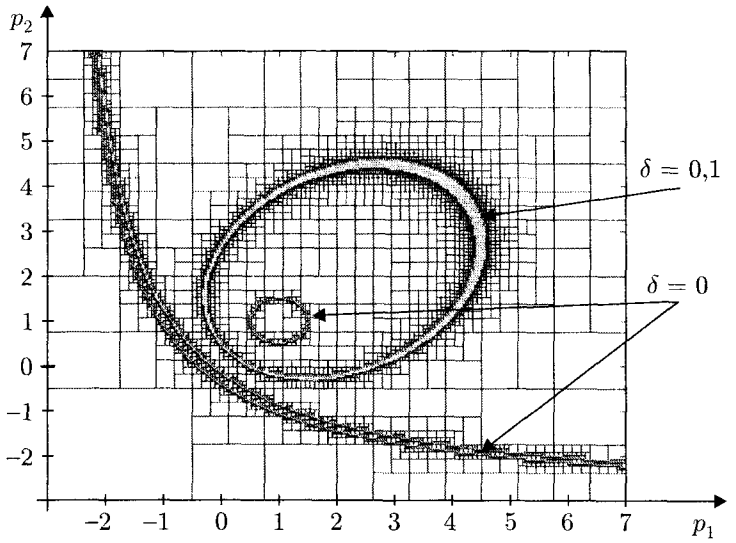


Рис. 7.6. Множества уровня степени устойчивости при $\sigma = 0{,}5$

Из таблицы Рауса для полинома $P(s - \delta, \mathbf{p})$ *можно легко показать, что при* $\delta \geqslant \frac{1}{3}$ *система является* δ*-неустойчивой для любых* $\mathbf{p} \in [\mathbf{p}]$ *и что при* $0 \leqslant \delta < \frac{1}{3}$ *система является* δ*-устойчивой тогда и только тогда, когда*

$$\begin{cases} p_1^2 + p_2^2 > \dfrac{4\delta(2\delta^2 - 2\delta + 1)}{-2\delta + 1} \triangleq \underline{\sigma}^2(\delta), \\ p_1^2 + p_2^2 < \dfrac{-\delta^3 + \delta^2 - \delta + 1}{\delta} \triangleq \overline{\sigma}^2(\delta). \end{cases} \tag{7.53}$$

Следовательно, система $\Sigma(\mathbf{p})$ *является* δ*-устойчивой для всех* $\mathbf{p}$, *расположенных между кругами, центрированными нулем, при их радиусах* $\underline{\sigma}(\delta)$ *и* $\overline{\sigma}(\delta)$. *Кроме того, единственной точкой, дающей неустойчивость, является* $\mathbf{0}$. *Для* $\delta_1 = 0{,}2$, $\delta_2 = 0{,}1$, $\delta_3 = 0{,}05$, $\delta_4 = 0$ *для параллелотопа* $[\mathbf{p}] = [-7,1{,}3] \times [-1,2{,}5]$ *и* $\varepsilon = 0{,}1$ *алгоритм* ISOCRIT *рассчитал покрытие, представленное на* рис. 7.7, *за* 0,3 *секунды на компьютере* Pentium 90 [Didrit, 1997]. ■

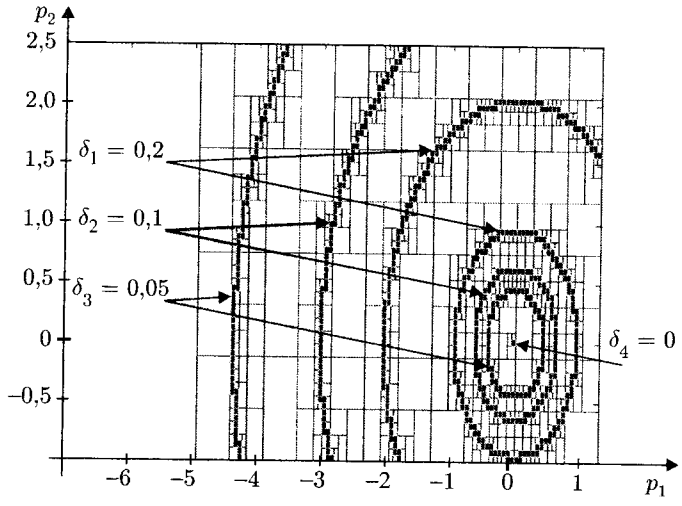


Рис. 7.7. Построение множеств уровня степени устойчивости

Этот пример был также исследован [Kokame и Mori, 1992]. Их метод применяется, когда передаточная матрица афинна по параметрам, и дает возможность только найти точку в пространстве параметров, которая является δ-устойчивой для заданной величины δ.

Степень робастной устойчивости. Определим *степень робастной устойчивости* $\delta_M([\mathbf{p}])$ системы $\Sigma(\mathbf{p})$ как степень устойчивости в наихудшем случае:

$$\delta_M([\mathbf{p}]) = \min_{\mathbf{p}\in[\mathbf{p}]} \max_{\mathbf{r}(\mathbf{p},\delta)\geqslant \mathbf{0}} \delta. \tag{7.54}$$

Если $\delta_M([\mathbf{p}]) > 0$, то тогда все корни характеристического полинома системы $\Sigma(\mathbf{p})$ лежат в левой полуплоскости $\mathbb{C}^-$ комплексной плоскости, и система $\Sigma([\mathbf{p}])$ с неопределенностями робастно устойчива (т. е. она устойчива для любых $\mathbf{p} \in [\mathbf{p}]$), как показано на рис. 7.8 (слева). Если $\delta_M([\mathbf{p}]) \leqslant 0$, то существует некоторое значение $\mathbf{p}$ из $[\mathbf{p}]$, такое, что система $\Sigma(\mathbf{p})$ неустойчива, как показано на рис. 7.8 (справа).

Алгоритм MINIMAX, описанный в параграфе 5.6. (с. 174), может быть использован для гарантированного вычисления степени робастной устойчивости $\delta_M([\mathbf{p}])$, что показывается следующими двумя примерами.

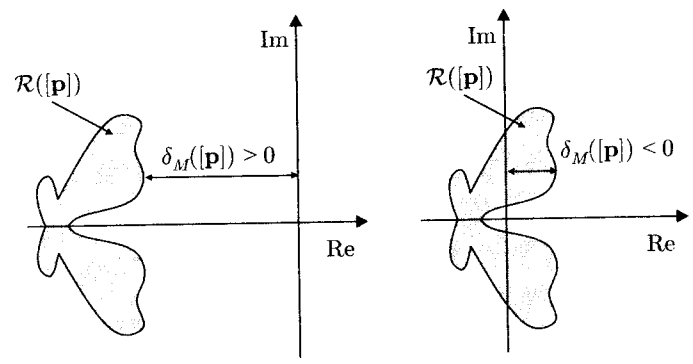


Рис. 7.8. Слева: область значений корней системы $\Sigma([\mathbf{p}])$ находится в левой полуплоскости $\mathbb{C}^-$, таким образом, $\delta_M([\mathbf{p}])$ положительно, и система $\Sigma([\mathbf{p}])$ робастно устойчива; справа: область значений корней системы $\Sigma([\mathbf{p}])$ не лежит полностью в $\mathbb{C}^-$, таким образом, $\delta_M([\mathbf{p}])$ отрицательно, и система $\Sigma([\mathbf{p}])$ робастно неустойчива для некоторых $\mathbf{p} \in [\mathbf{p}]$

Пример 7.7. *Рассмотрим систему с неопределенностью* [Balakrishnan, 1991a]

$$\dot{\mathbf{x}} = \begin{bmatrix} \dfrac{p_2}{1+p_2} & 2 \\ \dfrac{p_2}{1+p_1} & \dfrac{p_1}{1+p_2^2} \end{bmatrix} \mathbf{x}. \tag{7.55}$$

Для параллелотопа $[\mathbf{p}] = [1,2] \times [0,0{,}5]$ *при* $\varepsilon = 0{,}001$ *алгоритм* MINIMAX *на компьютере* Pentium 90 *за* 2 *секунды и за* 237 *итераций находит, что степень робастной устойчивости удовлетворяет неравенству*

$$-2{,}01590 \leqslant \delta_M([\mathbf{p}]) \leqslant -2{,}01451.$$

Таким образом, система робастно неустойчива. ■

Пример 7.8. *Рассмотрим систему с неопределенностью* [Balakrishnan, 1991a; 1991б]

$$\dot{\mathbf{x}} = \begin{bmatrix} \dfrac{1}{(p_1+3{,}5)^2 + (p_2+1)^2 + \frac{1}{0{,}9}} & 0 \\ 0 & \dfrac{1}{p_1^4 + p_2^4 + 1} \end{bmatrix} \mathbf{x}. \tag{7.56}$$

Для параллелотопа $[\mathbf{p}] = [-4, 0] \times [4, 4]$ *при* $\varepsilon = 0{,}01$ *алгоритм* MINIMAX *опять на компьютере* Pentium 90, *затратив только* 20 *итераций, находит, что степень робастной устойчивости удовлетворяет неравенству*

$$-1{,}0048866272 \leqslant \delta_M([\mathbf{p}]) \leqslant -0{,}9999999781. \tag{7.57}$$

Таким образом, система робастно неустойчива. ■

7.4.3. Подход на основе множества значений полинома

Идея *области значений* [Saeki, 1986; Barmish, 1988] допускает простую геометрическую интерпретацию робастной устойчивости на комплексной плоскости. Напомним, что система Σ с неопределенностью и с характеристическим полиномом

$$P(s, \mathbf{p}) = a_n(\mathbf{p})s_n + a_{n-1}(\mathbf{p})s^{n-1} + \ldots + a_1(\mathbf{p})s + a_0(\mathbf{p}) \tag{7.58}$$

робастно устойчива на параллелотопе $[\mathbf{p}]$, если задача выполнения ограничений

$$\mathcal{H} : (P(s, \mathbf{p}) = 0, \ \mathbf{p} \in [\mathbf{p}], \ \mathrm{Re}(s) \geqslant 0) \tag{7.59}$$

не имеет решения по s и $\mathbf{p}$. Этот факт может быть легко проверен с помощью алгоритма SIVIAX, описанного в параграфе 5.2 (с. 139). Поскольку s является комплексным числом, то размерность пространства поиска равна $\dim \mathbf{p} + 2$. Покажем сначала, что часто оказывается возможным понизить эту размерность до $\dim \mathbf{p} + 1$, пользуясь полезным свойством непрерывной зависимости корней характеристического полинома от его коэффициентов.

Предположим, что параллелотоп $[\mathbf{p}]$ содержит некоторый вектор параметров $\mathbf{p}_0$, дающий устойчивость, и некоторый вектор $\mathbf{p}_1$, дающий неустойчивость. Корни характеристического уравнения, соответствующие вектору $\mathbf{p}_0$, имеют отрицательные вещественные части, и по крайней мере один из корней, соответствующих вектору $\mathbf{p}_1$, имеет положительную вещественную часть. Это иллюстрируется на рис. 7.9. Предположим также, что коэффициенты полинома $P(s, \mathbf{p})$ непрерывны по $\mathbf{p}$ и что старший коэффициент $a_n(\mathbf{p})$ никогда не обнуляется. Когда вектор $\mathbf{p}$ движется от вектора $\mathbf{p}_0$ к вектору $\mathbf{p}_1$ в параллелотопе $[\mathbf{p}]$, по крайней мере один из корней пересекает мнимую ось комплексной плоскости (см. рис. 7.9), т. е. существуют некоторый вектор $\mathbf{p}$ из $[\mathbf{p}]$ и ω из $\mathbb{R}$, такие, что $P(j\omega, \mathbf{p}) = 0$. Это приводит к следующей теореме.

Теорема 7.4. *Если*

1) *коэффициенты полинома* $P(s, \mathbf{p})$ *являются непрерывными функциями от* $\mathbf{p}$,

2) *старший коэффициент* $a_n(\mathbf{p})$ *никогда не обнуляется на параллелотопе* $[\mathbf{p}]$,

3) *существует некоторый вектор* $\mathbf{p}_0$, *на котором полином* $P(s, \mathbf{p}_0)$ *устойчив,*

то полином $P(s, [\mathbf{p}])$ *устойчив тогда и только тогда, когда* задача выполнения ограничений

$$\mathcal{H} : (P(j\omega, \mathbf{p}) = 0, \ \mathbf{p} \in [\mathbf{p}], \ \omega \in \mathbb{R}) \tag{7.60}$$

не имеет решения. ■

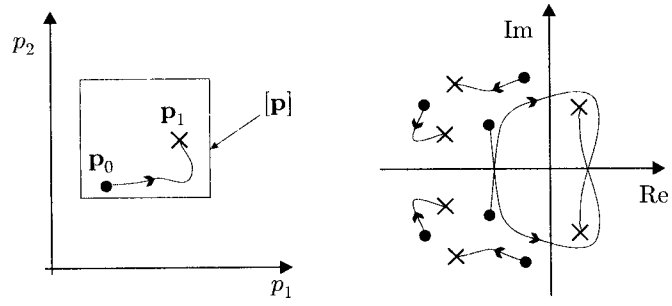


Рис. 7.9. Слева: траектория движения вектора $\mathbf{p}$ от вектора $\mathbf{p}_0$ (дающего устойчивость) к вектору $\mathbf{p}_1$ (дающему неустойчивость) в параллелотопе $[\mathbf{p}]$; справа: траектории движения корней на комплексной плоскости; по крайней мере один из корней пересекает мнимую ось

Устойчивость полинома $P(s, \mathbf{p}_0)$ может быть проверена, например, по критерию Рауса. Область для частоты ω может быть ограничена условием $\omega \geqslant 0$, так как если $(\mathbf{p}, \omega)$ является решением для (7.60), то $(\mathbf{p}, -\omega)$ также является решением. Размерность пространства поиска теперь составляет $\dim \mathbf{p} + 1$ вместо $\dim \mathbf{p} + 2$. Чтобы применить алгоритм SIVIAX для доказательства того, что (7.60) не имеет решения, важно ограничить область для частоты ω. Так как модуль полинома $P(j\omega, \mathbf{p})$ стремится к бесконечности с ростом ω, то существует значение угловой частоты ω_c (*частота среза*), за которой полином $P(j\omega, \mathbf{p})$ никогда не обратится в нуль для любых значений вектора параметров $\mathbf{p}$ из параллелотопа $[\mathbf{p}]$. Следующая теорема [Marden, 1966] дает средство для расчета некоторой верхней границы для ω_c.

Теорема 7.5. *Все корни полинома* $P(s) = a_n s^n + \ldots + a_1 s + a_0$ *при* $a_n \neq 0$ *лежат внутри круга, центрированного нулем, и радиуса*

$$\beta = 1 + \frac{\max\{|a_0|, |a_1|, \ldots, |a_{n-1}|\}}{|a_n|}. \tag{7.61}$$

■

Доказательство. Сначала докажем такой результат для монического полинома $P(s) = s^n + a_{n-1}s^{n-1} + \ldots + a_1 s + a_0$. Поскольку такой полином представляется в виде:

$$P_1(s) = s(s(s(s(\ldots) + a_4) + a_3) + a_2) + a_1) + a_0, \tag{7.62}$$

то полином $P_1(s)$ может быть получен по следующей последовательности:

$$\begin{aligned} &Q_0(s) = 1, \\ &Q_i(s) = sQ_{i-1}(s) + a_{n-i}, \quad i = 1, \ldots, n, \\ &P_1(s) = Q_n(s). \end{aligned} \tag{7.63}$$

Предположим, что $|s| > \beta$ и что $|Q_{i-1}(s)| \geqslant 1$ (это имеет место для $i = 1$). Тогда $|Q_i(s)| = |sQ_{i-1}(s) + a_{n-i}|$. Поскольку $|s| \geqslant \beta$, то $|Q_{i-1}(s)| \geqslant 1$ и (7.61) влекут, что $|a_{n-i}| \leqslant \beta$. Свойство $|\rho_1 e^{j\theta_1} + \rho_2 e^{j\theta_2}| \geqslant |\rho_1 + \rho_2|$, тогда влечет, что $|Q_i(s)| \geqslant \beta - (\beta - 1) = 1$. Следовательно, $|s| > \beta$ влечет, что $|P_1(s)| \geqslant 1$ и, поэтому, $|P_1(s)| \neq 0$. Теорема 7.5, таким образом, верна для монотонных полиномов. Теперь корни полинома $P(s)$ совпадают с корнями полинома

$$s^n + \frac{a_{n-1}}{a_n}s^{n-1} + \ldots + \frac{a_1}{a_n}s + \frac{a_0}{a_n}, \tag{7.64}$$

и, следовательно, расположены внутри круга, с центром в нуле, и радиуса

$$\beta = 1 + \max\left(|\frac{a_0}{a_n}|, \ldots, |\frac{a_{n-1}}{a_n}|\right) = 1 + \frac{\max(|a_0|, \ldots, |a_{n-1}|)}{|a_n|}. \tag{7.65}$$

■

Когда полином $P(s)$ зависит от вектора $\mathbf{p}$, радиус β становится функцией $\mathbf{p}$. Функция включения для функции $\beta(\mathbf{p})$ имеет вид:

$$[\beta]([\mathbf{p}]) = 1 + \frac{\max(|[a_0]([\mathbf{p}])|, \ldots, |[a_{n-1}]([\mathbf{p}])|)}{|[a_n]([\mathbf{p}])|}. \tag{7.66}$$

Если $\overline{\beta}$ – верхняя граница интервала $[\beta]([\mathbf{p}])$, то полином с неопределенностью $P(s, [\mathbf{p}])$ имеет все свои корни внутри круга, с центром в нуле

и радиуса $\overline{\beta}$. Таким образом, радиус $\overline{\beta}$ является верхней границей частоты среза ω_c. Область частоты ω берется из интервала $[0, \overline{\beta}]$, который является конечным. Для некоторых типов коэффициентных функций показано [Sideris, 1991; Ferreres и Magni, 1996], что исследование может быть сведено к рассмотрению задачи на конечном числе частот.

Пример 7.9. *Рассмотрим следующий полином* [Barmish и Tempo, 1995]:

$$P(s, \mathbf{p}) = s^3 + a_2(\mathbf{p})s^2 + a_1(\mathbf{p})s + a_0(\mathbf{p}), \tag{7.67}$$

где коэффициенты являются функциями вектора параметров

$$\begin{aligned} a_0(\mathbf{p}) &= (p_3 + 2)p_3^2 + p_1(\cos 2p_3 - p_2(p_4 - 0{,}5)^2) + 5, \\ a_1(\mathbf{p}) &= p_2(2\cos 2p_3 + p_1 \cos p_4) + 20, \\ a_2(\mathbf{p}) &= 4p_3 + p_2(1 + 2p_1) + 0{,}5, \end{aligned} \tag{7.68}$$

и возьмем параллелотоп $\mathbf{p} \in [\mathbf{p}] = [0, 1]^{\times 4}$. Уравнение (7.66) *дает* $\overline{\beta} = 24$ рад/с. *Алгоритм* SIVIAX *за* 0,005 *секунды на компьютере* Pentium 233 *подтверждает, что полином* $P(s, [\mathbf{p}])$ по (7.67) *робастно устойчив* (см. Упражнение 11.24, с. 405). ■

В литературе по робастному управлению Теорема 7.4 обычно представляется с помощью введения *множества значений* полинома

$$P(j\omega, [\mathbf{p}]) = \{P(j\omega, \mathbf{p}) \mid \mathbf{p} \in [\mathbf{p}]\}, \tag{7.69}$$

рассматриваемого в качестве функции частоты ω. Алгоритм IMAGESP из Главы 3 (параграф 3.4.2, с. 85) может быть использован для вычисления таких множеств значений.

Пример 7.10. *Снова рассмотрим задачу* Примера 7.9. *Аппроксимации сверху множеств значения полинома* $P(j\omega, [\mathbf{p}])$, *полученные алгоритмом* IMAGESP *для значений частоты* $\omega = 0{,}1$ рад/с, $\omega = 1{,}1$ рад/с *и* $\omega = 2{,}1$ рад/с, *показаны на* рис. 7.10. ■

Еще одна формулировка Теоремы 7.4 делается через *условие исключения нуля* [Frazer и Duncan, 1929].

Теорема 7.6. (Условие исключения нуля) *Если коэффициенты полинома* $P(s, \mathbf{p})$ *являются непрерывными функциями вектора параметров* $\mathbf{p}$, *и если существует некоторый вектор* $\mathbf{p}_0$ *из параллелотопа* $[\mathbf{p}]$, *такой, что полином* $P(s, \mathbf{p}_0)$ *устойчив, то полином* $P(s, [\mathbf{p}])$ *робастно устойчив тогда и только тогда, когда для любой частоты* $\omega \geqslant 0$, $\mathbf{0} \notin P(s, [\mathbf{p}])$. ■

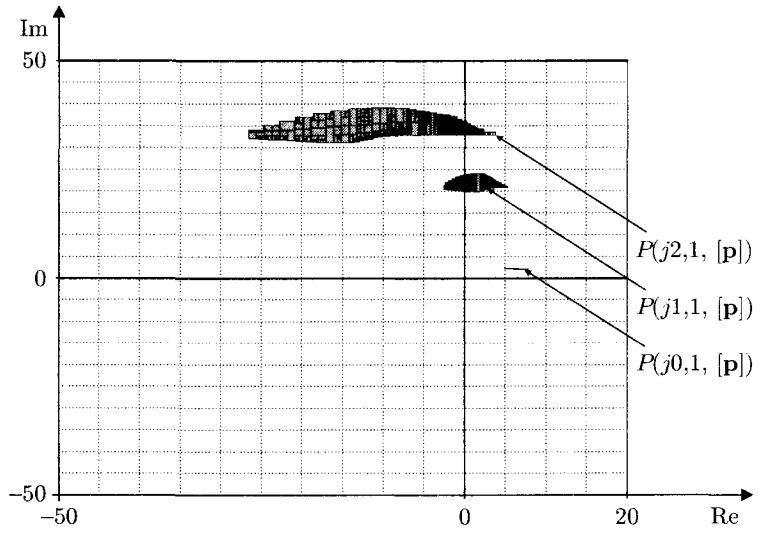


Рис. 7.10. Три аппроксимации сверху множеств значений в Примере 7.9

Пример 7.11. *Снова рассмотрим задачу* Примеров 7.9 и 7.10. *Алгоритм* IMAGESP *может быть использован для вычисления аппроксимации сверху множества*

$$P(j[0,24],[\mathbf{p}]) = \{P(j\omega,\mathbf{p}) \mid \omega \in [0,\,24],\ \ \mathbf{p} \in [\mathbf{p}]\}. \qquad (7.70)$$

Время вычислений может быть заметно снижено избежанием бисекций любого параллелотопа, образ которого не содержит **0** [Adrot, 2000]. *Результирующее покрытие, рассчитанное менее чем за* 2 *секунды на компьютере* Pentium 233, *представлено на* рис. 7.11. *Оно содержит три множества значений, показанных ранее на* рис. 7.10. *Поскольку множество* $P(j[0,24],[\mathbf{p}])$ *не содержит* **0**, *то условие исключения нуля влечет, что полином* $P(s,[\mathbf{p}])$ *является робастно устойчивым, так как легко показать, что полином* $P(s,\mathrm{mid}[\mathbf{p}])$ *устойчив.* ■

Замечание 7.3. Представленный здесь подход может быть использован даже если функция $P(s,\mathbf{p})$ не является полиномом. Это позволяет рассматривать системы с запаздыванием [Barmish, 1994] и, следовательно, дает возможность работы со значительно бóльшим классом задач, чем подход, основанный на критерии Рауса. ■

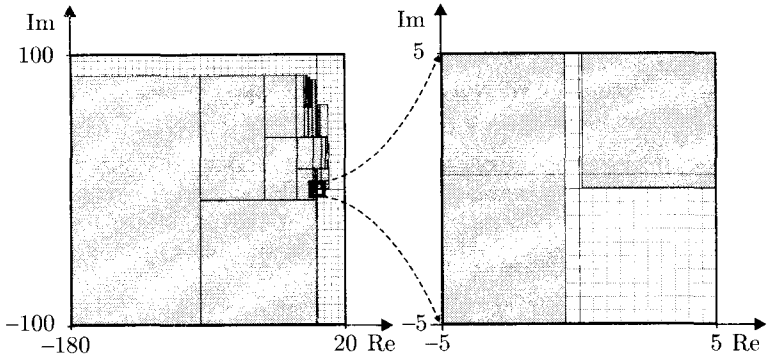


Рис. 7.11. Пересечение аппроксимации сверху множества всех множеств значений полинома с прямоугольниками $[-180,20]\times[-100,100]$ и $[-5,5]\times[-5,5]$; нуль **0** исключен

Чтобы вложить данную задачу в рамки задачи оптимизации [Didrit, 1997], можно преобразовать условие исключения нуля ($\exists\omega \geqslant 0,\ \mathbf{0} \notin P(j\omega,[\mathbf{p}])$) в эквивалентное условие

$$\eta([\mathbf{p}]) \triangleq \min_{\mathbf{p}\in[\mathbf{p}],\omega\geqslant 0} |P(j\omega,\mathbf{p})|^2 > 0. \qquad (7.71)$$

Размерность пространства поиска опять $\dim\mathbf{p}+1$. Область $[0,\overline{\beta}]$ частоты ω получается вычислением верхней границы частоты срезы, как в (7.66). Заметим, что $\eta([\mathbf{p}])$ может рассматриваться как запас устойчивости.

Пример 7.12. *Снова рассмотрим задачу* Примеров 7.9, 7.10 и 7.11. *В* [Barmish и Tempo, 1995] *показано, как построить аппроксимацию сверху множества значений полинома* $P(j\omega,[\mathbf{p}])$ *для заданного значения частоты* ω. *Из десяти таких аппроксимаций, полученных для десяти значений частоты* ω, *авторы без строгого доказательства заключают, что семейство полиномов* $P(s,[\mathbf{p}])$ *должно быть робастно устойчивым. Напомним, что верхняя граница частоты среза* $\overline{\beta}=24$ рад/с. *Область поиска составляет* $[\mathbf{x}]=[0,24]\times[0,1]^{\times 4}$. *Поскольку минимум целевой функции* $c(\omega,\mathbf{p}) = |P(j\omega,[\mathbf{p}])|^2$ *не зависит от параметра* p_4, *можно не делать бисекций интервала* $[p_4]$ [Didrit, 1997]. *После* 275 *бисекций за* 5,16 *секунд счета на компьютере* Pentium 90 *алгоритм Хансена, описанный в* параграфе 5.5.2 (с. 159), *отделил четыре параллелотопа решения. Каждый их них содержал одно значение вектора параметров* $\mathbf{p}$ *вида* $\mathbf{p}=(1,0,0,[p_4])^{\mathrm{T}}$. *Это может указывать на то, что решение лежит на границе рассматриваемой*

области, что не ставит проблем перед алгоритмом. Соответствующая область частот есть интервал [4,4609, 4,4611]; *доказано, что* $\eta(\mathbf{p})$ *принадлежит интервалу* [15,799, 15,802]. *Таким образом, мы не только доказали, что полином* $P(s, \mathbf{p})$ *робастно устойчив, но также количественно оценили запас устойчивости.* ■

Следующий пример показывает как Г-устойчивость может быть исследована, когда рассматриваемая область Г не является полуплоскостью.

Пример 7.13. *Для полинома*

$$P(s, \mathbf{p}) = s^3 + a_2(\mathbf{p})s^2 + a_1(\mathbf{p})s + a_0(\mathbf{p}) \tag{7.72}$$

коэффициентные функции имеют вид:

$$\begin{aligned} a_0(\mathbf{p}) &= \sin(p_2)\exp(p_2) + p_1 p_2 - 1, \\ a_1(\mathbf{p}) &= 2p_1 + 0{,}2p_1\exp(p_2), \\ a_2(\mathbf{p}) &= p_1 + p_2 + 4 \end{aligned} \tag{7.73}$$

и не являются ни линейными, ни полиномиальными. Используя некоторую афинную аппроксимацию сверху, [Amato *et al.*, 1995] *доказали, что этот полином устойчив для всех значений вектора параметров из параллелотопа* $[\mathbf{p}] = [1, 1{,}5]^{\times 2}$. *Сейчас мы исследуем робастную* Г*-устойчивость семейства полиномов* $P(s, [\mathbf{p}])$, *где* Г *— конус, симметричный относительно вещественной оси комплексной плоскости, с вершиной в* $\mathbf{0}$ *и углом полураствора* $\phi = \frac{\pi}{3}$. *Соображения, которым мы будем следовать, изображены на* рис. 7.12. *Область* Г *есть пересечение полуплоскостей* $\mathbb{D}^{-\frac{\pi}{3}}$ *и* $\mathbb{D}^{\frac{\pi}{3}}$. *Поскольку полином* $P(s, \mathbf{p})$ *имеет вещественные коэффициенты, его корни симметричны относительно вещественной оси, и некоторый полином является* Г*-устойчивым тогда и только тогда, когда он* $\mathbb{D}^{-\frac{\pi}{3}}$*-устойчив, и в этом случае он также является* $\mathbb{D}^{\frac{\pi}{3}}$*-устойчивым. Поэтому мы рассмотрим только задачу проверки* $\mathbb{D}^{-\frac{\pi}{3}}$*-устойчивости. Полином* (7.72), (7.73) *является* $\mathbb{D}^{-\frac{\pi}{3}}$*-устойчивым тогда и только тогда, когда уравнение*

$$P(s) = 0, \; s \notin \mathbb{D}^{-\frac{\pi}{3}} \tag{7.74}$$

не имеет решения по s. *Положим* $z = se^{-j\frac{\pi}{6}}$. *При этом уравнение* (7.74) *можно записать в следующей эквивалентной форме*:

$$P(ze^{j\frac{\pi}{6}}) = 0, \; z \notin \mathbb{C}^-, \tag{7.75}$$

где $\mathbb{C}^-$ *— множество всех комплексных чисел со строго отрицательными вещественными частями, и полином является* $\mathbb{D}^{-\frac{\pi}{3}}$*-устойчивым тогда и только тогда, когда уравнение* (7.75) *не имеет решения для* z. *Введем множество* $\widetilde{P}(z) = P(ze^{j\frac{\pi}{6}})$; *доказательство* Г*-устойчивости системы* $\Sigma([\mathbf{p}])$ *тогда сводится к доказательству устойчивости* $\widetilde{P}(z)$. *Это можно сделать проверкой того факта, что для некоторого заданного вектора* $\mathbf{p}$ *из параллелотопа* $[\mathbf{p}]$ *полином* $\widetilde{P}(s, \mathbf{p})$ *устойчив (что является тривиальным) и что*

$$\min_{\mathbf{p}\in[\mathbf{p}], \omega\in\mathbb{R}} |\widetilde{P}(j\omega, [\mathbf{p}])|^2 > 0 \tag{7.76}$$

(см. (7.71)). *Последнее условие может быть проверено по оптимизационному алгоритму Хансена при минимизации целевой функции*

$$c(\omega, \mathbf{p}) = |\widetilde{P}(j\omega, [\mathbf{p}])|^2. \tag{7.77}$$

Для $\varepsilon_{\mathrm{p}} = 10^{-3}$ *и* $\varepsilon_{\mathrm{c}} = 10^{-5}$ *после* 230 *итераций, выполненных за* 2,1 *секунды, на компьютере* Pentium 90 *был найден один параллелотоп решения, ширина которого меньше* ε_{p}. *Этот параллелотоп решения содержит точку с координатами* $p_1 = 1{,}5$, $p_2 = 1$ *и* $\omega = 0{,}678$ рад/с. *Алгоритм также выдает маленький интервал, который содержит минимум целевой функции* $c(\cdot)$, *ширина которого меньше чем* ε_{c}. *Этот минимум, приближенно найденный как* 0,217, *гарантированно является строго положительным. Робастная устойчивость семейства* $\widetilde{P}(z, [\mathbf{p}])$ *и, следовательно,* Г*-устойчивость семейства* $P(s, [\mathbf{p}])$, *таким образом, установлены.* ■

Приведение условия исключения нуля к такой оптимизационной задаче не дает возможности воспользоваться преимуществом геометрической интерпретации множеств значений, но оно допускает рассмотрение любого типа параметрической зависимости. Подход, подобный рассмотренному здесь, может также быть использован для выявления факта, является ли поведение системы с неопределенностью приемлемым в смысле нескольких критериев робастности.

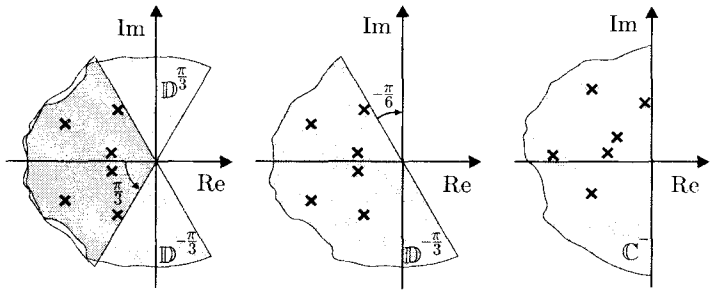


Рис. 7.12. Преобразование задачи оценки Γ-устойчивости в задачу оценки устойчивости полинома с комплексными коэффициентами

7.4.4. Запасы робастной устойчивости

Для иллюстрации понятия запаса робастной устойчивости рассмотрим систему с одним входом и одним выходом, определяемую передаточной функцией

$$G(s) = \frac{N(s)}{D(s)} = \frac{s+1}{s^2+0{,}4s+1}. \tag{7.78}$$

Отклик этой системы на единичное ступенчатое воздействие показан на рис. 7.13. (Отклик системы на единичное ступенчатое воздействие — это процесс на ее выходе, когда входное воздействие имеет вид $u = 0$ для $t < 0$ и $u = 1$ для $t \geqslant 0$.)

Охватим систему отрицательной обратной связью, как показано на рис. 7.14. Такие обратные связи обычно используются для противодействия влиянию внешних возмущений, подстройкой входа системы G на отклонение фактического выхода (y) от желаемого поведения (u). Пусть $H(s)$ — передаточная функция полученной замкнутой системы. При нулевых начальных условиях преобразование Лапласа $y(s)$ выхода $y(t)$ системы записывается как $y(s) = G(s)(u(s) - y(s))$, или, в эквивалентной форме, как

$$y(s) = \frac{G(s)}{1+G(s)}u(s). \tag{7.79}$$

Передаточная функция замкнутой системы с отрицательной обратной связью может быть записана как

$$H(s) = \frac{G(s)}{1+G(s)} = \frac{N(s)}{N(s)+D(s)} = \frac{s+1}{s^2+1{,}4s+2}. \tag{7.80}$$

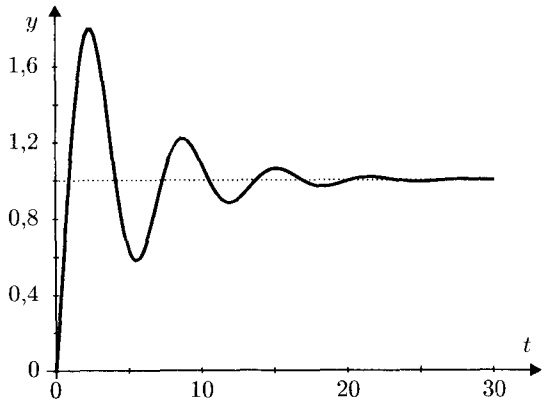


Рис. 7.13. Отклик незамкнутой системы (7.78) на единичное ступенчатое входное воздействие

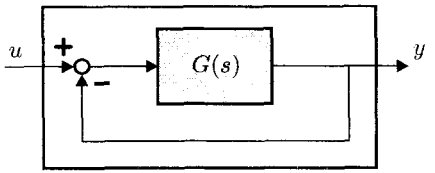


Рис. 7.14. Замкнутая система с отрицательной обратной связью

Отклик замкнутой системы с отрицательной обратной связью на единичное ступенчатое входное воздействие показан на рис. 7.15. Здесь $H(s)$ — устойчива, но если менять непрерывно коэффициенты передаточной функции $G(s)$, то $H(s)$ может стать неустойчивой. Такое происходит, когда один из нулей знаменателя $H(s)$ пересекает мнимую ось комплексной плоскости, т. е. когда

$$\exists\omega \,|\, G(j\omega) = \frac{N(j\omega)}{D(j\omega)} = -1. \tag{7.81}$$

Это сводится к тому, что $H(s)$ становится неустойчивой, когда множество

$$G(j\mathbb{R}) = \{G(j\omega) \mid \omega \in \mathbb{R}\} \tag{7.82}$$

пересекает критическую точку -1. Следовательно, возможно исследовать устойчивость замкнутой системы $H(s)$, анализируя свойства разомкну-

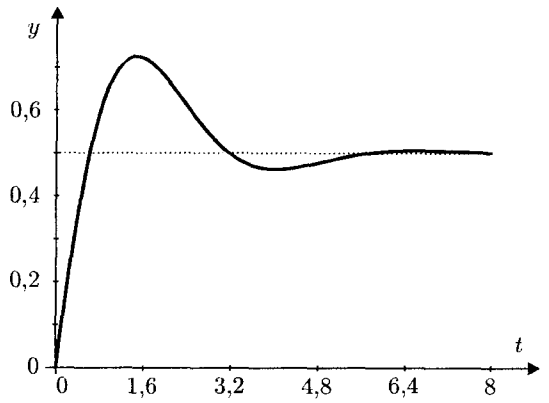


Рис. 7.15. Отклик замкнутой системы с отрицательной обратной связью на единичное ступенчатое входное воздействие

той системы $G(s)$. Особое значение имеет расположение кривой множества $G(j\omega)$, называемой *годографом Найквиста* передаточной функции G, относительно критической точки. Заметим, что эта критическая точка имеет модуль, равный 1, и фазу, равную $-\pi$. Годограф Найквиста системы (7.78) приведен на рис. 7.16.

Предположим, что номинальная модель $G(s)$ такова, что фукнция $H(s)$ устойчива. Чтобы оценить робастность устойчивости $H(s)$ относительно изменений модели $G(s)$, будем менять замкнутую систему, пока она не станет неустойчивой, умножая $G(s)$ на комплексный коэффициент $\rho e^{j\theta}$ при $\rho > 0$. Это показано на рис. 7.17. Передаточная функция разомкнутой системы теперь будет $\widetilde{G}(s) = \rho e^{j\theta} G(s)$, и соответствующая передаточная функция замкнутой системы будет обозначаться как $\widetilde{H}(s)$. При $\rho = 1$ и $\theta = 0$, система является невозмущенной, т. е. $\widetilde{G}(s) = G(s)$. Коэффициент ρ часто измеряется в децибелах (dB) как $\rho_{\text{dB}} = 20\log_{10}(\rho)$. Когда $\rho = 0$, 1 или ∞, $\rho_{\text{dB}} = -\infty$, 0 или ∞, соответственно, поэтому для невозмущенной системы $\rho_{\text{dB}} = 0$. Возмущенная передаточная функция может быть записана как

$$\widetilde{G}(s) = 10^{\frac{\rho_{\text{dB}}}{20}} \exp(j\theta) G(s). \tag{7.83}$$

Если ρ_{dB} или θ отклоняются от своих номинальных нулевых значений, то $\widetilde{H}$ становится неустойчивой, когда $\widetilde{G}(j\omega)$ пересекает критическую точку -1,

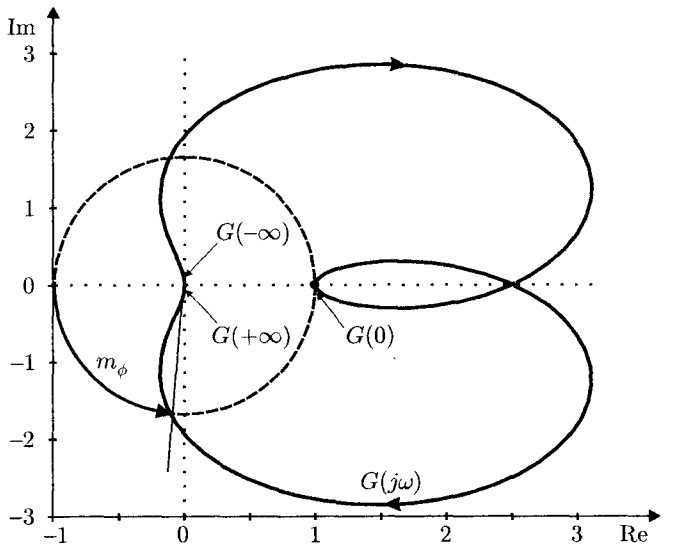


Рис. 7.16. Годограф Найквиста разомкнутой системы $G(s)$; его положение относительно критической точки -1 дает информацию о устойчивости замкнутой системы $H(s)$; угол m_ϕ показывает запас устойчивости по фазе системы $G(s)$

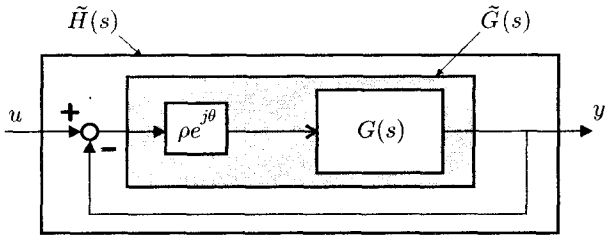


Рис. 7.17. Возмущенная система

т. е. когда

$$\exists \omega \in \mathbb{R} \mid 10^{\frac{\rho_{\text{dB}}}{20}} \exp(j\theta) G(j\omega) = -1. \tag{7.84}$$

Положим сначала $\theta = 0$ и определим *запас устойчивости* m_{G} *по амплитуде* как наименьшее значение $|\rho_{\text{dB}}|$, такое, при котором $\widetilde{H}(s)$ становится

неустойчивой. Это означает, что если коэффициент усиления разомкнутой системы изменен так, что абсолютное значение ρ_{dB} остается меньше m_{G}, то замкнутая система с отрицательной обратной связью остается устойчивой. Из (7.84)

$$m_{\text{G}} \triangleq \min\Big\{|\rho_{\text{dB}}| \mid \exists\omega \in \mathbb{R},\ 10^{\frac{\rho_{\text{dB}}}{20}} G(j\omega) = -1\Big\}. \tag{7.85}$$

Так как $10^{\frac{\rho_{\text{dB}}}{20}} G(j\omega) = -1 \Leftrightarrow \rho_{\text{dB}} = 20\log_{10}\dfrac{-1}{G(j\omega)}$, то запас устойчивости по амплитуде можно, эквивалентно, рассчитать как

$$\begin{cases} m_G = \min\left|20\log_{10}\dfrac{-1}{G(j\omega,\mathbf{p})}\right|, \\ \text{при условии } 10^{\frac{\rho_{\text{dB}}}{20}} G(j\omega) = -1. \end{cases} \tag{7.86}$$

Теперь, в (7.86) вещественное число ρ_{dB} является свободной переменной, и, таким образом, ограничение $10^{\frac{\rho_{\text{dB}}}{20}} G(j\omega) = -1$ эквивалентно утверждению, что функция $G(j\omega)$ — вещественна и строго отрицательна, т. е. что $\text{Im}(G(j\omega)) = 0$ и $\text{Re}(G(j\omega)) < 0$. Следовательно,

$$\begin{cases} m_G = \min\left|20\log_{10}\dfrac{-1}{G(j\omega,\mathbf{p})}\right|, \\ \text{при условии } (\text{Im}(G(j\omega)) = 0) \wedge (\text{Re}(G(j\omega)) < 0). \end{cases} \tag{7.87}$$

Кроме того, так как $G(j\omega) = G(-j\omega)$, когда ограничение $\text{Im}(G(j\omega)) = 0$ выполнено, то поиск может быть ограничен областью $\omega \geqslant 0$. Таким образом, нахождение запаса m_{G} сводится к решению задачи минимизации с ограничением

$$\begin{cases} m_G = \min\left|20\log_{10}\dfrac{-1}{G(j\omega,\mathbf{p})}\right|, \\ \text{при условии } (\text{Im}(G(j\omega)) = 0) \wedge (\text{Re}(G(j\omega)) < 0) \wedge (\omega \geqslant 0). \end{cases} \tag{7.88}$$

Положим теперь $\rho_{\text{dB}} = 0$ и определим *запас устойчивости* m_ϕ *по фазе* как наименьшую величину $|\theta|$, при которой передаточная функция $\tilde{H}(s)$ становится неустойчивой. Из (7.84)

$$m_\phi \triangleq \min\{|\theta| \mid \exists\omega \in \mathbb{R},\ \exp(j\theta)G(j\omega) = -1\}. \tag{7.89}$$

Поскольку $G(j\omega)$ и $G(-j\omega)$ — комплексно сопряжены, то фазу θ можно взять положительной,

$$m_\phi \triangleq \min_{\theta\geqslant 0}\{|\theta| \mid \exists\omega \in \mathbb{R},\ \exp(j\theta)G(j\omega) = -1\}. \tag{7.90}$$

При этом, нахождение m_ϕ сводится к решению задачи минимизации с ограничением

$$\begin{cases} m_\phi = \min\theta, \\ \text{при условии } (\theta \geqslant 0) \wedge (\exp(j\theta)G(j\omega) = -1). \end{cases} \tag{7.91}$$

Предположим теперь, что передаточная функция системы, которая охватывается обратной связью, имеет вид $G(s,\mathbf{p})$, где $\mathbf{p}$ — вектор параметров с неопределенностью. Предположим, что для любого вектора $\mathbf{p}$ из параллелотопа $[\mathbf{p}]$ замкнутая система устойчива, что можно сразу проверить с помощью аппарата, рассмотренного в параграфе 7.2.2 (с. 241). Понятия запасов устойчивости по амплитуде и фазе могут быть расширены в данной ситуации рассмотрением наихудшего случая, т. е. нахождением такого вектора $\mathbf{p}$ из параллелотопа $[\mathbf{p}]$, при котором оцениваемый запас — минимален. Это приводит к задаче нахождения *робастного запаса устойчивости по амплитуде* в виде

$$m_{\text{G}}([\mathbf{p}]) = \min_{\mathbf{p}\in[\mathbf{p}]} \min_{\substack{\text{Im}(G(j\omega,\mathbf{p}))=0 \\ \text{Re}(G(j\omega,\mathbf{p}))<0 \\ \omega\geqslant 0}} \left|20\log_{10}\frac{-1}{G(j\omega,\mathbf{p})}\right| \tag{7.92}$$

и к задаче нахождения *робастного запаса устойчивости по фазе* в виде

$$m_\phi([\mathbf{p}]) = \min_{\mathbf{p}\in[\mathbf{p}]} \min_{\substack{\theta\geqslant 0 \\ \exp(j\theta)G(j\omega,\mathbf{p})=-1}} \theta. \tag{7.93}$$

Эти величины могут быть рассчитаны с помощью алгоритма OPTIMIZE, описанного в Главе 5 (табл. 5.5, с. 157).

Пример 7.14. *Рассмотрим систему*

$$G(s,[\mathbf{p}]) = \frac{(2s+1)((1+p_1^2)s + \exp(-p_2))}{D(s,\mathbf{p})}, \tag{7.94}$$

где

$$D(s,\mathbf{p}) = s(s+5)\left(\frac{s}{1+p_2^2}+1+\cos^2 5p_1\right) * (s^2+\sqrt{p_2}(3+2\sin 3p_1)s+p_2^2-2p_2+2). \quad (7.95)$$

Для $[\mathbf{p}] = [-1,1]\times[0{,}3,1{,}5]$ *и* $\varepsilon_\mathrm{p} = \varepsilon_\mathrm{c} = 0{,}005$ *алгоритм* OPTIMIZE *находит, что*

$$\begin{aligned} 16{,}405\,\mathrm{dB} &\leqslant m_G([\mathbf{p}]) \leqslant 16{,}891\,\mathrm{dB}, \\ 1{,}537\,\mathrm{рад} &\leqslant m_\phi([\mathbf{p}]) \leqslant 1{,}544\,\mathrm{рад}. \end{aligned} \quad (7.96)$$

На компьютере Pentium 90 *вычисления занимают* 23,8 *секунды и* 1495 *итераций для построения* 56 *параллелотопов решения для нахождения запаса устойчивости по амплитуде и* 72,7 *секунды и* 6349 *итерации для построения* 1384 *параллелотопов решения для нахождения запаса устойчивости по фазе* [Didrit, 1997]. ■

7.4.5. Радиус устойчивости

Для нахождения запаса устойчивости системы $\Sigma(\mathbf{p})$ по отношению к неопределенности вектора параметров $\mathbf{p}$, окружим некоторое номинальное значение $\mathbf{p}^0$ вектора некоторой областью. Понятие *радиуса устойчивости* было независимо введено в [Safonov и Athans, 1981] и [Doyle, 1982].

Определение 7.1. *Радиусом устойчивости системы* $\Sigma(\mathbf{p})$ *в точке* $\mathbf{p}^0$ *назовем*

$$\begin{aligned} \rho &\triangleq \sup\{\eta \geqslant 0 \mid \Sigma(\mathbf{p}) \text{ устойчива для всех } \mathbf{p} \in [\mathbf{p}](\eta)\} \\ &= \min\{\eta \geqslant 0 \mid \Sigma(\mathbf{p}) \text{ неустойчива для одного } \mathbf{p} \in [\mathbf{p}](\eta)\}, \end{aligned} \quad (7.97)$$

где $[\mathbf{p}](\eta)$ *— параллелотоп с центром в точке* $\mathbf{p}^0$*, такой, что ширина его* j*-й компоненты удовлетворяет условию* $w([\underline{p}_j, \overline{p}_j) = 2\eta w_j$ *для некоторого наперед заданного положительного числа* w_j. ■

Таким образом, число η является радиусом гиперкуба $[\mathbf{p}](\eta)$ в пространстве с L_∞-нормой, взвешенной с коэффициентами w_j. Рис. 7.18 иллюстрирует это понятие для $\dim \mathbf{p} = 2$ и $w_1 = w_2 = 1$. Теперь, поскольку

$$\mathbf{p} \in [\mathbf{p}](\eta) \Leftrightarrow \forall j \in \{1,\ldots,n_\mathrm{p}\}, (p_j^0 - \eta\omega_j \leqslant p_j \leqslant p_j^0 + \eta\omega_j) \quad (7.98)$$

и вследствие эквивалентных соотношений

$$\begin{aligned} \Sigma(\mathbf{p}) \text{ неустойчива } &\Leftrightarrow \exists i, \text{ такое что } r_i(\mathbf{p}) \leqslant 0 \\ &\Leftrightarrow (r_1(\mathbf{p}) \leqslant 0) \vee \ldots \vee (r_n(\mathbf{p})) \leqslant 0), \end{aligned} \quad (7.99)$$

где символ $\vee$ соответствует булевскому оператору ИЛИ, радиус устойчивости может быть также определен как

$$\begin{cases} \rho = \min\limits_{\eta \geqslant 0} \eta, \\ \text{при условии } \begin{cases} ((r_1(\mathbf{p}) \leqslant 0) \vee \ldots \vee (r_n(\mathbf{p}) \leqslant 0)), \\ \wedge \left(\forall j \in \{1,\ldots,n_\mathrm{p}\}, \begin{cases} p_j^0 - \eta\omega_j - p_j \leqslant 0 \\ -p_j^0 - \eta\omega_j + p_j \leqslant 0 \end{cases}\right), \end{cases} \end{cases} \quad (7.100)$$

где символ $\wedge$ соответствует булевскому оператору И. Так как оператор $\vee$ включен в ограничения, алгоритм OPTIMIZE не может быть непосредственно применен.

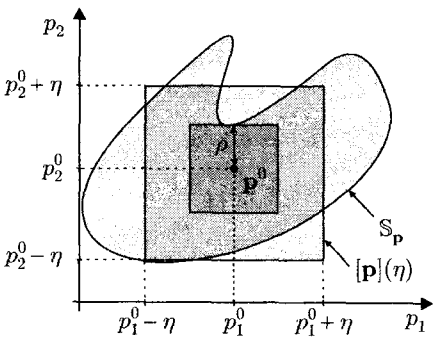


Рис. 7.18. Радиус устойчивости ρ в точке $\mathbf{p}^0$; $\mathbb{S}_\mathrm{p}$ — область устойчивости

Эквивалентным определением радиуса устойчивости ρ является

$$\rho = \min\{\rho_1, \ldots, \rho_n\}, \quad (7.101)$$

при

$$\begin{cases} \rho_i = \min\limits_{\eta \geqslant 0} \eta, \\ \text{при условии } \begin{cases} (r_i(\mathbf{p}) \leqslant 0), \\ \wedge \left(\forall j \in \{1,\ldots,n_\mathrm{p}\}, \begin{cases} p_j^0 - \eta\omega_j - p_j \leqslant 0 \\ -p_j^0 - \eta\omega_j + p_j \leqslant 0 \end{cases}\right). \end{cases} \end{cases} \quad (7.102)$$

Теперь уже компоненты ρ_i могут быть рассчитаны по алгоритму OPTIMIZE, так как соответствующие ограничения связаны с операторами $\wedge$. Радиус устойчивости тогда получается нахождением наименьшей величины из компонент ρ_i.

Число задач минимизации, которые надо решить, равно числу неравенств, проверяемых при анализе устойчивости. Если используется критерий Рауса, то это число равно порядку n исследуемой системы. Критерий Рауса–Гурвица может снизить это число до трех, как утверждается в следующей теореме.

Теорема 7.7. *Семейство полиномов*

$$P(s,[\mathbf{p}]) = a_n([\mathbf{p}])s^n + a_{n-1}([\mathbf{p}])s^{n-1} + \ldots + a_1([\mathbf{p}])s + a_0([\mathbf{p}]) \quad (7.103)$$

робастно устойчиво, если система $\Sigma(\mathrm{mid}([\mathbf{p}]))$ *устойчива и если для любого* $\mathbf{p}$ *из* $[\mathbf{p}]$

$$\begin{cases} q_1(\mathbf{p}) = a_0(\mathbf{p}) > 0, \\ q_2(\mathbf{p}) = a_n(\mathbf{p}) > 0, \\ q_3(\mathbf{p}) = D_{n-1}(\mathbf{p}) > 0, \end{cases} \quad (7.104)$$

где $D_{n-1}(\mathbf{p})$ *—* $(n-1)$*-й определитель Гурвица, связанный с полиномом* $P(s,\mathbf{p})$:

$$D_{n-1}(\mathbf{p}) = \begin{vmatrix} a_1(\mathbf{p}) & a_3(\mathbf{p}) & a_5(\mathbf{p}) & \ldots & a_{2n-1}(\mathbf{p}) \\ a_0(\mathbf{p}) & a_2(\mathbf{p}) & a_4(\mathbf{p}) & \ldots & a_{2n-2}(\mathbf{p}) \\ 0 & a_1(\mathbf{p}) & a_3(\mathbf{p}) & \ldots & a_{2n-3}(\mathbf{p}) \\ 0 & a_0 & a_2(\mathbf{p}) & \ldots & a_{2n-4}(\mathbf{p}) \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & a_{n-1}(\mathbf{p}) \end{vmatrix}. \quad (7.105)$$

■

Радиус устойчивости ρ удовлетворяет условиям

$$\begin{cases} \rho = \min\limits_{\eta \geqslant 0} \eta, \\ \text{при условии} \begin{cases} ((q_1(\mathbf{p}) \leqslant 0) \vee (q_2(\mathbf{p}) \leqslant 0) \vee (q_3(\mathbf{p}) \leqslant 0)), \\ \wedge \left(\forall j \begin{cases} p_j^0 - \eta\omega_j - p_j \leqslant 0 \\ -p_j^0 - \eta\omega_j + p_j \leqslant 0 \end{cases} \right), \end{cases} \end{cases} \quad (7.106)$$

или, в эквивалентном виде

$$\begin{cases} \rho = \min\{\rho_1, \rho_2, \rho_3\}, \\ \rho_i = \min\limits_{\eta \geqslant 0} \eta, \\ \text{при условии} \begin{cases} (q_i(\mathbf{p}) \leqslant 0), \\ \wedge \left(\forall j \begin{cases} p_j^0 - \eta\omega_j - p_j \leqslant 0 \\ -p_j^0 - \eta\omega_j + p_j \leqslant 0 \end{cases} \right). \end{cases} \end{cases} \quad (7.107)$$

Число выполняемых операций минимизации теперь равно трем вместо n.

Когда коэффициентная функция афинна, задача может быть решена использованием методов линейного программирования [Tesi и Vicino, 1989]. Когда это функция — полином, применялись методы, основанные на обобщенном геометрическом программировании (называемом также — *сигномиальное программирование*) [Vicino *et al.*, 1990]. Доказано [Braatz *et al.*, 1994], что задача вычисления радиуса устойчивости имеет полиномиальную сложность (NP-сложность).

Продемонстрируем на двух примерах эффективность интервальных методов расчета радиуса устойчивости.

Пример 7.15. *Полином*

$$P(s,\mathbf{p}) = s^3 + (p_1 + p_2 + 2)s^2 + (p_1 + p_2 + 2)s + 2p_1p_2 + 6p_1 + 6p_2 + 2 + \sigma^2$$

был рассмотрен в Примерах 7.3 (с. 252) *и* 7.5 (с. 256). *Когда параметры* p_1 *и* p_2 *положительны, этот полином устойчив для всех значений вектора параметров вне круга с центром* $\mathbf{p}^0 = (1,1)^{\mathrm{T}}$ *и радиусом* σ. *Основываясь на* (7.107) *используем оптимизационный алгоритм Хансена для вычисления радиуса устойчивости при различных значениях коэффициента* σ. *Как и в* [Murdock *et al.*, 1991; Psarris и Floudas, 1995; Malan *et al.*, 1997], *номинальное значение вектора параметров* $\mathbf{p}$ *берется* $\mathbf{p}^0 = (1{,}4,\ 0{,}85)^{\mathrm{T}}$, *а весовые коэффициенты заданы* $w_1 = 1{,}1$ *и* $w_2 = 0{,}85$. *Параллелотоп* $[\mathbf{p}](\eta)$ *определен как*

$$\begin{aligned} 1{,}4 - 1{,}1\eta \leqslant p_1 \leqslant 1{,}4 + 1{,}1\eta, \\ 0{,}85 - 0{,}85\eta \leqslant p_2 \leqslant 0{,}85 + 0{,}85\eta. \end{aligned} \quad (7.108)$$

Результаты, полученные на компьютере Pentium 90 *для разных значений коэффициента* σ, *приведены в* табл. 7.2. *Они подобны результатам, полученным в* [Malan *et al.*, 1997], *но наш подход может работать с непараметрической зависимостью общего вида, как показано в следующем примере.* ■

Таблица 7.2. Радиусы устойчивости при различных значениях коэффициента σ (Пример 7.15)

σ	10^{-1}	10^{-3}	10^{-5}	10^{-7}
ε_p и ε_c	10^{-5}	10^{-5}	10^{-5}	10^{-7}
Число итераций	66	113	55	63
Время счета	0,44	0,55	0,44	0,49
Число параллелотопов решения	1	5	1	2
Радиус устойчивости	0,2727	0,3627	0,3636	0,3636

Пример 7.16. *Рассмотрим полином*

$$P(s,\mathbf{p}) = s^3 + a_2(\mathbf{p})s^2 + a_1(\mathbf{p})s + a_0(\mathbf{p}) \tag{7.109}$$

с коэффициентами

$$\begin{aligned} a_0(\mathbf{p}) &= \sin(p_2)\exp(p_2) + p_1p_2 - 1,\\ a_1(\mathbf{p}) &= 2p_1 + 0{,}2p_1\exp(p_2),\\ a_2(\mathbf{p}) &= p_1 + p_2 + 4. \end{aligned} \tag{7.110}$$

Коэффициентная функция $\mathbf{a}(\mathbf{p})$ *не является ни линейной, ни полиномиальной. Вычисление радиуса устойчивости по* (7.107), *использующее оптимизационный алгоритм Хансена при* $\varepsilon_p = \varepsilon_c = 10^{-5}$, *дало радиус, приблизительно равный* 2,025 *в точке* $\mathbf{p}^0 = (-1{,}5,\ 1{,}5)^T$ [Didrit, 1997]. *Это согласуется с тем фактом, что полином* $P(s,\mathbf{p})$ *является устойчивым для любых векторов* $\mathbf{p}$ *из параллелотопа* $[\mathbf{p}] = [1,2]^{\times 2}$ [Amato *et al.*, 1995]. ■

7.5. Синтез регулятора

Хотя относительно мало статей посвящено синтезу регуляторов на основе интервального анализа [Kolev *et al.*, 1988; Kierfott, 1989б; Khlebalin, 1992; Kolev, 1993; Jaulin и Walter, 1996; Malan *et al.*, 1997], интерес к этому вопросу возрастает, как показал недавний специальный выпуск журнала *Reliable Computing* [Garloff и Walter, 2000]. В этом параграфе будет рассмотрено применение интервальных разрешающих операторов (описанных в Главе 5) к настройке параметров регулятора.

Рассмотрим линейную систему, которой надо управлять. Предположим сначала, что система полностью известна и что в ней нет параметров с неопределенностью. Вектор параметров $\mathbf{c}$ регулятора может быть выбран произвольно из некоторого параллелотопа $[\mathbf{c}]$. Обозначим через $\mathbf{r}(\mathbf{c},\delta)$

δ-функцию Рауса, связанную с этой управляемой системой. Регулятор, который максимизирует степень устойчивости, задается как

$$\mathbf{c} = \arg\max_{\mathbf{c}\in[\mathbf{c}]} \max_{\mathbf{r}(\mathbf{c},\delta)\geqslant\mathbf{0}} \delta. \tag{7.111}$$

Пример 7.17. *Снова рассмотрим систему* $\Sigma(\mathbf{p})$ *из* Примера 7.6 (с. 256). *Предположим теперь, что вектор* $\mathbf{p}$ *можно настроить так, чтобы максимизировать степень устойчивости, так как здесь* $\mathbf{p}$ *играет роль вектора* $\mathbf{c}$. *Характеристический полином системы* $\Sigma(\mathbf{p})$ *имеет вид*

$$P(s,\mathbf{p}) = s^3 + s^2 + (p_1^2 + p_2^2 + 1)s + 1, \tag{7.112}$$

где параметры p_1 *и* p_2 *можно выбрать произвольно из параллелотопа* $[\mathbf{p}] = [-7,\ 1{,}3]\times[-1,\ 2{,}5]$. *Напомним, что для любого вектора* $\mathbf{p}$ *система* $\Sigma(\mathbf{p})$ δ-*неустойчива при* $\delta \geqslant \frac{1}{3}$. *Для* $0 < \delta < \frac{1}{3}$ *полином* $P(s,\mathbf{p})$ *является* δ-*устойчивым для всех векторов* $\mathbf{p}$ *внутри области, расположенной между кругами с радиусами*

$$\underline{\sigma}(\delta) = \sqrt{\frac{4\delta(2\delta^2 - 2\delta + 1)}{-2\delta + 1}} \quad \text{и} \quad \overline{\sigma}(\delta) = \sqrt{\frac{-\delta^3 + \delta^2 - \delta + 1}{\delta}}. \tag{7.113}$$

Максимальная степень устойчивости получается, когда внутренний и внешний круги совпадают. Это происходит при $\delta = \frac{1}{3}$. *При этом* $\underline{\sigma}(\delta) = \overline{\sigma}(\delta) = \frac{2\sqrt{5}}{3}$. *Следовательно, имеется бесконечно много значений вектора* $\mathbf{p}$, *которые максимизируют степень устойчивости системы с обратной связью. Это происходит вследствие того факта, что система перепараметризована (содержит избыточные параметры). При* $\varepsilon_c = 0{,}005$ *и* $\varepsilon_\delta = 0{,}001$ *после* 588 *итераций за* 2,5 *секунд на компьютере* Pentium 90, *оптимизационный алгоритм Хансена дает* $\delta_M \in [0{,}3333,\ 0{,}3339]$. *Все точки максимумов находятся в покрытии, отмеченном черным цветом на* рис. 7.19. ■

Предположим теперь, что модель системы, которой надо управлять, зависит от вектора $\mathbf{p}$ параметров с неопределенностью. Будем работать с двумя типами параметров: собственно параметры $\mathbf{p}$ и настраиваемые параметры $\mathbf{c}$ регулятора.

Рассмотрим систему с обратной связью $\Sigma(\mathbf{p},\mathbf{c})$, прямая цепь которой состоит из регулятора $C(s,\mathbf{c})$ и последовательно соединенного с ним звена $G(s,\mathbf{p})$, которое является параметрической моделью системы с неопределенностью, и $\mathbf{p} \in [\mathbf{p}]$ (см. рис. 7.20). Исследуемая задача состоит в вычислении множества $\mathbb{S}_c$ векторов $\mathbf{c}$, которые максимизируют степень устойчивости в наихудшем случае.

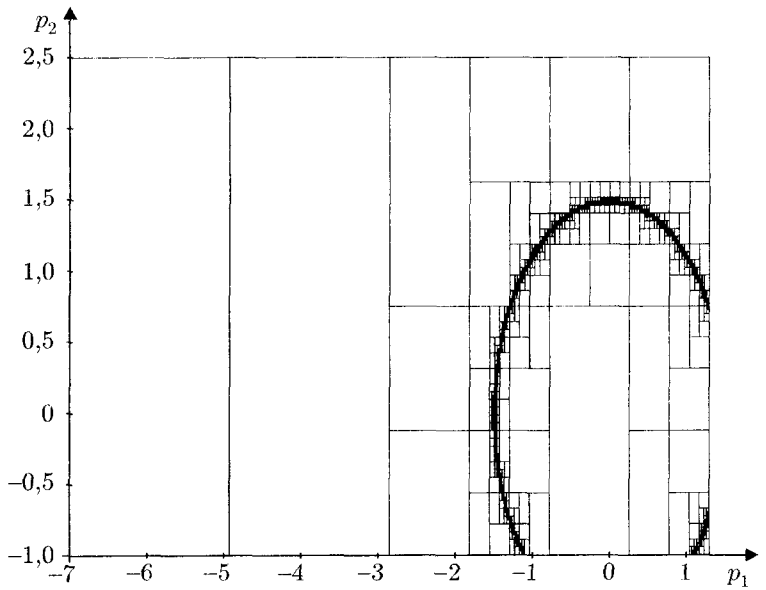


Рис. 7.19. Покрытие, выработанное для описания множества всех точек максимума степени устойчивости Примера 7.17; все точки максимума находятся в параллелотопах, отмеченных черным цветом

Это множество удовлетворяет условию

$$\mathbb{S}_c = \arg \max_{\mathbf{c}\in[\mathbf{c}]} \Big(\min_{\mathbf{p}\in[\mathbf{p}]} \max_{\mathbf{r}(\mathbf{p},\mathbf{c},\delta)\geqslant\mathbf{0}} \delta \Big). \tag{7.114}$$

Здесь, крайняя правая операция «max» соответствует нахождению степени устойчивости. Операция «min» обеспечивает условия наихудшего случая. Левая операция «max» соответствует выполнению требования оптимальности. Если в регуляторе выбрать некоторый элемент $\mathbf{c} \in \mathbb{S}_c$, то можно быть уверенным, что степень устойчивости управляемой системы по меньшей мере равна

$$\delta_M^c = \min_{\mathbf{p}\in[\mathbf{p}]} \max_{\mathbf{r}(\mathbf{p},\mathbf{c},\delta)\geqslant\mathbf{0}} \delta, \tag{7.115}$$

и δ_M^c — *оптимальная степень робастной устойчивости.*

Пример 7.18. [Jaulin и Walter, 1996; Didrit, 1997] *Для системы с обратной связью* $\Sigma(\mathbf{p}, \mathbf{c})$, *показанной на* рис. 7.20, *алгоритм* MINIMAX *дает результаты, представленные в* табл. 7.3. *В данной таблице* $\sharp\mathbb{S}_c$ — *число параллелотопов в покрытии* $\overline{\mathbb{S}}_c$, *содержащих все значения вектора* $\mathbf{c}$, *соответствующие глобально оптимальным робастным регуляторам,* $[\overline{\mathbb{S}}_c]$ — *интервальная оболочка множества* $\mathbb{S}_c$, *а* $[\delta_M^c]$ — *интервал, гарантированно содержащий соответствующее оптимальное значение робастной устойчивости. Время расчетов показано для компьютера* Pentium 90, *а порядок величины точности* $\varepsilon \sim 10^{-3}$. ■

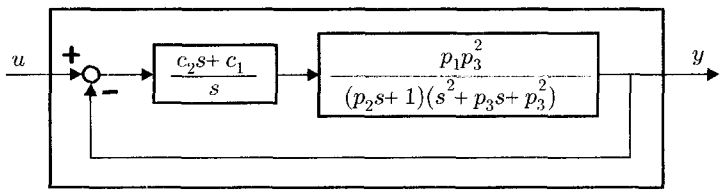


Рис. 7.20. Система с неопределенностью с ПИ-регулятором

Таблица 7.3. Результаты, полученные алгоритмом MINIMAX для оптимального робастного регулятора

$[\mathbf{p}]$	Время счета, с	$\#\mathbb{S}_c$	$[\overline{\mathbb{S}}_c]$	$[\delta_M^c]$
$(1,1,1)^{\mathrm{T}}$	5,5	66	$[0{,}257, 0{,}273] \times [0{,}305, 0{,}354]$	$[0{,}300, 0{,}326]$
$[0{,}99, 1{,}01]^{\times 3}$	85	69	$[0{,}239, 0{,}274] \times [0{,}264, 0{,}382]$	$[0{,}288, 0{,}299]$
$[-0{,}95, 1{,}05]^{\times 3}$	339	37	$[0{,}207, 0{,}277] \times [0{,}179, 0{,}437]$	$[0{,}261, 0{,}282]$
$[0{,}9, 1{,}1]^{\times 3}$	345	17	$[0{,}207, 0{,}254] \times [0{,}191, 0{,}367]$	$[0{,}230, 0{,}246]$

ЗАМЕЧАНИЕ 7.4. На практике часто достаточно найти только один вектор $\mathbf{c}$, для которого степень робастной устойчивости выше, чем некоторый предписанный уровень $\underline{\delta}$. Тогда задача может быть сформулирована следующим образом:

$$\text{найти один вектор } \mathbf{c} \in [\mathbf{c}] \mid \forall \mathbf{p} \in [\mathbf{p}],\ \mathbf{r}(\mathbf{p}, \mathbf{c}, \underline{\delta}) > \mathbf{0}. \tag{7.116}$$

Эта задача значительно проще, чем поиск множеств всех оптимальных робастных регуляторов, и алгоритм OPTIMIZE может быть адаптирован для решения этой задачи более эффективным образом [Jaulin и Walter, 1996]. ■

7.6. Выводы

Робастное управление дает целый пласт возможностей для применения интервального анализа. Почти каждая интересующая проблема из этой области может быть связана с обращением множеств, минимаксной оптимизацией или с оптимизацией при наличии ограничений. Мы надеемся, что примеры, рассмотренные в данной главе, убедили читателя, что интервальный анализ обладает достаточными средствами для получения соответствующих решений.

Разумеется, интервальный анализ в его нынешнем состоянии не может справиться со всеми задачами робастного управления, хотя бы уже вследствие проклятья размерностей. Но есть, однако, четыре причины, которые делают разрешимыми практически интересные задачи. Во-первых, в большинстве случаев интересуются управляющими устройствами с малым числом настраиваемых параметров (примерно тремя в широко распространенном ПИД-регуляторе). Во-вторых, параметрическая неопределенность модели процесса, которым необходимо управлять, может часто быть ограничена несколькими существенными, доминирующими факторами, часто связанными с параметрами, имеющими физический смысл. В-третьих, обычно легко выразить передаточную матрицу разомкнутой или замкнутой системы, которой надо управлять, в виде явной функции неопределенных и настраиваемых параметров, что упрощает построение эффективных функций включения. Наконец, что в общем случае действительно необходимо знать, так это — вектор удовлетворительно настраиваемых параметров, а не некоторое описание множества всех таких векторов.

Оценивание того, насколько сложность вычислений может затруднить расчет реального устройства управления, является захватывающей задачей, решение которой может потребовать совместных усилий специалистов по управлению и по интервальному анализу.

ГЛАВА 8
Робототехника

8.1. Введение

Под роботами понимаются механические системы, управляемые для достижения определенных целей, которые представляются слишком однообразными, слишком опасными или слишком трудными для человека. Как результат, робототехника является обширной междисциплинарной областью, которая основана на математике, механике, теории управления, искусственном интеллекте, эргономике и т. д. Эта глава, разумеется, не может претендовать на исчерпывающую полноту, и мы ограничиваемся иллюстрацией того, что интервальный анализ может дать в решении трех сложных задач.

Первая их них, представленная в параграфе 8.2, является задачей оценивания всех возможных конфигураций параллельного робота, известного как платформа Стюарта–Гофа, при заданной длине его звеньев. Эта задача стала классическим эталонным тестом для компьютерной алгебры, поскольку она включает в себя решение очень сложного набора нелинейных уравнений [Raghavan и Roth, 1995]. Мы покажем, что интервальный анализ делает возможным работать с этим типом задач на персональном компьютере даже в самом сложном и наиболее общем случае, и отметим преимущества получаемого решения по сравнению с решениями, базирующимися на работе с классическими символьными операциями.

Вторая задача, описанная в параграфе 8.3, относится к планированию свободного от столкновений маршрута движения жесткого объекта в известной среде. Эта задача решается путем объединения приемов интервального анализа и теории графов. Задача иллюстрируется примером на плоскости, когда объект является невыпуклым многоугольником, а препятствия содержат линейные отрезки.

Последняя задача, рассмотренная в параграфе 8.4, связана с определением местоположения робота и слежением за его движением в частично известной среде по информации от его бортовых дальномеров. Мы увидим, как эта задача может быть решена в рамках задачи оценивания параметров

и состояния при наличии ограничений, которая была описана в Главе 6. Показывается также, как можно учесть выход из строя дальномера, устаревшей карты и неоднозначности вследствие симметрии в окружающей среде.

8.2. Расширенная кинематическая задача для платформ Стюарта–Гофа

8.2.1. Платформы Стюарта–Гофа

Платформа Стюарта–Гофа состоит из жесткой подвижной платформы, связанной с жестким неподвижным основанием шестью прямолинейными звеньями, длиной которых можно управлять (см. рис. 8.1). Поскольку звенья одновременно влияют на положение и угловую ориентацию подвижной платформы по отношению к основанию, платформа является примером, известным как параллельный робот (в противоположность артикулируемой руке, где исполнительные органы присоединены к перемещаемым звеньям последовательно). Этот механизм был предложен Гофом в 1949 году для ложа, используемого в исследовании усталости [Gough, 1956], и использовался в [Stewart, 1965] для проектирования полетного имитатора. Платформы Стюарта–Гофа в настоящее время применяются во многих других случаях в задачах, где необходимы нужное усилие и точность [Merlet, 1990].

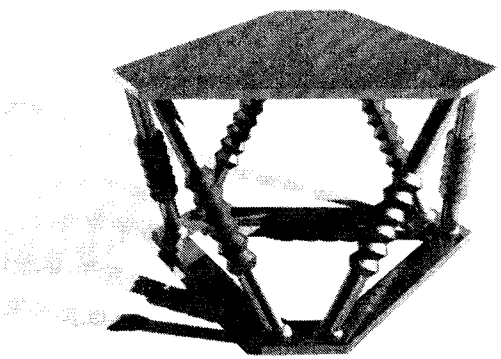


Рис. 8.1. Условное изображение платформы Стюарта–Гофа

Пусть $\mathbf{a}(i)$ и $\mathbf{b}(i)$ — крайние точки креплений i-го звена, присоединенного к основанию и к подвижной платформе, соответственно. *Расширенная кинематическая задача* состоит в расчете *всех* возможных конфигураций (состояний) платформы, описываемых: 1) координатами точек $\mathbf{a}(i)$ в системе координат основания; 2) координатами точек $\mathbf{b}(i)$ в системе координат платформы; 3) длинами $\mathbf{y}_i$ звеньев. Решение этой проблемы настолько сложно, что она превратилась в классический тестовый пример для символьных и численных вычислений [Nanua *et al.*, 1990; Lazard, 1992; Mourrain, 1993; Wang и Chen, 1993].

В настоящее время существует несколько методов для сведения этой задачи к решению некоторой системы нелинейных уравнений. Подход, основанный на использовании Эйлеровых углов, описывается в параграфах 8.2.2 и 8.2.3. Решение результирующих уравнений с использованием интервальных разрешающих операторов рассматривается в параграфе 8.2.4. Более детальное описание решения этой задачи можно найти в [Didrit *et al.*, 1998].

8.2.2. Переход от системы координат подвижной платформы к системе координат основания

Пусть $\mathcal{R}_0$ — система координат, связанная с основанием, а $\mathcal{R}_4$ — система координат, связанная с подвижной платформой (смысл этих обозначений поясняется чуть ниже). Конфигурация (положение и ориентация) подвижной платформы относительно основания может быть описана координатами $(c_{x0}, c_{y0}, c_{z0},)$ некоторой точки подвижной платформы в системе $\mathcal{R}_0$ и тремя Эйлеровыми углами ψ, θ и φ, как показано на рис. 8.2.

Координаты $\mathbf{a}(i)$ известны в системе $\mathcal{R}_0$, а координаты $\mathbf{b}(i)$ известны в системе $\mathcal{R}_4$. Эти координаты не зависят от положения платформы, но их преобразование из системы $\mathcal{R}_0$ в систему $\mathcal{R}_4$ зависит от ее положения. Для вычисления длин звеньев, как функций от положения платформы, нужно выразить координаты $\mathbf{a}(i)$ и $\mathbf{b}(i)$ в одной системе координат, скажем, $\mathcal{R}_0$. Таким образом, надо иметь формулу преобразования для вычисления координат $\mathbf{b}(i)$ в системе $\mathcal{R}_0$ по их значениям в системе $\mathcal{R}_4$. Данный параграф посвящен построению такого преобразования.

Пусть $\mathcal{R}_1$ — система координат, полученная после поворота системы $\mathcal{R}_0$ относительно оси $\mathbf{k}_0$ на угол ψ (см. рис. 8.2). Базис векторов системы $\mathcal{R}_1$ может быть выражен по отношению к базису системы $\mathcal{R}_0$ следующим образом:

$$\begin{aligned} \mathbf{i}_1 &= \cos\psi\mathbf{i}_0 + \sin\psi\mathbf{j}_0, \\ \mathbf{j}_1 &= -\sin\psi\mathbf{i}_0 + \cos\psi\mathbf{j}_0, \\ \mathbf{k}_1 &= \mathbf{k}_0, \end{aligned} \tag{8.1}$$

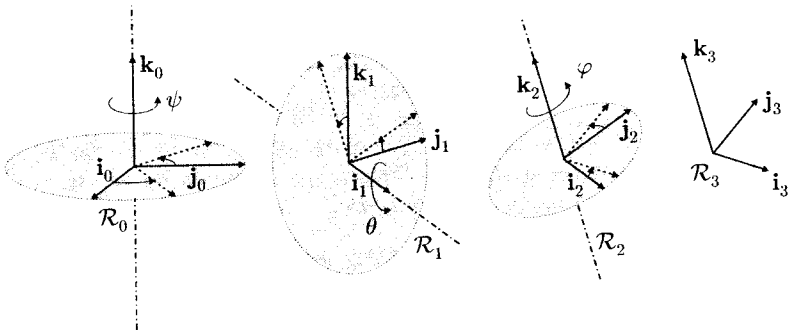


Рис. 8.2. Преобразование системы координат с использованием Эйлеровых углов ψ, θ и φ

или, в матричной форме,

$$\begin{pmatrix}\mathbf{i}_1\\ \mathbf{j}_1\\ \mathbf{k}_1\end{pmatrix} = \mathbf{P}_1 \begin{pmatrix}\mathbf{i}_0\\ \mathbf{j}_0\\ \mathbf{k}_0\end{pmatrix}, \quad \text{где } \mathbf{P}_1 = \begin{pmatrix}\cos\psi & \sin\psi & 0\\ -\sin\psi & \cos\psi & 0\\ 0 & 0 & 1\end{pmatrix}. \tag{8.2}$$

Пусть $\mathcal{R}_2$ — система координат, полученная поворотом системы $\mathcal{R}_1$ относительно оси $\mathbf{i}_1$ на угол θ. Базис векторов системы $\mathcal{R}_2$ удовлетворяет соотношениям

$$\begin{pmatrix}\mathbf{i}_2\\ \mathbf{j}_2\\ \mathbf{k}_2\end{pmatrix} = \mathbf{P}_2 \begin{pmatrix}\mathbf{i}_1\\ \mathbf{j}_1\\ \mathbf{k}_1\end{pmatrix}, \quad \text{где } \mathbf{P}_2 = \begin{pmatrix}1 & 0 & 0\\ 0 & \cos\theta & \sin\theta\\ 0 & -\sin\theta & \cos\theta\end{pmatrix}. \tag{8.3}$$

Для системы $\mathcal{R}_3$, полученной поворотом системы $\mathcal{R}_2$ относительно оси $\mathbf{k}_2$ на угол φ, имеем

$$\begin{pmatrix}\mathbf{i}_3\\ \mathbf{j}_3\\ \mathbf{k}_3\end{pmatrix} = \mathbf{P}_3 \begin{pmatrix}\mathbf{i}_2\\ \mathbf{j}_2\\ \mathbf{k}_2\end{pmatrix}, \quad \text{где } \mathbf{P}_3 = \begin{pmatrix}\cos\varphi & \sin\varphi & 0\\ -\sin\varphi & \cos\varphi & 0\\ 0 & 0 & 1\end{pmatrix}. \tag{8.4}$$

Объединение (8.2), (8.3) и (8.4) дает

$$\begin{pmatrix}\mathbf{i}_3\\ \mathbf{j}_3\\ \mathbf{k}_3\end{pmatrix} = \mathbf{P}_3\mathbf{P}_2\mathbf{P}_1 \begin{pmatrix}\mathbf{i}_0\\ \mathbf{j}_0\\ \mathbf{k}_0\end{pmatrix}. \tag{8.5}$$

Рассмотрим теперь некоторый вектор $\mathbf{v}$ с координатами (x_0, y_0, z_0) в системе $\mathcal{R}_0$ и координатами (x_3, y_3, z_3) в системе $\mathcal{R}_3$. Имеем

$$\mathbf{v} = x_0\mathbf{i}_0 + y_0\mathbf{j}_0 + z_0\mathbf{k}_0 = x_3\mathbf{i}_3 + y_3\mathbf{j}_3 + z_3\mathbf{k}_3, \tag{8.6}$$

или, в векторной форме,

$$(x_0, y_0, z_0) \begin{pmatrix}\mathbf{i}_0\\ \mathbf{j}_0\\ \mathbf{k}_0\end{pmatrix} = (x_3, y_3, z_3) \begin{pmatrix}\mathbf{i}_3\\ \mathbf{j}_3\\ \mathbf{k}_3\end{pmatrix}. \tag{8.7}$$

С учетом (8.5) это соотношение дает

$$(x_0, y_0, z_0) \begin{pmatrix}\mathbf{i}_0\\ \mathbf{j}_0\\ \mathbf{k}_0\end{pmatrix} = (x_3, y_3, z_3)\mathbf{P}_3\mathbf{P}_2\mathbf{P}_1 \begin{pmatrix}\mathbf{i}_0\\ \mathbf{j}_0\\ \mathbf{k}_0\end{pmatrix}. \tag{8.8}$$

Поскольку $\mathbf{i}_0, \mathbf{j}_0, \mathbf{k}_0$ — базис в $\mathbb{R}^3$, то (8.8) эквивалентно выражению

$$(x_0, y_0, z_0) = (x_3, y_3, z_3)\mathbf{P}_3\mathbf{P}_2\mathbf{P}_1, \tag{8.9}$$

или

$$\begin{pmatrix}x_0\\ y_0\\ z_0\end{pmatrix} = \mathbf{P}^{\mathrm{T}} \begin{pmatrix}x_3\\ y_3\\ z_3\end{pmatrix}, \tag{8.10}$$

где

$$\mathbf{P}^{\mathrm{T}} = \mathbf{P}_1^{\mathrm{T}}\mathbf{P}_2^{\mathrm{T}}\mathbf{P}_3^{\mathrm{T}}, \tag{8.11}$$

а матрицы угловых преобразований $\mathbf{P}_1$, $\mathbf{P}_2$ и $\mathbf{P}_3$ определяются выражениями (8.2)–(8.4). Наконец, пусть система координат $\mathcal{R}_4$ получается преобразованием (смещением) системы $\mathcal{R}_3$ на вектор $\mathbf{c}$ с координатами c_{x0}, c_{y0} и c_{z0} в системе $\mathcal{R}_0$. Тогда координаты m_{x0}, m_{y0} и m_{z0} некоторой точки $\mathbf{m}$ в системе $\mathcal{R}_0$ могут быть получены по ее координатам m_{x4}, m_{y4} и m_{z4} в системе $\mathcal{R}_4$ следующим образом:

$$\begin{pmatrix}m_{x0}\\ m_{y0}\\ m_{z0}\end{pmatrix} = \begin{pmatrix}c_{x0}\\ c_{y0}\\ c_{z0}\end{pmatrix} + \mathbf{P}^{\mathrm{T}} \begin{pmatrix}m_{x4}\\ m_{y4}\\ m_{z4}\end{pmatrix}. \tag{8.12}$$

8.2.3. Решаемые уравнения

Длина y_i i-го звена платформы является Эвклидовым расстоянием между точками $\mathbf{a}(i)$ и $\mathbf{b}(i)$. Теперь, координаты точек $\mathbf{a}_i$ задаются в системе $\mathcal{R}_0$,

а точек $\mathbf{b}_i$ — в системе $\mathcal{R}_4$. Чтобы вычислить это расстояние, координаты этих точек должны быть выражены в одной системе координат, например, в $\mathcal{R}_0$. Координаты точек $\mathbf{b}_i$ в системе $\mathcal{R}_0$ вычисляются по (8.12). Тогда длина y_i получается как

$$y_i = \|\mathbf{a}(i) - \mathbf{b}(i)\|_2 = \sqrt{(a_{x0}(i) - b_{x0}(i))^2 + (a_{y0}(i) - b_{y0}(i))^2 + (a_{z0}(i) - b_{z0}(i))^2}. \quad (8.13)$$

Процедура расчета длин шести звеньев как функций от положения $\mathbf{x} = (c_{x0}, c_{y0}, c_{z0}, \psi, \theta, \varphi)^{\mathrm{T}}$ платформы описывается в табл. 8.1. Коэффициенты r_{ij} соответствуют элементам матрицы $\mathbf{P}^{\mathrm{T}}$ в (8.11).

Таким образом, отыскание всех возможных положений платформы Стюарта–Гофа по известным длинам ее звеньев сводится к решению векторного уравнения

$$\mathbf{f}(\mathbf{x}) - \mathbf{y} = \mathbf{0} \quad (8.14)$$

относительно $\mathbf{x}$, где $\mathbf{x}$ — вектор положения, $\mathbf{f}(\mathbf{x})$ — вектор длин звеньев платформы, рассчитываемых по алгоритму табл. 8.1, а $\mathbf{y}$ — вектор, содержащий истинные численные величины длин y_i, $i = 1, \ldots, 6$ звеньев. Интервальные разрешающие операторы, описанные в Главе 5, могут быть использованы для решения этой задачи, как показывается в следующем параграфе.

8.2.4. Решение

В общем случае, уравнение (8.14) не допускает аналитического решения. Было показано [Lazard, 1993; Mourrain, 1993; Raghavan, 1993; Wampler, 1996], что, в общем случае, существует по крайней мере 40 комплексных решений, некоторые из которых могут быть связаны с конкретными конфигурациями [Lee и Roth, 1993; Faugère и Lazard, 1995]. Разумеется, на практике интерес представляют только *вещественные* решения.

Поскольку (8.14) включает в себя тригонометрические функции, методы формального исключения для решения систем полиномиальных уравнений, т. е. методов, основанных на построении базиса Грёбнера (Gröbner), как в [Lazard, 1992; 1993], не могут быть прямо применены. Однако уравнение (8.14) можно преобразовать к системе девяти полиномиальных уравнений за счет увеличения числа неизвестных с шести до девяти. Для платформ специальных типов задача может быть упрощена. Например, когда точки $\mathbf{a}(i)$ компланарны, точки $\mathbf{b}(1) = \mathbf{b}(2)$, $\mathbf{b}(3) = \mathbf{b}(4)$, $\mathbf{b}(5) = \mathbf{b}(6)$ и $(\mathbf{b}(1), \mathbf{b}(3), \mathbf{b}(5))$ образуют равнобедренный треугольник, оказывается возможным свести задачу к задаче решения единственного полиномиального уравнения степени 16 с одной неопределенностью [Merlet, 1990; Nauma

Таблица 8.1. Алгоритм вычисления вектора $\mathbf{y}_{\mathrm{m}}$ длины звеньев при заданном положении платформы

Алгоритм SGSIMULATION(вход : $c_{x0}, c_{y0}, c_{z0}, \psi, \theta, \varphi$; выход : $\mathbf{y}_{\mathrm{m}}$)

$r_{11} = \cos\psi\cos\varphi - \sin\psi\cos\theta\sin\varphi$;
$r_{12} = -\cos\psi\sin\varphi - \sin\psi\cos\theta\cos\varphi$;
$r_{13} = \sin\psi\sin\theta$;
$r_{21} = \sin\psi\cos\varphi + \cos\psi\cos\theta\sin\varphi$;
$r_{22} = -\sin\psi\sin\varphi + \cos\psi\cos\theta\cos\varphi$;
$r_{23} = -\cos\psi\sin\theta$;
$r_{31} = \sin\theta\sin\varphi$;
$r_{32} = \sin\theta\cos\varphi$;
$r_{33} = \cos\theta$;
для $i = 1$ до 6
  $b_{x0}(i) = c_{x0} + r_{11}b_{x4}(i) + r_{12}b_{y4}(i) + r_{13}b_{z4}(i)$;
  $b_{y0}(i) = c_{y0} + r_{21}b_{x4}(i) + r_{22}b_{y4}(i) + r_{23}b_{z4}(i)$;
  $b_{z0}(i) = c_{z0} + r_{31}b_{x4}(i) + r_{32}b_{y4}(i) + r_{33}b_{z4}(i)$;
  $y_i = \sqrt{(a_{x0}(i) - b_{x0}(i))^2 + (a_{y0}(i) - b_{y0}(i))^2 + (a_{z0}(i) - b_{z0}(i))^2}$;
$\mathbf{y}_{\mathrm{m}} = (y_1, \ldots, y_6)^{\mathrm{T}}$.

et al., 1990; Innocenti и Parenti-Castelli, 1991]. В [Husty, 1996] показано, как выполнять подобное преобразование в общем некомпланарном случае, но это преобразование включает в себя сложные алгебраические действия, что доступно на практике только когда геометрические параметры, описывающие задачу, принимают малые целые значения. Многие формальные методы, основанные на теории исключения, страдают по такой же причине. Кроме того, эти методы должны использовать численные алгоритмы для поиска решений полиномов высокой степени, которые могут вырождаться, и точность численных результатов может оказаться сомнительной, если не применять методы гарантированного решения, основанные на интервальном анализе. В противоположность этому подход, рассматриваемый в данном параграфе, в состоянии изолировать все вещественные решения уравнения (8.14) даже в общем некомпланарном случае с практическими значениями коэффициентов в геометрическом описании платформы.

Перед началом работы разрешающего оператора нужно определить априорный параллелотоп, гарантированно содержащий все интересующие нас решения. Примитивная начальная область

$$[\mathbf{x}](0) = \mathbb{R} \times \mathbb{R} \times \mathbb{R} \times [-\pi, \pi] \times [-\pi, \pi] \times [-\pi, \pi] \quad (8.15)$$

может быть уменьшена с учетом геометрии задачи. Поскольку

$$\mathbf{P}^{\mathrm{T}}(\psi+\pi,-\theta,\varphi+\pi)=\mathbf{P}^{\mathrm{T}}(\psi,\theta,\varphi), \tag{8.16}$$

где матрица $\mathbf{P}^{\mathrm{T}}$ задается выражением (8.11)[1], то положение платформы для углов $\psi+\pi,-\theta,\varphi+\pi$ — такое же, как для углов $\psi,-\theta,\varphi$. Следовательно, априорная область может быть уменьшена до вида

$$[\mathbf{x}](0)=\mathbb{R}\times\mathbb{R}\times\mathbb{R}\times[-\pi,\,\pi]\times[0,\,\pi]\times[-\pi,\,\pi]. \tag{8.17}$$

Чтобы ограничить области значений первых трех компонент вектора $\mathbf{x}$, заметим, что

$$\mathbf{c}=\mathbf{a}(i)+(\mathbf{b}(i)-\mathbf{a}(i))+(\mathbf{c}-\mathbf{b}(i)). \tag{8.18}$$

В системе координат $\mathcal{R}_0$ это уравнение принимает вид:

$$\begin{pmatrix}c_{x0}\\c_{y0}\\c_{z0}\end{pmatrix}=\begin{pmatrix}a_{x0}(i)\\a_{y0}(i)\\a_{z0}(i)\end{pmatrix}+(\mathbf{b}(i)-\mathbf{a}(i))_{\mathcal{R}_0}+(\mathbf{c}-\mathbf{b}(i))_{\mathcal{R}_0}. \tag{8.19}$$

Теперь, поскольку y_i — расстояние между точками $\mathbf{a}(i)$ и $\mathbf{b}(i)$, то каждая компонента $(\mathbf{b}(i)-\mathbf{a}(i))_{\mathcal{R}_0}$ принадлежит интервалу $[-y_i,y_i]$, и поскольку расстояние $d(\mathbf{c},\mathbf{b}(i))$ между точками $\mathbf{c}$ и $\mathbf{b}(i)$ известно, то каждая компонента $(\mathbf{c}-\mathbf{b}(i))_{\mathcal{R}_0}$ принадлежит интервалу $[-d(\mathbf{c},\mathbf{b}(i)),d(\mathbf{c},\mathbf{b}(i))]$. Таким образом, для любого i

$$\begin{pmatrix}c_{x0}\\c_{y0}\\c_{z0}\end{pmatrix}\in\begin{pmatrix}a_{x0}(i)\\a_{y0}(i)\\a_{z0}(i)\end{pmatrix}+\begin{pmatrix}[-y_i,y_i]\\[-y_i,y_i]\\[-y_i,y_i]\end{pmatrix}+\begin{pmatrix}[-d(\mathbf{c},\mathbf{b}(i)),\,d(\mathbf{c},\mathbf{b}(i))]\\[-d(\mathbf{c},\mathbf{b}(i)),\,d(\mathbf{c},\mathbf{b}(i))]\\[-d(\mathbf{c},\mathbf{b}(i)),\,d(\mathbf{c},\mathbf{b}(i))]\end{pmatrix}, \tag{8.20}$$

и вектор $\mathbf{c}$, выраженный в системе $\mathcal{R}_0$, принадлежит параллелотопу

$$[\mathbf{c}](0)=\bigcap_{i\in\{1,\ldots,6\}}\begin{pmatrix}a_{x0}(i)+[-y_i,y_i]+[-d(\mathbf{c},\mathbf{b}(i)),d(\mathbf{c},\mathbf{b}(i))]\\a_{y0}(i)+[-y_i,y_i]+[-d(\mathbf{c},\mathbf{b}(i)),d(\mathbf{c},\mathbf{b}(i))]\\a_{z0}(i)+[-y_i,y_i]+[-d(\mathbf{c},\mathbf{b}(i)),d(\mathbf{c},\mathbf{b}(i))]\end{pmatrix}. \tag{8.21}$$

ЗАМЕЧАНИЕ 8.1. Если и основание, и подвижная платформа — плоские (что часто имеет место на практике) и если $(c_{x0},c_{y0},c_{z0},\psi,\theta,\varphi)$ есть некоторое решение, то положение $(c_{x0},c_{y0},c_{z0},\psi+\pi,\theta,\varphi+\pi)$, симметричное относительно основания, также является решением. В таком случае область поиска может быть ограничена до положительных значений координат c_{z0}. К такому же выводу привело бы и априорное знание того, что подвижная платформа всегда находится над основанием. ■

[1]Для проверки (8.16) достаточно проверить его для всех элементов матриц. Например, для элемента, связанного с первой строкой и с первым столбцом, имеем $\cos(\psi+\pi)\cos(\varphi+\pi)-\sin(\psi+\pi)\cos(-\theta)\sin(\varphi+\pi)=(-\cos\psi)(-\cos\varphi)-(-\sin\psi)=\cos\psi\cos\varphi-\sin\psi\cos\theta\sin\varphi$.

Таблица 8.2. Данные для Примера 8.1

i	$a_{x0}(i)$	$a_{y0}(i)$	$a_{z0}(i)$	$b_{x4}(i)$	$b_{y4}(i)$	$b_{z4}(i)$	$(y_i)^2$
1	−3	2	0	−1	1	0	22
2	3	2	0	1	1	0	31
3	4	0	0	2	−1	0	39
4	1	−3	0	1	−2	0	29
5	−1	−3	0	−1	−2	0	22
6	−4	1	0	−2	−1	0	22

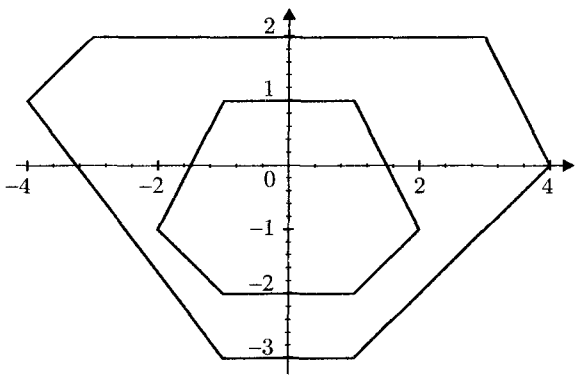


Рис. 8.3. Формы и размеры основания и подвижной платформы (Пример 8.1)

Пример 8.1 (плоский случай). *Основание и подвижная платформа являются плоскими, как задано в* табл. 8.2. *Их формы и размеры показаны на* рис. 8.3. *Используя* (8.17), (8.21) *и* Замечание 8.1, *получаем параллелотоп поиска в виде*

$$[-7{,}93,3{,}99]\times[-4{,}99,8{,}25]\times[0,6{,}25]\times[-\pi,\pi]\times[0,\pi]\times[-\pi,\pi].$$

Для $\varepsilon_{\mathrm{x}}=\varepsilon_{\mathrm{f}}=10^{-5}$, *после* 286345 *итераций, выполненных за* 92 *минуты на компьютере* Pentium 90, *алгоритм Хансена для решения уравнений* [Hansen, 1992б] *выработал четыре параллелотопа, каждый из которых удовлетворяет условию единственности* (с. 162). *Этот алгоритм подобен общему*

разрешающему алгоритму SIVIAX параграф 5.2, (с. 139) *со сжимающим оператором, описанном в* параграфе 5.5.2 (с. 159). *Аппроксимации этих четырех решений даются в* табл. 8.4. Рис. 8.4 *показывает соответствующие положения.* ■

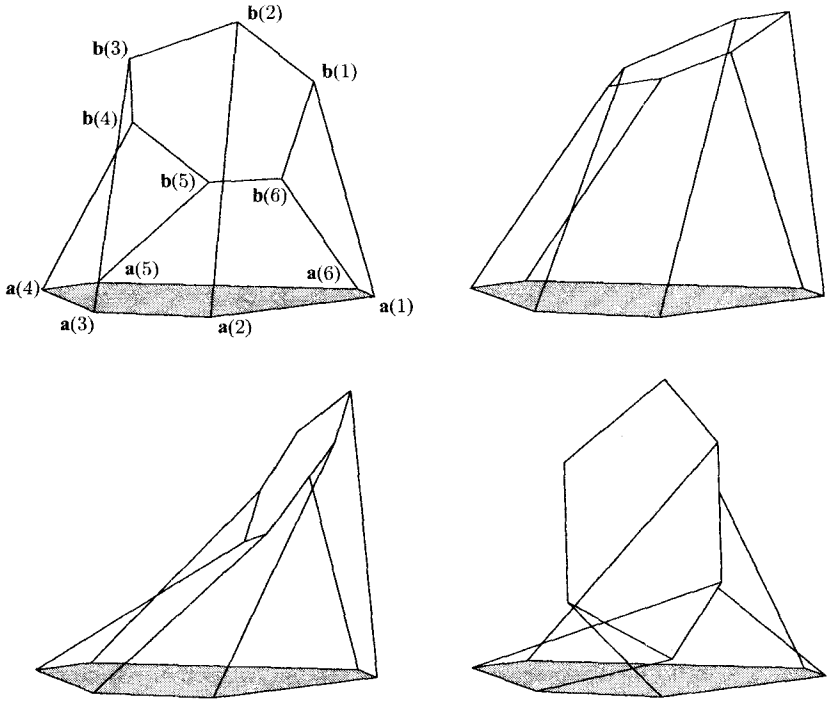


Рис. 8.4. Четыре конфигурации платформы по Примеру 8.1

Пример 8.2 (неплоский случай). *Основание и подвижная платформа не являются плоскими, как показано в* табл. 8.4. *Снова по соотношениям* (8.17) *и* (8.21) *априорный параллелотоп принимает вид*

$$[-30{,}37, 18{,}92] \times [-19{,}52, 33{,}1] \times [25, 23] \times [-\pi, \pi] \times [0, \pi] \times [\pi, \pi].$$

Для $\varepsilon_x = \varepsilon_f = 10^{-5}$, *после* 481800 *итераций, выполненных за* 153 *минуты на компьютере* Pentium 90, *алгоритм Хансена для решения уравнений*

Таблица 8.3. Конфигурации, полученные для Примера 8.1

	c_{x0}	c_{y0}	z_{y0}	ψ	θ	φ
1	0,823	1,359	4,600	−1,482	1,856	−1,726
2	−1,014	1,001	4,976	0,690	0,966	−0,531
3	−2,014	1,222	4,423	0,027	0,519	−0,015
4	−1,772	−1,574	1,921	−0,871	1,297	0,706

Таблица 8.4. Данные Примера 8.2

i	$a_{x0}(i)$	$a_{y0}(i)$	$a_{z0}(i)$	$b_{x4}(i)$	$b_{y4}(i)$	$b_{z4}(i)$	$(y_i)^2$
1	−9,70	9,1	1,0	−3,000	7,300	1,0	426,76
2	9,70	9,1	−1,0	3,000	−7,300	−1,0	576,27
3	12,76	3,9	1,0	7,822	−1,052	1,0	365,86
4	3,00	−13,0	−1,0	4,822	−6,248	−1,0	377,70
5	−3,00	−13,0	1,0	−4,822	−4,822	1,0	381,53
6	−12,76	3,9	−1,0	−7,822	−7,822	−1,0	276,30

нашел десять параллелотопов (см. табл. 8.5). *Для каждого из них условие единственности удовлетворяется. Соответствующие конфигурации представлены на* рис. 8.5. *Для некоторых из них подвижная платформа пересекается с основанием, такие случаи не являются реальными, хотя не запрещены в задаче при поставленной формулировке.* ■

Таблица 8.5. Конфигурации, полученные для Примера 8.2

	c_{x0}	c_{y0}	c_{z0}	ψ	θ	φ
1	4,945	−6,707	9,757	2,076	2,159	2,801
2	4,458	−4,606	−7,666	1,126	1,973	2,051
3	−10,406	−6,218	−7,269	0,692	1,860	0,225
4	−2,904	−1,346	−3,538	1,124	1,107	1,756
5	−4,835	5,507	−16,756	−2,790	0,503	3,040
6	−6,980	6,457	−14,916	−2,256	0,740	2,664
7	1,561	6,709	15,219	−0,928	1,227	0,566
8	−11,524	−0,882	10,701	0,172	1,582	−0,504
9	−7,674	−3,113	5,658	−1,612	1,953	−1,828
10	−5,000	5,000	17,000	0,000	0,524	0,000

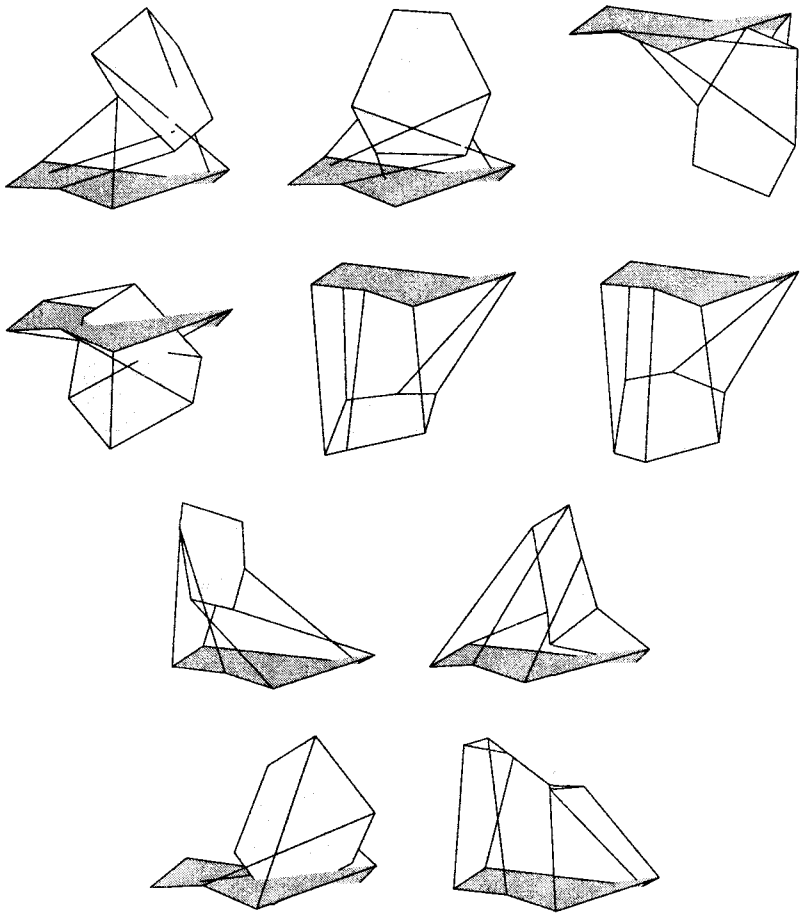


Рис. 8.5. Десять конфигураций платформы по Примеру 8.2

8.3. Планирование маршрута

В данном параграфе описывается новый подход к поиску свободного от столкновений маршрута движения объекта в среде, содержащей известные

препятствия [Jaulin и Godon, 1999; Jaulin, 2001a]. Эта задача *планирования маршрута в известной среде* привлекала серьезное внимание [Nilsson, 1969; Lozano-Pérez, 1981; O'Dunlaing и Yap, 1982; Rimon и Koditschek, 1992].

Большинство доступных из литературы методов основываются на концепции *пространства конфигураций* (состояний), или C-пространства [Lozano-Pérez и Wesley, 1979]. Каждая координата C-пространства представляет степень свободы данного объекта. Число независимых параметров, необходимых для описания конфигурации (состояния) этого объекта, соответствует размерности C-пространства. Начальная конфигурация и желаемая конечная конфигурация объекта становятся двумя точками $\mathbf{a}$ и $\mathbf{b}$ C-пространства. Примерами таких объектов являются промышленные роботы с n степенями свободы. Их конфигурация может быть описана n вещественными числами, которые являются координатами некоторого вектора размерности n в C-пространстве [Lozano-Pérez, 1983]. Пространство $\mathbb{S}$ *допустимых конфигураций* является подмножеством C-пространства и содержит только векторы состояний, на которых объект не сталкивается с препятствиями. Планирование маршрута сводится к нахождению некоторого маршрута, принадлежащего $\mathbb{S}$ и переводящего объект из начальной точки $\mathbf{a}$ в желаемую конечную точку $\mathbf{b}$.

Ряд подходов к решению этой задачи основывается на использовании потенциальных функций [Khatib, 1986]. Препятствия, которые необходимо обойти, окружаются отталкивающим потенциалом, а желаемое конечное состояние окружается притягивающим потенциалом. При этом ожидается, что объект, движимый силой, созданной этими потенциалами, достигнет желаемого состояния без столкновений с препятствиями (при этом необходимо обеспечить, чтобы движение не застряло ни в одном локальном минимуме потенциала).

Рассматривались также подходы, основанные на разбиении C-пространства [Brooks и Lozano-Pérez, 1985; Reboulet, 1988; Pruski, 1996]. В этих подходах C-пространство разбивается на три множества неперекрывающихся параллелотопов, а именно: на параллеdотопы, целиком лежащие внутри подпространства $\mathbb{S}$; параллелотопы, относительно которых можно доказать, что они целиком лежат вне $\mathbb{S}$; параллелотопы, относительно расположения которых нельзя ничего доказать.

Хотя это может звучать вызывающе, но доступные по литературе методы решения вопроса — лежит ли параллелотоп внутри или вне $\mathbb{S}$, не основываются на интервальном анализе и наталкиваются на трудности с описывающими параметрами. Интервальный анализ был уже применен для параметрического планирования маршрута в работах [Jualin и Walter, 1996] и [Piazzi и Visioli, 1998], но этот подход потребовал параметрическую модель, чтобы сделать маршрут допустимым, т. е. чтобы маршрут принадлежал бы некоторому семейству, параметризованному некоторым настраивае-

мым вектором $\mathbf{p}$, размерность которого должна была быть малой. В указанных работах модель маршрута выбиралась в виде кубического полинома. В отличие от этого, подход, который мы здесь описываем, не требует никакой параметрической модели маршрута.

В параграфе 8.3.1 приводятся основные сведения, необходимые для построения графа, связанного с задачей планирования маршрута. В параграфе 8.3.2 рассматриваются два алгоритма для нахождения некоторого допустимого маршрута, приводящего из состояния $\mathbf{a}$ в состояние $\mathbf{b}$.

Первый алгоритм описывает множество $\mathbb{S}$ с помощью некоторых покрытий перед тем как искать допустимый маршрут. За исключением того, что проверки, используемые для определения допустимости некоторого параллелотопа, базируются на интервальном анализе, этот алгоритм в значительной части является классическим [Brooks и Lozano-Pérez, 1985; Reboulet, 1988].

Второй более эффективный алгоритм исследует только те области С-пространства, которые могут привести к интересующему нас маршруту. Для иллюстрации его применения в параграфе 8.3.3 рассматривается построение маршрута перемещения объекта, имеющего форму невыпуклого многоугольника, в двумерном пространстве с препятствиями, описываемыми прямолинейными отрезками.

8.3.1. Графическая дискретизация пространства конфигураций

Гарантированное описание пространства $\mathbb{S}$ допустимых конфигураций может быть получено с использованием алгоритма SIVIA, вырабатывающего покрытие пространства и описанного в Главе 3. Далее можно построить граф, связанный с таким описанием. Вся процедура, называемая *графической дискретизацией*, будет использована в параграфе 8.3.2 для планирования маршрута. Введем теперь основные понятия, необходимые для понимания принципов графической дискретизации.

Напомним, что некоторое покрытие параллелотопа $[\mathbf{p}_0]$ — это множество непересекающихся параллелотопов, объединение которых равно $[\mathbf{p}_0]$. Это покрытие обозначается символом $\mathbb{P}$, когда оно рассматривается как множество, и символом $\mathcal{P}$, когда оно рассматривается как некоторый список параллелотопов (см. Замечание 3.1, с. 75). Рис. 8.6 показывает некоторое покрытие $\mathcal{P} = \{[\mathbf{p}_1], [\mathbf{p}_2], \ldots, [\mathbf{p}_9]\}$ параллелотопа $[\mathbf{p}_0] = [-2, 10] \times [-2, 6]$.

Два параллелотопа из $\mathbb{IR}^n$, занесенные в список $\mathcal{P}$, являются *смежными*, если они, по крайней мере частично, соприкасаются своими гранями, размерности $n-1$. Например, параллелотопы $[\mathbf{p}_1]$ и $[\mathbf{p}_4]$ являются смежными в покрытии, показанном на рис. 8.6, а параллелотопы $[\mathbf{p}_2]$ и $[\mathbf{p}_5]$ не являются смежными. Подпокрытие $\mathcal{P}_1$ покрытия $\mathcal{P}$, которое содержит все

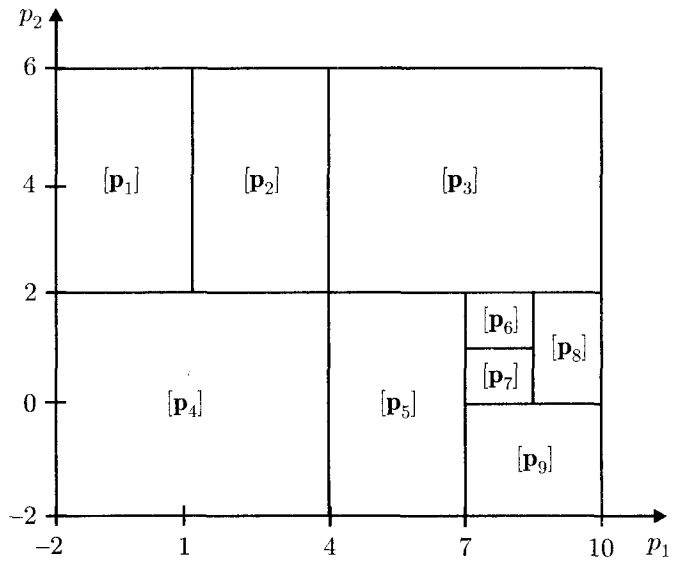


Рис. 8.6. Покрытие, обозначаемое символом $\mathbb{P}$, как множество, и символом $\mathcal{P}$, как список параллелотопов

параллелотопы покрытия $\mathcal{P}$, удовлетворяющие заданному условию, будем обозначать

$$\mathcal{P}_1 = \text{подпокрытие } (\mathcal{P},\ \text{условие}). \tag{8.22}$$

Опять в связи с покрытием, показанным на рис. 8.6, рассмотрим, например, проверку $t(\mathbf{p}) \triangleq (p_1 = 5)$, где p_1 — первая компонента вектора $\mathbf{p}$. Если $[t]([\mathbf{p}])$ является минимальной проверкой включения для $t(\mathbf{p})$, поскольку $[t]([\mathbf{p}_3]) = [t]([\mathbf{p}_5]) = [0, 1]$, то

$$\text{подпокрытие}(\mathcal{P}, [t]([\mathbf{p}]) = 0) = \{[\mathbf{p}_1], [\mathbf{p}_2], [\mathbf{p}_4], [\mathbf{p}_6], [\mathbf{p}_7], [\mathbf{p}_8], [\mathbf{p}_9]\}. \tag{8.23}$$

Потребуются также некоторые основные понятия теории графов [Deo, 1974]. *Граф* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ состоит из непустого множества $\mathcal{V}$ вершин и множества $\mathcal{E}$ неупорядоченных пар вершин множества $\mathcal{V}$, называемых *ребрами*. Если v_a и v_b — две вершины графа, то ребро, соединяющее эту пару (v_a, v_b), обозначается как $v_a v_b$. *Ветвь* на графе $\mathcal{G}$ есть последовательность вершин v_i $(i = 1, \ldots, k)$, таких, что для любого $i \in \{1, \ldots, k-1\}$

ребро v_iv_{i+1} принадлежит множеству $\mathcal{E}$. Ветвь является *маршрутом*, если $v_i \neq v_j$ при $i \neq j$. Ветвь является *циклом*, если $v_k = v_1$. Граф называется *связным*, если существует маршрут между его любыми двумя вершинами. Две различные вершины v_i и v_j графа $\mathcal{G}$ являются *соседними*, если множество $\mathcal{E}$ содержит ребро v_iv_j. *Подграф* графа $\mathcal{G}$ есть граф, вершины и ребра которого принадлежат $\mathcal{G}$.

Любое покрытие $\mathcal{P}$ параллелотопа $[\mathbf{p}_0]$ может быть представлено некоторым графом $\mathcal{G}$. Каждый элемент $[\mathbf{p}_i]$ (параллелотоп) покрытия $\mathcal{P}$ соответствует некоторой вершине v_i графа $\mathcal{G}$. Если два параллелотопа $[\mathbf{p}_i]$ и $[\mathbf{p}_j]$ из $\mathcal{P}$ являются соседними, то граф $\mathcal{G}$ содержит ребро v_iv_j. Например, граф $\mathcal{G}$, соответствующий покрытию рис. 8.6, показан на рис. 8.7, *а*. Подпокрытие также может быть представлено графом. Например, граф $\mathcal{G}_1$, соответствующий подпокрытию $\mathcal{P}_1$ по соотношению (8.23), показан на рис. 8.7, *б*. Он является несвязным подграфом графа $\mathcal{G}$. Граф, соответствующий некоторому покрытию (или подпокрытию) $\mathcal{P}$, обозначается как $\mathcal{G} = \text{graph}(\mathcal{P})$. Построение графа, соответствующего заданному покрытию, требует быстрых алгоритмов для нахождения параллелотопов, соседних некоторому заданному параллелотопу, и такие алгоритмы были разработаны в [Samet, 1982].

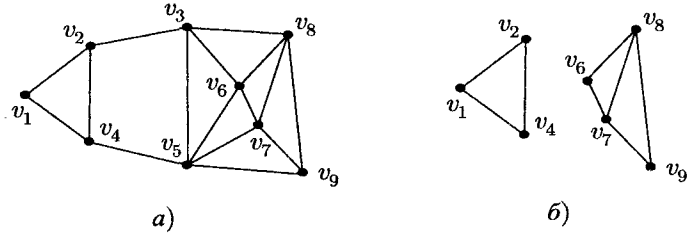


Рис. 8.7. *а*) Граф $\mathcal{G}$, соответствующий покрытию $\mathcal{P}$; *б*) Граф $\mathcal{G}_1$, соответствующий подпокрытию $\mathcal{P}_1$ покрытия $\mathcal{P}$; подграф $\mathcal{G}_1$ является несвязным подграфом графа $\mathcal{G}$

8.3.2. Алгоритмы нахождения допустимого маршрута

Рассмотрим компактное множество $\mathbb{S}$, заключенное в параллелотоп $[\mathbf{p}_0]$, и две точки $\mathbf{a}$ и $\mathbf{b}$ множества $\mathbb{S}$. Предположим, что имеется некоторая точная проверка включения $[t]$ для решения вопроса о том, лежит ли некоторый параллелотоп внутри или вне множества $\mathbb{S}$. *Движение* из начальной точки $\mathbf{a}$ к желаемой конечной точке $\mathbf{b}$ есть некоторая однозначная

Таблица 8.6. Алгоритм поиска кратчайшего маршрута на графе

Алгоритм DIJKSTRA(вход : $\mathcal{G}, v_a, v_b$; выход : $\mathcal{L}$)
1 для каждой вершины $v \in \mathcal{G}$, $d(v) := \infty$;
2 $d(v_a) := 0$; $d_{\min} := 0$;
3 повторить
4 если $\mathcal{G}(d_{\min}) = \varnothing$, то $\mathcal{L} := \varnothing$; возврат
5 $d_{\min} := d_{\min} + 1$;
6 для каждой вершины $v \in \mathcal{G}(d_{\min} - 1)$,
7 для каждой вершины w, соседней с v в графе $\mathcal{G}$ с $d(w) = \infty$, $d(\omega) := d_{\min}$;
8 до тех пор, пока $d(v_b) \neq \infty$;
9 $\ell := d(v_b)$; $v_\ell := v_b$;
10 для $i := \ell - 1$ уменьшая до 0, выбрать вершину v_i, соседнюю к v_{i+1}, такую, что $d(v_i) = i$;
11 $\mathcal{L} := \{v_a, v_1, v_2, \ldots, v_{\ell-1}, v_b\}$.

непрерывная функция $\mathbf{m} : [0,1] \to \mathbb{R}^n$; $\tau \mapsto \mathbf{m}(\tau)$, такая, что $\mathbf{m}(0) = \mathbf{a}$ и $\mathbf{m}(1) = \mathbf{b}$. Соответствующий маршрут есть множество

$$\mathbb{L} = \{\mathbf{m}(\tau) \,|\, \tau \in [0,1]\}. \tag{8.24}$$

Маршрут $\mathbb{L}$ является допустимым, если $\mathbb{L} \subset \mathbb{S}$ (т. е. маршрут целиком лежит в исходном множестве). В этом параграфе предлагаются два алгоритма FEASIBLEPATH1 и FEASIBLEPATH2, находящие такие допустимые маршруты. Оба алгоритма, когда они успешно завершают свою работу, выдают *параллелотопный маршрут*, т. е. список связанных параллелотопов $\{[\mathbf{p}_a], [\mathbf{p}_1], \ldots, [\mathbf{p}_{l-1}], [\mathbf{p}_b]\}$, таких, что $\mathbf{a} \in [\mathbf{p}_a]$ и $\mathbf{b} \in [\mathbf{p}_b]$, и таких, что все они лежат внутри множества $\mathbb{S}$. Но еще будет необходимо найти точечный маршрут $\mathbb{L}$ из $\mathbf{a}$ в $\mathbf{b}$. В общем случае, выбор этого окончательного точечного маршрута должен основываться на конкретных соображениях, таких как кинематические или динамические характеристики (объекта), а не только на чисто геометрических критериях [Lammond, 1986]. Например, желаемым свойством этого окончательного маршрута является гладкость. Для простоты, в наших исследованиях будет считаться приемлемой ломаная линия от $\mathbf{a}$ в $\mathbf{b}$, целиком лежащая в параллелотопном маршруте.

Среди алгоритмов, которые предлагались для поиска кратчайшего маршрута между конкретными вершинами v_a и v_b на графе $\mathcal{G}$, наиболее эффективным является алгоритм, разработанный в [Dijkstra, 1959]. Он вызывается алгоритмами FEASIBLEPATH1 и FEASIBLEPATH2. Хотя он был

первоначально разработан для взвешенных *диграфов* (т. е. графов с ориентированными ребрами), мы будем использовать для неориентированных графов некоторую его упрощенную версию, представленную в табл. 8.6. С каждой вершиной v графа $\mathcal{G}$ свяжем некоторое целое число $d(v)$, представляющее минимальное число ребер в маршруте от вершины v_a к вершине v. Символ $\mathcal{G}(i)$, $i \in \mathbb{N}$, будет обозначать множество всех вершин графа $\mathcal{G}$, таких, что $d(v) = i$. Если алгоритм DIJKSTRA выдает пустой список $\mathcal{L}$, то вершины v_a и v_b *не находятся в одной связной компоненте графа* $\mathcal{G}$. В остальных случаях алгоритм выдает один из кратчайших маршрутов из v_a в v_b, в терминах номеров проходимых вершин.

При работе алгоритма DIJKSTRA$(\mathcal{G}, v_1, v_6)$ на графе рис. 8.7, *а* получаем $d(v_1) = 0, d(v_2) = d(v_4) = 1, d(v_3) = d(v_5) = 2, d(v_6) = d(v_7) = d(v_8) = d(v_9) = 3$. Алгоритм DIJKSTRA выдает либо маршрут $\{v_1, v_2, v_3, v_6\}$, либо маршрут $\{v_1, v_4, v_5, v_6\}$.

Первая часть алгоритма FEASIBLEPATH1, приведенная в табл. 8.7, есть алгоритм SIVIA (см. Главу 3), который используется для построения покрытия $\mathbb{P}$ и двух покрытий $\underline{\mathbb{P}}$ и $\overline{\mathbb{P}}$ покрытия $\mathbb{P}$, удовлетворяющих условию

$$\underline{\mathbb{P}} \subset \mathbb{S} \subset \overline{\mathbb{P}}. \tag{8.25}$$

Вспомогательный набор $\mathcal{Q}$ параллелотопов служит для хранения всех тех параллелотопов, которые еще не обработаны. Далее строятся графы $\underline{\mathcal{G}}$ и $\overline{\mathcal{G}}$, соответствующие $\underline{\mathcal{P}}$ и $\overline{\mathcal{P}}$, и из $\overline{\mathcal{P}}$ выбираются два параллелотопа $[\mathbf{p}_a]$ и $[\mathbf{p}_b]$, такие, что $\mathbf{a} \in [\mathbf{p}_a]$ и $\mathbf{b} \in [\mathbf{p}_b]$. Приемлемых кандидатов на параллелотопы $[\mathbf{p}_a]$ или $[\mathbf{p}_b]$ может быть несколько, если $\mathbf{a}$ или $\mathbf{b}$ находится на границе некоторого параллелотопа из $\overline{\mathcal{P}}$. В таком случае алгоритм выбирает любые их них. Пусть v_a и v_b — две вершины из графа $\overline{\mathcal{G}}$, соответствующие паралелотопам $[\mathbf{p}_a]$ и $[\mathbf{p}_b]$. Алгоритм FEASIBLEPATH1 вызывает процедуру DIJKSTRA для построения маршрута $\overline{\mathcal{L}}$, идущего от v_a к v_b на графе $\overline{\mathcal{G}}$. Если такового маршрута не найдено, то этим показывается, что точки $\mathbf{a}$ и $\mathbf{b}$ лежат в несвязанных компонентах[2] множества $\mathbb{S}$, и алгоритм FEASIBLEPATH1 сообщает, что маршрута нет. Если найден непустой маршрут $\overline{\mathcal{L}}$, то процедура DIJKSTRA запускается снова для поиска некоторого маршрута $\underline{\mathcal{L}}$, соединяющего v_a с v_b, из графа $\underline{\mathcal{G}}$. Если такой маршрут $\underline{\mathcal{L}} = \{v_a, v_1, \ldots, v_{l-1}, v_b\}$ найден, то соответствующий параллелотопный маршрут в $\underline{\mathbb{P}}$ включается в множество $\mathbb{S}$, и, следовательно, может быть построен соответствующий

[2] Если $\overline{\mathcal{L}} = \varnothing$, то вершины v_a и v_b принадлежат различным связным компонентам графа $\overline{\mathcal{G}}$, т. е., точки $\mathbf{a}$ и $\mathbf{b}$ принадлежат различным связным компонентам покрытия $\overline{\mathbb{P}}$. Теперь, поскольку $\mathbf{a} \in \mathbb{S}$, $\mathbf{b} \in \mathbb{S}$ и $\mathbb{S} \subset \overline{\mathbb{P}}$, то $\mathbf{a}$ и $\mathbf{b}$ принадлежат различным связным компонентам множества $\mathbb{S}$. Таким образом, когда алгоритм FEASIBLEPATH1 выдает сообщение «нет маршрута», это заключение является гарантированным (с учетом того, что при вычислениях использовалось округление с избытком).

Таблица 8.7. Основной алгоритм поиска допустимого маршрута

Алгоритм FEASIBLEPATH1(вход: $t(\cdot), \mathbf{a}, \mathbf{b}, [\mathbf{p}_0], \varepsilon$; выход: $\underline{\mathcal{L}}$, сообщение)

```
если [t](a) ≠ 1 или [t](b) ≠ 1, то
  L := ∅; сообщение := "ошибка: a и b должны быть
        допустимыми"; возврат;
если a ∉ [p0] или b ∉ [p0], то
  L := ∅; сообщение := "ошибка: a и b должны
        принадлежать [p0]"; возврат;
Q := {[p0]}; ΔP := ∅; P := ∅;
пока Q ≠ ∅;
    захватить некоторый параллелотоп из Q
                        в параллелотоп [p];
    если [t]([p]) = 1 то P := P ∪ {[p]};
    если [t]([p]) = [0,1] и ω([p]) ⩽ ε, то ΔP := ΔP ∪ {[p]};
    если [t]([p]) = [0,1] и ω([p]) > ε, то
      выполнить бисекцию [p] и записать два полученных
          подпараллелотопа в конец списка Q;
закончить цикл "пока";
P̄ = P ∪ ΔP; Ḡ = граф(P̄); G = граф(P);
v_a := вершина([p_a]), где [p_a] ∈ P̄ и a ∈ [p_a];
v_b := вершина([p_b]), где [p_b] ∈ P̄ и b ∈ [p_b];
L̄ := DIJKSTRA(Ḡ, v_a, v_b);
если L̄ = ∅, то
    L := ∅; сообщение := "нет маршрута"; возврат;
если v_a ∉ G или v_b ∉ G, то
    L := ∅; сообщение := "нет маршрута"; возврат;
L := DIJKSTRA(G, v_a, v_b);
если L ≠ ∅, то
    сообщение := "маршрут найден";
иначе
    сообщение := "отказ".
```

Таблица 8.8. Улучшенный алгоритм поиска допустимого маршрута

Алгоритм FEASIBLEPATH2(вход : $t(\cdot), \mathbf{a}, \mathbf{b}, [\mathbf{p}_0]$; выход : $\underline{\mathcal{L}}$, сообщение)

1 если $[t](\mathbf{a}) \neq 1$ или $[t](\mathbf{b}) \neq 1$, то
2 $\quad\underline{\mathcal{L}} := \varnothing$; сообщение := "ошибка: **a** и **b** должны быть допустимыми"; возврат;
3 если $\mathbf{a} \notin [\mathbf{p}_0]$ или $\mathbf{b} \notin [\mathbf{p}_0]$, то
4 $\quad\underline{\mathcal{L}} := \varnothing$; сообщение := "ошибка: **a** и **b** должны принадлежать $[\mathbf{p}_0]$"; возврат;
5 пусть $\mathcal{P}$ – покрытие, состоящее из единственного параллелотопа $[\mathbf{p}_0]$;
6 повторять
7 $\quad\overline{\mathcal{P}} :=$ покрытие($\mathcal{P}$, $1 \in [t]([\mathbf{p}])$); $\overline{\mathcal{G}} :=$ граф($\overline{\mathcal{P}}$)
8 $\quad v_a :=$ вершина($[\mathbf{p}_a]$), где $[\mathbf{p}_a] \in \overline{\mathcal{P}}$ и $\mathbf{a} \in [\mathbf{p}_a]$;
9 $\quad v_b :=$ вершина($[\mathbf{p}_b]$), где $[\mathbf{p}_b] \in \overline{\mathcal{P}}$ и $\mathbf{b} \in [\mathbf{p}_b]$;
10 $\quad\overline{\mathcal{L}} :=$ DIJKSTRA($\overline{\mathcal{G}}, v_a, v_b$);
11 $\quad$если $\overline{\mathcal{L}} = \varnothing$, то $\underline{\mathcal{L}} := \varnothing$; сообщение := "нет маршрута"; возврат;
12 $\quad\underline{\mathcal{P}} :=$ покрытие($\mathcal{P}$, $[t]([\mathbf{p}]) = 1$); $\underline{\mathcal{G}} =$ граф($\underline{\mathcal{P}}$);
13 $\quad$если $v_a \in \underline{\mathcal{G}}$ и $v_b \in \underline{\mathcal{G}}$, то $\underline{L} :=$ DIJKSTRA($\underline{\mathcal{G}}, v_a, v_b$);
14 $\quad$если $\underline{L} \neq \varnothing$, то сообщение := "маршрут найден"; возврат;
15 $\quad\mathcal{C} := \{[\mathbf{p}] \in \overline{\mathcal{P}} \mid$ вершина($[\mathbf{p}]$) $\in \overline{\mathcal{L}}$ и $[t]([\mathbf{p}]) = [0,1]\}$;
16 $\quad$выполнить бисекцию всех подпараллелотопов из $\mathcal{C}$, получая, таким образом, новое покрытие $\mathcal{P}$;
17 постоянно

точечный маршрут. Если маршрут $\underline{\mathcal{L}}$ пуст, то алгоритм сообщает об отсутствии решения, так как пока ничего не удается доказать относительно существования или не существования допустимого маршрута из **a** в **b**. Можно запустить алгоритм снова с меньшей величиной допуска ε.

Обоснованием алгоритма FEASIBLEPATH2, приведенного в табл. 8.8, является то, что время, затрачиваемое на работу алгоритма DIJKSTRA, весьма мало по сравнению с затратами на построение детального описания пространства допустимых конфигураций.

На каждой итерации алгоритма FEASIBLEPATH2 алгоритм DIJKSTRA используется для выделения областей пространства конфигураций, которые

представляются наиболее привлекательными, и алгоритм останавливается сразу, как только найден некоторый допустимый маршрут.

Пусть $\mathbb{P}$ – текущее покрытие параллелотопа поиска $[\mathbf{p}_0]$. Как и алгоритм FEASIBLEPATH1, алгоритм FEASIBLEPATH2 сперва просматривает кратчайший маршрут $\overline{\mathcal{L}}$ на графе, связанном с допустимым покрытием $\overline{\mathbb{P}}$ множества $\mathbb{P}$, которое удовлетворяет условию $\mathbb{S} \subset \overline{\mathbb{P}}$. Если такого маршрута не найдено, то точки **a** и **b** не находятся в одной связной компоненте множества $\mathbb{S}$, и алгоритм сообщает, что такого маршрута нет. Если такой маршрут существует, то алгоритм FEASIBLEPATH2 пытается найти кратчайший маршрут $\underline{\mathcal{L}}$ на графе $\underline{\mathcal{G}}$, связанном с покрытием $\underline{\mathbb{P}}$ покрытия $\mathbb{P}$, которое удовлетворяет условию $\underline{\mathbb{P}} \subset \mathbb{S}$. Если такой маршрут найден, то алгоритм заканчивает работу. В противном случае, поскольку параллелотопный маршрут, соответствующий списку $\overline{\mathcal{L}}$, может тем не менее содержать некоторый допустимый маршрут, все подпараллелотопы этого маршрута подвергаются бисекции, и, таким образом, получается новое покрытие $\mathbb{P}$.

8.3.3. Тестовый пример

Пространство конфигураций (состояний) в этом тестовом примере выбрано двумерным, так что множество $\mathbb{S}$ допустимых состояний может быть легко визуализировано, но достаточно попытаться найти некоторое решение, чтобы понять, что задача, все-таки, весьма сложна.

Рассмотрим двумерное рабочее пространство, которое содержит $\bar{j}$ препятствий в виде отрезков. Крайние точки каждого j-го препятствия обозначим как $\mathbf{a_j}$ и $\mathbf{b_j}$ для $j \in \mathcal{J} = \{1, \ldots, \bar{j}\}$. Передвигаемый объект является невыпуклым многоугольником с $\bar{i}$ вершинами, и обозначим их как $\mathbf{s_i} \in \mathbb{R}^2$, $i \in \mathcal{I} = \{1, \ldots, \bar{i}\}$. В рассматриваемом примере $\bar{j} = 2$, а $\bar{i} = 14$. На положение вершины $\mathbf{s}_1$ наложено ограничение: она всегда должна оставаться на горизонтальной оси в рабочем пространстве (рис. 8.8, слева). Таким образом, состояние объекта представляется двумерным вектором $(p_1, p_2)^{\mathrm{T}}$, где p_1 – координата $\mathbf{s}_1$ на горизонтальной оси, а p_2 – угол его поворота (в радианах). Рис. 8.8 иллюстрирует понятие пространства конфигураций (состояний, положений) данного тестового примера. Рис. 8.9, *а* представляет начальное положение, а на рис. 8.9, *б* показано конечное желаемое положение объекта.

Некоторый вектор $\mathbf{p}$, связанный с заданной конфигурацией (положением) объекта, является допустимым тогда и только тогда, когда ни одна из его сторон не пересекает какого-нибудь отрезка из препятствия и крайние точки каждого отрезка препятствия лежат вне объекта. Как показано на рис. 8.8, вектор положения $\mathbf{p} = (8, \pi/4)^{\mathrm{T}}$ – допустим. Далее, символ segm(**a**, **b**) обозначает отрезок с концевыми точками **a** и **b**, а символ line(**a**, **b**) – прямую

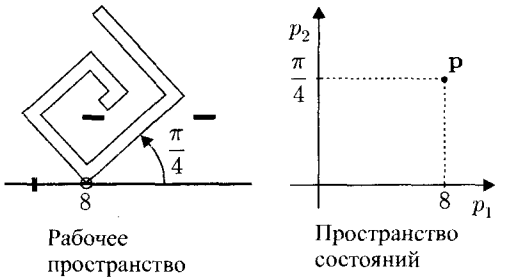


Рис. 8.8. Любому заданному положению объекта в рабочем пространстве соответствует единственная точка $\mathbf{p}$ в С-пространстве состояний

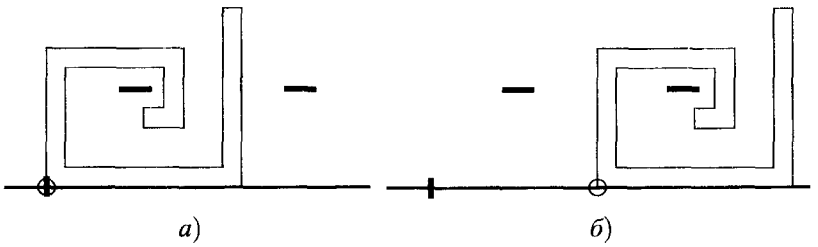


Рис. 8.9. Желаемое положения объекта: *а*) начальное; *б*) конечное

линию, проходящую через $\mathbf{a}$ и $\mathbf{b}$. Обозначим как $[\mathbb{A}]$ интервальную оболочку множества $\mathbb{A}$ (т. е. наименьший параллелотоп, содержащий $\mathbb{A}$), тогда $[\mathbf{a} \cup \mathbf{b}]$ будет представлять наименьший параллелотоп, который содержит $\mathbf{a}$ и $\mathbf{b}$. Если $\mathbf{s}_{i+1}$ взять равным $\mathbf{s}_1$, то

$$(\mathbf{p} \in \mathbb{S}) \Leftrightarrow \begin{pmatrix} \forall i \in \mathcal{I},\ \forall j \in \mathcal{J},\ \mathrm{segm}(\mathbf{s}_i,\ \mathbf{s}_{i+1}) \cap \mathrm{segm}(\mathbf{a}_j,\ \mathbf{b}_j) = \varnothing \\ \text{и } \mathbf{a}_j \text{ и } \mathbf{b}_j \text{ находятся вне объекта} \end{pmatrix}. \quad (8.26)$$

Проверка факта пересекается ли $\mathrm{segm}(\mathbf{s}_i, \mathbf{s}_{i+1})$ с $\mathrm{segm}(\mathbf{a}_j, \mathbf{b}_j)$ эквивалентна проверке выполнения одновременно следующих трех условий:

$$\begin{cases} 1) & \mathrm{line}(\mathbf{s}_i, \mathbf{s}_{i+1}) \cap \mathrm{segm}(\mathbf{a}_j, \mathbf{b}_j) \neq \varnothing, \\ 2) & \mathrm{segm}(\mathbf{s}_i, \mathbf{s}_{i+1}) \cap \mathrm{line}(\mathbf{a}_j, \mathbf{b}_j) \neq \varnothing, \\ 3) & [\mathbf{s}_{i+1} \cup \mathbf{s}_i] \cap [\mathbf{a}_j \cup \mathbf{b}_j] \neq \varnothing. \end{cases} \quad (8.27)$$

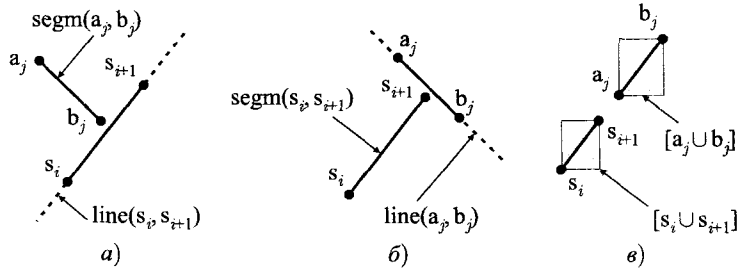


Рис. 8.10. Три положения объекта, при которых отрезок препятствия и сторона объекта имеют пустое пересечение

Таблица 8.9. Алгоритм проверки допустимости вектора $\mathbf{p}$ состояния (положения) объекта

Алгоритм t(вход : $\mathbf{p}$; выход : t)

1 для $j = 1$ до $\bar{j}$
2 $\widetilde{x}_a = (x_a(j) - p_1)\cos p_2 + y_a \sin p_2$;
3 $\widetilde{y}_a = -(x_a(j) - p_1)\sin p_2 + y_a(j)\cos p_2$;
4 $\widetilde{x}_b = (x_b(j) - p_1)\cos p_2 + y_a(j)\sin p_2$;
5 $\widetilde{y}_b = -(x_a(j) - p_1)\sin p_2 + y_a(j)\cos p_2$;
6 $\widetilde{\mathbf{a}} = (\widetilde{x}_a,\ \widetilde{y}_a)^{\mathrm{T}}$; $\widetilde{\mathbf{b}} = (\widetilde{x}_b,\ \widetilde{y}_b)^{\mathrm{T}}$;
7 если $\widetilde{\mathbf{a}}$ находится внутри объекта, то $t := 0$; возврат;
8 если $\widetilde{\mathbf{b}}$ находится внутри объекта, то $t := 0$; возврат;
9 для $i = 1$ до $\bar{i}$
10 $\quad d_1 = \det\left(\mathbf{s}_i - \widetilde{\mathbf{b}}, \mathbf{s}_i - \widetilde{\mathbf{a}}\right) * \det\left(\mathbf{s}_{i+1} - \widetilde{\mathbf{b}}, \mathbf{s}_{i+1} - \widetilde{\mathbf{a}}\right)$;
11 $\quad d_2 = \det\left(\mathbf{s}_{i+1} - \mathbf{s}_i, \mathbf{s}_i - \widetilde{\mathbf{a}}\right) * \det\left(\mathbf{s}_{i+1} - \mathbf{s}_i, \mathbf{s}_i - \widetilde{\mathbf{b}}\right)$;
12 $\quad$ если $(d_1 \leqslant 0)$ и $(d_2 \leqslant 0)$ и $\left([\mathbf{s}_{i+1} \cup \mathbf{s}_i] \cap [\widetilde{\mathbf{a}} \cup \widetilde{\mathbf{b}}] \neq \varnothing\right)$, то
13 $\quad\quad t := 0$; возврат;
14 $t := 1$; возврат.

Условие 1) не выполняется на рис. 8.10, *а*, условие 2) не выполняется на рис. 8.10, *б*, а условие 3) не выполняется на рис. 8.10, *в*. В каждом из этих случаев, хотя два других условия выполняются, пересечение $\mathrm{segm}(\mathbf{s}_i, \mathbf{s}_{i+1}) \cap \mathrm{segm}(\mathbf{a}_j, \mathbf{b}_j) = \varnothing$.

Алгоритм, представленный в табл. 8.9, проверяет допустимость вектора $\mathbf{p}$ состояния (положения) объекта. Для заданного сегмента с номером j препятствия, Шаги 2–6 вычисляют точку $\widetilde{\mathbf{a}} = (\widetilde{x}_a, \widetilde{y}_a)^{\mathrm{T}}$ и точку $\widetilde{\mathbf{b}} = (\widetilde{x}_b, \widetilde{y}_b)^{\mathrm{T}}$ — координаты крайних точек сегмента $\mathrm{segm}(\mathbf{a}_j, \mathbf{b}_j)$ в системе координат объекта. Чтобы доказать, что точка $\widetilde{\mathbf{a}}$ лежит внутри объекта, как требуется на Шаге 7, достаточно проверить, что

$$\sum_{i=1}^{\bar{i}} \arg\left(\mathbf{s}_i - \widetilde{\mathbf{a}}, \mathbf{s}_{i+1} - \widetilde{\mathbf{a}}\right) \neq 0. \qquad (8.28)$$

Эта сумма равна нулю, если точка $\widetilde{\mathbf{a}}$ лежит вне многоугольника (рис. 8.11, *б*), и равна 2π, если $\widetilde{\mathbf{a}}$ лежит внутри. Тот же тип условий используется для проверки точки $\widetilde{\mathbf{b}}$ на Шаге 8.

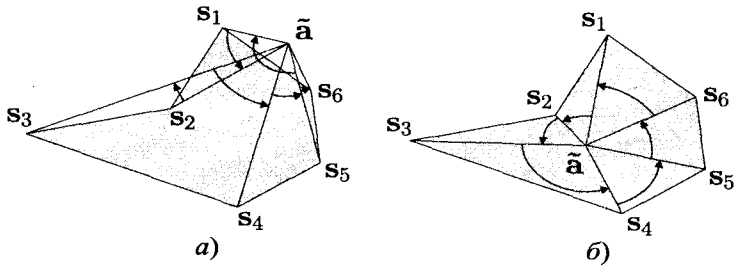


Рис. 8.11. Проверка нахождения точки $\widetilde{\mathbf{a}}$ внутри или вне многоугольника: *а*) сумма углов равна нулю; *б*) сумма углов равна 2π

Если $d_1 \leqslant 0$, $d_2 \leqslant 0$ и $[\mathbf{s}_{i+1} \cup \mathbf{s}_i] \cap [\widetilde{\mathbf{a}}, \widetilde{\mathbf{b}}] \neq \varnothing$, то (8.27) показывает, что i-я сторона объекта пересекается с j-м сегментом препятствия; следовательно, вектор $\mathbf{p}$ состояния — недопустим, и алгоритм выдает нулевое значение признака проверки на Шаге 13. Проверка включения $[t]([\mathbf{p}])$ для $t(\mathbf{p})$ дается алгоритмом, представленным в табл. 8.10. Для оценки величин $[\widetilde{x}_a], [\widetilde{y}_a], [\widetilde{x}_b], [\widetilde{y}_b], [d_1], [d_2]$ использована центрированная форма функции включения по отношению к параметрам p_1 и p_2.

Менее чем за 10 минут на компьютере Pentium 133 при $\varepsilon = 0{,}1$ алгоритм FEASIBLEPATH1 выработал покрытие, показанное на рис. 8.12, и соответствующий граф. Параллелотопы, отмеченные серым цветом, являются допустимыми, а черным — недопустимыми. Ничего не известно относительно параллелотопов, отмеченных белым цветом. Менее чем за 0,1 секунды алгоритм FEASIBLEPATH1 находит кратчайший маршрут на графе.

Таблица 8.10. Алгоритм проверки допустимости конфигурации параллелотопа $[\mathbf{p}]$

Алгоритм $[t]$(вход : $[\mathbf{p}]$; выход : $[t]$)

1 $[t] := 1$;
2 для $j = 1$ до $\bar{j}$
3 $\quad [\widetilde{x}_a] = (x_a(j) - p_1)\cos[p_2] + y_a(j)\sin[p_2]$;
4 $\quad [\widetilde{y}_a] = -(x_a(j) - p_1)\sin[p_2] + y_a(j)\cos[p_2]$;
5 $\quad [\widetilde{x}_b] = (x_b(j) - p_1)\cos[p_2] + y_a(j)\sin[p_2]$;
6 $\quad [\widetilde{y}_b] = -(x_a(j) - p_1)\sin[p_2] + y_a(j)\cos[p_2]$;
7 $\quad [\widetilde{\mathbf{a}}] := ([\widetilde{x}_a], [\widetilde{y}_a])^{\mathrm{T}}$; $[\widetilde{\mathbf{b}}] := ([\widetilde{x}_b], [\widetilde{y}_b])^{\mathrm{T}}$;
8 $\quad$ если $[\widetilde{\mathbf{a}}]$ находится внутри объекта, то $[t] := 0$; возврат;
9 $\quad$ если $[\widetilde{\mathbf{b}}]$ находится внутри объекта, то $t := 0$; возврат;
10 $\quad$ для $i = 1$ до $\bar{i}$
11 $\quad\quad [d_1] := \det\left(\mathbf{s}_i - [\widetilde{\mathbf{b}}], \mathbf{s}_i - [\widetilde{\mathbf{a}}]\right) * \det\left(\mathbf{s}_{i+1} - [\widetilde{\mathbf{b}}], \mathbf{s}_{i+1} - [\widetilde{\mathbf{a}}]\right)$;
12 $\quad\quad [d_2] = \det\left(\mathbf{s}_{i+1} - \mathbf{s}_i, \mathbf{s}_i - [\widetilde{\mathbf{a}}]\right) * \det\left(\mathbf{s}_{i+1} - \mathbf{s}_i, \mathbf{s}_i - [\widetilde{\mathbf{b}}]\right)$;
13 $\quad\quad$ если $([d_1] < 0)$ и $[d_2] < 0$, то $[t] := 0$; возврат;
14 $\quad\quad$ если $0 \in [d_1]$ или $0 \in [d_2]$ и $\left[[\widetilde{\mathbf{a}}] \cup [\widetilde{\mathbf{b}}]\right] \cap [\mathbf{s}_{i+1} \cup \mathbf{s}_i] \neq \varnothing$, то
15 $\quad\quad\quad [t] := [0, 1]$.

Соответствующее движение объекта показано на рис. 8.13. Как и ожидалось, два отрезка препятствия видны на рисунке (не затенены объектом ни в одном из его положений). При $\varepsilon = 0{,}2$ алгоритм FEASIBLEPATH1 не в состоянии найти допустимый маршрут и выдает сообщение об отказе.

Менее чем за 1 минуту алгоритм FEASIBLEPATH2 находит маршрут, показанный на рис. 8.14. Доказано, что параллелотопы, отмеченные серым цветом, лежат внутри множества $\mathbb{S}$, черные параллелотопы находятся вне $\mathbb{S}$, и ничего не известно относительно параллелотопов, отмеченных белым цветом. Алгоритм FEASIBLEPATH2 предпринимает усилия по бисекции и анализу зон C-пространства конфигураций (состояний), только когда это необходимо, что и объясняет его эффективность по сравнению с алгоритмом FEASIBLEPATH1.

Замечание 8.2. Конфигурация (состояние), показанное на рис. 8.15, является тупиковым, оно обычно возникает при попытке решить задачу «вручную». Это состояние соответствует точке z, указанной на рис. 8.12. ■

Замечание 8.3. Паралелотоп поиска $[\mathbf{p}_0] = [-28, 57] \times [-1{,}4, 2{,}7]$ был выбран достаточно малым, чтобы сделать его обозримым, но достаточно большим, чтобы охватывать весь маршрут. Та же задача была решена для параллелотопа поиска

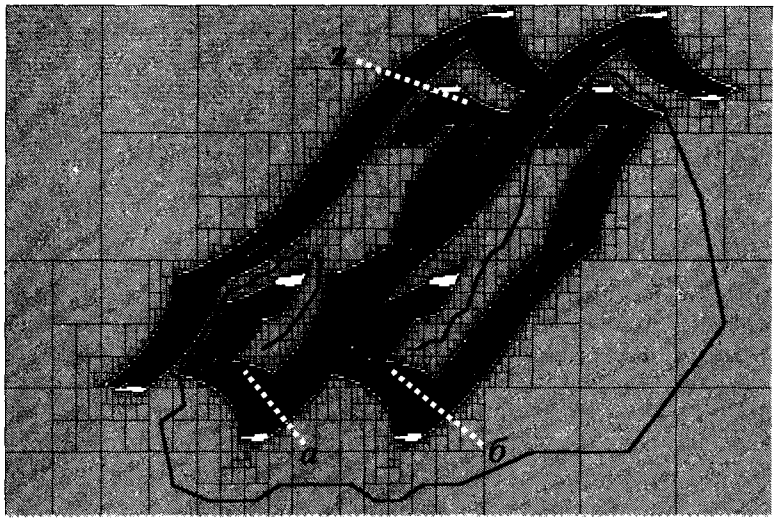


Рис. 8.12. Покрытие и маршрут, выработанные алгоритмом FEASIBLEPATH1; наружные размеры всего поля соответствуют параллелотопу поиска $[\mathbf{p}_0] = [-28, 57] \times [-1{,}4, 2{,}7]$; начальное *а* и конечное *б* состояния объекта; кусочная ломаная — маршрут; точка **z** соответствует тупиковому состоянию

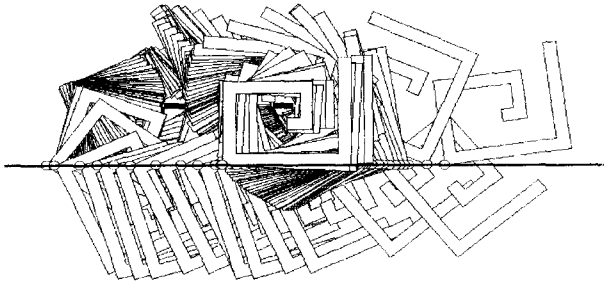


Рис. 8.13. Картина движения объекта; видны незатененные отрезки препятствия

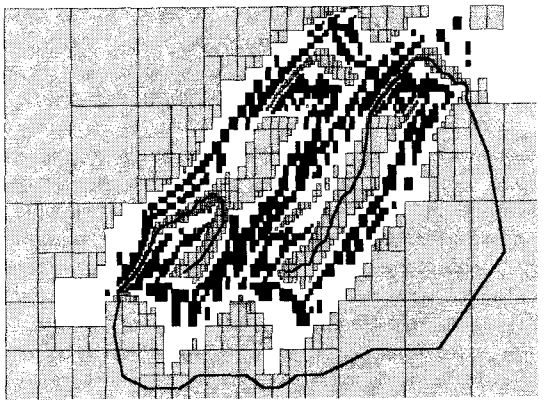


Рис. 8.14. Покрытие и маршрут, выработанные алгоритмом FEASIBLEPATH2; наружные размеры всего поля соответствуют параллелотопу поиска $[\mathbf{p}_0] = [-28, 57] \times [-1{,}4, 2{,}7]$

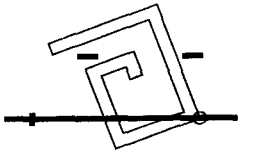


Рис. 8.15. Тупиковое состояние

$[\mathbf{p}_0] = [-100, 100] \times [-10, 10]$. Время вычислений по алгоритму FEASIBLEPATH1 в три раза больше, чем время, затраченное при прежнем параллелотопе $[\mathbf{p}_0]$, при том, что время работы алгоритма FEASIBLEPATH2 существенно не изменилось. ■

8.4. Определение местоположения и слежение за мобильным роботом

Автономное определение местоположения аппарата в частично известной среде является ключевой проблемой в мобильных роботах. Может использоваться целый набор чувствительных датчиков, каждый из которых выдает измерения с неопределенностью. Эти измерения долж-

ны быть объединены, и такая локализация является весьма типичной задачей объединения данных от источников информации [Crowley, 1989; Castelianos *et al.*, 1999]. Трудность такой задачи возрастает вследствие того, что карты окружающей среды, занесенные в робот, часто неточны или устарели.

Чтобы в динамике определить свое местоположение, робот, например, такой как на рис. 8.16, должен сперва оценить свое начальное положение и начальную ориентацию. Эта первоначальная задача является более сложной, чем последующее слежение. Разумеется, если более менее точное начальное местоопределение выполнено, часто становится возможным использовать хорошо отработанные приемы локального слежения, такие как обобщенная калмановская фильтрация [Leonard и Duttant–Whyte, 1991], или ее аналог при ограниченных ошибках [Hanebeck и Schmidt, 1996; Meizel *et al.*, 1996]. Мы будем предполагать, что первоначально роботу известно только, что он находится в некоторой области, описываемой картой. В этой связи определение начального местоположения относится к распознаванию образов [Drumheller, 1987] и выполняется с использованием деревьев представления данных [Crimson и Lozano-Pérez, 1987; Halbwachs и Meizel, 1997] и, совсем недавно, к глобальному оцениванию максимума правдоподобия [Olson, 2000].

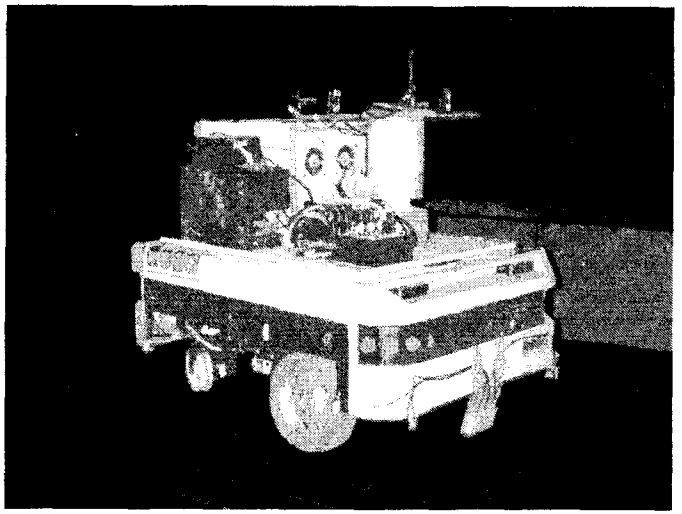


Рис. 8.16. Мобильный робот *Robuter* фирмы *Robosoft*

Метод, представленный в данной главе, делает возможным решать задачи местоопределения и слежения даже тогда, когда результат начального местоопределения неоднозначен или очень неточен. Метод основывается на предположении, что ошибки измерения и возмущения состояния принадлежат известным компактным множествам. Но данный метод будет сделан робастным по отношению к данным, которые могут и не удовлетворять указанному предположению.

Сначала будет рассмотрен самый простой статический случай, когда робот неподвижен. В параграфе 8.4.1 местоопределение робота формализуется как задача параметрического оценивания с ограничением на ошибки измерения. В параграфе 8.4.2 описывается модель измерений, даваемых ультразвуковыми радарами (сонарами). Конструкция функции включения, для получения выходных данных из этой модели, обсуждается в параграфе 8.4.3. Эта функция включения для этой модели позволяет использовать приемы, описанные в Главе 6. Параграф 8.4.4 поясняет как обрабатывать выбросы (*Прим. перев.*: в отечественной литературе и стандартах для недостоверных замеров принят термин — промахи), а иллюстрирующий пример рассматривается в параграфе 8.4.5.

Во второй части рассматривается случай движущегося робота, где задача параметрического оценивания заменяется на задачу оценивания состояния. Методология слежения, основанная на рекурсивном последовательном алгоритме оценивания состояния из параграфа 6.4, описывается в параграфе 8.4.6 и иллюстрируется в параграфе 8.4.7.

8.4.1. Формулировка статической задачи определения местоположения робота

Вычисления будут использовать две системы координат, а именно, систему координат $\mathcal{W}$ «внешнего мира» и локальную систему $\mathcal{R}$, связанную с роботом. Начало с системы координат $\mathcal{R}$ выбирается на середине оси между ведущими колесами. Координаты этого центра во внешней системе $\mathcal{W}$ обозначаются как x_c и y_c. Угол θ между системами $\mathcal{R}$ и $\mathcal{W}$ соответствует направлению движения робота (см. рис. 8.17). Точки и их координаты будут обозначаться строчными символами в системе $\mathcal{W}$ и строчными символами с тильдой в системе $\mathcal{R}$. Так, точка $\widetilde{\mathbf{m}}$ с координатами $(\widetilde{x}, \widetilde{y})$ из $\mathcal{R}$ в системе $\mathcal{W}$ обозначается как $\mathbf{m}$ с координатами

$$\mathbf{m} = \begin{pmatrix} x_c \\ y_c \end{pmatrix} + \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} \widetilde{x} \\ \widetilde{y} \end{pmatrix}. \tag{8.29}$$

Подлежат оцениванию три параметра, а именно, x_c, y_c и θ. Они образуют *вектор конфигурации* (состояния) $\mathbf{p} = (x_c, y_c, \theta)^T$ (рис. 8.17).

Задача состоит в оценивании значения вектора $\mathbf{p}$, полагаемого постоянным на рассматриваемом интервале времени, по *карте*, описывающей окружающую робота среду, и по *измерениям расстояния* от n_s бортовых сонаров по периметру робота, некоторые из них видны на рис. 8.16. Полагаются известными ограничения на ошибки измерений (величины ограничений могут быть получены специальными лабораторными экспериментами), а результирующие интервалы расстояний запоминаются в форме интервального вектора $[\mathbf{d}] = ([d_1], \ldots, [d_{n_s}])^{\mathrm{T}}$.

Полагается, что имеется некоторая модель расчета вектора $\mathbf{d}_m(\mathbf{p})$ расстояний, который возможен при положении $\mathbf{p}$ робота, и задача местоопределения робота становится известной задачей параметрического оценивания при ограниченных ошибках, именно, задачей построения множества

$$\mathbb{P} = \{\mathbf{p} \in [\mathbf{p}_0] \mid \mathbf{d}_m(\mathbf{p}) \in [\mathbf{d}]\}, \tag{8.30}$$

где $\mathbf{p}_0$ — параллелотоп начального поиска, который полагается достаточно большим, чтобы содержать все интересующие нас состояния. Тогда $\mathbb{P}$ содержит все векторы состояний, которые совместны с картой и замерами.

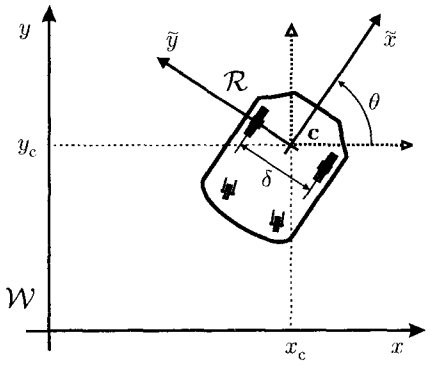


Рис. 8.17. Вектор положения робота $\mathbf{p} = (x_c, y_c, \theta)^{\mathrm{T}}$

Чтобы быть в состоянии построить вектор $\mathbf{d}_m(\mathbf{p})$ (параграф 8.4.2), мы поясним сначала, как карта описывает окружающую среду, и сформулируем предположения о процессе измерений.

Карта. Предполагается, что карта $\mathbb{M} = \{[\mathbf{a}_j, \mathbf{b}_j] \mid j = 1, \ldots, n_w\}$ окружающей среды состоит из n_w ориентированных отрезков $[\mathbf{a}_j, \mathbf{b}_j]$, которые

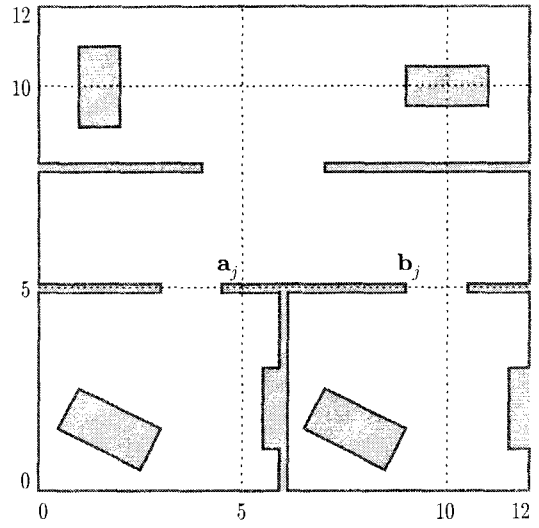


Рис. 8.18. Карта среды, окружающей робота; расстояния указаны в метрах

описывают местные ориентиры (стены, столбы и т. д.). Такая карта показана на рис. 8.18, где препятствия отмечены серым цветом. По договоренности, когда движение идет от $\mathbf{a}_j$ к $\mathbf{b}_j$, отражающая (внешняя) сторона отрезка препятствия находится слева. Полуплоскость $\Delta_{\mathbf{a}_j\mathbf{b}_j}$, расположенная с отражающей стороны отрезка $[\mathbf{a}_j, \mathbf{b}_j]$, описывается

$$\Delta_{\mathbf{a}_j\mathbf{b}_j} = \left\{\mathbf{m} \in \mathbb{R}^2 \mid \det\left(\overrightarrow{\mathbf{a}_j\mathbf{b}_j}, \overrightarrow{\mathbf{a}_j\mathbf{m}}\right) \geqslant 0\right\}. \tag{8.31}$$

ЗАМЕЧАНИЕ 8.4. Аналитический тест (8.31) будет часто использоваться для проверки принадлежности некоторой точки заданной полуплоскости. Другой вариант — использовать скалярное произведение векторов (см. рис. 8.19). ■

Измерения. Положение i-го датчика в системе координат $\mathcal{R}$ робота задается как $\tilde{\mathbf{s}}_i = (\tilde{x}_i, \tilde{y}_i)$. Этот датчик излучает ультразвуковые волны, распространяющиеся в некотором конусе с вершиной в точке $\tilde{\mathbf{s}}_i$, направлением оси $\tilde{\theta}_i$ и углом $\tilde{\gamma}_i$ полураствора конуса излучения (рис. 8.20).

Так как этот угол не зависит от системы координат, то $\tilde{\gamma}_i = \gamma_i$. Конус излучения будем обозначать как $\mathbb{E}(\tilde{\mathbf{s}}_i, \tilde{\theta}_i, \gamma_i)$. Излученная волна будет отражаться от объекта в окружающей робота среде. От объекта с шероховатой поверхностью происходит как зеркальное, так и диффузное отражение

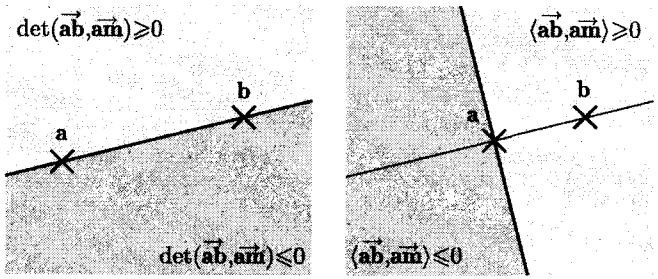


Рис. 8.19. Аналитические проверки для описания полуплоскости: *а*) граница, проходящая через точки **a** и **b**; *б*) граница ортогональна отрезку (**a**, **b**) и проходит через точку **a**; конусы и полоски получаются пересечением таких полуплоскостей

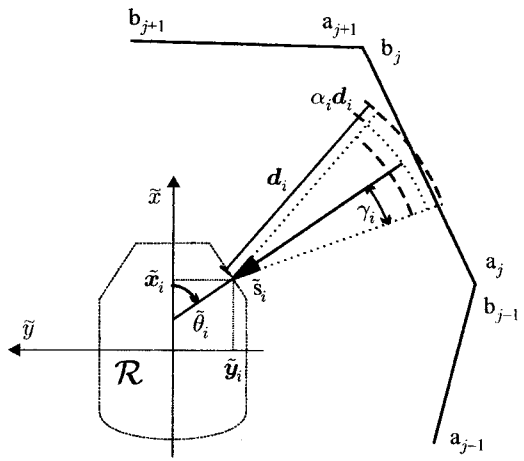


Рис. 8.20. Конус излучения сонара

[Hecht, 1987; Kuc и Siegel, 1987]. Угол падения волны на поверхность данного объекта определяет, дойдет ли отраженная волна до датчика. Можно определить *предельный угол падения* β_j, за которым отраженная волна не была бы принята, если бы угол γ_i полураствора диаграммы конуса излучения был нулевым. Когда γ_i — ненулевой, средний угол падения волны

должен быть меньше, чем $\gamma_i + (\beta/2)$, чтобы отраженная волна была принята [Kieffer *et al.*, 2001в]. Величина β зависит от шероховатости поверхности препятствия, представляемого j-м сегментом карты. Этот угол велик для грубых поверхностей и весьма мал для гладких поверхностей. Две типичных ситуации показаны на рис. 8.21.

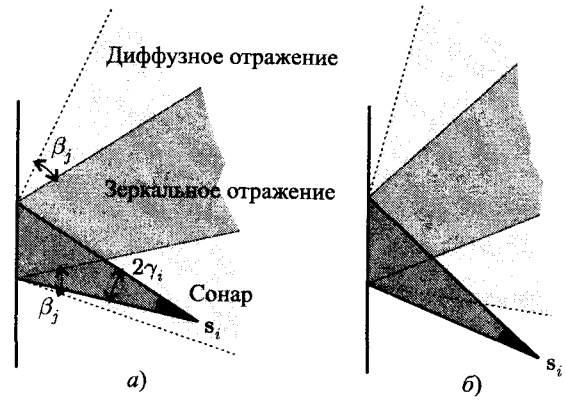


Рис. 8.21. Слева: отраженная волна принимается, если отражение диффузное; справа: никакая отраженная волна не принимается

Когда отраженная волна принимается i-м датчиком, время, прошедшее с момента излучения, пересчитывается в расстояние и интерпретируется как расстояние d_i между этим датчиком и поверхностью ближайшего объекта, по крайней мере частично находящегося внутри конуса излучения. На основе лабораторных экспериментов получают, что относительная ошибка такого дальномерного устройства будет предположительно принадлежать интервалу $[-\alpha_i, \alpha_i]$, где α — характеристика i-го сонара. Таким образом, предполагается, что интервал $[(1-\alpha_i)d_i, (1+\alpha_i)d_i]$ содержит истинное значение дальности до ближайшего препятствия, по крайней мере находящегося внутри конуса излучения.(*Прим. перев.*: данные формулы являются приближенными: для рассматриваемой области значений замеров $d_i > 0$ границы множества неопределенности замера оказывается смещенным вниз, а его ширина занижается, и истинное значение дальности может быть потеряно; для $d_i > 0$ точные формулы имеют вид $[d_i] = [d_i/(1+\alpha_i), d_i/(1-\alpha_i)]$.)

Однако иногда волна принимается только после нескольких переотражений (или даже вовсе не принимается). Строгая интерпретация результирующего измерения расстояния была бы слишком сложной, вместо этого

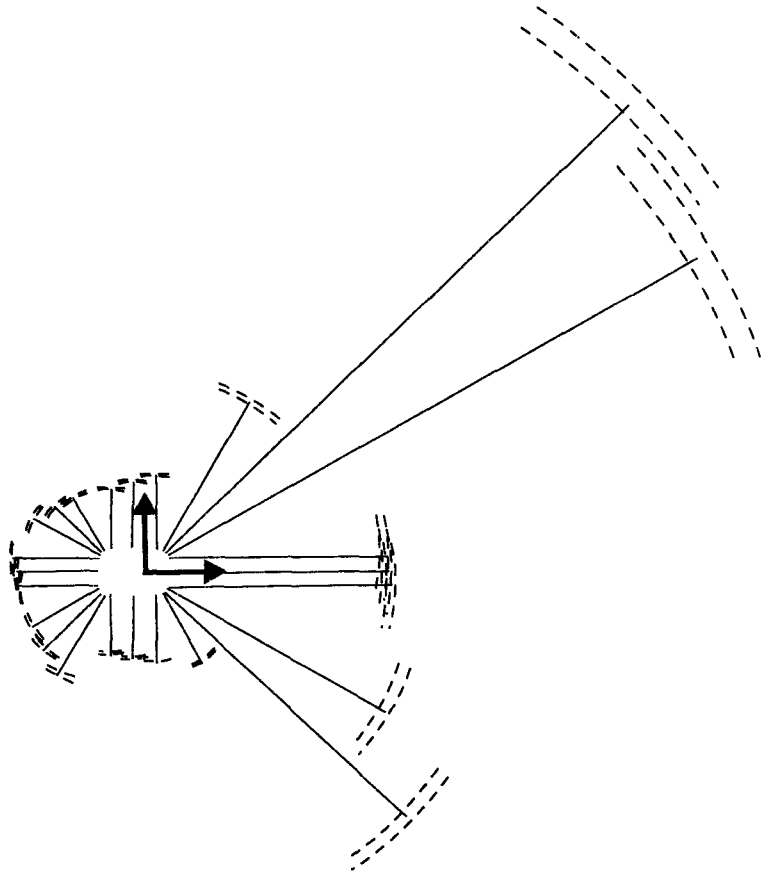


Рис. 8.22. Диаграмма излучения набора сонаров

мы будем рассматривать такие данные как выбросы, по отношению к которым оценивание положения робота должно быть сделано робастным. Почти всегда неизбежное присутствие таких выбросов делает задачу особенно трудной.

Замеры дальностей от n_s объектов могут присутствовать одновременно в *диаграмме* излучения (см. рис. 8.22). По результатам замеров сонара, каждый отрезок препятствия в диаграмме соответствует некоторой дальности d_i. Некоторое препятствие должно лежать по крайней мере частично между двумя соответствующими дугами окружностей, материализующих относительную ошибку α_i измерения дальности и угла γ_i полураствора конуса излучения данного сонара.

8.4.2. Модель процесса измерения

Простоты ради, одна и таже модель будет использоваться для описания всех сонаров и объектов среды, поэтому можно опустить индекс i в записи величин α_i и γ_i и индекс j в β_j. Для любого заданного датчика i и вектора положения $\mathbf{p}$ рассчитывается расстояние, которое можно было бы получить, если бы только один сегмент карты присутствовал. Эта операция повторяется для всех сегментов карты, чтобы рассчитать наименьшее из расстояний, которое берется в качестве i-й компоненты вектора $\mathbf{d}_m(\mathbf{p})$. Подробности этой операции приводятся ниже, но их чтение можно опустить и перейти прямо к параграфу 8.4.3.

Рассмотрим некоторый сонар с конусом излучения $\mathbb{E}(\widetilde{\mathbf{s}}, \widetilde{\theta}, \gamma)$. Для любого заданного состояния $\mathbf{p} = (x_c, y_c, \theta)^T$ конус $\mathbb{E}$ может быть эквивалентно описан в системе $\mathcal{W}$ его вершиной $\mathbf{s}(\mathbf{p})$ и двумя единичными векторами $\overrightarrow{\mathbf{u}_1}(\mathbf{p}, \widetilde{\theta}, \gamma)$ и $\overrightarrow{\mathbf{u}_2}(\mathbf{p}, \widetilde{\theta}, \gamma)$, соответствующими крайним образующим конуса:

$$\overrightarrow{\mathbf{u}_1} = \begin{pmatrix} \cos(\theta + \widetilde{\theta} - \gamma) \\ \sin(\theta + \widetilde{\theta} - \gamma) \end{pmatrix}, \qquad \overrightarrow{\mathbf{u}_2} = \begin{pmatrix} \cos(\theta + \widetilde{\theta} + \gamma) \\ \sin(\theta + \widetilde{\theta} + \gamma) \end{pmatrix}. \tag{8.32}$$

Опуская величины $\mathbf{p}$, $\widetilde{\theta}$ и γ в записи зависимостей, можно написать $\mathbb{E} = \mathbb{E}(\mathbf{s}, \overrightarrow{\mathbf{u}_1}, \overrightarrow{\mathbf{u}_2})$. Поскольку γ всегда меньше, чем $\pi/2$, то условием принадлежности любой точки $\mathbf{m} \in \mathbb{R}^2$ конусу излучения является

$$\mathbf{m} \in \mathbb{E}(\mathbf{s}, \overrightarrow{\mathbf{u}_1}, \overrightarrow{\mathbf{u}_2}) \Leftrightarrow \left(\det \overrightarrow{\mathbf{u}_1}, \overrightarrow{\mathbf{sm}}\right) \geqslant 0) \wedge \left(\det(\overrightarrow{\mathbf{u}_2}, \overrightarrow{\mathbf{sm}}) \leqslant 0\right) \tag{8.33}$$

(см. рис. 8.19, *а*).

Модель формирования величины дальности, измеренной сонаром, опирается на следующее определение, в котором $[\mathbf{a}, \mathbf{b}]$ является сегментом карты.

Определение 8.1. *Удаленность конуса излучения* $\mathbb{E}(\mathbf{s}, \overrightarrow{\mathbf{u}_1}, \overrightarrow{\mathbf{u}_2})$ *от сегмента* $[\mathbf{a}, \mathbf{b}]$ *карты, обозначаемая как* $r(\mathbf{s}, \overrightarrow{\mathbf{u}_1}, \overrightarrow{\mathbf{u}_2}, \mathbf{a}, \mathbf{b})$, *является наименьшим расстоянием между вершиной* $\mathbf{s}$ *и пересечением сегмента* $[\mathbf{a}, \mathbf{b}]$ *с конусом* $\mathbb{E}$, *при условии, что это пересечение существует и что вершина* $\mathbf{s}$ *принадлежит полуплоскости* $\Delta_{\mathbf{ab}}$, *расположенной с отражающей стороны сегмента* $[\mathbf{a}, \mathbf{b}]$; *в противном случае, удаленность полагается бесконечной.* ■

По (8.31), проверка принадлежности ($\mathbf{s} \in \Delta_{\mathbf{ab}}$) эквивалентна проверке ($\det(\overrightarrow{\mathbf{ab}}, \overrightarrow{\mathbf{as}}) \geqslant 0$), и всюду ниже мы предполагаем, что эти проверки верны. Далее необходимо оценить минимум величины $\|\overrightarrow{\mathbf{sm}}\|$, когда $\mathbf{m}$ описывает пересечение $[\mathbf{a}, \mathbf{b}] \cap \mathbb{E}$. Пусть $\mathbf{h}$ — ортогональная проекция $\mathbf{s}$ на линию $(\mathbf{a}, \mathbf{b})$. Если $\mathbf{h} \in [\mathbf{a}, \mathbf{b}] \cap \mathbb{E}$, то $r(\mathbf{s}, \overrightarrow{\mathbf{u}_1}, \overrightarrow{\mathbf{u}_2}, \mathbf{a}, \mathbf{b}) = \|\overrightarrow{\mathbf{sh}}\|$. Проверка условия $\mathbf{h} \in [\mathbf{a}, \mathbf{b}] \cap \mathbb{E}$ эквивалентна проверке принадлежности $\mathbf{s}$ полоске, ограниченной прямыми линиями, ортогональными к $[\mathbf{a}, \mathbf{b}]$ и проходящими через точки $\mathbf{a}$ и $\mathbf{b}$, поэтому

$$(\mathbf{h} \in [\mathbf{a}, \mathbf{b}]) \Leftrightarrow \left(\langle \overrightarrow{\mathbf{ab}}, \overrightarrow{\mathbf{as}} \rangle \geqslant 0\right) \wedge \left(\langle \overrightarrow{\mathbf{ba}}, \overrightarrow{\mathbf{bs}} \rangle \geqslant 0\right) \tag{8.34}$$

(см. рис. 8.19, *б*). Опираясь на (8.33), можно записать

$$(\mathbf{h} \in \mathbb{E}) \Leftrightarrow \left(\det(\overrightarrow{\mathbf{u}_1}, \overrightarrow{\mathbf{sh}}) \geqslant 0\right) \wedge \left(\det(\overrightarrow{\mathbf{u}_2}, \overrightarrow{\mathbf{sh}}) \leqslant 0\right). \tag{8.35}$$

Эта проверка не совсем обычна, потому что трудно получить координаты точки $\mathbf{h}$. Поскольку $\mathbf{s} \in \Delta_{\mathbf{ab}}$, то $\overrightarrow{\mathbf{sh}}/\|\overrightarrow{\mathbf{sh}}\|$ получается из $\overrightarrow{\mathbf{ab}}/\|\overrightarrow{\mathbf{ab}}\|$ поворотом на угол $-\pi/2$, поэтому соотношение (8.35) может быть записано как

$$(\mathbf{h} \in \mathbb{E}) \Leftrightarrow \left(\langle \overrightarrow{\mathbf{u}_1}, \overrightarrow{\mathbf{ab}} \rangle \leqslant 0\right) \wedge \left(\langle \overrightarrow{\mathbf{u}_2}, \overrightarrow{\mathbf{ab}} \rangle \geqslant 0\right). \tag{8.36}$$

Объединяя (8.34) и (8.36), получаем

$$\begin{aligned}(\mathbf{h} \in [\mathbf{a}, \mathbf{b}] \cap \mathbb{E}) \Leftrightarrow &\left(\langle \overrightarrow{\mathbf{ab}}, \overrightarrow{\mathbf{as}} \rangle \geqslant 0\right) \wedge \left(\langle \overrightarrow{\mathbf{ba}}, \overrightarrow{\mathbf{bs}} \rangle \geqslant 0\right) \\ &\wedge \left(\langle \overrightarrow{\mathbf{u}_1}, \overrightarrow{\mathbf{ab}} \rangle \leqslant 0\right) \wedge \left(\langle \overrightarrow{\mathbf{u}_2}, \overrightarrow{\mathbf{ab}} \rangle \geqslant 0\right).\end{aligned} \tag{8.37}$$

Если $\mathbf{h} \notin [\mathbf{a}, \mathbf{b}] \cap \mathbb{E}$, то удаленность $r(\mathbf{s}, \overrightarrow{\mathbf{u}_1}, \overrightarrow{\mathbf{u}_2}, \mathbf{a}, \mathbf{b})$ либо бесконечна (если $[\mathbf{a}, \mathbf{b}] \cap \mathbb{E} = \varnothing$), либо получена для одной из крайних точек сегмента $[\mathbf{a}, \mathbf{b}] \cap \mathbb{E}$. Эти крайние точки принадлежат множеству $\mathbb{K} = \{\mathbf{a}, \mathbf{b}, \mathbf{h}_1, \mathbf{h}_2\}$, где $\mathbf{h}_1$ и $\mathbf{h}_2$ — точки пересечений линии $(\mathbf{a}, \mathbf{b})$ с линиями $(\mathbf{s}, \overrightarrow{\mathbf{u}_1})$ и $(\mathbf{s}, \overrightarrow{\mathbf{u}_2})$. Для примера на рис. 8.23 $r(\mathbf{s}, \overrightarrow{\mathbf{u}_1}, \overrightarrow{\mathbf{u}_2}, \mathbf{a}, \mathbf{b}) = \|\overrightarrow{\mathbf{sb}}\|$. Проверка принадлежности некоторого элемента из $\mathbb{K}$ пересечению $[\mathbf{a}, \mathbf{b}] \cap \mathbb{E}$ легко выводится из (8.33). Для $\mathbf{v} \in \{\mathbf{a}, \mathbf{b}\}$ имеем

$$(\mathbf{v} \in \mathbb{E}) \Leftrightarrow \left(\det(\overrightarrow{\mathbf{u}_1}, \overrightarrow{\mathbf{sv}}) \geqslant 0\right) \wedge \left(\det(\overrightarrow{\mathbf{u}_2}, \overrightarrow{\mathbf{sv}}) \leqslant 0\right). \tag{8.38}$$

По построению, точки $\mathbf{h}_1$ и $\mathbf{h}_2$ принадлежат пересечению $\mathbb{E} \cap (\mathbf{a}, \mathbf{b})$; поэтому нужно только проверить их принадлежность отрезку $[\mathbf{a}, \mathbf{b}]$, что эквивалентно проверке их принадлежности конусу с вершиной $\mathbf{s}$ и крайними

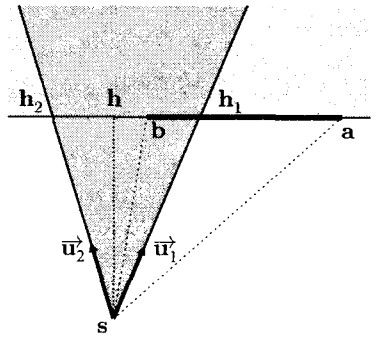


Рис. 8.23. Удаленность отдельного сегмента $[\mathbf{a}, \mathbf{b}]$ от датчика $\mathbf{s}$; полуплоскость, расположенная с неотражающей стороны сегмента $[\mathbf{a}, \mathbf{b}]$, отмечена светло-серым цветом; конус $\mathbb{E}$ излучения отмечен темно-серым

образующими $\overrightarrow{\mathbf{sa}}$ и $\overrightarrow{\mathbf{sb}}$. Из (8.33) получаем для $i = 1,2$

$$(\mathbf{h}_i \in [\mathbf{a}, \mathbf{b}] \cap \mathbb{E}) \Longleftrightarrow \left(\det(\overrightarrow{\mathbf{sa}}, \overrightarrow{\mathbf{u}_1}) \geqslant 0\right) \wedge \left(\det(\overrightarrow{\mathbf{sb}}, \overrightarrow{\mathbf{u}_i}) \leqslant 0\right). \tag{8.39}$$

Наконец, если ни точка $\mathbf{h}$, ни элементы из $\mathbb{K}$ не принадлежат пересечению $[\mathbf{a}, \mathbf{b}] \cap \mathbb{E}$, то пустота пересечения $[\mathbf{a}, \mathbf{b}] \cap \mathbb{E} = \varnothing$ доказана, и удаленность конуса $\mathbb{E}$ от отрезка $[\mathbf{a}, \mathbf{b}]$ полагается равной бесконечности.

Табл. 8.11 представляет функцию оценивания удаленности $r(\mathbf{s}, \overrightarrow{\mathbf{u}_1}, \overrightarrow{\mathbf{u}_2}, \mathbf{a}, \mathbf{b})$ конуса излучения от отдельного сегмента $[\mathbf{a}, \mathbf{b}]$, которая основывается на рассмотренных проверках. На Шаге 4 рассчитывается расстояние $\ell(\mathbf{s}, (\mathbf{a}, \mathbf{b}))$ от вершины $\mathbf{s}$ до линии $(\mathbf{a}, \mathbf{b})$ (рис. 8.24) по соотношению

$$\ell(\mathbf{s}, (\mathbf{a}, \mathbf{b})) = \|\overrightarrow{\mathbf{sh}}\| = \frac{|\det(\overrightarrow{\mathbf{ab}}, \overrightarrow{\mathbf{as}})|}{\|\overrightarrow{\mathbf{ab}}\|}, \tag{8.40}$$

и на Шаге 11 расстояние $\ell_{\overrightarrow{\mathbf{u}}}(\mathbf{s}, (\mathbf{a}, \mathbf{b}))$ от вершины $\mathbf{s}$ до линии $(\mathbf{a}, \mathbf{b})$ вдоль единичного вектора $\overrightarrow{\mathbf{u}}$ (рис. 8.24) находится по соотношению

$$\ell_{\overrightarrow{\mathbf{u}}}(\mathbf{s}, (\mathbf{a}, \mathbf{b})) = \|\overrightarrow{\mathbf{sm}}\| = \frac{\|\overrightarrow{\mathbf{ah}}\|}{|\sin\theta|} = \frac{|\det(\overrightarrow{\mathbf{ab}}, \overrightarrow{\mathbf{as}})|}{\|\overrightarrow{\mathbf{ab}}\|\,|\sin\theta|} = \frac{|\det(\overrightarrow{\mathbf{ab}}, \overrightarrow{\mathbf{as}})|}{|\overrightarrow{\mathbf{ab}}\,\overrightarrow{\mathbf{u}}|}. \tag{8.41}$$

Когда имеется n_{w} сегментов препятствия, то должен учитываться тот факт, что какие-нибудь из них могут не быть освещены сонаром, так как они

Таблица 8.11. Функция оценивания удаленности конуса излучения от отдельного сегмента препятствия

Алгоритм r(вход : $\mathbf{s}, \overrightarrow{\mathbf{u}_1}, \overrightarrow{\mathbf{u}_2}, \mathbf{a}, \mathbf{b}$; выход : r)

1 если $\left(\det(\overrightarrow{\mathbf{ab}}, \overrightarrow{\mathbf{as}}) < 0\right)$, то
2 $\quad r := +\infty$; возврат;
3 если $\left(\langle\overrightarrow{\mathbf{ab}}, \overrightarrow{\mathbf{as}}\rangle \geqslant 0\right) \wedge \left(\langle\overrightarrow{\mathbf{ba}}, \overrightarrow{\mathbf{bs}}\rangle \geqslant 0\right) \wedge \left(\langle\overrightarrow{\mathbf{u}_1}, \overrightarrow{\mathbf{ab}}\rangle \leqslant 0\right)$
$\quad \wedge \left(\langle\overrightarrow{\mathbf{u}_2}, \overrightarrow{\mathbf{ab}}\rangle \geqslant 0\right)$,
4 $\quad$ то $r_{\mathbf{h}} := \|\overrightarrow{\mathbf{sh}}\| = \ell(\mathbf{s}, (\mathbf{a}, \mathbf{b}))$, иначе $r_{\mathbf{h}} := +\infty$;
5 если $\left(\det(\overrightarrow{\mathbf{u}_1}, \overrightarrow{\mathbf{sa}}) \geqslant 0\right) \wedge \left(\det(\overrightarrow{\mathbf{u}_2}, \overrightarrow{\mathbf{sa}}) \leqslant 0\right)$,
6 $\quad$ то $r_{\mathbf{a}} := \|\overrightarrow{\mathbf{sa}}\|$, иначе $r_{\mathbf{a}} := +\infty$;
7 если $\left(\det(\overrightarrow{\mathbf{u}_1}, \overrightarrow{\mathbf{sb}}) \geqslant 0\right) \wedge \left(\det(\overrightarrow{\mathbf{u}_2}, \overrightarrow{\mathbf{sb}}) \leqslant 0\right)$,
8 $\quad$ то $r_{\mathbf{b}} := \|\overrightarrow{\mathbf{sb}}\|$, иначе $r_{\mathbf{b}} := +\infty$;
9 для $i := 1, 2$
10 $\quad$ если $\left(\det(\overrightarrow{\mathbf{sa}}, \overrightarrow{\mathbf{u}_i}) \geqslant 0\right) \wedge \left(\det(\overrightarrow{\mathbf{sb}}, \overrightarrow{\mathbf{u}_i}) \leqslant 0\right)$,
11 $\quad$ то $r_{\mathbf{h}_i} := \|\overrightarrow{\mathbf{sh}_i}\| = \ell_{\overrightarrow{\mathbf{u}_i}}(\mathbf{s}, (\mathbf{a}, \mathbf{b}))$, иначе $r_{\mathbf{h}_i} := +\infty$;
12 $r := \min(r_{\mathbf{h}}, r_{\mathbf{a}}, r_{\mathbf{b}}, r_{\mathbf{h}_1}, r_{\mathbf{h}_2})$.

лежат в тени других, более близких к датчику сегментов. Пусть $r_{ij}(\mathbf{p})$ — удаленность j-го сегмента, взятого как отдельный от i-го датчика при состоянии $\mathbf{p}$ робота

$$r_{ij}(\mathbf{p}) = r(\mathbf{s}_i(\mathbf{p}), \overrightarrow{\mathbf{u}_{1i}}(\mathbf{p}, \widetilde{\theta}_i, \gamma), \overrightarrow{\mathbf{u}_{2i}}(\mathbf{p}, \widetilde{\theta}_i, \gamma), \mathbf{a}_j, \mathbf{b}_j). \tag{8.42}$$

Далее, модель измерения, которая будет реализована i-м датчиком при состоянии $\mathbf{p}$, описывается как

$$(\mathbf{d}_{\mathrm{m}})_i(\mathbf{p}) = \min_{j=1,\ldots,n_{\mathrm{w}}} r_{ij}(\mathbf{p}). \tag{8.43}$$

Оцениванием (8.43) для $i = 1, \ldots, n_{\mathrm{s}}$ получаем вектор $\mathbf{d}_{\mathrm{m}}(\mathbf{p})$, который сравнивается с интервальными данными $[\mathbf{d}]$ по расстоянию. Функция $\mathbf{d}_{\mathrm{m}}(\mathbf{p})$ оценивается по алгоритму, представленному в табл. 8.12.

ЗАМЕЧАНИЕ 8.5. Сложность оценивания вектора дальностей $\mathbf{d}_{\mathrm{m}}$ — билинейна по величинам n_{s} и n_{w}. ■

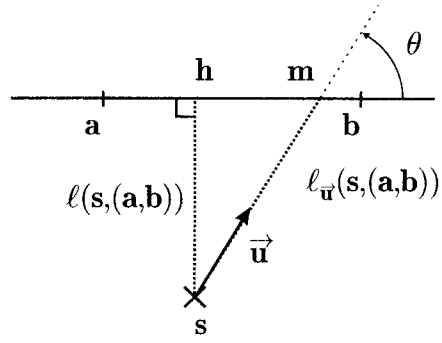


Рис. 8.24. Дальности от точки $\mathbf{s}$ до линии $(\mathbf{a}, \mathbf{b})$

Таблица 8.12. Функция вычисления дальностей, ожидаемых в состоянии $\mathbf{p}$

Алгоритм $\mathbf{d}_{\mathrm{m}}$(вход : $\mathbf{p}$; выход : $\mathbf{d}_{\mathrm{m}}$)

1 для $i := 1$ до n_{s}
2 $\quad \mathbf{s}_i := \begin{pmatrix} x_{\mathrm{c}} \\ y_{\mathrm{c}} \end{pmatrix} + \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \widetilde{\mathbf{s}}_i$;
3 $\quad \overrightarrow{\mathbf{u}_{1i}} := \begin{pmatrix} \cos(\theta + \widetilde{\theta}_i - \gamma) \\ \sin(\theta + \widetilde{\theta}_i - \gamma) \end{pmatrix}$; $\overrightarrow{\mathbf{u}_{2i}} := \begin{pmatrix} \cos(\theta + \widetilde{\theta}_i + \gamma) \\ \sin(\theta + \widetilde{\theta}_i + \gamma) \end{pmatrix}$;
4 $\quad (\mathbf{d}_{\mathrm{m}})_i(\mathbf{p}) := +\infty$;
5 $\quad$ для $j := 1$ до n_{w}
6 $\quad\quad (\mathbf{d}_{\mathrm{m}})_i(\mathbf{p}) := \min((\mathbf{d}_{\mathrm{m}})_i(\mathbf{p}), r(\mathbf{s}_i, \overrightarrow{\mathbf{u}_{1i}}, \overrightarrow{\mathbf{u}_{2i}}, \mathbf{a}_j, \mathbf{b}_j))$.

8.4.3. Обращение множеств

Задача описания множества

$$\mathbb{P} = \{\mathbf{p} \in [\mathbf{p}_0] \mid \mathbf{d}_m(\mathbf{p}) \in [\mathbf{d}]\} = [\mathbf{p}_0] \cap (\mathbf{d}_m)^{-1}([\mathbf{d}]) \tag{8.44}$$

может далее быть решена использованием алгоритма SIVIA, описанного в табл. 3.1 (с. 82). Единственным предварительным условием является наличие некоторой функции включения $[\mathbf{d}_{\mathrm{m}}](\cdot)$ для функции $\mathbf{d}_{\mathrm{m}}(\cdot)$. Модельная функция $\mathbf{d}_{\mathrm{m}}(\cdot)$ основана на оценивании удаленности и содержит некоторое число разветвлений по логическим условиям. При оценивании параллелотопа $[\mathbf{d}_{\mathrm{m}}]([\mathbf{p}])$ на любом заданном параллелотопе из пространства параметров надо решать, какая ветвь (ветви) должна быть выполнена. Рассмотрим сейчас метод, предложенный в [Jaulin *et al.*, 2000] и основанный на понятии

χ-функции [Kearfott, 1996a]. Если t — булевский результат проверки, а y и z — два некоторых вещественных числа, то

$$\chi(t,\,y,\,z) = \begin{cases} y, & \text{если } t = 1, \\ z, & \text{если } t = 0. \end{cases} \tag{8.45}$$

Интервальный аналог функции $\chi(t, y, z)$ имеет вид

$$[\chi]([t],[y],[z]) = \begin{cases} [y], & \text{если } [t] = 1, \\ [z], & \text{если } [t] = 0, \\ [y] \sqcup [z], & \text{если } [t] = [0,1]. \end{cases} \tag{8.46}$$

Результат оценивания проверки, основанной на $[\chi]$, всегда является интервалом.

Интервальный аналог алгоритма табл. 8.11 приведен в табл. 8.13. В табл. 8.13 вектор $\overrightarrow{\mathbf{a}[\mathbf{s}]}$ сохраняется для множества всех векторов с началом в точке $\mathbf{a}$ и крайней точкой из параллелотопа $[\mathbf{s}]$.

Параллелотоп $[\mathbf{s}]$, гарантированно содержащий положение датчика $\mathbf{s}$ для любого состояния робота из параллелотопа $[\mathbf{p}] = ([x_c], [y_c], [\theta])^{\mathrm{T}}$, оценивается заменой всех вещественных переменных, появляющихся в (8.29), их интервальными аналогами. Подобным образом, характеристики конуса (8.32) оцениваются как $[\mathbb{E}] = \mathbb{E}([\mathbf{s}], \overrightarrow{[\mathbf{u}_1]}, \overrightarrow{[\mathbf{u}_2]})$. Наконец, минимум из двух интервалов находится как

$$\min([a],[b]) = \left[\min(\underline{a},\underline{b}), \min(\overline{a},\overline{b})\right], \tag{8.47}$$

и эта формула непосредственно расширяется на большее число интервалов.

ЗАМЕЧАНИЕ 8.6. Алгоритм табл. 8.13 может быть ускорен удалением Шага 12. Цена, которую придется заплатить, — возможное увеличение числа выбросов (см. параграф 8.4.4). ■

Теперь тривиально получить функцию включения $[\mathbf{d}_{\mathrm{m}}](\cdot)$ для функции $\mathbf{d}_{\mathrm{m}}(\cdot)$, опираясь на табл. 8.12 и 8.13. Эта функция включения представлена в табл. 8.14. Сложность оценивания $[\mathbf{d}_{\mathrm{m}}]$ — билинейна по величинам n_{s} и n_{w}.

8.4.4. Обработка выбросов

В связи с задачей определения местоположения робота, выбросы являются почти неизбежными. Выбросы — это данные, на величинах которых

Таблица 8.13. Функция включения для функции оценивания удаленности конуса излучения от отдельного сегмента препятствия

Алгоритм $[r]$(вход : $[\mathbf{s}], [\overrightarrow{\mathbf{u}_1}], [\overrightarrow{\mathbf{u}_2}], \mathbf{a}, \mathbf{b}$; выход : $[r]$)

1 $[t_r] := \left(\det(\overrightarrow{\mathbf{ab}}, \overrightarrow{\mathbf{as}}) \geqslant 0\right)$;
  если $[t_r] = 0$, то $[r] := +\infty$; возврат;
2 $[t_{\mathbf{h}}] := \left(\left\langle \overrightarrow{\mathbf{ab}}, \overrightarrow{\mathbf{a}[\mathbf{s}]} \right\rangle \geqslant 0\right) \wedge \left(\left\langle \overrightarrow{\mathbf{ba}}, \overrightarrow{\mathbf{b}[\mathbf{s}]} \right\rangle \geqslant 0\right)$
  $\wedge \left(\left\langle \overrightarrow{\mathbf{ab}}, \overrightarrow{[\mathbf{u}_1]} \right\rangle \leqslant 0\right) \wedge \left(\left\langle \overrightarrow{\mathbf{ab}}, \overrightarrow{[\mathbf{u}_2]} \right\rangle \geqslant 0\right)$;
3 $[r_{\mathbf{h}}] := [\chi]([t_{\mathbf{h}}], [\ell]([\mathbf{s}], (\mathbf{a}, \mathbf{b})), +\infty)$;
4 $[t_{\mathbf{a}}] := \left(\det(\overrightarrow{[\mathbf{u}_1]}, \overrightarrow{[\mathbf{s}]\mathbf{a}}) \geqslant 0\right) \wedge \left(\det(\overrightarrow{[\mathbf{u}_2]}, \overrightarrow{[\mathbf{s}]\mathbf{a}}) \leqslant 0\right)$;
5 $[r_{\mathbf{a}}] := [\chi]([t_{\mathbf{a}}], \|\overrightarrow{[\mathbf{s}], (\mathbf{a}}\|, +\infty)$;
6 $[t_{\mathbf{b}}] := \left(\det(\overrightarrow{[\mathbf{u}_1]}, \overrightarrow{[\mathbf{s}]\mathbf{b}}) \geqslant 0\right) \wedge \left(\det(\overrightarrow{[\mathbf{u}_2]}, \overrightarrow{[\mathbf{s}]\mathbf{b}}) \leqslant 0\right)$;
7 $[r_{\mathbf{b}}] := [\chi]([t_{\mathbf{b}}], \|\overrightarrow{[\mathbf{s}]\mathbf{b}}\|, +\infty)$;
8 для $i := 1$ до 2
9 $[t_{\mathbf{h}_i}] := \left(\det(\overrightarrow{[\mathbf{s}]\mathbf{a}}, \overrightarrow{[\mathbf{u}_i]}) \geqslant 0\right) \wedge \left(\det(\overrightarrow{[\mathbf{s}]\mathbf{b}}, \overrightarrow{[\mathbf{u}_i]}) \leqslant 0\right)$;
10 $[r_{\mathbf{h}_i}] := [\chi]\left([t_{\mathbf{h}_i}], \left[\ell_{\overrightarrow{[\mathbf{u}_i]}}\right]([\mathbf{s}], (\mathbf{a}, \mathbf{b})), +\infty\right)$;
11 $[r] := \min([r_{\mathbf{h}}], [r_{\mathbf{a}}], [r_{\mathbf{b}}], [r_{\mathbf{h}_1}], [r_{\mathbf{h}_2}])$;
12 $[r] := [\chi]([t_r], [r], +\infty)$.

Таблица 8.14. Функция включения для модели измерения

Алгоритм $[\mathbf{d}_{\mathrm{m}}]$(вход : $[\mathbf{p}]$); выход : $[\mathbf{d}_{\mathrm{m}}]$

1 для $i := 1$ до n_{s}
2 $[\mathbf{s}_i] := \begin{pmatrix} x_{\mathrm{c}} \\ y_{\mathrm{c}} \end{pmatrix} + \begin{pmatrix} \cos[\theta] & -\sin[\theta] \\ \sin[\theta] & \cos[\theta] \end{pmatrix} \widetilde{\mathbf{s}}_i$;
3 $\overrightarrow{[\mathbf{u}_{1i}]} := \begin{pmatrix} \cos([\theta] + \widetilde{\theta}_i - \gamma) \\ \sin([\theta] + \widetilde{\theta}_i - \gamma) \end{pmatrix}$; $\overrightarrow{[\mathbf{u}_{2i}]} := \begin{pmatrix} \cos([\theta] + \widetilde{\theta}_i + \gamma) \\ \sin([\theta] + \widetilde{\theta}_i + \gamma) \end{pmatrix}$;
4 $[\mathbf{d}_{\mathrm{m}}]_i([\mathbf{p}]) := +\infty$;
5 для $j := 1$ до n_{w}
6 $[\mathbf{d}_{\mathrm{m}}]_i([\mathbf{p}]) := \min\left([\mathbf{d}_m]_i([\mathbf{p}]), [r]([\mathbf{s}_i], \overrightarrow{[\mathbf{u}_{1i}]}, \overrightarrow{[\mathbf{u}_{2i}]}, \mathbf{a}_j, \mathbf{b}_j)\right)$.

нарушаются гипотезы об ограниченности ошибки измерений. Такие замеры могут возникать из-за многократных отражений, присутствия людей или мебели, неисправностей датчика, устаревания карты и т. д. В присутствии таких выбросов множество $\mathbb{P}$, определяемое по (8.30), может стать пустым. Введение функции релаксации (см. параграф 6.3.3, с. 206)

$$\lambda(\mathbf{p}) = \frac{\sum_{i=1}^{n_s} \pi_{[d_i]}(\mathbf{p})}{n_s}, \tag{8.48}$$

где

$$\pi_{[d_i]}(\mathbf{p}) = \begin{cases} 1, & \text{если } (\mathbf{d}_m)_i(\mathbf{p}) \in [d_i], \\ 0, & \text{если } (\mathbf{d}_m)_i(\mathbf{p}) \notin [d_i], \end{cases} \tag{8.49}$$

и построение множества

$$\mathbb{P}^q = \left\{ \mathbf{p} \in [\mathbf{p}_0] \mid \lambda(\mathbf{p}) \geqslant 1 - \frac{q}{n_s} \right\} \tag{8.50}$$

допускает наличие до q выбросов. Для выбора величины q можно применить алгоритм GOMNE из работы [Jaulin *et al.*, 1996], суть которого была описана в параграфе 6.3.3 (с. 206).

ЗАМЕЧАНИЕ 8.7. В [Kieffer *et al.*, 1999; 2000] были предложены проверки, которые дают возможность быстро исключать большие куски априорной области поиска при выполнении обращения множества, и, следовательно, весьма существенно ускоряя местоопределение. Следующий пример рассматривается с использованием этих проверок. ■

8.4.5. Пример статической задачи определения местоположения

Хотя этот пример и основан на моделировании, он является весьма реалистичным, и подобные результаты были получены на реальных данных [Lévêque, 1998]. Здесь характеристики робота — такие же, как и у робота на рис. 8.16, который снабжен $n_s = 24$ сонарами. Для каждого из них были экспериментально найдены угол $\gamma = 0{,}2$ рад полураствора конуса излучения и максимальная величина относительной ошибки $\alpha = 2\,\%$ замера дальности в рабочем диапазоне сонара.

Робот помещен в среду, описанную картой на рис. 8.18. Все препятствия имели конечный угол $\beta = 0{,}6$ рад падения излучения. Начальное (неизвестное) положение робота было $(x_c, y_c, \theta) = (8\text{ м}, 3{,}5\text{ м}, 6\text{ рад})$. Замеры дальности, полученные бортовыми сонарами, приведены в табл. 8.15 и соответствуют диаграмме излучения как на рис. 8.22. Чтобы промоделировать ситуацию, которая может привести к появлению выбросов, когда

Таблица 8.15. Замеры дальности, полученные от сонаров

Датчик i	1	2	3	4	5	6	7	8
d_i, м	3,24	3,21	9,02	9,60	2,58	1,11	1,01	0,91

Датчик i	9	10	11	12	13	14	15	16
d_i, м	0,89	0,95	1,10	1,27	1,21	1,14	1,14	1,21

Датчик i	17	18	19	20	21	22	23	24
d_i, м	1,39	0,91	0,96	1,02	1,19	4,95	3,71	3,30

угол падения волны, излученной конкретным сонаром, больше, чем $\gamma + (\beta/2)$, замер дальности брался случайным образом из диапазона между 0,5 м и 10 м с равномерным распределением. Параллелотоп поиска $[\mathbf{x}_0]$ брался $[0\text{ м}, 12\text{ м}] \times [0\text{ м}, 12\text{ м}] \times [0\text{ рад}, 2\pi\text{ рад}]$.

Процедура статического местоопределения не находит решения до тех пор, пока в замерах допускается не менее трех выбросов. Когда число выбросов $q = 3$, аппроксимация сверху $\overline{\mathbb{P}}^3$ множества решения, показанная на рис. 8.25, состоит из двух несвязных подмножеств, одно из которых *гарантированно* содержит истинное состояние, что доказывает присутствие не более чем трех выбросов. Эта аппроксимация сверху описывается как

$$\overline{\mathbb{P}}^3 \subset [7{,}93\text{ м}, 8{,}07\text{ м}] \times [3{,}43\text{ м}, 3{,}57\text{ м}] \times [5{,}90\text{ рад}, 6{.}10\text{ рад}]$$
$$\cup\, [1{,}93\text{ м}, 2{,}07\text{ м}] \times [3{,}43\text{ м}, 3{,}57\text{ м}] \times [5{,}90\text{ рад}, 6{,}10\text{ рад}].$$

Рис. 8.26 показывает два состояния, принадлежащие множеству $\overline{\mathbb{P}}^3$. Замеры, являющиеся выбросами, выделены жирными штриховыми дугами. (*Прим. перев.*: в каждом представленном расположении и ориентации робота указанным образом отмечаются по три выброса информации — ложные замеры на трех направлениях.) Неоднозначность местоопределения является следствием локальной симметричности карты и, разумеется, не может быть интерпретирована как недостаток метода оценивания. Только обращение гарантированного множества обнаруживает существование задачи идентификации, в то время как большинство других методов местоопределения ограничились бы выдачей точечной оценки состояния робота, без какого бы то ни было предупреждения о возможном существовании радикально иных решений.

Когда количество q выбросов увеличивается, размер аппроксимации сверху множества решения (вычисленного добавлением размеров параллелотопов $\overline{\mathbb{P}}^q$) может также увеличиваться вследствие того, что некоторые ин-

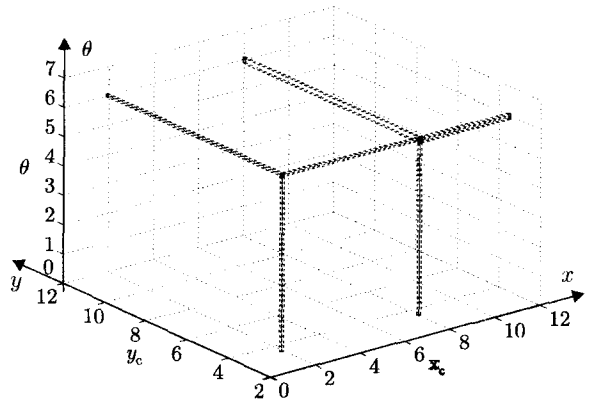


Рис. 8.25. Аппроксимация сверху $\overline{\mathbb{P}}^3$ и ее проекции

Таблица 8.16. Результаты работы процедуры статического местоопределения в зависимости от максимально допустимого числа выбросов

q	Размер $\overline{\mathbb{P}}^q$	Время вычислений, с	Суммарное время, с
0	0	25	25
1	0	48	73
2	0	89	162
3	0,0035	129	291
4	0,0043	167	458
5	0,0078	216	674

формативные замеры будут теперь игнорироваться. Таким образом, нужно принимать некий компромисс между робастностью и точностью алгоритма оценивания.

В рассматриваемом примере, при $q = 5$, размер множества $\overline{\mathbb{P}}^q$ заметно увеличивается при появлении дополнительных несвязных компонент, указывающих на то, что замеры, рассматриваемые как выбросы, не всегда являются таковыми. Объем вычислений также увеличивается, так как исключение некоторых частей исходного параллелотопа поиска становится все более трудным. Табл. 8.16 показывает полученные размеры аппроксимирующего множества, время вычислений и суммарное время вычислений при количестве выбросов от $q = 0$ до $q = 5$, при расчетах на компьютере Pentium-II 450.

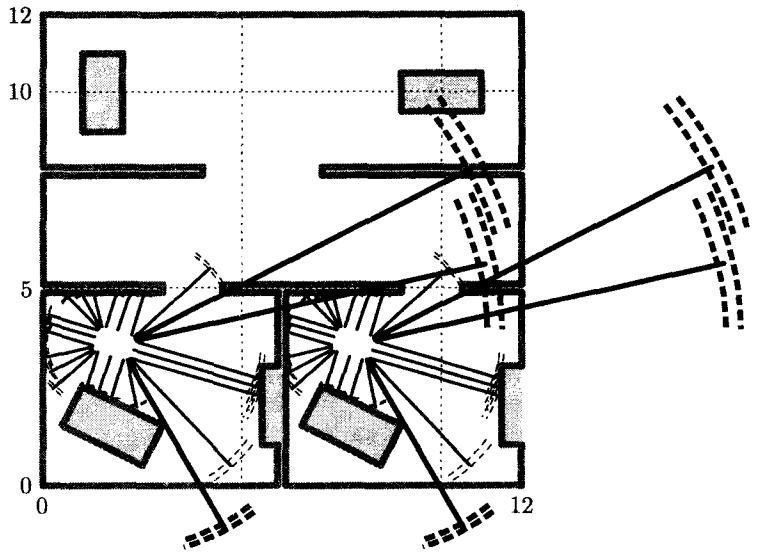


Рис. 8.26. Два возможных состояния робота (расположения и ориентации), полученные процедурой местоопределения; выбросы информации отмечены жирными лучами и дугами

ЗАМЕЧАНИЕ 8.8. В [Lévêque, 1998] результаты, даваемые одной ранней версией этого алгоритма местоопределения, сравнивались с результатами, полученными с помощью обобщенного фильтра Калмана. Как и ожидалось, временные затраты, разумеется, были в пользу последнего, но там надо было заранее удалять выбросы и вводить связь между замерами и объектами среды на карте перед тем, как применять фильтр Калмана. Разумеется, обобщенный фильтр Калмана не был в состоянии обнаружить неоднозначность местоопределения вследствие локальной симметричности карты. ■

8.4.6. Слежение

Предположим теперь, что робот может передвигаться. Следовательно, его конфигурация является функцией времени, и будем называть ее *состоянием*. Для оценивания этого состояния в реальном времени будем использовать алгоритм рекурсивного последовательного оценивания, приведенный в табл. 6.13 (с. 233). При этом в расчет могут браться только результаты $\overline{\mathbb{P}}_0 = \overline{\mathbb{P}}^q$ начального статического местоопределения и замеры дальности, получаемые во время движения робота.

На каждом временно́м шаге с номером k алгоритм вычисляет некоторое множество $\overline{\mathbb{P}}_k$, гарантированно содержащее все значения состояния, совместные с информацией, накопленной к шагу с номером k, на основе этапов возможного прогнозирования и корректирования информации. Напомним, что уравнения состояния (6.91) модели, лежащие в основе алгоритма оценивания состояния, включали две функции $\mathbf{f}(\cdot)$ и $\mathbf{g}(\cdot)$. Предсказание развития вектора состояния на шаге времени выполняется с помощью функции $\mathbf{f}(\cdot)$, а функция $\mathbf{g}(\cdot)$ моделирует замеры на k-м шаге и используется для коррекции информации. Пренебрегая перемещением, имеющем место во время выполнения замера на шаге с номером k (*Прим. перев.*: т. е. условно считая замер мгновенным.), можно рассматривать этап коррекции как статическую процедуру, поэтому расчет вектора $\mathbf{d}_m(\cdot)$ играет роль функции наблюдения $\mathbf{g}(\cdot)$, и надо построить только функцию $\mathbf{f}(\cdot)$.

Полагается, что робот движется достаточно медленно, чтобы описание этого движения кинематическими уравнениями было реалистичным. Компоненты вектора состояния удовлетворяют уравнениям

$$\begin{aligned}\frac{d\theta}{dt} &= \rho\frac{\omega_\mathrm{r}-\omega_\mathrm{l}}{\delta},\\ \frac{dx_\mathrm{c}}{dt} &= \rho\frac{\omega_\mathrm{r}-\omega_\mathrm{l}}{2}\cos\theta,\\ \frac{dy_\mathrm{c}}{dt} &= \rho\frac{\omega_\mathrm{r}-\omega_\mathrm{l}}{2}\sin\theta,\end{aligned} \tag{8.51}$$

где ω_l и ω_r — непрерывные угловые скорости вращения левого и правого ведущих колес, соответственно; ρ — радиус этих колес; δ — среднее расстояние между точками контакта колес с землей (см. рис. 8.17). Поведение робота управляется действием входных переменных (управлений) ω_l и ω_r. Точная дискретизация системы (8.51) по времени выполняется в предположении, что скорости вращения ведущих колес постоянны на периоде τ наблюдения. Ориентация робота в момент с номером $k+1$ по его ориентации в момент с номером k описывается как

$$\theta_{k+1} = \theta_k + \tau\rho\frac{\omega_\mathrm{r}-\omega_\mathrm{l}}{\delta}. \tag{8.52}$$

Если скорости ω_l и ω_r одинаковы, то координаты центра робота рассчитываются

$$\begin{aligned}(x_\mathrm{c})_{k+1} &= (x_\mathrm{c})_k + \tau\rho\frac{\omega_\mathrm{r}+\omega_\mathrm{l}}{2}\cos\theta_k,\\ (y_\mathrm{c})_{k+1} &= (y_\mathrm{c})_k + \tau\rho\frac{\omega_\mathrm{r}+\omega_\mathrm{l}}{2}\sin\theta_k,\end{aligned} \tag{8.53}$$

если же скорости ω_l и ω_r неодинаковы, то расчетные формулы записываются

$$\begin{aligned}(x_\mathrm{c})_{k+1} &= (x_\mathrm{c})_k + \delta\frac{\omega_\mathrm{r}+\omega_\mathrm{l}}{\omega_\mathrm{r}-\omega_\mathrm{l}}\cos\left(\theta_k+\tau\rho\frac{\omega_\mathrm{r}-\omega_\mathrm{l}}{2\delta}\right)\sin\left(\tau\rho\frac{\omega_\mathrm{r}-\omega_\mathrm{l}}{2\delta}\right),\\ (y_\mathrm{c})_{k+1} &= (y_\mathrm{c})_k + \delta\frac{\omega_\mathrm{r}+\omega_\mathrm{l}}{\omega_\mathrm{r}-\omega_\mathrm{l}}\sin\left(\theta_k+\tau\rho\frac{\omega_\mathrm{r}-\omega_\mathrm{l}}{2\delta}\right)\sin\left(\tau\rho\frac{\omega_\mathrm{r}-\omega_\mathrm{l}}{2\delta}\right).\end{aligned} \tag{8.54}$$

При поворотах расстояние δ между точками контакта ведущих колес точно неизвестно. Вот почему δ будет браться как интервал $[\delta] = [0{,}57\text{ м}, 0{,}63\text{ м}]$. Прогноз развития состояния робота зависит от нелинейности уравнений по $[\delta]$, и отклонение между истинным значением δ и средней точкой интервала $[\delta]$ может рассматриваться как ограниченное по величине возмущение состояния.

Этап коррекции информации выполняется применением статической процедуры местоопределения. Исследуется только область состояний, соответствующая множеству, полученному на предыдущем этапе прогнозирования, что эффективно ускоряет процедуру коррекции.

Возможные выбросы должны приниматься во внимание только на этапе коррекции информации и, таким образом, обрабатываться в статическом режиме, за исключением условия, что максимально допустимое число q выбросов может теперь зависеть от номера момента времени k. Одним из возможных правил подстройки числа q_k является поиск первого множества векторов состояния, совместного со всеми замерами расстояний, полученных в момент с номером k, при этом $q_k = 0$, и далее увеличивать q_k на единицу до тех пор, пока множество $\overline{\mathbb{P}}_k^{q_k}$ остается пустым.

На практике можно посоветовать придавать q_k бóльшее значение, чем то, что потребовалось для получения непустого множества $\overline{\mathbb{P}}_k^{q_k}$. По сути, рассматриваемая здесь процедура робастного оценивания состояния является гарантирующей, только если на каждом временном шаге действительное число выбросов меньше или равно подстраиваемому максимально допустимому числу q_k выбросов. В примере, рассматриваемом в следующем параграфе, предупреждающее действие было реализовано путем задания на каждом шаге максимально допустимого числа выбросов, на единицу бóльшим числа, необходимого для получения непустого множества $\overline{\mathbb{P}}_k^{q_k}$. Эта предупреждающая мера, разумеется, обычно проходит за счет ухудшения точности оценивания, поскольку все бóльшим числом информативных замеров приходится пренебрегать, и опять необходим компромисс между робастностью и точностью оценивания.

8.4.7. Пример

Рассмотрим снова случай параграфа 8.4.5, но уже при движении робота. Карта окружающей среды по-прежнему описывается рис. 8.18, и начальная аппроксимация сверху $\overline{\mathbb{P}}_0$ вектора состояния соответствует результату статического местоопределения, выполненного в параграфе 8.4.5.

В реальности робот движется снизу вверх из комнаты, расположенной на рис. 8.18 справа. Новые измерения принимаются через каждую секунду для коррекции прогнозируемого множества, содержащего истинное состояние робота. Эволюция проекции множества $\overline{\mathbb{P}}_k^{q_k}$ на плоскость (x, y) по шагам k показана в левой стороне рис. 8.27 и 8.28 (*Прим. перев.*: отсчет номеров шагов начинается с $k = 0$.). На правой стороне этих рисунков показано состояние робота в положениях, соответствующих оцениванию множества решения. До временного шага $k = 6$ это множество состоит из двух несвязных частей. При $k = 6$ левая часть прогнозируемого множества может быть исключена, так как только состояния из правой части множества совместны со всеми замерами кроме пяти замеров (*Прим. перев.*: выбросов, отмеченных жирными отрезками на рис. 8.27, нижний справа.), в то время как для того, чтобы сохранить левую часть множества, при $k = 6$ нужно допустить наличие восьми выбросов.

Моделирование 20-секундного сценария занимает 15 секунд на компьютере Pentium-II 450. Каждый этап коррекции занимает меньше времени, чем этап статического местоопределения, так как априорное (*Прим. перев.*: в момент прихода замера.) пространство поиска вытекает из прогнозирования с предыдущего шага, и оно имеет меньший размер, чем при начальном запуске процедуры оценивания. Таким образом, становится возможным отслеживать движение робота в реальном времени, учитывая все 24 замера в секунду от сонаров, что представляется разумным для роботов, с относительно медленными движениями.

Табл. 8.17 показывает изменение по времени числа q_k — максимально допустимого числа промахов. Пусть $\underline{q}_k$ — наименьшее из чисел q_k, такое, что $\overline{\mathbb{P}}_k^{q_k}$ непусто; чтобы защититься от выбросов, величина q_k бралась равной $\underline{q}_k + 1$. Положения робота с информацией, соответствующей промахам, отмечены на рис. 8.27 и 8.28. Временами, число q_k становилось очень большим (больше одной трети всего числа замеров), что в моделируемом примере соответствовало положениям робота, при которых углы падения излучения многих сонаров лежали за предельным углом β. Время вычислений в ситуациях такого типа увеличивалось незначительно, вследствие малого размера областей поиска.

ЗАМЕЧАНИЕ 8.9. В реальных приложениях поступление данных обычно — последовательное (например, датчики опрашиваются группами по четыре). Это ограничение может быть легко учтено путем использования только последних данных, полученных для обновления прогнозируемого состояния. ■

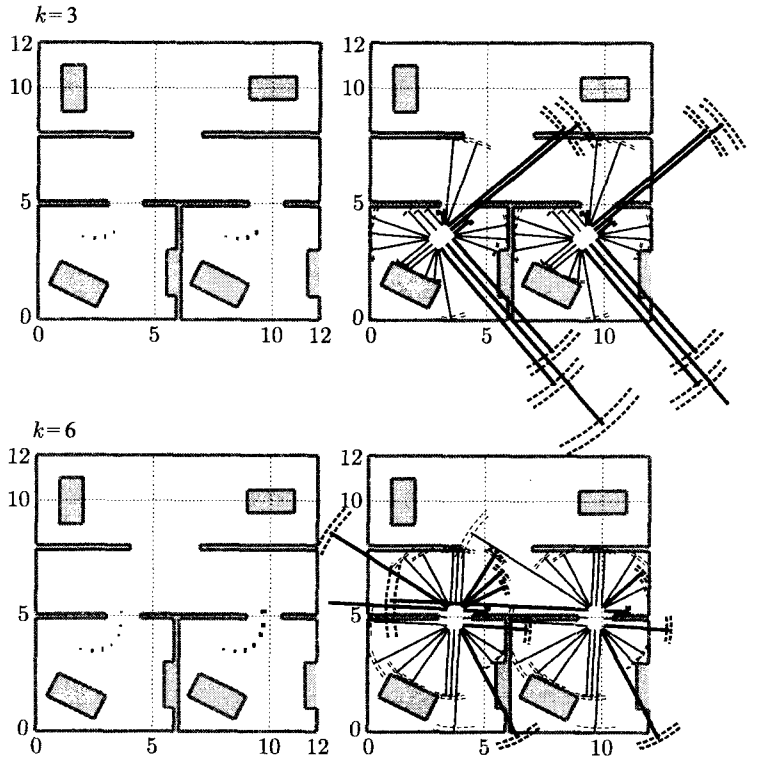


Рис. 8.27. Эволюция проекции на плоскость (x, y) аппроксимации сверху множества решения (слева) и состояний робота (справа), принадлежащих этому множеству; замеры поступают каждую секунду; до момента с номером $k = 6$ локальная симметрия окружающего пространства допускает наличие двух типов радикально различных решений

8.5. Выводы

Рассмотрены три проблемы из робототехники, для которых интервальный анализ оказался способным дать гарантированные решения.

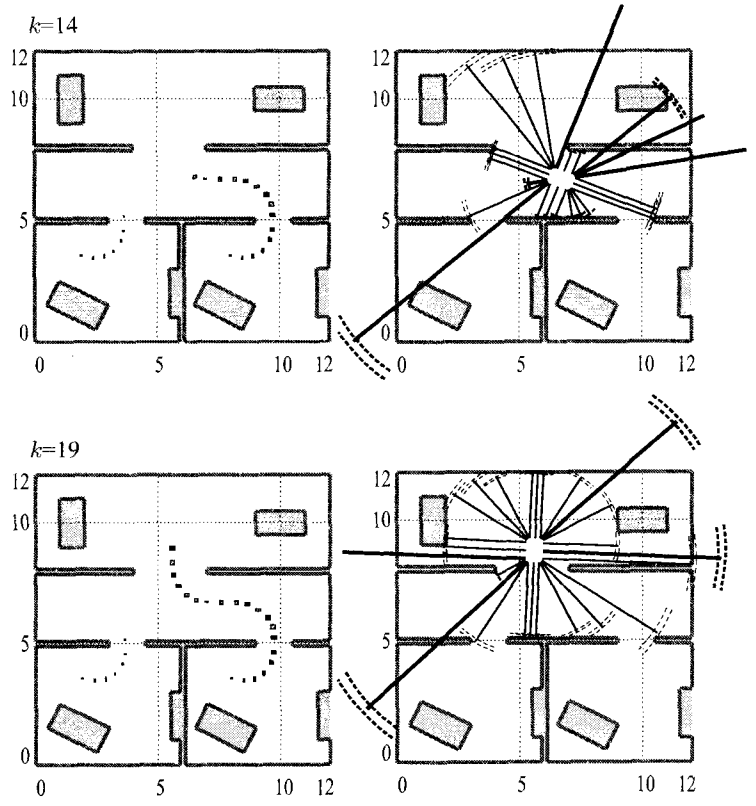


Рис. 8.28. Эволюция проекции на плоскость (x, y) аппроксимации сверху множества решения (слева) и состояний робота (справа), принадлежащих этому множеству; замеры поступают каждую секунду; после момента с номером $k = 6$ исключается неоднозначность решения, обусловленная локальной симметрией окружающего пространства

Интервальные методы решения дали возможность решить задачу управления платформой Стюарта-Гофа в наиболее общем случае неплоского движения. Тригонометрические функции решались прямо, без увеличения числа неизвестных и их последующего исключения. Реальные решения, которые, собственно, и представляли интерес, были очень просто выделены. Не нужно было предполагать, что геометрические коэффициенты являются малыми целыми числами, и решения были получены с оценкой их точностей. Было также очень просто учитывать неопределенность в геометрических параметрах, определяющих платформу или длины ее звеньев, путем задания их интервальных величин.

Таблица 8.17. Эволюция числа q_k — максимально допустимого числа выбросов, как функция от номера k шага времени

Момент i	1	2	3	4	5	6	7	8	9	10
d_k	3	7	8	9	6	6	7	8	9	8

Момент i	11	12	13	14	15	16	17	18	19	20
d_k	8	10	9	7	10	9	6	7	5	5

Для планирования маршрута предложены два алгоритма, основанные на комбинировании интервальных методов и методов теории графов. Интервальный анализ использовался для проверки допустимости параллелотопов в пространстве конфигураций. Главное ограничение на применение данного подхода состоит в том, что время вычислений растет экспоненциально от числа степеней свободы объекта, который необходимо перемещать.

Последняя рассмотренная задача связана с определением местоположения автономного робота, которое в частном случае удается решить с помощью интервального анализа, вследствие небольшого числа параметров, которые надо оценивать. Метод решения, к которому мы обратились в этой главе, имеет определенные преимущества по сравнению с обычными численными методами.

Оказывается ненужным ни нумеровать все возможные связи между данными от чувствительного датчика и объектами среды, ни рассматривать все возможные наборы q выбросов (ложных замеров) среди n_s измеренных точек. Как результат, удается избежать взрывного увеличения числа рассматриваемых комбинаций. Полученные результаты являются глобальными, и не может быть потеряна ни одна из конфигураций, совместных с априорной информацией и замером. Результаты являются весьма робастными, и использованный метод вычисления может даже справляться с большинством выбросов. Найдено, что в большинстве случаев число действительных выбросов может быть равно их максимально допустимому числу, при котором гарантированность результатов еще сохраняется.

Фактическое время вычислений допускает реализацию алгоритмов в реальном времени. Метод является достаточно гибким и позволяет прямо учитывать дополнительную информацию о физической сути задачи. Можно учесть, например, тот факт, что дальность работы сонара (ультразвукового дальномерного устройства) ограничена. Другие типы датчиков, например,

такого как вращающийся лазерный измеритель дальности [Borenstein *et al.*, 1996; Crowly *et.al.*, 1998], а также объединение данных от мультисенсорных устройств [Kam *et al.*, 1997), будут составлять предмет дальнейших исследований в плане применения интервальных методов, как те, к которым мы обращались в данной главе.

Этим мы завершаем Раздел III, посвященный применению методов интервального анализа в инженерных задачах. Раздел IV будет посвящен вопросам реализации этих методов.

Раздел IV

Реализация

Глава 9

Автоматическое дифференцирование

9.1. Введение

Интервальные разрешающие операторы требуют повторного интервального оценивания производных от функций. Например, вычисления производных от функций с интервальным аргументом необходимы при оценивании центрированных функций включения (с. 54), Ньютоновских сжимающих операторов (с. 117), сжимающих операторов, основанных на параллельной линеаризации (с. 118), и при выборе направления бисекции в алгоритме SIVIAX (см. (5.4), с. 141).

Такое вычисление может быть разделено на два шага. Первый состоит в получении точечного алгоритма для оценивания производной функции $\mathbf{f}$, которую надо дифференцировать. За исключением простых академических примеров, функция $\mathbf{f}$ не имеет аналитического выражения (она может быть описана только некоторым алгоритмом) и оценивание ее производной является трудной задачей, которая может привести к ошибкам, если решать ее вручную. Вот почему требуется стройная (систематическая) методология, такая, как описывается в настоящей главе. Второй шаг состоит в получении гарантированного поглощения производной, когда используются интервалы. Этого легко можно достичь при использовании методов Главы 2, так что мы рассмотрим только первый шаг процедуры дифференцирования. Более детально см. [Rall, 1980; 1981; Corliss, 1988; Bischof, 1991; Evtushenko, 1991; Iri, 1991; Bischof *et al.*, 1992; Corliss, 1992; Rall и Corliss, 1999].

Автоматическое дифференцирование может быть двух типов. *Прямое дифференцирование* находит производные по тому же направлению, что и при оценивании самой функции $\mathbf{f}$, которую надо дифференцировать. *Обратное дифференцирование* находит производные в противоположном направлении. Принципы расчета прямого и обратного дифференцирования излагаются в параграфе 9.2. Параграф 9.3 дает правила построения программ, вычисляющих производную функции $\mathbf{f}$ относительно ее аргументов, когда $\mathbf{f}$ рассчитывается по программе. Два очень простых иллюстрационных примера подробно обсуждаются в параграфе 9.4.

9.2. Прямое и обратное дифференцирование

Рассмотрим последовательность

$$\mathbf{v}^{k+1} = \boldsymbol{\phi}^{k+1}(\mathbf{v}^k), \quad k \in \{0, \ldots, \overline{k}-1\}, \tag{9.1}$$

где размерность n_k вектора $\mathbf{v}^k$ может зависеть от k и начальный вектор $\mathbf{v}^0$ полагается известным. Определим функцию:

$$\mathbf{f} \triangleq \boldsymbol{\phi}^{\overline{k}} \circ \ldots \circ \boldsymbol{\phi}^1. \tag{9.2}$$

В настоящем параграфе предлагается подход к расчету численной величины $\frac{d\mathbf{f}}{d\mathbf{v}^0}$ для заданной численно величины $\mathbf{v}^0$. Определим функции:

$$\boldsymbol{\lambda}^k \triangleq \boldsymbol{\phi}^k \circ \boldsymbol{\phi}^{k-1} \circ \ldots \circ \boldsymbol{\phi}^1, \; k \in \{1, \ldots, \overline{k}\}, \; \boldsymbol{\lambda}^0 \triangleq \mathbf{I}_{n_0}, \tag{9.3}$$

$$\boldsymbol{\psi}^k \triangleq \boldsymbol{\phi}^{\overline{k}} \circ \ldots \circ \boldsymbol{\phi}^{k+1}, \; k \in \{0, \ldots, \overline{k}-1\}, \; \boldsymbol{\psi}^{\overline{k}} \triangleq \mathbf{I}_{n_{\overline{k}}}, \tag{9.4}$$

где $\mathbf{I}_n$ — n-мерная тождественная функция (или матрица). Следующие свойства имеют место (рис. 9.1):

$$\mathbf{v}^k = \boldsymbol{\lambda}^k(\mathbf{v}^0), \tag{9.5}$$

$$\mathbf{v}^{\overline{k}} = \boldsymbol{\psi}^k(\mathbf{v}^k), \tag{9.6}$$

$$\mathbf{f} = \boldsymbol{\psi}^0 = \boldsymbol{\lambda}^0 \circ \boldsymbol{\psi}^0 = \ldots = \boldsymbol{\psi}^k \circ \boldsymbol{\lambda}^k = \ldots = \boldsymbol{\psi}^{\overline{k}} \circ \boldsymbol{\lambda}^{\overline{k}} = \boldsymbol{\lambda}^{\overline{k}}, \tag{9.7}$$

$$\frac{d\mathbf{f}}{\mathbf{v}_0} = \frac{d\boldsymbol{\psi}^0}{d\mathbf{v}_0} = \frac{d\boldsymbol{\lambda}^{\overline{k}}}{d\mathbf{v}_0}. \tag{9.8}$$

9.2.1. Прямое дифференцирование

Из уравнения (9.3) следует, что $\boldsymbol{\lambda}^{k+1} = \boldsymbol{\phi}^{k+1} \circ \boldsymbol{\lambda}^k$. Дифференцируя это выражение, получаем

$$\frac{d\boldsymbol{\lambda}^{k+1}}{d\mathbf{v}^0} = \frac{d\boldsymbol{\phi}^{k+1}}{d\mathbf{v}^k}\frac{d\boldsymbol{\lambda}^k}{d\mathbf{v}^0}. \tag{9.9}$$

Если

$$\mathbf{A}^k \triangleq \frac{d\boldsymbol{\lambda}^k}{d\mathbf{v}^0}, \tag{9.10}$$

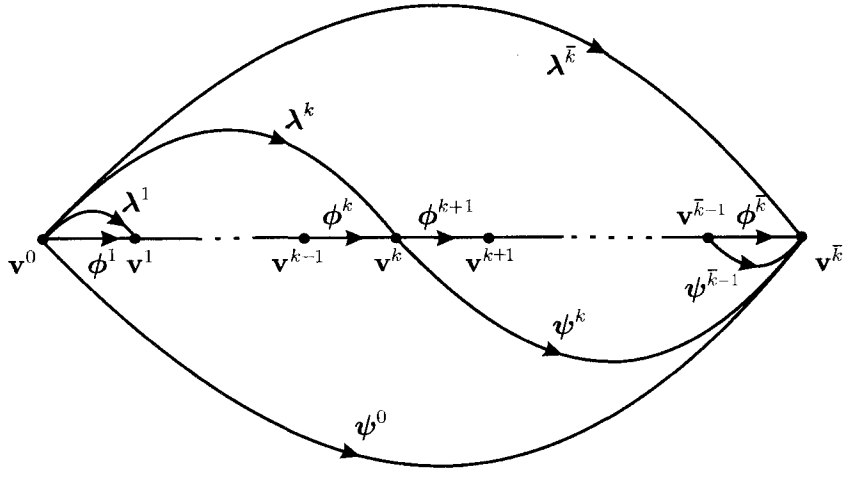


Рис. 9.1. Множество путей оценивания $\mathbf{v}^{\overline{k}} = \mathbf{f}(\mathbf{v}^0)$

то первая производная функции $\mathbf{f}$ по $\mathbf{v}^0$ задается соотношением

$$\frac{d\mathbf{f}}{d\mathbf{v}^0} = \frac{d\boldsymbol{\lambda}^{\overline{k}}}{d\mathbf{v}^0} = \mathbf{A}^{\overline{k}}. \tag{9.11}$$

Матрица $\mathbf{A}^{\overline{k}}$ может быть рассчитана рекурсивно оцениванием последовательностей

$$\begin{cases} \mathbf{A}^{k+1} = \dfrac{d\boldsymbol{\phi}^{k+1}}{d\mathbf{v}^k}\mathbf{A}^k, \\ \mathbf{v}^{k+1} = \boldsymbol{\phi}^{k+1}(\mathbf{v}^k), \end{cases} \qquad k = 0, \ldots, \overline{k}-1, \tag{9.12}$$

где $\mathbf{A}^0 = \mathbf{I}_{n_0}$. Это соответствует алгоритму прямого дифференцирования FD1 (табл. 9.1). Когда алгоритм FD1 завершает работу, то $\mathbf{A} = \mathbf{A}^{\overline{k}} = \frac{d\mathbf{f}}{d\mathbf{v}^0}$.

ЗАМЕЧАНИЕ 9.1. Порядок выполнения Шагов 4 и 5 алгоритма в Таблице 9.1 важен, так как $d\boldsymbol{\phi}^{k+1}/d\mathbf{v}^k$ является функцией от $\mathbf{v}$. ■

Таблица 9.1. Первая версия алгоритма прямого дифференцирования

Алгоритм FD1(вход : $\mathbf{v}^0$; выход : $\mathbf{A}$)	
1	$\mathbf{v} := \mathbf{v}^0$;
2	$\mathbf{A} := \mathbf{I}_{n_0}$;
3	для $k := 0$ до $\overline{k} - 1$
4	$\quad \mathbf{A} := \left(d\boldsymbol{\phi}^{k+1}/d\mathbf{v}^k\right)\mathbf{A}$;
5	$\quad \mathbf{v} := \boldsymbol{\phi}^{k+1}(\mathbf{v})$.

9.2.2. Обратное дифференцирование

Из уравнения (9.4) следует, что $\boldsymbol{\psi}^k = \boldsymbol{\psi}^{k+1} \circ \boldsymbol{\phi}^{k+1}$. Дифференцируя это выражение, получаем

$$\frac{d\boldsymbol{\psi}^k}{d\mathbf{v}^k} = \frac{d\boldsymbol{\psi}^{k+1}}{d\mathbf{v}^{k+1}} \frac{d\boldsymbol{\phi}^{k+1}}{d\mathbf{v}^k}. \tag{9.13}$$

Если

$$\mathbf{B}^k \triangleq \frac{d\boldsymbol{\psi}^k}{d\mathbf{v}^k}, \tag{9.14}$$

то первая производная функции $\mathbf{f}$ по $\mathbf{v}^0$ задается соотношением

$$\frac{d\mathbf{f}}{d\mathbf{v}^0} = \frac{d\boldsymbol{\psi}^0}{d\mathbf{v}^0} = \mathbf{B}^0. \tag{9.15}$$

Матрица $\mathbf{B}^0$ может быть рассчитана рекурсивно оцениванием последовательностей

$$\begin{cases} \mathbf{v}^{k+1} = \boldsymbol{\phi}^{k+1}(\mathbf{v}^k), & k = 0, \ldots, \overline{k} - 1, \\ \mathbf{B}^k = \mathbf{B}^{k+1} \dfrac{d\boldsymbol{\phi}^{k+1}}{d\mathbf{v}^k}, & k = \overline{k} - 1, \ldots, 0, \end{cases} \tag{9.16}$$

где $\mathbf{B}^{\overline{k}} = \mathbf{I}_{n_{\overline{k}}}$. Это соответствует алгоритму обратного дифференцирования BD1 (табл. 9.2). Когда алгоритм BD1 завершает работу, то $\mathbf{B} = \mathbf{B}^0 = \dfrac{d\boldsymbol{\phi}^0}{d\mathbf{v}^0} = \dfrac{d\mathbf{f}}{d\mathbf{v}^0}$.

Таблица 9.2. Первая версия алгоритма обратного дифференцирования

Алгоритм BD1(вход : $\mathbf{v}^0$; выход : $\mathbf{B}$)	
1	для $k := 0$ до $\overline{k} - 1$
2	$\quad \mathbf{v}^{k+1} := \boldsymbol{\phi}^{k+1}(\mathbf{v}^k)$;
3	$\mathbf{B} := \mathbf{I}_{n_{\overline{k}}}$;
4	для $k := \overline{k} - 1$ уменьшая до 0
5	$\quad \mathbf{B} := \mathbf{B}(d\boldsymbol{\phi}^{k+1}/d\mathbf{v}^k)$.

ЗАМЕЧАНИЕ 9.2. В (9.12) и FD1 счетчик k итераций в обеих последовательностях увеличивается, в то время как в (9.16) и BD1 счетчик k итераций увеличивается при оценивании $\mathbf{v}$ и уменьшается при оценивании $\mathbf{B}$. Как результат, все компоненты векторов $\mathbf{v}^k$, которые нужны для оценивания $d\boldsymbol{\phi}^{k+1}/d\mathbf{v}^k$, должны запоминаться. Это не так в алгоритме FD1, и в задачах большого объема ограниченная память может привести к использованию алгоритма FD1, а не алгоритма BD1. ■

ЗАМЕЧАНИЕ 9.3. Как (9.12), так и (9.16) рассчитывают следующее произведение $\overline{k}$ матриц:

$$\frac{d\boldsymbol{\phi}^{\overline{k}}}{d\mathbf{v}^{\overline{k}-1}} \cdots \frac{d\boldsymbol{\phi}^3}{d\mathbf{v}^2} \frac{d\boldsymbol{\phi}^2}{d\mathbf{v}^1} \frac{d\boldsymbol{\phi}^1}{d\mathbf{v}^0}. \tag{9.17}$$

Единственное отличие состоит в том, что (9.12) вычисляет это произведение справа налево, в то время как (9.16) вычисляет его слева направо. В зависимости от размерности матриц, тот или другой подход является более эффективным. Например, произведение

$$\begin{pmatrix} 1 & 2 & 1 \\ 2 & 2 & 3 \\ 1 & 4 & 5 \end{pmatrix} \begin{pmatrix} 4 & 7 & 1 \\ 8 & 2 & 0 \\ 1 & 4 & 5 \end{pmatrix} \begin{pmatrix} 9 & 2 & 9 \\ 2 & 5 & 3 \\ 4 & 4 & 5 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix} \tag{9.18}$$

требует значительно меньше вычислений при оценивании справа налево. Следовательно, вычисление произведения $(\mathbf{AB})\mathbf{C}$ может потребовать значительно бо́льших (или меньших) затрат, чем вычисление $\mathbf{A}(\mathbf{BC})$. ■

ЗАМЕЧАНИЕ 9.4. Поскольку суперпозиция $\boldsymbol{\psi}^k \circ \boldsymbol{\lambda}^k$ постоянна при $k \in \{0, \ldots, \overline{k}\}$, то после дифференцирования $(d\boldsymbol{\psi}^k/d\mathbf{v}^k)(d\boldsymbol{\lambda}^k/d\mathbf{v}^0)$ также не зависит от k. Тогда из уравнений (9.10) и (9.14) следует, что

$$\mathbf{B}^0 = \mathbf{B}^0\mathbf{A}^0 = \mathbf{B}^1\mathbf{A}^1 = \ldots = \mathbf{B}^k\mathbf{A}^k = \ldots = \mathbf{B}^{\overline{k}}\mathbf{A}^{\overline{k}} = \mathbf{A}^{\overline{k}}. \tag{9.19}$$

Это свойство может быть использовано для проверки правильности реализации алгоритмов FD1 и BD1. ■

9.3. Дифференцирование алгоритмов

Алгоритм может рассматриваться как специальный случай последовательности (9.1), где k увеличивается на единицу каждый раз, когда выполняется некоторое предписанное утверждение, и где $\mathbf{v}^k$ состоит из всех переменных, которые надо запоминать в компьютере, когда значение счетчика утверждений равно k. Для широкого класса алгоритмов функции $\boldsymbol{\phi}^k$ – элементарны в том смысле, что они изменяют только одну переменную (т. е. только одну компоненту в $\mathbf{v}^k$). Как результат, матрицы $\mathbf{A}$ и $\mathbf{B}$ (возможно очень большой размерности), вовлеченные в алгоритмы FD1 и BD1, являются разреженными. Это можно выгодно использовать для снижения объема вычислений, как поясняется в данном параграфе.

Сделаем теперь три предположения относительно функций $\boldsymbol{\phi}^k$, включенных в (9.1). Каждое из них повлечет адаптацию алгоритмов FD1 и BD1.

9.3.1. Первое предположение

Предположение 9.1. *Векторы* $\mathbf{v}^1, \ldots, \mathbf{v}^{\overline{k}-1}$ *все имеют одинаковую размерность* n, *и каждая из функций* $\boldsymbol{\phi}^2, \ldots, \boldsymbol{\phi}^{\overline{k}-1}$ *изменяет только одну компоненту своего аргумента, т. е.* $\forall k \in \{1, \ldots, \overline{k}-2\}$, $\exists \mu \mid \forall i \neq \mu,\ \phi_i^{k+1}(\mathbf{v}^k) = v_i^k$. ■

Следовательно, функция $\boldsymbol{\phi}^{k+1}$ может быть выражена как

$$\boldsymbol{\phi}^{k+1}(v_1^k, \ldots, v_n^k) = (v_1^k, \ldots, v_{\mu-1}^k, \phi_\mu^{k+1}(v_1^k, \ldots, v_n^k), v_{\mu+1}^k, \ldots, v_n^k)^{\mathrm{T}}, \quad (9.20)$$

и соответствующая матрица Якоби имеет вид

$$\frac{d\boldsymbol{\phi}^{k+1}}{d\mathbf{v}^k} = \begin{pmatrix} 1 & \ldots & 0 & 0 & 0 & \ldots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & & \vdots \\ 0 & \ldots & 1 & 0 & 0 & \ldots & 0 \\ \dfrac{\partial\phi_\mu^{k+1}}{\partial v_1^k} & \ldots & \dfrac{\partial\phi_\mu^{k+1}}{\partial v_{\mu-1}^k} & \dfrac{\partial\phi_\mu^{k+1}}{\partial v_\mu^k} & \dfrac{\partial\phi_\mu^{k+1}}{\partial v_{\mu+1}^k} & \ldots & \dfrac{\partial\phi_\mu^{k+1}}{\partial v_n^k} \\ 0 & \ldots & 0 & 0 & 1 & \ldots & 0 \\ \vdots & & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \ldots & 0 & 0 & 0 & \ldots & 1 \end{pmatrix}. \quad (9.21)$$

Всюду далее, индекс μ компоненты $\mathbf{v}^k$, изменяемой функцией $\boldsymbol{\phi}^{k+1}$, будет обозначаться как $\mu(\boldsymbol{\phi}^{k+1})$. Алгоритмы FD1 и BD1 будут теперь модифицированы, чтобы учесть выгодный факт, даваемый Предположением 9.1.

Прямое дифференцирование. Пусть $\mathbf{a}_i^{\mathrm{T}}$ – i-я вектор-строка матрицы $\mathbf{A}$ в алгоритме FD1 прямого дифференцирования. Предписанное утверждение

$$\mathbf{A} := \frac{d\boldsymbol{\phi}^{k+1}}{d\mathbf{v}^k}\mathbf{A} \quad (9.22)$$

на Шаге 4 может быть записано как

$$\begin{pmatrix} \mathbf{a}_1^{\mathrm{T}} \\ \vdots \\ \mathbf{a}_n^{\mathrm{T}} \end{pmatrix} := \frac{d\boldsymbol{\phi}^{k+1}}{d\mathbf{v}^k} \begin{pmatrix} \mathbf{a}_1^{\mathrm{T}} \\ \vdots \\ \mathbf{a}_n^{\mathrm{T}} \end{pmatrix}. \quad (9.23)$$

Вследствие (9.21), выражение (9.23) сводится к виду:

$$\mathbf{a}_\mu^{\mathrm{T}} := \sum_{j=1}^{n} \frac{\partial\phi_\mu^{k+1}}{\partial v_j^k}\mathbf{a}_j^{\mathrm{T}}, \quad (9.24)$$

а все другие вектор-строки $\mathbf{a}_i^{\mathrm{T}}$ матрицы $\mathbf{A}$ должны быть оставлены без изменений. Следовательно, алгоритм FD1 прямого дифференцирования может быть переписан как алгоритм FD2 (табл. 9.3).

Таблица 9.3. Вторая версия алгоритма прямого дифференцирования

Алгоритм FD2(вход : $\mathbf{v}^0$; выход : $\mathbf{A}$)		
1	$\mathbf{v} := \mathbf{v}^0$;	
2	$\mathbf{A} := d\boldsymbol{\phi}^1/d\mathbf{v}^0$;	// шаг 4 в FD1, $k = 0$
3	$\mathbf{v} := \boldsymbol{\phi}^1(\mathbf{v})$;	// шаг 5 в FD1, $k = 0$
4	для $k := 1$ до $\overline{k} - 2$	
5	$\quad \mu := \mu(\boldsymbol{\phi}^{k+1})$;	
6	$\quad \mathbf{a}_\mu^{\mathrm{T}} := \sum_{j=1}^{n} (\partial\phi_\mu^{k+1}/\partial v_j^k)\mathbf{a}_j^{\mathrm{T}}$;	// см. (9.24)
7	$\quad v_\mu := \phi_\mu^{k+1}(\mathbf{v})$;	
8	$\mathbf{A} := (d\boldsymbol{\phi}^{\overline{k}}/d\mathbf{v}^{\overline{k}-1})\mathbf{A}$.	// шаг 4 в FD1, $k = \overline{k} - 1$

В алгоритме FD2 Шаг 4 алгоритма FD1 работает специальным образом при $k = 0$ и $k = \overline{k} - 1$. В конец алгоритма FD2 предписанное утверждение $\mathbf{v} := \boldsymbol{\phi}^{\overline{k}}(\mathbf{v})$, вроде необходимое, чтобы FD2 был строго эквивалентен алгоритму FD1, на самом деле не включено, так как оно не влияет на

результат работы алгоритма FD2. Матрица $\mathbf{A}$, выдаваемая FD2, соответствует матрице $\mathbf{A}^{\overline{k}} = d\mathbf{f}/d\mathbf{v}^0$.

Обратное дифференцирование. Пусть $\mathbf{b}_j$ — j-й вектор-столбец матрицы $\mathbf{B}$ в алгоритме BD1 обратного дифференцирования. Выражение присваивания

$$\mathbf{B} := \mathbf{B}\frac{d\boldsymbol{\phi}^{k+1}}{d\mathbf{v}^k} \tag{9.25}$$

на Шаге 5 может быть переписано как

$$\left(\mathbf{b}_1 \dots \mathbf{b}_\mu \dots \mathbf{b}_n\right) := \left(\mathbf{b}_1 \dots \mathbf{b}_\mu \dots \mathbf{b}_n\right)\frac{d\boldsymbol{\phi}^{k+1}}{d\mathbf{v}^k}. \tag{9.26}$$

Вследствие (9.21), выражение (9.26) сводится к виду:

$$\begin{aligned} \mathbf{b}_i &:= \mathbf{b}_i + \frac{\partial \phi_\mu^{k+1}}{\partial v_i^k}\mathbf{b}_\mu, \text{ если } i \neq \mu, \\ \mathbf{b}_\mu &:= \frac{\partial \phi_\mu^{k+1}}{\partial v_\mu^k}\mathbf{b}_\mu. \end{aligned} \tag{9.27}$$

Поскольку вектор $\mathbf{b}_\mu$ входит в первое уравнение (9.27), то он должен быть заслан *после* векторов $\mathbf{b}_i$ для $i \neq \mu$, поэтому указанный порядок уравнений в (9.27) является важным. Алгоритм BD1 обратного дифференцирования теперь может быть переписан как BD2 (табл. 9.4). Чтобы алгоритм BD2 был строго эквивалентен алгоритму BD1 (см. Шаг 2 в BD1 для $k = \overline{k} - 1$), утверждение $\mathbf{v}^{\overline{k}} := \boldsymbol{\phi}^{\overline{k}}(\mathbf{v}^{\overline{k}-1})$ должно бы быть включено после цикла Шага 2 в BD2, но это делать не нужно, так как это утверждение не влияет на результат. Шаг 5 создает много избыточных переменных, которые алгоритм BD2 должен запоминать. Этого можно избежать использованием стека, как показано в алгоритме BD2BIS (табл. 9.5). Этот стек делает возможным на Шаге 8 алгоритма BD2BIS обновлять только компоненту $\mathbf{v}^k$, которая отлична от $\mathbf{v}^{k+1}$. Напомним, что вектор $\mathbf{v}^k$ используется на Шаге 9 и Шаге 10 алгоритма BD2BIS для оценивания производной $\partial \phi_\mu^{k+1}/\partial v_i^k$.

9.3.2. Второе предположение

Предположение 9.2. *Первые компоненты вектора* $\mathbf{v}^1$ *соответствуют* n_0 *входным переменным* v_i^0 *функции* $\mathbf{f}$, *т. е.*

$$\mathbf{v}^1 = \boldsymbol{\phi}^1(\mathbf{v}^0) = \begin{pmatrix} \mathbf{I}_{n_0} \\ \mathbf{O}_{(n-n_0)\times n_0} \end{pmatrix}\mathbf{v}^0 = \left(v_1^0 \dots v_{n_0}^0 0 \dots 0\right)^{\mathrm{T}}, \tag{9.28}$$

Таблица 9.4. Вторая версия алгоритма обратного дифференцирования

Алгоритм BD2(вход : $\mathbf{v}^0$; выход : $\mathbf{B}$)	
1 $\mathbf{v}^1 := \boldsymbol{\phi}^1(\mathbf{v}^0)$;	// шаг 2 в BD1, $k = 0$
2 для $k := 1$ до $\overline{k} - 2$	// шаг 1 в BD1
3 $\quad \mu := \mu(\boldsymbol{\phi}^{k+1})$;	
4 $\quad v_\mu^{k+1} := \phi_\mu^{k+1}(v_1^k, \dots, v_n^k)$;	// (9.20)
5 $\quad$ для $i := 1$ до n, $i \neq \mu$, $v_i^{k+1} := v_i^k$;	
6 $\mathbf{B} := (d\boldsymbol{\phi}^{\overline{k}}/d\mathbf{v}^{\overline{k}-1})$;	// шаг 5 в BD1, $k = \overline{k} - 1$
7 для $k := \overline{k} - 2$ уменьшая до 1	// шаг 5 в BD1
8 $\quad \mu := \mu(\boldsymbol{\phi}^{k+1})$;	
9 $\quad$ для $i := 1$ до n, $i \neq \mu$,	// (9.27)
10 $\quad\quad \mathbf{b}_i := \mathbf{b}_i + (\partial \phi_\mu^{k+1}/\partial v_i^k)\mathbf{b}_\mu$;	
11 $\quad \mathbf{b}_\mu := (\partial \phi_\mu^{k+1}/\partial v_\mu^k)\mathbf{b}_\mu$;	
12 $\mathbf{B} := \mathbf{B}(d\boldsymbol{\phi}^1/d\mathbf{v}^0)$.	// шаг 5 в BD1, $k = 0$

Таблица 9.5. Более эффективная версия алгоритма BD2

Алгоритм BD2BIS(вход : $\mathbf{v}^0$; выход : $\mathbf{B}$)
1 $\mathbf{v}^1 := \boldsymbol{\phi}^1(\mathbf{v}^0)$;
2 для $k := 1$ до $\overline{k} - 2$
3 $\quad\quad \mu := \mu(\boldsymbol{\phi}^{k+1})$;
4 $\quad\quad v_\mu$ в стек;
5 $\quad\quad v_\mu := \phi_\mu^{k+1}(v_1, \dots, v_n)$;
6 $\mathbf{B} := \left(d\boldsymbol{\phi}^{\overline{k}}/d\mathbf{v}^{\overline{k}-1}\right)$;
7 для $k := \overline{k} - 2$ уменьшая до 1
8 $\quad\quad \mu := \mu(\boldsymbol{\phi}^{k+1})$; извлечь из стека в v_μ;
9 $\quad\quad$ для $i := 1$ до $n, i \neq \mu$, $\mathbf{b}_i := \mathbf{b}_i + \left(\partial \phi_\mu^{k+1}/\partial v_i^k\right)\mathbf{b}_\mu$;
10 $\quad\quad \mathbf{b}_\mu := \left(\partial \phi_\mu^{k+1}/\partial v_\mu^k\right)\mathbf{b}_\mu$;
11 $\mathbf{B} := \mathbf{B}\left(d\boldsymbol{\phi}^1/d\mathbf{v}^0\right)$.

где $\mathbf{O}_{m\times n}$ *обозначает ненулевую* $m \times n$ *матрицу, а* $\mathbf{I}_n$ — *тождественная* $n \times n$ *матрица.* ■

Алгоритмы FD2 и BD2BIS могут теперь быть модифицированы с учетом Предположения 9.2.

Прямое дифференцирование. Присваивание $\mathbf{A} := \dfrac{d\boldsymbol{\phi}^1}{d\mathbf{v}^0}$ на Шаге 2 алгоритма FD2 принимает вид:

$$\begin{pmatrix} \mathbf{a}_1^{\mathrm{T}} \\ \vdots \\ \mathbf{a}_n^{\mathrm{T}} \end{pmatrix} := \begin{pmatrix} \mathbf{I}_{n_0} \\ \mathbf{O}_{(n-n_0)\times n_0} \end{pmatrix}, \tag{9.29}$$

поэтому алгоритм FD2 прямого дифференцирования может быть переписан как FD3 (табл. 9.6).

Обратное дифференцирование. Из уравнения (9.28) следует, что

$$\mathbf{B}\frac{d\boldsymbol{\phi}^1}{d\mathbf{v}^0} = (\mathbf{b}_1 \ldots \mathbf{b}_{n_0}, \mathbf{b}_{n_0+1} \ldots \mathbf{b}_n) \begin{pmatrix} \mathbf{I}_{n_0} \\ \mathbf{O}_{(n-n_0)\times n_0} \end{pmatrix} = (\mathbf{b}_1 \ldots \mathbf{b}_{n_0}).$$

Таблица 9.6. Третья версия алгоритма прямого дифференцирования

Алгоритм FD3(вход : $\mathbf{v}^0$; выход : $\mathbf{A}$)	
1 $\mathbf{v} := (v_1^0, \ldots, v_{n_0}, 0, \ldots, 0)^{\mathrm{T}}$;	// шаг 3 в FD2
2 для $i := 1$ до n_0, $\mathbf{a}_i^{\mathrm{T}} = (0, \ldots, 0, 1, 0, \ldots, 0)$;	// см. (9.29)
3 для $i := n_0 + 1$ до n, $\mathbf{a}_i^{\mathrm{T}} = (0, \ldots, 0)$;	// см. (9.29)
4 для $k := 1$ до $\overline{k} - 2$	
5 $\quad \mu := \mu(\boldsymbol{\phi}^{k+1})$;	
6 $\quad \mathbf{a}_\mu^{\mathrm{T}} := \sum_{j=1}^{n} \left(\partial\phi_\mu^{k+1}/\partial v_j^k\right) \mathbf{a}_j^{\mathrm{T}}$;	
7 $\quad v_\mu := \phi_\mu^{k+1}(\mathbf{v})$;	
8 $\mathbf{A} := (d\boldsymbol{\phi}^{\overline{k}}/d\mathbf{v}^{\overline{k}-1})\mathbf{A}$.	

Таким образом, присваивание $\mathbf{B} := \mathbf{B}\dfrac{d\boldsymbol{\phi}^1}{d\mathbf{v}^0}$ на Шаге 11 алгоритма BD2BIS сводится к удалению последних $n - n_0$ столбцов матрицы $\mathbf{B}$. Алгоритм обратного дифференцирования BD2BIS может быть теперь переписан как BD3 (табл. 9.7).

9.3.3. Третье предположение

Предположение 9.3. *Последние $n_{\overline{k}}$ компонент вектора $\mathbf{v}^{\overline{k}-1}$ соответствуют выходным переменным $v_i^{\overline{k}}$ функции* $\mathbf{f}$, *т. е.*

$$\mathbf{v}^{\overline{k}} = \boldsymbol{\phi}^{\overline{k}}\left(\mathbf{v}^{\overline{k}-1}\right) = \left(\mathbf{O}_{n_{\overline{k}}\times(n-n_{\overline{k}})} \mid \mathbf{I}_{n_{\overline{k}}}\right)\mathbf{v}^{\overline{k}-1}. \tag{9.30}$$

Таблица 9.7. Третья версия алгоритма обратного дифференцирования

Алгоритм BD3(вход : $\mathbf{v}^0$; выход : $\mathbf{B}$)
1 $\mathbf{v} := (v_1^0, \ldots, v_{n_0}^0, 0, \ldots, 0)^{\mathrm{T}}$;
2 для $k := 1$ до $\overline{k} - 2$
3 $\quad \mu := \mu(\boldsymbol{\phi}^{k+1})$;
4 $\quad v_\mu$ в стек;
5 $\quad v_\mu := \phi_\mu^{k+1}(v_1, \ldots, v_n)$;
6 $\mathbf{B} := d\phi^{\overline{k}}/d\mathbf{v}^{\overline{k}-1}$;
7 для $k := \overline{k} - 2$ уменьшая до 1
8 $\quad \mu := \mu(\boldsymbol{\phi}^{k+1})$; извлечь из стека в v_μ;
9 $\quad$ для $i := 1$ до n, $i \neq \mu$, $\mathbf{b}_i := \mathbf{b}_i + \left(\partial\phi_\mu^{k+1}/\partial v_i^k\right)\mathbf{b}_\mu$;
10 $\quad \mathbf{b}_\mu := \left(\partial\phi_\mu^{k+1}/\partial v_\mu^k\right)\mathbf{b}_\mu$;
11 удалить последние $n - n_0$ столбцов $\mathbf{B}$.

■

Теперь алгоритмы FD3 и BD3 могут быть модифицированы с учетом Предположения 9.3.

Прямое дифференцирование. Из уравнения (9.30) следует, что

$$\frac{d\boldsymbol{\phi}^{\overline{k}}}{d\mathbf{v}^{\overline{k}-1}}\mathbf{A} = \left(\mathbf{O}_{n_{\overline{k}}\times(n-n_{\overline{k}})} \mid \mathbf{I}_{n_{\overline{k}}}\right)\begin{pmatrix} \mathbf{a}_1^{\mathrm{T}} \\ \vdots \\ \mathbf{a}_n^{\mathrm{T}} \end{pmatrix} = \begin{pmatrix} \mathbf{a}_{n-n_{\overline{k}}+1}^{\mathrm{T}} \\ \vdots \\ \mathbf{a}_n^{\mathrm{T}} \end{pmatrix}. \tag{9.31}$$

Следовательно, присваивание $\mathbf{A} := (d\boldsymbol{\phi}^{\overline{k}}/d\mathbf{v}^{\overline{k}-1})\mathbf{A}$ на Шаге 8 алгоритма FD3 сводится к удалению первых $n - n_{\overline{k}}$ строк матрицы $\mathbf{A}$. Алгоритм FD3 прямого дифференцирования теперь может быть переписан как алгоритм FD4 (табл. 9.8). Когда алгоритм FD4 заканчивает работу, матрица $\mathbf{A}$ равна производной $d\mathbf{f}/d\mathbf{v}^0$.

Обратное дифференцирование. Предписанное утверждение

$$\mathbf{B} := \frac{d\boldsymbol{\phi}^{\overline{k}}}{d\mathbf{v}^{\overline{k}-1}} \tag{9.32}$$

на Шаге 6 в алгоритме BD3 теперь можно разбить на

$$\left(\mathbf{b}_1 \ldots \mathbf{b}_{n-n_{\overline{k}}}\right) := \mathbf{O}_{n_{\overline{k}}\times(n-n_{\overline{k}})} \tag{9.33}$$

Таблица 9.8. Четвертая версия алгоритма прямого дифференцирования

Алгоритм FD4(вход : $\mathbf{v}^0$; выход : $\mathbf{A}$)
1 $\mathbf{v} := (v_1^0, \ldots, v_{n_0}^0, 0, \ldots, 0)^{\mathrm{T}}$;
2 для $i := 1$ до n_0, $\mathbf{a}_i^{\mathrm{T}} = (0, \ldots, 0, 1, 0, \ldots, 0)$;
3 для $i := n_0 + 1$ до n, $\mathbf{a}_i^{\mathrm{T}} = (0, \ldots, 0)$;
4 для $k := 1$ до $\overline{k} - 2$
5 $\quad \mu := \mu(\boldsymbol{\phi}^{k+1})$;
6 $\quad \mathbf{a}_\mu^{\mathrm{T}} := \sum_{j=1}^{n} (\partial \phi_\mu^{k+1} / \partial v_j^k) \mathbf{a}_j^{\mathrm{T}}$;
7 $\quad v_\mu := \phi_\mu^{k+1}(\mathbf{v})$;
8 удалить первые $n - n_{\overline{k}}$ строк матрицы $\mathbf{A}$.

и

$$\left(\mathbf{b}_{n-n_{\overline{k}}+1} \ldots \mathbf{b}_n\right) = \mathbf{I}_{n_{\overline{k}}}. \tag{9.34}$$

Алгоритм обратного дифференцирования BD3 теперь можно окончательно переписать как алгоритм BD4 (Таблица 9.9). Когда алгоритм BD4 заканчивает работу, матрица $\mathbf{B}$ равна производной $\mathrm{d}\mathbf{f}/\mathrm{d}\mathbf{v}^0$.

Таблица 9.9. Четвертая версия алгоритма обратного дифференцирования

Алгоритм BD4(вход : $\mathbf{v}^0$; выход : $\mathbf{B}$)
1 $\mathbf{v} := (v_1^0, \ldots, v_{n_0}^0, 0, \ldots, 0)^{\mathrm{T}}$;
2 для $k := 1$ до $\overline{k} - 2$
3 $\quad \mu := \mu(\boldsymbol{\phi}^{k+1})$;
4 $\quad v_\mu$ в стек;
5 $\quad v_\mu := \phi_\mu^{k+1}(v_1, \ldots, v_n)$;
6 для $i := 1$ до $n - n_{\overline{k}}$, $\mathbf{b}_i = (0, \ldots, 0)^{\mathrm{T}}$;
7 для $i := n - n_{\overline{k}+1}$ до n, $\mathbf{b}_i = (0, \ldots, 0, 1, 0, \ldots, 0)^{\mathrm{T}}$;
8 для $k := \overline{k} - 2$ уменьшая до 1
9 $\quad \mu := \mu(\boldsymbol{\phi}^{k+1})$; извлечь из стека в v_μ;
10 $\quad$ для $i := 1$ до n, $i \neq \mu$, $\mathbf{b}_i = \mathbf{b}_i + \left(\partial \phi_\mu^{k+1} / \partial v_i^k\right) \mathbf{b}_\mu$;
11 $\quad \mathbf{b}_\mu := \left(\partial \phi_\mu^{k+1} / \partial v_\mu^{k+1}\right) \mathbf{b}_\mu$;
12 удалить последние $n - n_0$ столбцов $\mathbf{B}$.

9.4. Примеры

В этом параграфе показывается, как алгоритмы FD4 и BD4 могут быть построены, когда функция, подлежащая дифференцированию, задается некоторым алгоритмом. Напомним, что алгоритм, который надо дифференцировать, может рассматриваться как некоторый специальный случай последовательности (9.1).

9.4.1. Пример 1

Предположим, что необходимо дифференцировать следующий алгоритм

Алгоритм f(вход : u; выход : y)
1 $x := 2u$;
2 $x := 3x^2 + u$;
3 $y := x - u$.

ЗАМЕЧАНИЕ 9.5. Этот первый пример был выбран достаточно простым, чтобы можно было продифференцировать его без знаний об автоматическом дифференцировании. Функция, которую надо дифференцировать, записывается $f(u) = (3(2u)^2 + u) - u$. ■

Чтобы работать по правилам, описанным в параграфах 9.2 и 9.3, перепишем этот алгоритм, используя переменные v_1, v_2 и v_3 вместо u, x и y, чтобы получить

Алгоритм f(вход : v_1; выход : v_3)
1 $v_2 := 2v_1$;
2 $v_2 := 3v_2^2 + v_1$;
3 $v_3 := v_2 - v_1$.

Предположение 9.1 удовлетворяется, так как не более чем одна переменная меняется на каждом заданном шаге. Предположения 9.2 и 9.3 также удовлетворяются, поскольку входная переменная v_1 соответствует первой компоненте вектора $\mathbf{v} = (v_1, v_2, v_3)^{\mathrm{T}}$, а выходная переменная v_3 – его последней компоненте. Функции $\boldsymbol{\phi}^k(\mathbf{v})$ в (9.1) описываются соотношениями

$$\begin{aligned} \boldsymbol{\phi}^1 &= \begin{pmatrix} v_1 \\ 0 \\ 0 \end{pmatrix}, \quad \boldsymbol{\phi}^2 = \begin{pmatrix} v_1 \\ 2v_1 \\ 0 \end{pmatrix}, \quad \boldsymbol{\phi}^3 = \begin{pmatrix} v_1 \\ 3v_2^2 + v_1 \\ 0 \end{pmatrix}, \\ \boldsymbol{\phi}^4 &= \begin{pmatrix} v_1 \\ v_2 \\ v_2 - v_1 \end{pmatrix}, \quad \boldsymbol{\phi}^5 = v_3. \end{aligned} \tag{9.35}$$

Кроме того, $\mu(\boldsymbol{\phi}^2) = 2$, $\mu(\boldsymbol{\phi}^3) = 2$, $\mu(\boldsymbol{\phi}^4) = 4$ и $\overline{k} = 5$.

Прямое дифференцирование. Алгоритм FD4 сводится к алгоритму, представленному в табл. 9.10. Поскольку функция f имеет только одну входную (независимую) переменную v_1, то матрица $\mathbf{A}$ — матрица-столбец, и ее вектор-строки $\mathbf{a}_i^{\mathrm{T}}$ являются скалярными числами a_i. Возвращаясь к начальным обозначениям переменных u, x и y и упрощая псевдокод, получаем алгоритм, представленный в табл. 9.11. По договоренности, элементы матрицы $\mathbf{A}$, связанные с переменными u, x и y, обозначены как a_u, a_x и a_y. Когда алгоритм завершает работу, переменная a_y равна производной df/du при численном значении, принимаемом переменной u.

Таблица 9.10. Первая версия алгоритма прямого дифференцирования в приложении к Примеру 1

Алгоритм $\frac{df}{dv_1}^{\mathrm{FD}}$ (вход : v_1; выход : a_3)	
1	$\mathbf{v} := (v_1, 0, 0)^{\mathrm{T}}$;
2	$a_1 := 1;\ a_2 := 0;\ a_3 := 0;$ // шаги 2 и 3 в FD4
3	$a_2 := \frac{\partial \phi_2^2}{\partial v_1} a_1 + \frac{\partial \phi_2^2}{\partial v_2} a_2 + \frac{\partial \phi_2^2}{\partial v_3} a_3 = 2a_1;$ // $k = 1,\ \mu = 2$
4	$v_2 := 2v_1;$
5	$a_2 := \frac{\partial \phi_2^3}{\partial v_1} a_1 + \frac{\partial \phi_2^3}{\partial v_2} a_2 + \frac{\partial \phi_2^3}{\partial v_3} a_3 = a_1 + 6v_2 a_2;$ // $k = 2,\ \mu = 2$
6	$v_2 := 3v_2^2 + v_1;$
7	$a_3 := \frac{\partial \phi_3^4}{\partial v_1} a_1 + \frac{\partial \phi_3^4}{\partial v_2} a_2 + \frac{\partial \phi_3^4}{\partial v_3} a_3 = -a_1 + a_2;$ // $k = 3,\ \mu = 3$
8	$v_3 := v_2 - v_1.$ // можно не делать

Обратное дифференцирование. Алгоритм BD4 сводится к алгоритму, представленному в табл. 9.12. Поскольку функция f имеет только одну выходную переменную v_3, то матрица $\mathbf{B}$ — матрица-строка, и ее вектор-столбцы $\mathbf{b}_i$ являются скалярными числами b_i. Возвращаясь к начальным обозначениям переменных u, x и y, и удаляя ненужные утверждения, получаем алгоритм, представленный в табл. 9.13.

По договоренности, элементы матрицы $\mathbf{B}$, связанные с переменными u, x и y, обозначены как b_u, b_x и b_y. Когда алгоритм, представленный в табл. 9.13, завершает работу, переменная b_u равна производной df/du при численном значении, принимаемом переменной u.

После небольшой практики становится очень легко программировать алгоритмы прямого и обратного дифференцирования непосредственно по программе оценивания функции, подлежащей дифференцированию.

Таблица 9.11. Окончательный вид алгоритма прямого дифференцирования в приложении к Примеру 1

Алгоритм $\frac{df}{dv_1}^{\mathrm{FD}}$ (вход : v_1; выход : a_y)	
1	$a_u := 1;$
2	$a_x := 2a_u;$
3	$x := 2u;$
4	$a_x := a_u + 6x a_x;$
5	$x := 3x^2 + u;$
6	$a_y := -a_u + a_x.$

Таблица 9.12. Первая версия алгоритма обратного дифференцирования в приложении к Примеру 1

Алгоритм $\frac{df}{dv_1}^{\mathrm{BD}}$ (вход : v_1; выход : b_1)	
1	$\mathbf{v} := (v_1, 0, 0)^{\mathrm{T}}$;
2	v_2 в стек; $v_2 := 2v_1;$ // $\mu = 2,\ k = 1$
3	v_2 в стек; $v_2 := 3v_2^2 + v_1;$ // $\mu = 2,\ k = 2$
4	v_3 в стек; $v_3 := v_2 - v_1;$ // $\mu = 3,\ k = 3$
5	$b_1 := 0;\ b_2 := 0; b_3 := 1;$ шаги 6 и 7 в BD4
6	из стека в v_3; // $\mu = 3,\ k = 3$
7	$b_1 := b_1 + \frac{\partial \phi_3^4}{\partial v_1} b_3 = b_1 - b_3;$
8	$b_2 := b_2 + \frac{\partial \phi_3^4}{\partial v_2} b_3 = b_2 + b_3;$
9	$b_3 := \frac{\partial \phi_3^4}{\partial v_2} b_3 = 0;$
10	из стека в v_2; // $\mu = 2,\ k = 2$
11	$b_1 := b_1 + \frac{\partial \phi_2^3}{\partial v_1} b_2 = b_1 + b_2;$
12	$b_3 := b_3 + \frac{\partial \phi_2^3}{\partial v_3} b_2 = b_3;$
13	$b_2 := \frac{\partial \phi_2^3}{\partial v_2} b_2 = 6v_2 b_2;$
14	из стека в v_2; // $\mu = 2,\ k = 1$
15	$b_1 := b_1 + \frac{\partial \phi_2^2}{\partial v_1} b_2 = b_1 + 2b_2;$
16	$b_3 := b_3 + \frac{\partial \phi_2^2}{\partial v_3} b_2 = b_3;$
17	$b_2 := \frac{\partial \phi_2^2}{\partial v_2} b_2.$

Таблица 9.13. Окончательная версия алгоритма обратного дифференцирования в приложении к Примеру 1

Алгоритм $\frac{df}{du}^{\mathrm{BD}}$(вход : u; выход : b_u)
1 $x := 2u$;
2 x в стек; $x := 3x^2 + u$;
3 $y := x - u$;
4 $b_u := 0$; $b_x := 0$; $b_y := 1$;
5 $b_u := b_u - b_y$; $b_x := b_x + b_y$; $b_y := 0$;
6 из стека в x;
7 $b_u := b_u + b_x$; $b_x := 6xb_x$; $b_u := b_u + 2b_x$.

9.4.2. Пример 2

Рассмотрим дискретную динамическую систему [Walter и Pronzato, 1997] вида

$$y(k+1) = p_1 y(k), \qquad \text{где } y(0) = p_2. \tag{9.36}$$

Предположим, что доступны некоторые априорные значения $\check{y}(1), \ldots, \check{y}(n)$ переменных $y(1), \ldots, y(n)$. Требуется оценить вектор параметров $\mathbf{p} = (p_1, p_2)^{\mathrm{T}}$ путем минимизации целевой функции

$$\begin{aligned} c(\mathbf{p}) &= \sum_{k=1}^{n} \left(y(p_1, p_2, k) - \check{y}(k)\right)^2 \\ &= (p_1^n p_2 - \check{y}(n))^2 + \ldots + (p_1^2 p_2 - \check{y}(2))^2 + (p_1 p_2 - \check{y}(1))^2. \end{aligned} \tag{9.37}$$

Оценивание производной функции c по вектору $\mathbf{p}$ требуется в большинстве алгоритмов оптимизации. Опять в рассматриваемом примере можно просто вручную получить аналитическое выражение для вектора градиента. Чтобы проиллюстрировать использование алгоритма автоматического дифференцирования, представим функцию $c(\mathbf{p})$ в виде следующего алгоритма:

Алгоритм c(вход : p_1, p_2; выход : s)
1 $s := 0$;
2 $y := p_2$;
3 для $k := 1$ до n
4 $\quad y := p_1 y$;
5 $\quad s := s + (y - \check{y}(k))^2$.

Прямое дифференцирование. Алгоритм FD4 сводится к алгоритму, представленному в табл. 9.14. Поскольку функция c имеет две входных (независимых) переменных p_1 и p_2, то все вектор-строки $\mathbf{a}_{p_1}^{\mathrm{T}}$, $\mathbf{a}_{p_2}^{\mathrm{T}}$, $\mathbf{a}_s^{\mathrm{T}}$ и $\mathbf{a}_y^{\mathrm{T}}$ тоже имеют по два элемента. Когда алгоритм табл. 9.14 завершает работу, переменная $\mathbf{a}_s^{\mathrm{T}}$ равна вектору производной $(\frac{dc}{dp_1}(\mathbf{p}), \frac{dc}{dp_2}(\mathbf{p}))$.

Таблица 9.14. Алгоритм прямого дифференцирования в приложении к Примеру 2

Алгоритм $\frac{dc}{d\mathbf{p}}^{\mathrm{FD}}$(вход : p_1, p_2; выход : $\mathbf{a}_s^{\mathrm{T}}$)
1 $\mathbf{a}_{p_1}^{\mathrm{T}} := (1, 0)$; $\mathbf{a}_{p_2}^{\mathrm{T}} := (0, 1)$; $\mathbf{a}_s^{\mathrm{T}} := (0, 0)$; $\mathbf{a}_y^{\mathrm{T}} := (0, 0)$;
2 $s := 0$;
3 $\mathbf{a}_y^{\mathrm{T}} := \mathbf{a}_{p_2}^{\mathrm{T}}$; $y := p_2$;
4 для $k := 1$ до n
5 $\quad \mathbf{a}_y^{\mathrm{T}} := p_1 \mathbf{a}_y^{\mathrm{T}} + y\mathbf{a}_{p_1}^{\mathrm{T}}$; $y := p_1 y$;
6 $\quad \mathbf{a}_s^{\mathrm{T}} := \mathbf{a}_s^{\mathrm{T}} + 2(y - \check{y}(k))\mathbf{a}_y^{\mathrm{T}}$.

Обратное дифференцирование. Алгоритм BD4 сводится к алгоритму, представленному в табл. 9.15. Поскольку функция c имеет один (скалярный) выход, то вектор-столбцы $\mathbf{b}_y$, $\mathbf{b}_{p_1}$, $\mathbf{b}_{p_2}$ и $\mathbf{b}_s$ все тоже являются скалярами, которые обозначаются как b_y, b_{p_1}, b_{p_2} и b_s. Когда алгоритм табл. 9.15 завершает работу, то b_1 и b_2 равны, соответственно, $\frac{dc}{dp_1}(\mathbf{p})$ и $\frac{dc}{dp_2}(\mathbf{p})$.

9.5. Выводы

В данной главе представлены некоторые основные понятия автоматического дифференцирования алгоритма оценивания некоторой функции $\mathbf{f}$, заданной в виде программы. Когда длина кода этой программы не слишком велика, легко построить программу дифференцирования вручную, используя методологию, рассмотренную в параграфе 9.4, и максимально упрощая результирующий код. Разумеется, компьютерная реализация этой методики более предпочтительна, когда нужно работать со сложными программами.

Для выбора прямого или обратного дифференцирования необходимо учитывать разные критерии. Для *простой реализации* более предпочтительным может быть прямое дифференцирование, поскольку оно может быть выполнено с использованием оператора перегружения [Hammer *et al.*, 1995]. Для *экономии памяти* прямое дифференцирование может быть предпочтительнее, так как оно не требует запоминания значений, принимаемых

Таблица 9.15. Алгоритм обратного дифференцирования в приложении к Примеру 2

Алгоритм $\frac{dc}{dp}^{\mathrm{BD}}$(вход : p_1, p_2; выход : b_{p1}, b_{p2})
1 $s := 0;$
2 $y := p_2;$
3 для $k := 1$ до n
4 $\quad y$ в стек; $y := p_1 y;$
5 $\quad s$ в стек; $s := s + (y - \breve{y}(k))^2;$
6 $b_y := 0;\ b_{p_1} := 0;\ b_s := 1;$
7 для $k := n$ уменьшая до 1
8 $\quad$ из стека в s;
9 $\quad b_y := b_y + 2(y - \breve{y}(k))b_s;$
11 $\quad$ из стека в y;
12 $\quad b_{p_1} := b_{p_1} + y b_y;\ b_y := p_1 b_y;$
13 $b_{p_2} := b_y.$

переменными в программе, которую надо дифференцировать. Для *уменьшения времени вычислений* выбор между прямым и обратным дифференцированием зависит от числа n_{u} входных переменных и числа n_{y} выходных переменных подпрограммы расчета функции $\mathbf{f}$. Напомним, что каждая переменная v_i алгоритма расчета $\mathbf{f}$ соответствует n_{u} переменным (через вектор-строку $\mathbf{a}_i^{\mathrm{T}}$) для прямого дифференцирования и n_{y} переменным (через вектор-столбец $\mathbf{b}_i$) для обратного дифференцирования. В параграфе 9.4.2, например, алгоритм, который надо было дифференцировать, имел две входных переменных и одну выходную, поэтому вектор $\mathbf{a}_i$ имеет размерность два, в то время как $\mathbf{b}_i$ — скаляр. При этом вектора $\mathbf{a}_i$ и $\mathbf{b}_i$ должны запоминаться и обновляться на каждой итерации. Если число входных переменных больше числа выходных переменных, то обратное дифференцирование будет в общем случае выполняться быстрее, а в противоположном случае предпочтительным должно быть прямое дифференцирование.

ГЛАВА 10

Гарантированные вычисления с числами с плавающей точкой

10.1. Введение

Одним из главных свойств интервального анализа является его способность строить параллелотопы, гарантированно содержащие образ параллелотопа, заданного по некоторой функции. Это *свойство включения* (т. е. содержания истинного значения в интервальном результате) должно быть сохранено в компьютерной реализации интервальных вычислений. Интервалы, вычисляемые при представлении вещественных чисел с конечной точностью, должны, следовательно, всегда содержать интервалы, которые бы получались при вычислениях с бесконечной точностью. Кроме того, необходимо искать компромисс между временем вычислений и точностью интервального оценивания.

Первая часть этой краткой главы посвящена последствиям представления вещественных чисел с плавающей точкой при реализации вычислительных программ. Параграф 10.2 дает некоторые сведения по двоичной арифметике с плавающей точкой по стандарту IEEE 754. Этот стандарт компилируется процессорами, стоящими в большинстве современных персональных компьютеров и рабочих станций, и оказался очень полезным для упорядочения существовавшей ранее весьма хаотической ситуации [Severance, 1998]. Мы увидим, что он обладает свойствами, делающими возможным реализацию интервального вычисления, такого как управляемое округление, хотя и оставляет нерешенными некоторые проблемы. В параграфе 10.3 предлагается реализация интервального вычисления с учетом пустых интервалов и интервалов с бесконечными границами. Наконец, в параграфе 10.4 даются указания на доступное программное обеспечение.

10.2. Числа с плавающей точкой и стандарт IEEE 754

Среди способов, предложенных для приближенного представления вещественных чисел в компьютерах (см. [Sartzlander и Alexopoulos, 1975],

представление *знак/логарифм*, или представление через *систему дополнительных чисел*) [Matula и Kornerup, 1985], представление с плавающей точкой используется наиболее широко. Число с плавающей точкой может быть записано как

$$\pm d_0, d_1 d_2 \ldots d_{p-1} \times \beta^e, \tag{10.1}$$

где $d_0, d_1 d_2 \ldots d_{p-1}$ — p-значная *мантисса*, β — основание, e — *несмещенный* заданный *показатель*, ограниченный некоторым интервалом $[e_{\min}, e_{\max}]$. Причина называть показатель несмещенным станет скоро понятной. Обозначение (10.1) соответствует вещественному числу

$$\pm\left(d_0\beta^0 + d_1\beta^{-1} + d_2\beta^{-2} + \ldots + d_{p-1}\beta^{p-1}\right) \times \beta^e, \tag{10.2}$$

при $(0 \leqslant d_i < \beta)$.

Пример 10.1. *Вещественное число* $0,5$ *неявно выражено в десятичной системе. При* $\beta = 10$ *и* $p = 4$ *представление этого числа с плавающей точкой имеет вид* $5{,}000 \times 10^{-1}$. *При* $\beta = 2$ *и* $p = 6$ *представление имеет вид* $1{,}000 \times 2^{-1}$. ■

Однако такое представление не является единственным. Например,

$$2{,}500 \times 10^{-1} = 0{,}025 \times 10^{1}. \tag{10.3}$$

Это усложняет разработку некоторых алгоритмов, какие требуются для сравнения чисел. Чтобы обеспечить единственность представления, старший разряд d_0 в (10.1) принудительно задается ненулевым. Результирующее представление называется *нормализованным*.

ЗАМЕЧАНИЕ 10.1. Обычно в компьютерах, когда $\beta = 2$, d_0 равен или 0 или 1; поэтому, в нормализованном двоичном представлении d_0 всегда равен 1. ■

10.2.1. Представление

Стандарт IEEE 754 определяет норматив двоичного представления чисел с плавающей точкой ($\beta = 2$). Подробности можно найти в IEEE Computer Society (1985), [Goldberg, 1991] и [Kahan, 1996]. Определены четыре формата с плавающей точкой, а именно, *одинарный*, *двойной*, *одинарный расширенный* и *двойной расширенный*. Два первых используются наиболее широко и соответствуют типам **float** и **double** в языках **C** и **C++**. Каждый формат описывается длиной p мантиссы и интервалом допустимых значений показателя (см. табл. 10.1).

Таблица 10.1. Стандарт IEEE 754 по форматам с плавающей точкой

Формат	p	$e_{\min}$	$e_{\max}$	Длина показателя мантиссы	Длина формата
одинарный	24	-126	127	8	32
одинарный расшир.	32	$\leqslant -1022$	$\geqslant 1023$	$\geqslant 11$	$\geqslant 43$
двойной	53	-1022	1023	11	64
двойной расшир.	64	$\leqslant -16382$	$\geqslant 16383$	$\geqslant 15$	$\geqslant 79$

В обоих расширенных форматах задается только нижняя граница числа разрядов показателя. Следовательно, это число разрядов может зависеть от реализации. Заметим, что $e_{\max}$ больше $|e_{\min}|$. Это делается для того, чтобы избежать переполнения при вычислении $1/x$, если x — некоторое ненулевое число с плавающей точкой с наименьшей абсолютной величиной в рассматриваемом формате (например, $\pm 1{,}000 \ldots \times 2^{-126}$ в одинарном формате). Однако возможно исчезновение мантиссы, что считается разработчиками стандарта менее сложной проблемой, чем переполнение. Показатели $e_{\min} - 1$ и $e_{\max} + 1$ зарезервированы для представления специальных чисел (см. параграф 10.2.3).

Представление числа с плавающей точкой требует также кодирования знаков мантиссы и показателя. Для знака мантиссы используется специальный бит со значением 0 для положительного числа (при знаке + числа) и 1 для отрицательного числа. Знак показателя кодируется с использованием смещения показателя на заданное поле (например, +127 для одинарного формата). Результат называется *смещенным* показателем. При этом обеспечивается, что числа с плавающей точкой запоминаются следующим образом: показатель, а далее мантисса. Этот подход имеет преимущество сохранения упорядочения чисел с плавающей точкой в их представлении.

Так как для кодирования знака мантиссы требуется двоичный бит, то для кодирования знака в одинарном формате могло бы потребоваться 33 бита. Чтобы остаться в пределах 32 бит, возможно использовать тот факт, что старший разряд d_0 в (10.1) всегда равен единице при нормализованном двоичном представлении и, следовательно, не нуждается в запоминании. Далее, старший разряд присоединяется к мантиссе как скрытый разряд. Табл. 10.2 иллюстрирует некоторые примеры битового представления. Скрытый разряд отмечен в скобках.

Таблица 10.2. Десятичное и битовое представление чисел с плавающей точкой

Десятичное представление	Двоичное представление			Интерпретация двоичного представления
	Знак	Показатель	Мантисса	
1	0	01111111	(1)000...000	$(-1)^0 \times 1{,}000 \times 2^{127-127}$
-2	1	10000000	(1)000...000	$(-1)^1 \times 1{,}000 \times 2^{128-127}$
16,5	0	10000011	(1)000010...0	$(-1)^0 \times 1{,}00001 \times 2^{131-127} = 1 \times 2^4 + 1 \times 2^{-1}$

10.2.2. Округление

Одним их последствий описания вещественных чисел конечным числом битов является то, что они обычно не могут быть представлены точно. Следовательно, необходимо иметь механизмы округления для получения правильного представления чисел. Даже сам результат вычислений при использовании чисел с плавающей точкой должен часто округляться чтобы получить число, с плавающей точкой.

Ошибка округления обычно измеряется в *единицах последнего (младшего) разряда*. При этом, разность между истинным значением числа и его аппроксимацией выражается как единица наименьшего бита мантиссы в рассматриваемом формате представления с плавающей точкой. Например, если $\beta = 2$ и $p = 4$, то представление числа 1,001 в виде 1,000 приводит к ошибке в единицу последнего разряда; представление числа 1,10001 в виде 1,100 соответствует ошибке $0{,}01 = 1{,}0 \times 2^{-2} = 0{,}25$ единицы последнего разряда. В общем случае, если вещественное число z аппроксимируется числом с плавающей точкой как $\pm d_0, d_1 d_2 \ldots d_{p-1} \times \beta^e$, то ошибка округления выражается в единицах последнего разряда следующим образом:

$$\left| \pm d_0, d_1 d_2 \ldots d_{p-1} - \frac{z}{\beta^e} \right| \beta^{p-1}. \tag{10.4}$$

Когда ошибка округления меньше, чем 0,5 единицы последнего разряда, говорится, что число с плавающей точкой *округлено правильно* [Kahan, 1996].

В стандарте оговорены четыре способа округления. Способ округления по умолчанию состоит в округлении к ближайшему числу с плавающей точкой. При округлении результатов четырех основных арифметических операций, квадратного корня, операции нахождения процентов (если имеется представление $a = n \times b + r$ при $(a, b) \in \mathbb{R}^2$ и $n \in Z$, то остаток $r = a\%b$ удовлетворяет неравенству $|r| < |b|$), и чтобы корректно выполнить округление в способе по умолчанию, требуется преобразование от целых чисел к числам с плавающей точкой. Остальные три способа округления

состоят в округлении к 0, к $+\infty$ и к $-\infty$. Два последних способа округления оказываются особенно полезными при выполнении гарантированных вычислений, как будет показано в параграфе 10.3. Рис 10.1 иллюстрирует эти четыре способа округления. Жирные вертикальные засечки соответствуют числам, которые можно точно представить в рассматриваемой системе с плавающей точкой. Вещественные числа, соответствующие двум последовательным жирным засечкам различаются на единицу последнего разряда. Вещественное число x не может быть представлено точно, и четыре способа округления приводят к двум различным возможным результатам.

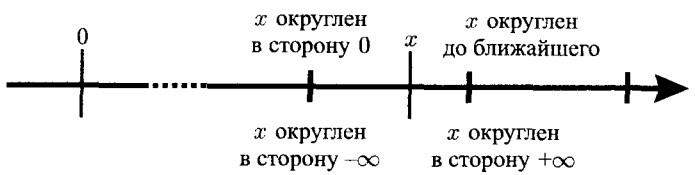


Рис. 10.1. Четыре способа округления, определенные стандартом IEEE 754

Результаты расчета трансцендентных функций не нуждаются в корректном округлении вследствие факта, известного как *дилемма составителя таблицы* [Goldberg, 1991; Daumas *et al.*, 1995; Muller, 1997; Lefèvre *et al.*, 1998]. Рассмотрим систему с плавающей точкой с основанием 2 и мантиссой длины p. Предположим, например, что логарифм некоторого числа a с плавающей точкой оценен с точностью $p + 2$ разрядов:

$$\log(a) \simeq \underbrace{\mathbf{1}, \mathbf{x} \ldots \mathbf{x} \mathbf{0} 111}_{p \text{ разрядов}}. \tag{10.5}$$

Члены $\mathbf{x}$ в (10.5) представляют разряды, не играющие роли в упомянутой дилемме. Когда округление выполняется к $-\infty$, величина $\log(a)$ должна быть округлена к $\mathbf{1}, \mathbf{x} \ldots \mathbf{x01}$, если вычисление с бóльшим числом разрядов даст $\log(a) = \mathbf{1}, \mathbf{x} \ldots \mathbf{x01110} \ldots$, и округлена к $\mathbf{1}, \mathbf{x} \ldots \mathbf{x10}$, если видно, что $\log(a) = \mathbf{1}, \mathbf{x} \ldots \mathbf{x10010} \ldots$. Предположим теперь, что оценивание выполняется с увеличивающейся точностью, и что результат имеет вид $\mathbf{1}, \mathbf{x} \ldots \mathbf{x01111}$ и далее $\mathbf{1}, \mathbf{x} \ldots \mathbf{x011111}$, и т. д. .... Так как логарифм — трансцендентная функция, то из-за дилеммы выбора результата округления заранее неизвестно, когда точность вычисления станет достаточной. Поэтому для некоторых особых чисел вычисление потребовалось бы делать с весьма высокой точностью (и, следовательно, с весьма большими вычислительными затратами), чтобы разрешить дилемму. Но делать этого по стандарту IEEE 754 не требуется.

Указанная дилемма имеет важное следствие для сложности оценивания замкнутых интервалов, гарантированно содержащих значения трансцендентных функций, как будет видно в параграфе 10.3.3.

10.2.3. Специальные величины

Специальные величины определены стандартом IEEE 754 *для обеспечения корректной обработки особых ситуаций, таких как деление на нуль или извлечение квадратного корня из отрицательного числа. Коды этих специальных величин приведены в табл.* 10.3.

Таблица 10.3. Специальные величины и их кодирование по стандарту IEEE 754

Число	Знак	Показатель	Мантисса
$\pm\infty$	$\pm$	$e_{\max}+1$	$1{,}00\ldots0$
±0	$\pm$	$e_{\min}-1$	$1{,}00\ldots0$
денормализованное	$\pm$	$e_{\min}-1$	$\neq 1{,}00\ldots0$
NaN	любой	$e_{\max}+1$	$\neq 1{,}00\ldots0$

Бесконечные величины. Величины $\pm\infty$ обеспечивают способ продолжения вычислений после переполнения. Расширение $+\infty$ более безопасно, чем представление результата очень большим конечным числом, как показывается в следующем примере.

Пример 10.2. *Когда величина* $\log(\exp(1000))$ *оценивается с использованием чисел с плавающей точкой по одинарной точности, вычисление* $\exp(1000)$ *дает переполнение. Процессор, совместимый со стандартом* IEEE 754, *закодирует это число как* $+\infty$, *и окончательный результат будет* $+\infty$, *что укажет на переполнение в вычислении. Если же вместо* $+\infty$ *будет использовано некоторое очень большое число, например,* 10^{38}, *то окончательный результат вычисления будет* $\log(10^{38})\approx 87$, *выглядит разумно, но, очевидно, бессмысленно, так как* $\log(\exp(1000))=1000$. ■

Нули со знаками. Поскольку старший разряд d_0 мантиссы нормализованного двоичного числа с плавающей точкой равен 1, то представление нуля оказывается проблематичным. По договоренности, используются *нули со знаками* $+0=1{,}0\times\beta^{e_{\min}-1}$ и $-0=-1{,}0\times\beta^{e_{\min}-1}$. Стандарт предполагает, что равенство $+0=-0$ справедливо, в противном случае простейшая проверка ($\mathbf{x == 0}$) имела бы непредсказуемый результат. Необходимо однако помнить, что эти два числа различны. Это имеет особо важное значение при интервальных делениях. Так, $[4,5]\ /\ [+0,2]=[2,+\infty]$, в то время как $[4,5]/[-0,2]=[-\infty,+\infty]$. Поскольку стандарт IEEE 754 рекомендует

реализацию функции **CopySign()**, которая обнаруживает знак нуля, некоторые компиляторы обрабатывают эту функцию (это делается в **Borland C++ Builder**, но не было доступно в его предыдущих версиях).

Денормализованные числа. Такие числа введены, чтобы обеспечить представление чисел меньших, чем те, что даются при нормализованном представлении (за счет снижения числа разрядов мантиссы). Для денормализованных чисел старший разряд полагается равным 0.

Числа *NaN*. Как показано в табл. 10.3, существует много чисел вида *NaN*. Каждое из них означает *Не число*, но может также пониматься и как *Не какое-либо число* [Kahan, 1996], когда оно используется для указания на то, что результат несостоятелен. Таким образом, от каждой несостоятельной операции требуется вырабатывать число вида *NaN*. Примерами несостоятельных операций являются $\sqrt{x}$ при $x<0$, $0\times\infty$, $0,0\,/\,0,0$, ∞/∞, $(\pm\infty)-(\pm\infty)$ — когда знаки в скобках одинаковы, $y\%0,0$ и $\infty\%y$ при $y\in\mathbb{R}$. Любая арифметическая операция с числом типа *NaN* дает на выходе тоже *NaN*, за исключением случая, когда результат, полученный при замене каждого из чисел типа *NaN* некоторым произвольным числом (оно может быть конечным или бесконечным), не зависит от величины замещающего числа.

Концепция числа типа *NaN* была введена для численного расчета и еще не используется широко в математическом анализе. Она дает элегантный путь преодоления особых ситуаций, которые могут встречаться, например, при поиске корня или при глобальной оптимизации. Обычно нужно указать начальную область поиска. Если оцениваемая функция неопределена на некоторых частях этой области, то плохо организованный алгоритм поиска может застрять при исследовании этих частей. При наличии типа *NaN* становится возможным апостериори, без глобального прерывания процесса поиска, выявить, что эта функция не была определена в некоторых точках исследуемой области. Другой возможный прием состоит в ведении исключения там, где обнаруживается, что функция неопределена (см. параграф 11.13).

Даже если это не оговорено стандартом IEEE 754, другие вещественные функции могут иногда выдавать число типа *NaN* в таких ситуациях, как $\log(x)$ при $x\leqslant 0$, $\sin(\infty)$, $\arcsin(y)$ при $|y|>1$, и т. п. Так, например, происходит в процессоре Pentium фирмы INTEL.

10.3. Интервалы и стандарт IEEE 754

Стандарт IEEE 754 не определяет, как должны выполняться интервальные вычисления, однако он облегчает удовлетворение таких требований, как

включение и переносимость интервальных операций. Этот параграф резюмирует все предложения, представленные в работах [Chiriaev и Walster, 1998; Walster, 1998; Lerch и Wolff von Gudenberg, 2000], для выполнения интервальных арифметических действий на компьютерах, совместимых со стандартом IEEE 754. Эти предложения были отслежены компанией SUN при разработке компиляторов языков **Fortran** и **C++**, вырабатывающих свой интервальный тип переменной, которая обрабатывается на том же уровне, что и переменная вещественного типа. Сначала мы рассмотрим машинное представление интервалов, перед тем, как описывать арифметические действия, удовлетворяющие *свойству включения* и работающие на замкнутых интервалах. Наконец, мы увидим, что возможно получать гарантированные функции включения даже для трансцендентных функций.

10.3.1. Машинные интервалы

Пусть $\mathcal{R}$ — множество всех *конечных машинных чисел*, т. е. всех вещественных чисел, которые представимы в некотором заданном формате с плавающей точкой. Обозначим символом $\aleph$ наибольший элемент из $\mathcal{R}$. Имеем

$$-\infty < -\aleph < \ldots < 0 < \ldots < \aleph < \infty, \tag{10.6}$$

где 0 — здесь и в остальной части данного параграфа представляет $+0$ или -0, без различения, $\aleph$ — мощность множества натуральных чисел.

Пусть $\overline{\mathcal{R}}$ — множество всех машинных чисел $\overline{\mathcal{R}} = \mathcal{R} \cup \{-\infty, +\infty\}$, где $-\infty$ и $+\infty$ — бесконечные величины по стандарту IEEE 754. Любой элемент из $\mathbb{R}$ может быть отображен без неоднозначности в один элемент из $\overline{\mathcal{R}}$, использованием либо округления к $+\infty$, обозначаемого как $\uparrow(\cdot)$, либо округления к $-\infty$, обозначаемого как $\downarrow(\cdot)$. Оба эти типа округления сохраняют отношение упорядоченности $\leqslant$ множества вещественных чисел и являются *идемпотентными*, т. е. $\uparrow(\uparrow(x)) = \uparrow(x)$ и $\downarrow(\downarrow(x)) = \downarrow(x)$. Они обеспечиваются стандартом IEEE 754, как описано в параграфе 10.2.2. Для реальных чисел с абсолютной величиной, меньшей чем $\aleph$, ошибка округления гарантированно будет меньше, чем единица последнего разряда в обоих типах округления.

Пусть $\mathcal{IR}$ — *множество всех конечных машинных интервалов*

$$\mathcal{IR} = \{[a,b] \mid a \in \mathcal{R},\ b \in \mathcal{R},\ a \leqslant b\}, \tag{10.7}$$

а $\overline{\mathcal{IR}}$ — *множество всех машинных интервалов*

$$\overline{\mathcal{IR}} = \mathcal{IR} \cup \{[a,+\infty] \mid a \in \mathcal{R}\} \cup \{[-\infty,b] \mid b \in \mathcal{R}\} \cup [-\infty,+\infty] \cup [NaN_\varnothing, NaN_\varnothing], \tag{10.8}$$

где интервал $[NaN_\varnothing, NaN_\varnothing]$ представляет пустое множество $\varnothing$, полученное, например, пересечением двух несвязных интервалов (см. параграф 2.3). Заметим, что интервалы $[-\infty,-\infty]$ и $[+\infty,+\infty]$ не принадлежат множеству $\overline{\mathcal{IR}}$. Интервалы $[-\infty,-\aleph]$ и $[\aleph,+\infty]$ имею ясную теоретико-множественную интерпретацию и облегчают реализацию, так как операции вида $-\infty + \infty$ и $+\infty - \infty$ обходятся при интервальном сложении или вычитании.

Любая арифметическая операция с пустым множеством $\varnothing$ или любая элементарная функция, действующая на $\varnothing$, выдает пустое множество. Число $NaN_\varnothing$ есть число NaN, используемое только для кодировки пустого множества, которую нельзя получить никак иначе. Эта кодировка использует преимущество того факта, что в стандарте IEEE 754 числа NaN *являются поглощающими*, т. е. *что любое вычисление, содержащее числа NaN*, выдает числа NaN. Использование интервала $[NaN_\varnothing, NaN_\varnothing]$ значительно упрощает реализацию интервальных алгоритмов; часто интервал $[NaN_\varnothing, NaN_\varnothing]$ может обрабатываться подобно другому стандартному интервалу, и не требуется проверка на пустоту.

Операции округления $\uparrow(\cdot)$ и $\downarrow(\cdot)$ могут быть использованы для отображения любого интервала из $\mathbb{IR}$ в элемент множества $\overline{\mathcal{IR}}$ без неоднозначности. Это делается благодаря интервальной функции $\updownarrow(\cdot)$ двустороннего округления с избытком, определяемой следующим образом:

$$\forall [x] = [\underline{x}, \overline{x}] \in \mathbb{IR}, \quad \updownarrow([x]) = [\downarrow(\underline{x}), \uparrow(\overline{x})] \in \overline{\mathcal{IR}}. \tag{10.9}$$

Следовательно, интервалы, нижние границы которых больше числа $\aleph$, представляются как $[\aleph,+\infty]$, а интервалы, верхние границы которых меньше числа $-\aleph$, представляются как $[-\infty,-\aleph]$.

ЗАМЕЧАНИЕ 10.2. Возможно выполнить двустороннее округление с избытком, используя только один тип направленного округления, так

$$\updownarrow([x]) = [\downarrow(\underline{x}), \uparrow(\overline{x})] = [\downarrow(\underline{x}), -\downarrow(-\overline{x})] = [-\uparrow(-\underline{x}), \uparrow(\overline{x})]. \tag{10.10}$$

Это свойство может быть использовано для снижения числа переключений типа операции округления, занимающих много времени в процессоре [Knofel, 1993]. ■

10.3.2. Арифметика замкнутых интервалов

Для любого оператора $\diamond$ из операторов $\{+,-,*,/\}$, некоторого $[x] \in \overline{\mathcal{IR}}$ и $[y] \in \overline{\mathcal{IR}}$

$$\updownarrow([x] \diamond [y]) \in \overline{\mathcal{IR}}, \tag{10.11}$$

и

$$[x] \diamond [y] \subset \updownarrow ([x] \diamond [y]). \tag{10.12}$$

Таким образом, оператор $\updownarrow(\cdot)$ на множестве $\overline{\mathcal{IR}}$ делает возможным определить арифметические действия на машинных интервалах, которые удовлетворяют *требованию включения*. Эта арифметика является замкнутой, так как арифметические операторы, которые неопределены, когда интервальные операнды содержат нуль или бесконечные величины, расширяются применением правил вычислений с бесконечными величинами [Kahan, 1968; Chiriaev и Walster, 1998].

Пример 10.3. *В описанной арифметике имеем*

$$\begin{aligned}
&\updownarrow([2,3] * [5,7]) = [10,21],\\
&\updownarrow(2 * [500,\aleph]) = [1000,+\infty],\\
&\updownarrow([0,3] * [10,+\infty]) = [0,+\infty],\\
&\updownarrow([-\infty,-\aleph] + [\aleph,+\infty]) = [-\infty,\infty],\\
&\updownarrow([1,3]/[0,3]) = [1/3,+\infty],\\
&\updownarrow([-5,2]/[0,0]) = [-\infty,+\infty],\\
&\updownarrow([0,0]/[0,0]) = [-\infty,+\infty].
\end{aligned} \tag{10.13}$$

■

Когда выполняются только интервальные арифметические операции, стандарт IEEE 754 гарантирует, что при вычислении границ ошибка будет меньше, чем единица последнего разряда. Следовательно, максимальная точность получается при минимальном объеме вычислений, и не нужно искать компромисса между точностью и машинными затратами.

Пример 10.4. *Результат деления числа $x = 5$ на число $y = 50$ равен* 0,1*, и он не может быть точно представлен числом с плавающей точкой в двоичной системе. Операция направленного округления делает возможным вычислить некоторый интервал $[z]$, гарантированно содержащий результат деления x/y, с границами*

$$\underline{z} = 1{,}10011001100110011001100\mathbf{0} \times 2^{-4}$$

и

$$\overline{z} = 1{,}10011001100110011001101 \times 2^{-4},$$

если используется одинарная точность. Границы интервала $[z]$ различаются только на единицу последнего разряда. ■

Псевдокод операции интервального сложения представлен в табл. 10.4.

Таблица 10.4. Операция интервального сложения

Алгоритм OPERATOR+(вход : $[a], [b]$; выход : $[r]$)
1 запомнить тип округления ();
2 $[r] := [\downarrow(\underline{a} + \underline{b}), \uparrow(\overline{a} + \overline{b})]$;
3 восстановить тип округления ().

Необходимо отметить три момента. Во-первых, нет необходимости проверять, является ли $[a]$ или $[b]$ пустым множеством $\varnothing$, так как представление пустого множества $\varnothing$ числом типа $[NaN_{\varnothing}, NaN_{\varnothing}]$ гарантирует, что результат будет представлен как $[NaN_{\varnothing}, NaN_{\varnothing}]$, стоит лишь одному из операндов быть пустым. Во-вторых, используемый метод округления запоминается до выполнения и восстанавливается после выполнения интервального сложения, чтобы обеспечивать оценивание вещественных чисел с максимальной точностью. В-третьих, метод округления переключается дважды при оценивании $[r]$, но может быть включен только один раз (см. Замечание 10.2).

10.3.3. Работа с элементарными функциями

Основная проблема при работе с элементарными функциями состоит в том, что некоторые из них, такие как log или tg, не определены на всей вещественной прямой. Действия, которые надо предпринять при оценивании, например, интервальной функции $\log([-3,1])$, являются предметом разногласий между специалистами по интервальным вычислениям. В данной книге мы предпочитаем обратиться к диапазону значений функции от заданного интервала, даже если эта функция неопределена на части этого интервала, (см. (2.30)). Если функция неопределена на любой части интервального аргумента, то результатом вычисления функции является пустое множество, как это имеет место в языке **Prolog 4.**

Реализация операции квадратного корня (sqrt), удовлетворяющей *свойству включения*, легко достигается с точностью в единицу числа последнего разряда на каждой границе, так как в операции sqrt по стандарту IEEE 754 должно выполняться корректное округление (см. табл. 10.5).

Проверка пустоты интервала $[b]$ доступна на любом компьютере, в котором любая элементарная функция от NaN в качестве результата также выдает NaN, как это требуется по стандарту IEEE 754. Здесь потребуется два переключения метода округления.

Таблица 10.5. Интервальная операция извлечения квадратного корня

Алгоритм SQRT(вход : $[a]$; выход : $[r]$)
1 $[b] := [a] \cap [0, \infty]$;
2 если ($[b] = \varnothing$) (необязательно)
3 $[r] := \varnothing$; возврат;
4 запомнить тип округления ();
5 $[r] := [\downarrow (\mathrm{sqrt}(\underline{b})), \uparrow (\mathrm{sqrt}(\overline{b}))]$;
6 восстановить тип округления ().

Пример 10.5. *Для рассмотренной выше интервальной операции извлечения квадратного корня имеем*

$$\updownarrow \left(\sqrt{[5,8]}\right) = \left[\downarrow \left(\sqrt{5}\right), \uparrow \left(\sqrt{8}\right)\right], \tag{10.14}$$

$$\updownarrow \left(\sqrt{[-3,7]}\right) = \left[0, \uparrow \left(\sqrt{7}\right)\right], \tag{10.15}$$

$$\updownarrow \left(\sqrt{[15,+\infty]}\right) = \left[\downarrow \left(\sqrt{15}\right), (+\infty)\right], \tag{10.16}$$

$$\updownarrow \left(\sqrt{[-5,-2]}\right) = \varnothing. \tag{10.17}$$

■

Для трансцендентных функций, вследствие дилеммы составителя таблицы, оказывается невозможным гарантировать охват корректного математического результата результатом интервального вычисления простым включением подходящего метода округления (см. параграф 10.2.2). Решение заключается в расширении интервала, вычисленного так, чтобы он гарантированно содержал истинный математический результат. Число единиц последнего разряда, на которое нужно расширить интервал, зависит от качества реализации трансцендентных функций. Рис. 10.2 иллюстрирует ситуацию, когда вычисления, совместные со стандартом IEEE 754, дают некоторый интервал $[y]$, не содержащий интервала $\log([x])$. Эта проблема решается расширением каждой границы интервала $[y]$ на единицу последнего разряда. При работе с трансцендентными функциями приходится искать компромисс между объемом и точностью вычислений.

10.3.4. Улучшение интервальных оценок

Рассмотренная реализация интервальных операций соответствует *простой* расширенной интервальной системе, описанной в [Walster, 1998].

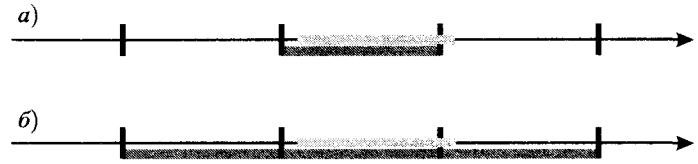


Рис. 10.2. Оценивание образа интервала $[x]$ по функции log; истинный образ отмечен светло серым отрезком: *а*) образ аппроксимирован по стандарту IEEE 754 (темно-серый отрезок); вследствие дилеммы составителя таблицы, аппроксимация не содержит истинный образ целиком; *б*) гарантированная аппроксимация сверху образа, полученная расширением каждой границы предыдущей аппроксимации на одну единицу последнего разряда (темно-серый отрезок)

Для арифметических операций это было пояснено в параграфе 10.3.2 на примере реализации интервального сложения, в рамках того, что требуется по стандарту IEEE 754.

Более утонченные системы интервальных вычислений представлены в [Walster, 1998], например, *полная* расширенная интервальная система, которая в операциях с интервалами обеспечивает наилучшее управление ситуацией исчезновения мантиссы. Это снижает число ситуаций, в которых в качестве результата выдается вся числовая ось. Главное отличие полной системы от простой системы состоит в различении нулей со знаками; в простой расширенной системе интервал $[+0, 5]$ рассматривается как равный интервалу $[-0, 5]$, что не имеет места в полной системе. Следующий пример иллюстрирует последствия возможности такого различения.

Пример 10.6. *Возьмем* $[a] = [4, 5]$ *и* $[b] = [0{,}5, \aleph]$. *При оценивании с библиотекой, реализующей простую расширенную систему, имеем*

$$\updownarrow ([a] \,/\, \updownarrow (1/[b])) = \updownarrow ([4,5]/[0,2]) = [-\infty, +\infty]. \tag{10.18}$$

Та же величина, оцененная по библиотеке, реализующей полную расширенную систему, была бы

$$\updownarrow ([a] \,/\, \updownarrow (1/[b])) = \updownarrow ([4,5] \,/\, [+0,2]) = [2, +\infty]. \tag{10.19}$$

Такой результат требует уточнения того, что $\downarrow (1/\aleph) = +0$, *и выявления знака нуля во втором делении.* ■

Реализация полной расширенной системы является сложной задачей, и к настоящему времени реализованы только простые расширенные системы.

10.4. Источники программного обеспечения для интервальных вычислений

Цель настоящего параграфа — дать указания на доступные источники программного обеспечения для интервальных вычислений. Регулярно обновляемый перечень таких источников можно найти на сайте:

http://www.cs.utep.edu/interval-comp/main.html

Читателю рекомендуется также заглянуть в работу [Kearfott, 1996б] для ознакомления с историей развития программного обеспечения для интервальных вычислений.

Серия программного обеспечения была разработана с 1960-х годов в Университете Карлсруэ, и существующее семейство библиотек **XSC** (для расширения научных вычислений — e**X**tention for **S**cientific **C**omputing) доказало свои преимущества на основе накопленного опыта применения. Так, системы **Pascal-XSC** [Klatte *et al.*, 1992], **Fortran-XSC** и **C-XSC** [Klatte *et al.*, 1992; Hammer *et al.*, 1995] расширяют **Pascal, Fortran** и **C++** на интервалы. Обычным свойством этих инструментов является то, что они управляют одним из главных источников вычислительных ошибок путем реализации точного точечного умножения [Kulisch и Mirander, 1981; Wolff von Gudenberg, 1994]. Разработаны стандартные алгоритмы для таких задач, как глобальная оптимизация функций с возможной мультимодальностью, или решение систем уравнений с возможным нелинейным характером. Эти библиотеки доступны для большинства платформ и дают программное обеспечение, поддерживающее стандарт IEEE 754, когда этот стандарт не полностью реализован в аппаратуре компьютера. Подробности и фрагменты программного обеспечения можно найти на сайте:

http://www.xsc.de

Новая версия библиотеки **C-XSC**, которая согласована с новым стандартом **ISO-C++**, в настоящее время разрабатывается под именем **C-XSC++**. Ее версия **beta** доступна на сайте:

http://www.uni-karlsruhe.de/~uad3/cxsc/

Библиотека **Basic Interval Arithmetic Subroutines Library** [Corliss, 1991] может быть образцом для любой интервальной библиотеки, независимо от языка, на котором эта библиотека будет программироваться. Несколько программных средств были разработаны на основе этого источника; наиболее популярной является, возможно, библиотека **Profil/Bias**, написанная на **C++** (Knüppel, 1994). Эта библиотека свободно доступна для большинства платформ, но она менее полна, чем **C-XSC**. Библиотеку можно взять

с сайта:

ftp://ti3sun.ti3.tu-harburg.de/pub/profil/

Подробности можно найти в Главе 11.

Библиотека **INTLIB** (**INT**erval **LIB**rary) была разработана Kierfott'ом на языке **Fortran** специально для решения систем нелинейных уравнений [Kierfott *et al.*, 1992; Kierfott, 1996б]. Эта библиотека и другие интересные программные средства могут быть взяты с сайта:

http://interval.louisiana.edu/kearfott.html

Совсем новая реализация компанией SUN компиляторов **Forte Fortran/HPC** и **C++** особенно заслуживает внимания. Эти компиляторы являются по своей сути первыми настоящими интервальными компиляторами, которые позволяют программировать алгоритмы, основанные на интервальных вычислениях без какой-либо дополнительной библиотеки. Система интервальной арифметики на замкнутых интервалах также реализована (см. параграф 10.3). В последнее время эти компиляторы запускаются только на компьютерах с операционной системой **Solaris**. Пробные версии библиотеки и операционной системы Solaris могут быть взяты (в ограниченном по времени доступе) с сайта:

http://www.sun.com/forte/index.html

Еще один пакет интервальных вычислений на **C++**, дающий интервальную систему вычислений на замкнутых интервалах, называется **fi_lib++** [Lerch и Wolff von Gutenberg, 2000]. Эта библиотека свободно доступна на сайте:

www.math.uni-wuppertal.de/org/WRST/xsc/download.html

Все эти программные продукты предпочтительны в значительной степени по сравнению с объектно-ориентированным программированием и перегрузкой операторов и позволяют работать с интервалами и интервальными векторами так же свободно, как и с данными других стандартных типов.

Чтобы решать относительно простые задачи, для которых компиляцию можно избежать, можно применять интерпретирующие языки. Для этого был разработан интервальный пакет **Maple Add-ons** [Connell и Corless, 1993]. Для **Matlab** Zemke разработал учебный пакет **B4M (Bias for Matlab 5)**, в начале разработки находится многообещающая библиотека **IntLab**, в которой особое внимание уделено вычислениям трансцендентных функций [Rump, 1999; 2000]. Библиотека **IntLab** запускается под операционными системами **Windows, Unix и Linux**. Библиотеки **IntLab** и **B4M** могут быть взяты с сайта:

http://www.ti3.tu-harburg.de/english/index.html

10.5. Выводы

Стандарт IEEE 754 для двоичного представления чисел с плавающей точкой обеспечивает среду для гарантированной реализации алгоритмов, рассмотренных в настоящей книге. Особое значение имеет управление округлением, которое позволяет таким образом учесть ограничение на точность (ограничение вытекает из представления вещественных чисел с плавающей точкой), что результаты все еще являются гарантированными. Хотя первые версии компиляторов и, следовательно, первые версии библиотек для интервальных вычислений составлялись только по части спецификаций этого стандарта, направление разработок идет к более строгому их построению.

Реализация на компьютерах SUN серии компиляторов FORTE в настоящее время делает возможным прямо покрывать потребности интервальных вычислений. Одной из главных задач в настоящее время является проектирование вычислительной аппаратуры для этих же целей. Были предприняты усилия для проектирования таких со-процессоров [Wolff von Gudenberg, 1996; Schulte и Swartzlander, jr., 2000], а также делались предложения по модификации архитектуры процессоров [Stine и Schulte, 1998a; 1998b; Kolla *et al.*, 1999]. К сожалению, большинство из доступных в настоящее время результатов ограничены лишь описаниями или реализациями прототипов.

ГЛАВА 11

Материал для самостоятельных упражнений

11.1. Введение

Первой целью данной главы будет показать, как ВАША БИБЛИОТЕКА, базовая библиотека для интервальных вычислений, может быть реализована на языке **C++**. Как вскорости выяснится, такая реализация потребует много работы. При этом возникнет вопрос, а не будет ли лучше **не использовать** уже доступную библиотеку; и вторая цель настоящей главы — объяснить, как можно создать свою библиотеку. Мы выбрали библиотеку PROFIL/BIAS, потому что она лицензируется бесплатно и запускается на широком наборе платформ. Время, потраченное на построение ВАШЕЙ БИБЛИОТЕКИ, будет способствовать пониманию программ из PROFIL/BIAS, так как они используют общий синтаксис. Последняя цель данной главы — дать некоторые подробности того, как алгоритмы, описанные в остальной части книги, могут быть реализованы с использованием либо ВАШЕЙ БИБЛИОТЕКИ, либо библиотеки PROFIL/BIAS.

Материал главы организован следующим образом. Сперва приводится некоторый минимум сведений о **C++**, необходимых для начала работы. Далее вводятся базовые объекты ВАШЕЙ БИБЛИОТЕКИ, а именно, интервалы, интервальные векторы и матрицы. Описания каждого из объектов сходны. Перед тем, как шаг за шагом строить структуру объекта, мы напоминаем специальные понятия языка **C++**, если это потребуется читателю. Типовые проблемы реализации далее поясняются в деталях, что должно позволить завершить построение ВАШЕЙ БИБЛИОТЕКИ, как предлагается в упражнениях. Далее мы объясняем, как эти объекты реализуются и используются в библиотеке PROFIL/BIAS. Упражнения, иллюстрирующие реализацию некоторых алгоритмов, описанных ранее, завершают представление этих объектов. Операции покрытия, которые не являются частью библиотеки PROFIL/BIAS, описываются с бóльшими подробностями. Глава заканчивается параграфом по вопросу обработки ошибок в программах.

Читатели, не интересующиеся созданием библиотеки, могут пропустить параграфы 11.6 и 11.9. Решения упражнений можно найти на сайте:

http://www.lss.supelec.fr/books/intervals

11.2. Сведения о C++

Этот параграф, очевидно, не сможет заменить курс языка C++. Подробности этого языка могут быть найдены во многих превосходных учебниках, таких как [Stroustrup, 1991; Capper, 1994] и [Anderson и Anderson, 1998], или в Интернете, например, на сайте:

http : //www.cetus – links.org/oo_c_plus_plus.html

или

http : //www.math.nist.gov/~RPozo/c + +class/

11.2.1. Структура программы

Было бы весьма неразумным, если большая программа состояла бы из единого модуля. Лучше, когда программа организована из более-менее независимых модулей, хранимых в *исходных файлах* (которые идентифицируются по их расширению, зависящему от типа компилятора, и могут иметь вид `.cpp` или `.cc`). Одни модули могут обслуживать графику, другие — интервальные вычисления, третьи — интервальные векторы, и т. д. Одни модули могут вызывать функции из других модулей, что требует знания описаний и функций в этих модулях. Эти данные обеспечиваются *заголовочным файлом* (*заголовком* у функции), связанным с каждым из модулей и идентифицируемым по расширению `.hpp` или `.h`.

Рассмотрим, например, модуль `myapp.cpp`, содержащий главную функцию `main`, требуемую в любой программе. Предположим, что этот модуль использует библиотеку интервальных вычислений, исходные коды которой находятся в `ival.cpp`. Заголовочный файл `ival.h` этой библиотеки должен быть включен в `myapp.cpp` следующим образом.

```
//-------------------------------------------------------------
// Файл:    myapp.cpp
// Назначение: иллюстрирует подключение файлов

#include "ival.h"          // чтобы использовать интервальную
                                                  //библиотеку
```



```
main ()
{
  // остальная часть модуля
  //...
}
//-------------------------------------------------------------
```

Перед компиляцией модуля `myapp.cpp` *препроцессор* замещает строку `#include "ival.h"` на содержимое файла `ival.h`. Следовательно, заголовочные файлы должны содержать как можно меньше кода, для того, чтобы избежать ненужного увеличения размера компилируемого кода и результирующего исполняемого файла. Исходные файлы, расширенные таким образом, обрабатываются *компилятором* для получения *объектных файлов*. Объектные файлы далее связываются друг с другом *редактором связей, компоновщиком* для создания *исполняемого файла*.Компоновщик может также подключать нужные части уже откомпилированных библиотек (файлов с расширением `.lib`). Схематично, структура проекта на C++ представлена на рис. 11.1.

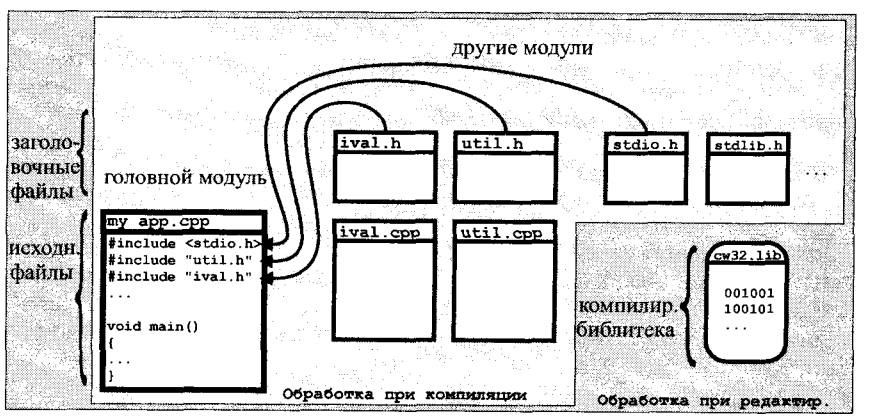


Рис. 11.1. Структура проекта на C++

ЗАМЕЧАНИЕ 11.1. Любая последовательность символов, идущая за символом // в заданной строке, обрабатывается как комментарий и игнорируется компилятором. Равносильно, последовательность, занимающая несколько строк, должна восприниматься как комментарий при ее заключении в символы /* и */. ■

Могут потребоваться другие заголовочные файлы для использования стандартных библиотек языка **C++**, например,

```
#include <iostream.h>
```

для использования стандартных операций ввода-вывода,

```
#include <math.h>
```

для математических функций, таких как синус или косинус.

Имя заголовочного файла, который нужно включить, ставится между скобками (< >), когда это соответствует стандартной библиотеке, поставляемой с компилятором. Для библиотек, задаваемых пользователем, это имя чаще всего ставится в кавычках (" "). Тип ограничителя показывает папку (каталог), в которой этот файл должен быть найден.

11.2.2. Стандартные типы

Наиболее используемыми стандартными типами языка **C++** являются `char` для символов, `int` для целых чисел со знаком, `float` и `double` для чисел с плавающей точкой с одинарной или двойной точностью. Любая переменная должна быть объявлена до того, как она используется. Это объявление может быть сделано в любом месте перед ее использованием и выполняется следующим образом:

```
int i;       // целочисленная переменная
double x;    // вещественная переменная с двойной точностью
```

В программе на языке **C++** учитывается регистр символов. Так, `int x;` и `int X;` описывают разные переменные. Все выражения заканчиваются символом ";". Переменной, которая не должна изменяться при выполнении программы, должен предшествовать *квалификатор* `const`, как во фрагменте

```
const int i = 3; // целая переменная i не может быть
                 // изменена
```

11.2.3. Указатели

Оператор адреса `&` используется, чтобы получить доступ к участку памяти, где хранится переменная. Таким образом, `&x` содержит адрес, где переменная `x` сохраняется; `&x` называется *указателем на* `x`. Возможно также объявлять переменные, которые являются указателями:


```
double* px;  // px~---- указатель на вещественную
             // переменную двойной точности
px = &x;     // px теперь содержит адрес переменной x
```

Чтобы получить доступ к переменной по указателю, хранящему ее адрес, используется *оператор разыменования* `"*"`:

```
double x;
double* px; px = &x;
*px = 3; // *px есть x, поэтому x = 3
```

ЗАМЕЧАНИЕ 11.2. Строго говоря, запись `double* px` (объявление указателя `px` на `double`) отличается от `double *px` (объявление `double`, указываемого с помощью `px`). На практике оба указателя обрабатываются компилятором одним и тем же образом, поэтому расположение пробелов не имеет значения. Однако рекомендуется второй способ, чтобы избежать потенциальных трудностей, например, когда в одном выражении хочется объявить два указателя на `double`. Правильный синтаксис имеет вид `double *px, *py`, поскольку `double* px, py` или `double *px, py` соответствовали бы объявлению указателя `px` и переменной `double py`. ■

11.2.4. Передача параметров в функцию

Чтобы проиллюстрировать, как функции обмениваются информацией, рассмотрим сначала функцию, вычисляющую среднее двух вещественных чисел с двойной точностью.

```
double MeanValue(double a, double b)
{
  double m;       // m - локальная переменная функции
  m = (a + b) / 2;
  return m;
}
```

В этом примере переменная `m` типа `double` возвращается вызывающей функции. Скобки { } определяют область видимости функции или используются для группировки выражений. Функция `MeanValue` может быть вызвана следующим образом:

```
double alpha, beta, gamma;
alpha = 3; beta = 5;
gamma = MeanValue(alpha, beta);
```

Параметры функции `MeanValue` передаются *по значению*. При каждом вызове делается копия каждого из них, и функция обрабатывает каждую из этих копий. Таким образом, `alpha` и `a` запоминаются в разных местах. Следовательно, любое изменение переменных, локальных для данной функции, оставляет неизменными исходные параметры в вызывающей функции.

Рассмотрим теперь функцию обмена значений двух переменных типа `double`. Поскольку параметры в вызывающей функции должны быть изменены, они не могут быть переданы величиной. Возможным инструментом для их передачи служит передача *ссылок*:

```
void Exchange(double& a, double& b)
{
  double temp;
  temp = a;
  a = b;
  b = temp;
}
```

Символ `&` в списке аргументов не должен смешиваться с оператором *получения адреса*, представляемым с помощью того же символа. Он показывает, что следующий параметр передается по ссылке. Снова, расположение пробелов не имеет значения, поэтому `double&a`, `double& a` и `double &a` эквивалентны. Но по той же причине, как в Замечании 11.2, в заголовке функции рекомендуется писать `double &a`. Оператор `void` возвращающего типа показывает, что вызывающей функции ничего не передастся. Функция `Exchange` может быть вызвана следующим образом:

```
  double alpha, beta;
  alpha = 3; beta = 4;
  Exchange(alpha, beta);  // теперь alpha = 4, a beta = 3
```

Здесь, `alpha` и `a` представляют одну и ту же переменную и располагаются в одном и том же месте памяти.

ЗАМЕЧАНИЕ 11.3. Хорошей практикой является помещать квалификатор `const` перед любым аргументом функции, который не будет изменяться этой функцией. В последнее время это стало обязательным для переменных, передаваемых по ссылке. ■

Наконец, параметры могут быть переданы *адресом*. В этом случае передаются указатели на параметры, которые подлежат передаче. Помимо всего прочего, этот подход дает возможность передавать функцию как аргумент другой функции, как будет показано в параграфе 11.8 (с. 401), и параграфе 11.12.3 (с. 419).

11.3. Класс INTERVAL

Введем еще некоторые понятия языка **C++** для того, чтобы позволить создавать объекты типа `INTERVAL`. Объекты — это образования, описываемые их *свойствами* (или *членами-переменными*) и *членами-функциями* (или *методами*), которые к ним применяются. Методы обеспечивают интерфейсы для доступа к свойствам объекта и их модификации. В языке **C++** понятие объекта реализуется использованием *классов*. Класс `INTERVAL` будет сконструирован таким образом, чтобы позволить программисту-прикладнику не обращать внимания на детали реализации интервалов, подобно тому, как обычно нет надобности знать детали того, как представляются числа с плавающей точкой. Класс `INTERVAL` должен *включать* (или *инкапсулировать*) свойства интервалов, определения арифметических операций на интервалах, функции ввода-вывода, и т. д.

Первым шагом в создании класса `INTERVAL` в ВАШЕЙ БИБЛИОТЕКЕ является определение заголовочного файла `ival.h`, включающего все свои свойства и заголовки его членов-функций (методов):

```
//-------------------------------------------------------------
// Файл:     ival.h
// Назначение:  описание класса INTERVAL
#ifndef __INTERVAL__
#define __INTERVAL__

#include <iostream.h>     // для базового неформатированного
                          // ввода-вывода

class INTERVAL{
  private:
    double inf, sup;
  public:
// конструкторы
    INTERVAL()                    // конструктор по умолчанию
      {inf = 0; sup = 0;}
    INTERVAL(const double a, const double b)  // конструктор
                                              // инициализации
      {inf = a; sup = b;}
    INTERVAL(const INTERVAL& a)   // конструктор копирования
      {inf = a.inf; sup = a.sup;}
// деструктор
    ~INTERVAL() {};
// другие методы (члены-функции)
    INTERVAL& operator=(const INTERVAL&);    // присваивание
// функции-друзья
```

```
  // функции чтения
    friend double Inf(const INTERVAL& a)
      { return a.inf };
    friend double Sup(const INTERVAL& a)
      { return a.sup };
    friend double Diam(const INTERVAL&);
    friend INTERVAL Hull(const INTERVAL&,
                                          const INTERVAL&);
  // перегруженные операторы
    friend INTERVAL operator+(const INTERVAL&,
                                          const INTERVAL&);
    friend INTERVAL operator/(const INTERVAL&,
                                            const double);
    friend ostream& operator<<(ostream&,
                                          const INTERVAL&);
//...
};                  // блоки определения класса заканчиваются
                                         // точкой с запятой
#endif
//-------------------------------------------------------------
```

Заголовочный файл может служить документацией класса. В дальнейшем изложении различные части этого заголовка будут объяснены, и читателя попросят к этому вернуться. Здесь заголовочные файлы часто начинаются с `#ifndef...`, `#define...` и заканчиваются на `#endif`. Эти директивы используются препроцессором, чтобы предотвратить множественные включения, путем проверки того, что соответствующий заголовок был уже включен. Имя `__INTERVAL__` выбрано произвольно, таким образом, что соответствующая последовательность символов не могла бы быть найдена еще где-нибудь в исходной программе.

Определение класса начинается с ключевого слова `class`, за которым следует его имя и блок «`{...};`», содержащий описание его свойств и методов (членов-функций). Заметим, что символ ";" закрывает блок. Интервал может быть описан своими нижней и верхней границами, таким образом, наш класс `INTERVAL` включает два `private` свойства, `inf` и `sup`, которые соответствуют этим двум границам. Частные свойства могут быть прочитаны или изменены (с использованием *оператора доступа* "`.`") только членами-функциями класса `INTERVAL`.

Следовательно, если `a` — экземпляр класса `INTERVAL`, то `a.inf` содержит его нижнюю границу. Такая защита частных свойств дает возможность предотвращать установку (инициализацию) неправильных интервалов, у которых нижняя граница больше верхней границы. Свойства и члены-функции, доступ к которым должен обеспечиваться извне класса, помещаются после ключевого слова `public`.Обычно принято (но не обязательно)

давать свойствам класса имена, начинающиеся с использованием строчных букв, и членам-функциям имена, начинающиеся с заглавных букв.

ЗАМЕЧАНИЕ 11.4. Могут быть рассмотрены и другие типы описания интервалов (например, интервал может описываться своим центром и шириной). Очевидно, это бы имело важные последствия для реализации остальной части рассматриваемого класса. ■

11.3.1. Конструкторы и деструкторы

Инициализация класса `INTERVAL` весьма подобна инициализации типа `float`. Таким образом, можно написать программу `firstapp.cpp`, которая использует модуль `ival`, как описано в `ival.h`

```
//-------------------------------------------------------------
// Файл:    firstapp.cpp
// Назначение: инициализация класса INTERVAL

#include "ival.h"       // для использования класса INTERVAL

int main()
{
  INTERVAL x;         // использован конструктор по умолчанию
  INTERVAL y(2,3);  // использован конструктор инициализации
  INTERVAL z(y);       // использован конструктор копирования
  //...
}
//-------------------------------------------------------------
```

Эти инициализации неявно вызывают *конструкторы*. Конструкторы — это члены-функции с тем же именем, как и у класса, без описания типа. Они позволяют инициализировать создаваемые объекты. Здесь они определяют границы интервалов. Представляются три конструктора, реализация которых выполняется прямо внутри заголовочного файла `ival.h`, так как они очень короткие. *Конструктор по умолчанию*, вызываемый для инициализации переменной `x`, формирует обе ее границы равными 0. *Конструктор инициализации* вызывается для инициализации переменной `y` и формирует ее нижнюю границу равной 2, а ее верхнюю границу равной 3. Этот конструктор может проверять правильность интервалов, которые он создает, и может также выдавать сообщение об ошибке, если это необходимо (см. параграф 11.13). Наконец, *конструктор копирования*, вызываемый для инициализации переменной `z`, копирует интервал, переданный как аргумент, поэтому здесь $z = [2, 3]$.

Деструктор `~INTERVAL()` вызывается автоматически в конце программы или по выходу из функции, для каждого экземпляра класса

INTERVAL, который был создан внутри этой программы или функции. Таким образом, при выходе из функции main() в файле firstapp.cpp деструктор будет вызван три раза. Деструкторы делают возможным, кроме всего прочего, динамически освобождать выделенную память (см. Параграф 11.6).

Упражнение 11.1. *Создайте файл* ival.h, *как показано, и напишите конструктор с единственным аргументом* double *так, чтобы он инициализировал точечные интервалы.* ■

11.3.2. Другие члены-функции

Теперь возможно создавать и инициализировать интервалы. Последующие члены-функции, упомянутые в файле ival.h (присваивание, арифметические операции), являются более сложными, поэтому их описание будет отнесено в файл ival.cpp, чтобы сохранить файл ival.h обозримым. Первая член-функция, упомянутая после деструктора

```
INTERVAL& operator=(const INTERVAL&);
```

выполняет *перегрузку* оператора присваивания "=" для интервалов. Эта член-функция присваивает значение аргумента класса INTERVAL вызывающему экземпляру INTERVAL. Ее аргумент предшествуется квалификатором const, показывающим, что эта величина не должна изменяться внутри этой функции. Эта защита особенно полезна при последовательных обращениях к функциям с одним и тем же параметром, так как компилятор заметит любую попытку изменения постоянного аргумента.

Программа, перегружающая оператор присваивания, может выглядеть следующим образом:

```
//-------------------------------------------------------------
// Файл:    ival.cpp
// Назначение: реализация класса INTERVAL

#include "ival.h"       // чтобы использовать класс INTERVAL

INTERVAL& INTERVAL::operator=(const INTERVAL& a)
{
  if (this==&a)      // предотвращает самоприсваивание a = a
    return (*this);
  inf = a.inf; sup = a.sup;
  return (*this);
}
//...
//-------------------------------------------------------------
```


INTERVAL: показывает, что эта член-функция принадлежит классу INTERVAL. Ключевое слово this соответствует указателю на адрес текущего объекта (таким образом, *this является ссылкой на сам объект). Этот адрес сравнивается с адресом &a аргумента a, чтобы избежать присваивания самому себе. Оставшаяся часть этой член-функции подобна конструктору копирования. Оператор «=», который был перегружен, может использоваться следующим образом:

```
INTERVAL x, y(3,4), z(1,3);
x = y;
y = x = z;  // возможно последовательное выполнение
            // x и y теперь равны z
```

Последние функции, занесенные в файл ival.h, являются *функциями-друзьями* (дружественными функциями) для класса INTERVAL. Функции-друзья и члены-функции имеют доступ к частным свойствам этого класса, но синтаксис их вызовов различен. Обращения дружественной функции подчиняется обычному синтаксису для математических функций. Функции-друзья Inf и Sup класса INTERVAL обеспечивают только считывание интервальных границ:

```
INTERVAL x(3,4);
double lowerbound;
lowerbound = Inf(x);  // вызов дружественной функции
```

Член-функция требует использования оператора доступа "." и поэтому менее интуитивно понятна. Вот почему функции-друзья будут применяться для получения свойств интервала, таких как его длина или центр, или для реализации обычных математических функций (cos, sin, exp...).

Возможная реализация дружественной функции, вычисляющей ширину интервала (и называемой Diam для совместимости с библиотекой PROFIL/BIAS), имеет вид

```
//-------------------------------------------------------------
// Файл: ival.cpp (продолжение)
//...
double Diam(const INTERVAL& a)          // вычисление ширины
{ return (a.sup - a.inf); }
//...
//-------------------------------------------------------------
```

Поскольку функции-друзья не рассматриваются как члены класса, то им не предшествует INTERVAL::. Бинарные операторы будут так же реализованы, как и функции-друзья. Функция оператора сложения «+» может иметь вид:

```
//-------------------------------------------------------------
// Файл:  ival.cpp (продолжение)
//...
#include <float.h>          // для управления типом округления
//...
INTERVAL operator+(const INTERVAL& a, const INTERVAL& b)
{
  INTERVAL res;
  unsigned int cw = _control87(NULL,NULL); // запоминает тип
                                           // текущего округления
  _control87(RC_DOWN,MCW_RC); // округление вниз к --oo
  res.inf = a.inf + b.inf;
  _control87(RC_UP,MCW_RC);  // округление вверх к +oo
  res.sup = a.sup + b.sup;
  _control87(cw,MCW_RC); // восстанавливается первоначальный
                                                 // тип округления
  return res;
}
//...
//-------------------------------------------------------------
```

Включение модуля `float` дает возможность применить выражение `_control87(.,.)` для получения и изменения управляющей переменной INTEL-совместимых математических сопроцессоров, соответствующих стандарту IEEE 754 (см. Главу 10). В частности, он полезен для установки типа округления, использованного при обработке результатов арифметических операций при выдаче из сопроцессора. Как упоминалось в параграфе 10.3.2 (с. 365), желательно запомнить тип округления перед его изменением и восстановить его после выхода.

Упражнение 11.2. *Напишите в файле* `ival.cpp` *дружественную функцию* `Hull`, *вычисляющую интервальную оболочку двух интервалов* `INTERVAL`. *Заголовок функции* `Hull` *уже есть в* `ival.h`. ■

Упражнение 11.3. *На основе программы* `Diam` *напишите в файле* `ival.h` *и в файле* `ival.cpp` *другие функции-друзья*:

```
friend double Mid(const INTERVAL& a);
```

вычисляющую центр интервала `a` *и*

```
friend int Intersection(INTERVAL& r, const INTERVAL& a,
                                     const INTERVAL& b);
```

где `r` *— пересечение интервалов* `a` *и* `b`. *Эта функция должна выдавать* `1`, *если пересечение непусто, и* `0` *в противном случае.* ■

Упражнение 11.4. *Перегрузите оператор вычитания* `operator-` *и оператор умножения* `operator*` *для двух интервалов* `INTERVAL`. ■

Упражнение 11.5. *Перегрузите оператор сравнения* `operator<=` *для проверки, включен ли* `INTERVAL a` *в интервал* `INTERVAL b`. *Заголовок этой функции должен быть*

```
friend int operator<=(const INTERVAL& a, const INTERVAL& b);
```

Эта функция должна выдавать `1`, *если интервал* `a` *включен в интервал* `b`, *и* `0` *в противном случае.* ■

Перегрузка оператора деления `/` для деления интервала на интервал, не вызывает проблем, если делитель не равен нулю. В последнем случае, можно отследить различные подходы, см. параграф 11.13, (с. 427). Рассмотрим сначала упрощенную ситуацию, когда делитель не является интервалом, а числом `double`. Если делитель равен нулю, то приведенная ниже программа выдает аппроксимацию всей вещественной прямой в виде интервала `[-Infinity, Infinity]` (см. параграф 10.2.3, с. 360):

```
//-------------------------------------------------------------
// Файл: ival.cpp (продолжение)
//...
INTERVAL operator/(const INTERVAL& a, const double b)
{
  INTERVAL res;
  unsigned int cw = _control87(NULL,NULL);     // запоминает
                                       // текущий тип округления
  if (b > 0)
  { _control87(RC_DOWN,MCW_RC);                 // округление
                                          // вниз к -oo
    res.inf = a.inf / b;
    _control87(RC_UP,MCW_RC);                   // округление
                                         // вверх к +oo
    res.sup = a.sup / b; }
  else if (b < 0)
  { _control87(RC_DOWN,MCW_RC);                 // округление
                                          // вниз к -oo
    res.inf = a.sup / b;
    _control87(RC_UP,MCW_RC);                   // округление
                                         // вверх к +oo
    res.sup = a.inf / b; }
  else
  { res.inf = -Infinity; res.sup = Infinity; }
```

```
  _control87(cw,MCW_RC);   // восстанавливает первоначальный
                                          // тип округления
  return res;
}
//...
//------------------------------------------------------------
```

Величина `Infinity`, соответствующая $+\infty$, еще не была определена и не принадлежит стандарту языка **C++**. Поэтому она должна быть определена с использованием своего битового представления (см. параграф 10.2.4). Так как величина `Infinity` будет использоваться многими функциями, она должна быть помещена в начало файла `ival.h`.

```
//------------------------------------------------------------
// Файл:     ival.h
// Назначение:  описание класса INTERVAL
#ifndef __INTERVAL__
#define __INTERVAL__

#include <iostream.h>      // для базового неформатированного
                                          // ввода-вывода

// Определение величины Infinity через ее
// битовое представление

union UREAL { unsigned short ushort[4]; double real; };

static union UREAL PosInfty = {{ 0x0000, 0x0000,
                                 0x0000, 0x7FF0 }};
static double Infinity = PosInfty.real;

class INTERVAL{
...
//------------------------------------------------------------
```

Упражнение 11.6. *Перегрузите оператор деления* `operator/` *двух интервалов* `INTERVAL`. *Когда* 0 *принадлежит интервальному делителю, то должен выдаваться интервал* `[-Infinity, Infinity]` (см. параграф 10.3.2, с. 365). ■

Последняя дружественная функция, занесенная в файл `ival.h`, перегружает бинарный *оператор вывода* «, чтобы обеспечить печать интервала в выходной информационный поток. Это дает возможность выводить две границы интервала на экран или в файл. Данная дружественная функция может быть реализована следующим образом:


```
//------------------------------------------------------------
// Файл: ival.cpp (продолжение)
//...
#include <iostream.h>       // дает возможность пользоваться
                                    // информационными потоками
//...
ostream& operator<<(ostream& os, const INTERVAL& a)
{
  os << "[" << a.inf << "," << a.sup << "]";
  return (os);
}
//...
//------------------------------------------------------------
```

Выражения

```
INTERVAL a(2,3);
cout << "a = " << a << endl;  // вывод на экран
```

дают следующий вывод на экран:

```
a = [2,3]
```

Эта версия класса `INTERVAL` вместе с результатами упражнений позволяет выполнять большинство основных арифметических операций на интервалах.

11.3.3. Математические функции

Еще предстоит реализовать функции включения для стандартных математических функций. Поскольку они не всегда нужны, их можно вынести в отдельный модуль, называемый `func`, для совместимости с библиотекой PROFIL/BIAS. Этот модуль представляет собой интервальный аналог стандартной математической библиотеки `math`.

Заголовок `func.h` содержит функции, которые должны быть реализованы:

```
//------------------------------------------------------------
// Файл:     func.h
// Назначение: стандартные математические
// функции для класса INTERVAL
#ifndef __FUNCTIONS__
#define __FUNCTIONS__

#include "ival.h"        // чтобы использовать класс INTERVAL
```

```
#include <math.h>        // подключение стандартной
                         // математической библиотеки

INTERVAL Exp    (const INTERVAL& x);
INTERVAL Log    (const INTERVAL& x);
//...
INTERVAL Sin    (const INTERVAL& x);
INTERVAL Cos    (const INTERVAL& x);
INTERVAL Tan    (const INTERVAL& x);
//...
INTERVAL Sqr    (const INTERVAL& x);
INTERVAL Sqrt   (const INTERVAL& x);
//...
INTERVAL ArcSin (const INTERVAL& x);
INTERVAL ArcTan (const INTERVAL& x);
//...
#endif
//-------------------------------------------------------------
```

Детально будут описаны только некоторые из этих функций. При использовании нижеследующих примеров реализация других функций не должна вызвать трудностей.

Монотонные функции особенно просты в реализации. Например, функция включения для экспоненциальной функции может быть реализована в `func.cpp` следующим образом:

```
//-------------------------------------------------------------
// Файл:    func.cpp
// Назначение: стандартные математические
// функции для класса INTERVAL
#include "func.h"
//...
INTERVAL Exp(const INTERVAL& x)
{ return INTERVAL(exp(Inf(x)), exp(Sup(x))); }
//...
//-------------------------------------------------------------
```

Округление с избытком еще не было реализовано. По сути дела, по стандарту IEEE 754 управление округлением не требуется для стандартных функций, за исключением функции `sqrt`, а гарантированное округление для трансцендентных функций еще является областью активных исследований (см. параграф 10.2.2, стр. 358).

Реализация функции включения для логарифмической функции делается аналогично, за исключением того, что областью определения этой функции является положительная вещественная полуось $\mathbb{R}^+$. Здесь можно

применить подход, описанный в параграфе 10.3.3 (с. 365). Другой подход состоит в реализации специальной обработки ошибки, когда интервальный аргумент $[x]$ не полностью содержится в $\mathbb{R}^+$ (см. параграф 11.13, с. 427).

Упражнение 11.7. *Опираясь на программу для функции* `Exp`, *напишите в* `func.cpp` *функции включения для других монотонных функций, таких как* `Sqrt` (*интервальная операция извлечения квадратного корня*). ■

Для реализации немонотонных функций обычно необходимо использование какого-нибудь алгоритма. Поэтому, чтобы получить функцию включения для квадратичной функции, можно написать:

```
//-------------------------------------------------------------
// Файл:    func.cpp (продолжение)
//...
INTERVAL Sqr(const INTERVAL& x)
{
  double infsqr = Inf(x)*Inf(x);
  double supsqr = Sup(x)*Sup(x);
  if (Inf(x) >= 0)
    return INTERVAL(infsqr, supsqr);
  else if (Sup(x) <= 0)
    return INTERVAL(supsqr, infsqr);
  else
    return INTERVAL(0, max(infsqr, supsqr));
}
//...
//-------------------------------------------------------------
```

Упражнение 11.8. *Постройте функцию включения* `Sin` *для функции синуса. По этому результату постройте функции включения* `Cos` *и* `Tan` *для функций косинуса и тангенса.* ■

Библиотека, построенная в таком объеме, может быть использована для решения упражнений параграфа 11.5. Но можно предпочесть использовать какую-нибудь уже готовую библиотеку, как предлагается в следующем параграфе.

11.4. Интервалы с использованием библиотеки PROFIL/BIAS

PROFIL/BIAS является библиотекой для интервальных вычислений. Она работает под многими операционными системами, от UNIX до DOS и OS/2. Она также может также работать под WINDOWS после небольшой модификации исходных файлов. Управление типом округления обеспечено для

многих процессоров, включая RS/6000, SPARC и PENTIUM. Она легко конфигурируется (можно, например, для ускорения вычислений исключить управление внешним округлением). Эта библиотека может быть взята с сайта:

ftp://ti3sun.ti3.tu-harburg.de/pub/profil/

Несколько модифицированная версия этой библиотеки (см. `readme.txt`) может быть взята с сайта:

http://www.lss.supelec.fr/books/intervals

Документация по библиотеке PROFIL/BIAS может быть найдена на сайте:

http://www.ti3.tu-harburg.de/Software/PROFILEnglisch.html

Библиотека состоит из двух компонентов, именно: BIAS и PROFIL.

11.4.1. BIAS

Библиотека BIAS (**B**asic **I**nterval **A**rithmetic **S**ubroutines — основные интервальные арифметические процедуры) была написана на языке **C** в духе Фортановской библиотеки FORTRAN BLAS (Basic Linear Algebra Subroutines — основные процедуры линейной алгебры). Она позволяет выполнять вычисления на точечных и интервальных скалярах, векторах и матрицах. Реализованы основные арифметические операции, управление округлением и элементарные математические функции. Гарантированные результаты получаются с использованием управления типом округления во всех операциях, для которых это возможно по стандарту IEEE 754. BIAS состоит из следующих четырех главных модулей:

- `bias0` обрабатывает вещественные и интервальные скаляры;
- `bias1` обрабатывает вещественные и интервальные векторы;
- `bias2` обрабатывает вещественные и интервальные матрицы;
- `biasf` реализует функции включения для обычных математических функций.

Так как уровень библиотеки BIAS прозрачен для пользователей, мы не будем здесь описывать его детально; более подробную информацию можно получить в [Knuppel, 1993; 1994] или в документации на сайтах Internet.

11.4.2. PROFIL

PROFIL (**P**rogrammer's **R**un time **O**ptimized **F**ast **I**nterval **L**ibrary — оптимизированная по времени быстрая интервальная библиотека для программистов) является объектной оболочкой к BIAS, написанной на языке **C++**.

Она использует удобство, предоставленное классами и перегрузкой операторов, для придания интервальным выражениям естественного вида, сходного с синтаксисом обычных числовых выражений. Переносимость этой библиотеки обеспечивается использованием стандарта **C++**, но для некоторых объектов были введены альтернативные имена через ключевое слово `typedef`. Так,

```
typedef double REAL;
```

делает `REAL` синонимом `double`. Соответствие наиболее часто употребляемых типов переменных показано в табл. 11.1. Для указателей библиотека PROFIL/BIAS определяет `PREAL` как синоним `REAL*`. В общем, любой специальный тип в библиотеке PROFIL/BIAS с предшествующим символом `P` соответствует указателю на тип переменной.

Таблица 11.1. Некоторые стандартные типы языка **C++** и их эквиваленты в библиотеке PROFIL/BIAS

Стандартный **C++**	PROFIL/BIAS	Тип переменной
`void`	`VOID`	пустой тип
`char`		символ
`int`	`INT`	целочисленная величина со знаком
`float`		число с плавающей точкой одинарной точности
`double`	`REAL`	число с плавающей точкой двойной точности

11.4.3. Первое знакомство

Нижеследующая программа выполняет сложение интервалов

```
//-------------------------------------------------------------
// Файл:    add.cpp
// Назначение: сложение двух интервалов INTERVAL
#include "ival.h"   // интервальный модуль библиотеки PROFIL

void main()
{
  INTERVAL a(3,4);                                  // a = [3,4]
  INTERVAL b(5,7);                                  // b = [5,7]
```

```
  INTERVAL c;
  c = a + b;
// выводится на экран
  cout << a << " + " << b << " = " << c << endl;
}
//-------------------------------------------------------------
```

Чтобы позволить выполнять вычисления по этой программе, в общую программу должны включены модули `bias0` и `ival`. Интервальное сложение (или какая-нибудь другая стандартная операция) просто записывается как ее вещественный аналог. Вывод на экран выполняется с использованием стандартного оператора вывода «.

Все стандартные математические функции (`Cos`, `Sin`, `Tan`, `Exp`, `Log`, `Log10`, `Sqrt`, `Sqr`...) обеспечиваются модулем `func` библиотеки PROFIL (см. заголовочный файл `func.h`). Использование модуля `func` требует, чтобы модуль `biasf` (или некоторая библиотека, его содержащая) был включен в общий проект.

Заметим, что в библиотеке PROFIL/BIAS имена стандартных математических функций начинаются с заглавных букв. Эти функции могут вычисляться с плавающей точкой и с интервальными аргументами. Имена, начинающиеся со строчных букв, могут использоваться только с аргументами с плавающей точкой.

Следующий пример иллюстрирует использование элементарных математических функций:

```
//-------------------------------------------------------------
// Файл:    standmat.cpp
// Назначение: стандартные математические
// функции для INTERVAL
#include "ival.h"   // интервальный модуль библиотеки PROFIL
#include "func.h"        // стандартные функции для INTERVAL

void main()
{
  INTERVAL a(3,4);
  INTERVAL b(-2,3);
  INTERVAL c,d;

  c = Exp(a); d = Sqr(b);

  // выводится на экран
  cout << "Exp(" << a << ") = " << c << endl;
  cout << "Sqr(" << b << ") = " << d << endl;
}
//-------------------------------------------------------------
```


11.5. Упражнения на вычисление интервалов

Эти упражнения могут быть выполнены одинаково как с использованием ВАШЕЙ БИБЛИОТЕКИ, так и по библиотеке PROFIL/BIAS.

Упражнение 11.9. *Напишите программу сравнения значений следующих двух функций включения для функции $x^2 - x$ на интервале $[x] = [-1, 3]$:*

$$[f]_1([x]) = [x]^2 - [x],$$
$$[f]_2([x]) = ([x] - 1/2)^2 - 1/4.$$

Эта программа требует включения модуля `ival` *в общий проект.* ■

Упражнение 11.10. *Рассмотрите функцию f, определенную выражением $f(x) = x^2 + \sin(x)$. Вычислите образы, полученные для интервалов $[x]_1 = \left[\frac{2\pi}{3}, \frac{4\pi}{3}\right]$ и $[x]_2 = \left[\frac{99\pi}{100}, \frac{101\pi}{100}\right]$ с использованием естественной, средней, тейлоровской и минимальной функций включения, связанных с функцией f, заданной выражением* (2.122) (с. 58). *Сравните результаты вычислений по этим функциям включения.* ■

Упражнение 11.11. *Рассмотрите уравнение*

$$f(\mathbf{x}) = 0, \tag{11.1}$$

где

$$f(\mathbf{x}) = x_1 \exp(x_2) + \sin(x_3). \tag{11.2}$$

Априорные области значений переменных x_1, x_2 и x_3 выбираются как

$$[-1000, 0] \times [-10, 10] \times [-3{,}14, 3{,}14]. \tag{11.3}$$

Целью данного упражнения является реализация алгоритма метода вперед-назад (см. параграф 4.2.4, с. 107) *для вычисления областей переменных x_1, x_2 и x_3, совместных с уравнением* (11.1). *Сначала напишите алгоритм вычисления области $[y] = [f]([x_1], [x_2], [x_3])$ на основе прямого этапа метода вперед-назад, и проверьте, что $0 \in [y]$. Затем, используя данные* табл. 4.6 (с. 110) *реализуйте обратный этап метода вперед-назад. Приводит ли это к эффективному сужению оцениваемых областей? Не забудьте проверить, не является ли пересечение пустым, что будет сигнализировать вам, что решения для* (11.1) *в области* (11.3) *не существует. Исследуйте другие априорные области.* ■

11.6. Интервальные векторы

Чтобы построить класс интервальных векторов, потребуется привлечь еще несколько понятий языка **C++**. Доступно несколько подходов к созданию интервальных векторов. Самый простой из них — использование массива. Например,

```
float a[5];
```

создает массив a из пяти элементов типа `float`, т. е. отводит место под пять элементов типа `float` в памяти машины. Доступ к элементам этого массива выполняется следующим образом:

```
a[0] = 4;                    // заносит первое число массива a
a[3] = 3.5;
a[4] = a[0] * a[3];          // заносит последнее число массива a
```

Заметим, что в отличие от обычной математической индексации первый элемент этого массива индексируется 0, а не 1. Размер массива *задается во время компиляции*, что оказывается неудобным. Чтобы можно было обрабатывать *динамические* массивы (т. е. массивы изменяемых размеров) необходимо обратиться к использованию указателей (параграф 11.2.3, стр. 374). Прежде всего должен быть создан указатель, который будет указывать на место, куда будет заноситься первая компонента этого вектора:

```
float *pa;  // pa указывает на float или на массив из float.
```

Далее память должна быть *динамически* выделена (т. е. во время работы), чтобы запомнить n элементов типа `float`. Это выполняется следующим выражением:

```
pa = new float[n];      // массив из n float создан
```

Разумеется, положительной целочисленной переменной n должно быть задано соответствующее числовое значение. Доступ к этим элементам должен производиться следующим образом:

```
pa[n-1] = pa[0] + pa[4];                  // заносится последний
                                          //элемент массива pa
```

Чтобы защититься от нехватки памяти, надо позаботиться об освобождении памяти, когда она больше не нужна. Это делается оператором `delete` для скаляров и `delete[]` для массивов.

```
delete[] pa;  // освобождает память отведенную для массива pa
```


Рассмотрим массив a из пяти элементов типа `float`. Выражение

```
a[5] = 6.4;                          // записывает вне массива
```

заносит число в шестой элемент массива a, который находится вне памяти, отведенной данному массиву. В результате фрагмент памяти, в принципе доступный для других программ, перезаписывается, возможно, с фатальными последствиями, хотя при компиляции не вырабатывается сообщения об ошибке. Этого можно избежать, если реализовано управление доступом. В языке **C++**, как будет показано в следующем параграфе, для решения проблем управления памятью могут создаваться векторные классы.

11.6.1. Класс INTERVAL_VECTOR

Интервальный вектор может быть описан числом своих компонент, записываемым целым числом, и адресом в памяти, где размещается его первая компонента. Этот адрес может быть запомнен в указателе на INTERVAL. Упомянутое целое число и указатель являются частными свойствами INTERVAL_VECTOR, заголовочный файл которого (называемый `ivalvec.h` снова для совместимости с библиотекой PROFIL/BIAS) может иметь следующий вид:

```
//-------------------------------------------------------------
// Файл:    ivalvec.h
// Назначение: описание класса INTERVAL_VECTOR
#ifndef __INTERVAL_VECTOR__
#define __INTERVAL_VECTOR__

#include "ival.h"           // чтобы использовать INTERVAL
#include <iostream.h>       // для базового неформатированого
                            // ввода-вывода
class INTERVAL_VECTOR{
  private:
    int nElements;
    INTERVAL* theElements;
  public:
// конструкторы
    INTERVAL_VECTOR()              // конструктор по умолчанию
      {nElements = 0; theElements = NULL;}
    INTERVAL_VECTOR(int n)         // конструктор инициализации
      {// здесь может быть вставлена некоторая операция,
       // контролирующая что n > 0
     nElements = n; theElements = new INTERVAL[n];}
    INTERVAL_VECTOR(const INTERVAL_VECTOR&);  // конструктор
                                              // копирования
```

```
// деструктор
    ~INTERVAL_VECTOR()
      {delete[] theElements;}
// другие члены-функции
  // присваивание
    INTERVAL_VECTOR& operator=(const INTERVAL_VECTOR&);
  // оператор вызова функции
    INTERVAL& operator()(int);
// функции-друзья
  // функции чтения
    friend int Dimension(const INTERVAL_VECTOR& a)
      { return a.nElements; }
  // перегружение операторов
    friend INTERVAL_VECTOR operator+
      (const INTERVAL_VECTOR&, const INTERVAL_VECTOR&);
    friend INTERVAL_VECTOR operator-
      (const INTERVAL_VECTOR&, const INTERVAL_VECTOR&);
    friend INTERVAL_VECTOR operator*
      (const INTERVAL_VECTOR&, const INTERVAL&);
    friend ostream& operator<<(ostream&,
                               const INTERVAL_VECTOR&);
//...
};
#endif
//-------------------------------------------------------------
```

В дальнейшем изложении различные части этого заголовка будут пояснять-ся, и читателю нужно будет возвращаться к этому фрагменту.

ЗАМЕЧАНИЕ 11.5. Различные векторные классы библиотеки PROFIL/BIAS содержат еще одно свойство, именно, целое `IsTemporary`, которое служит флагом, показывающим, что некоторый вектор является временным результатом некоторой промежуточной операции и может быть уничтожен после ее завершения. Использование этого флага улучшает эффективность программы, хотя и за счет некоторого ограничения синтаксиса. Этот вопрос не будет рассматриваться в ВАШЕЙ БИБЛИОТЕКЕ. ■

11.6.2. Конструкторы, операторы присваивания и вызова функции

Динамическое выделение памяти необходимо всегда, когда создается некоторый непустой вектор. Это выполняется всеми конструкторами класса INTERVAL_VECTOR, за исключением конструктора «по умолчанию», который создает вектор без компонент. В качестве иллюстрации рассмотрим конструктор копирования, который помещается в `ivalvec.cpp` (конструктор по умолчанию и конструктор инициализации, существенно более короткие, находятся в `ivalvec.h`):


```
//-------------------------------------------------------------
// Файл:      ivalvec.cpp
// Назначение:  реализация класса INTERVAL_VECTOR
#include "ivalvec.h"    // для использования INTERVAL_VECTOR

INTERVAL_VECTOR::INTERVAL_VECTOR
                                  (const INTERVAL_VECTOR& v)
{
   nElements = v.nElements;
   if (v.theElements == NULL) theElements = NULL;
   else theElements = new INTERVAL[nElements];
   for (int i = 0; i < nElements; i++)
     theElements[i] = v.theElements[i];
}
//...
//-------------------------------------------------------------
```

ЗАМЕЧАНИЕ 11.6. Для простоты содержание каждой компоненты v здесь копируются в цикле. Это можно было бы сделать быстрее копированием содержимого массива, указываемого как `v.theElements`, в массив, указываемый как `theElements`, используя функцию `memcpy` стандартного модуля `memory`. ■

Таким образом, число компонент нового объекта класса INTERVAL_VECTOR делается равным числу компонент v, и динамически выделяется такой же объем памяти, что и для v. Далее, каждая компонента v копируется в свой аналог в массиве, отведенном оператором `new`. Когда выделение памяти невозможно из-за ее нехватки, оператор `new` выдает указатель NULL, который может быть использован для обнаружения и обработки этой ошибки (см. параграф 11.13 с. 427).

Деструктор вызывается системой каждый раз, когда экземпляр класса больше не будет использоваться. Этим решается вопрос об освобождении соответствующей памяти, как это было показано выше в `ivalvec.h`.

Оператор присваивания похож на конструктор копирования:

```
//-------------------------------------------------------------
// Файл:      ivalvec.cpp (продолжение)
//...
INTERVAL_VECTOR& INTERVAL_VECTOR::operator=
                                  (const INTERVAL_VECTOR& v)
{
   if (this == &v) return (*this);
   if (nElements != v.nElements)
```

```
    { nElements = v.nElements; delete[] theElements; }
  if (v.theElements == NULL) theElements = NULL;
  else theElements = new INTERVAL[nElements];
  for (int i = 0; i < nElements; i++)
    theElements[i] = v.theElements[i];
  return (*this);
}
//...
//-------------------------------------------------------------
```

Сначала адрес v сравнивается с адресом вызывающего объекта (this), чтобы предотвратить самоприсваивание. Если два вектора имеют разное число элементов, то размер вызывающего объекта должен быть изменен. Далее память, отведенная для theElements, освобождается перед тем, как она будет перераспределена. Содержание v копируется в this. В качестве результата выдается ссылка на вызывающий объект, чтобы позволить организовать последовательные присваивания как в a = b = c;, где a, b и c являются векторами класса INTERVAL_VECTOR.

Упражнение 11.12. *На основе* ivalvec.h *напишите заголовочный файл* vector.h *модуля, реализующего векторы типа* double. ■

Оператор *вызова функции*, обозначенный как Operator()(...), может быть использован для доступа к элементам вектора с проверкой диапазона приема компонент векторов при управлении границами. Возможная реализация имеет следующий вид:

```
//-------------------------------------------------------------
// Файл:      ivalvec.cpp (продолжение)
//...
#include <stdlib.h>                          // для функции exit
//...
INTERVAL& INTERVAL_VECTOR::operator()(int i)
{
  if ((i < 1)||(i > nElements))  // базовая обработка ошибки
  { cout << "попытка нарушения границы"
         << "INTERVAL_VECTOR " << (*this) << endl;
    exit(EXIT_FAILURE); }
  return (*(theElements + i - 1)); // возвращает i-ый элемент
}
//...
//-------------------------------------------------------------
```

Индекс i сначала проверяется на принадлежность диапазону допустимых индексов (от 1 до nElements). После проверки либо генерируется ошибка, и инструкция exit(EXIT_FAILURE) вызывает прекращение работы функции, либо выдается ссылка на INTERVAL, соответствующий данному индексу. Эта ссылка обеспечивает доступ к компонентам данного вектора на чтение и на запись.


```
INTERVAL_VECTOR a(4);    // создается 4-компонентный вектор
a(3) = 4;
a(4) = a(3);
a(6) = 0;              // выдает ошибку
```

Возможность введения инструкций для обнаружения и обработки ошибок в теле членов-функций проиллюстрирована на операторе вызова функции. Для простоты, строки, реализующие подобные обнаружение и обработку ошибок в коде следующих функций, будут опущены. Но на практике функции не должны обходиться без этих фрагментов, если нужно построить устойчивую библиотеку (см. также параграф 11.13, с. 427).

ЗАМЕЧАНИЕ 11.7. Оператор operator()(...) был реализован так, что наименьший допустимый индекс равен 1 (как обычно в математических выражениях). Когда i равно 1, выдается *(theElements), т. е. первая компонента этого вектора. ■

ЗАМЕЧАНИЕ 11.8. Проверка if((i<1)||(i>nElements)) содержит логический оператор ИЛИ, обозначаемый в C++ как "||". Логический оператор И обозначается как &&, а оператор дополнения ¬ обозначается восклицательным знаком. Например, !(5>0) есть ложь. ■

11.6.3. Функции-друзья

Первая из функций-друзей в ivalvec.h обеспечивает только чтение размера этого вектора. Последующие функции-друзья реализуют операции сложения, вычитания и умножения на скаляр для INTERVAL_VECTOR. Это делается покомпонентным применением соответствующих скалярных операций. Перегрузка оператора operator+ сложения двух INTERVAL_VECTOR выполняется, например, функцией

```
//-------------------------------------------------------------
// Файл:      ivalvec.cpp (продолжение)
//...
INTERVAL_VECTOR operator+(const INTERVAL_VECTOR& a,
                          const INTERVAL_VECTOR& b)
{
  // создает вектор для запоминания результата
  INTERVAL_VECTOR res(a.nElements);
  for (int i = 1; i <= a.nElements; i++)
    res(i) = a(i) + b(i);
```

```
  return (res);
}
//...
//-------------------------------------------------------------
```

Здесь выполняется покомпонентное сложение. Эта реализация далека от оптимальной, так как для каждого сложения скалярных интервалов трижды происходит обращение к перегружаемому оператору вызова функции, что влечет за собой много ненужных проверок диапазонов индексов. Эта операция сложения может быть реализована более эффективно в виде

```
res.theElements[i-1] = a.theElements[i-1]
                              + b.theElements[i-1];
```

что не использует оператора вызова функции.

Упражнение 11.13. *Введите в* ivalvec.h *и* ivalvec.cpp *перегружение операторов*:

1) operator- *для вычитания двух* INTERVAL_VECTOR,
2) operator* *для умножения* INTERVAL_VECTOR *на* INTERVAL,
3) operator* *для умножения* INTERVAL *на* INTERVAL_VECTOR,
4) operator« *для вывода результата* INTERVAL_VECTOR *в выходной информационный поток.*
   Эта функция может быть реализована как дружественная функция с заголовком

```
friend ostream& operator<<(ostream&,
                           const INTERVAL_VECTOR&);
```

■

Упражнение 11.14. *Напишите в* ivalvec.h *и* ivalvec.cpp *дружественную функцию, вычисляющую* INTERVAL_VECTOR, *соответствующий интервальной оболочке двух* INTERVAL_VECTOR. *Заголовок этой функции может иметь вид*

```
friend INTERVAL_VECTOR Hull(const INTERVAL_VECTOR& a,
                            const INTERVAL_VECTOR& b);
```

■

Упражнение 11.15. *Напишите в* ivalvec.h *и* ivalvec.cpp *дружественную функцию, вычисляющую пересечение двух* INTERVAL_VECTOR. *Заголовок этой функции может иметь вид*

```
friend int Intersection(INTERVAL_VECTOR& r,
        const INTERVAL_VECTOR& a, const INTERVAL_VECTOR& b);
```


где r — *пересечение* a *и* b. *Эта функция должна выдавать* 1, *если интервальные векторы пересекаются, и* 0 *в противном случае.* ■

Упражнение 11.16. *Напишите в* ivalvec.h *и* ivalvec.cpp *дружественную функцию, проверяющую включение одного интервального вектора в другой интервальный вектор. Это можно сделать перегруженным оператором* operator<=, *и вид заголовка может быть*:

```
friend int operator<=(const INTERVAL_VECTOR& a,
                      const INTERVAL_VECTOR& b);
```

Проверка должна выдавать 1, *если вектор* a *включен в вектор* b, *и* 0 *в противном случае.* ■

11.6.4. Сервисные процедуры (утилиты)

Реализация таких алгоритмов, как SIVIA, требует специальных процедур, которые мы будем хранить в модуле util. Этот модуль реализует, например, бисекцию интервального вектора x по его i-й компоненте с помощью функций Lower и Upper, которые вычисляют два интервальных вектора, полученных в результате бисекции.

```
//-------------------------------------------------------------
// Файл:      util.h
// Назначение:  утилиты для INTERVAL
#ifndef __UTILITIES__
#define __UTILITIES__
#include "ivalvec.h"    // чтобы использовать INTERVAL_VECTOR
INTERVAL_VECTOR Lower (const INTERVAL_VECTOR& x, int i);
INTERVAL_VECTOR Upper (const INTERVAL_VECTOR& x, int i);
//...
#endif
//-------------------------------------------------------------
```

ЗАМЕЧАНИЕ 11.9. В оставшейся части книги мы предпочитаем называть *левым параллелотопом* и *правым параллелотопом* два параллелотопа, появляющихся после бисекции, по аналогии с левым и правым поддеревьями (см. параграф 3.3.2, с. 76). Здесь обозначения *нижний* и *верхний* используются вместо *левый* и *правый*, чтобы обеспечить совместимость с библиотекой PROFIL/BIAS. ■

Реализация Upper может иметь следующий вид:

```
//-------------------------------------------------------------
// Файл:      util.cpp
// Назначение:  утилиты для INTERVAL
#include "util.h"
```

```
INTERVAL_VECTOR Upper (const INTERVAL_VECTOR& x, int i)
{
  INTERVAL_VECTOR t(x);
  t(i) = INTERVAL(Mid(x(i)), Sup(x(i)));
  return t;
}
//...
//-------------------------------------------------------------
```

Упражнение 11.17. *Реализуйте* `Lower`. ■

11.7. Векторы с использованием библиотеки PROFIL/BIAS

В библиотеке PROFIL/BIAS имеются три векторных класса:

Имя модуля	Имя класса	Описание
`vector`	VECTOR	векторы типа `double`
`intvec`	INTEGER_VECTOR	векторы типа `int`
`ivalvec`	INTERVAL_VECTOR	интервальные векторы

Использование класса INTERVAL_VECTOR требует включения в общий проект модулей `bias0` и `bias1` (или библиотеки, их содержащей). Следующая простая программа иллюстрирует некоторые основные свойства векторов, реализованных с использованием библиотеки PROFIL/BIAS.

```
//-------------------------------------------------------------
// Файл:    addvect.cpp
// Назначение: сложение двух INTERVAL_VECTOR
#include "ivalvec.h"           // библиотека INTERVAL_VECTOR

void main()
{
  INTERVAL_VECTOR a(3);                    // размерность a
  INTERVAL_VECTOR b(3);                   // и b равна трем
  INTERVAL_VECTOR c;                  // c не инициализирован

  a(1) = INTERVAL(5,7);
  a(2) = INTERVAL(-2,3);
  a(3) = -2;     // все компоненты a теперь инициализированы
  b = a;
  b(3) = INTERVAL(4,5);
  c = a + b;
```



```
  // выводится на дисплей
  cout << a << " + " << b << " = " << c << endl;
}
//-------------------------------------------------------------
```

После объявления трех векторов INTERVAL_VECTOR, два из них инициализируются и складываются. Результат запоминается в третьем векторе, и результат отображается на дисплее. Размер вектора c подстраивается под размер суммы векторов `a + b`; таким образом, управление памятью прозрачно для пользователя.

Также доступны и другие функции, такие как умножение или деление вектора на скаляр, скалярное произведение и т. д., (см. документацию библиотеки PROFIL/BIAS).

11.8. Упражнения на интервальные векторы

Целью данного параграфа является создать и использовать первую версию алгоритма SIVIA.Потребуются только средства интервальных вычислений, разработанные для ВАШЕЙ БИБЛИОТЕКИ, но все упражнения этого параграфа могут быть выполнены также с помощью библиотеки PROFIL/BIAS.

Разрабатываемые функции будут помещены в модуль, названный `sivia`.

```
//-------------------------------------------------------------
// Файл:     sivia.h
// Назначение:  первая версия алгоритма Sivia
#include "ivalvec.h"  // чтобы использовать INTERVAL_VECTOR

// определим новый тип "интервальные булевские переменные"
typedef enum{IB_TRUE, IB_FALSE, IB_INDET} INTERVAL_BOOL;
// PIBT означает "Pointer to an Interval Boolean Test~---
// указатель на интервальную булевскую проверку"
typedef INTERVAL_BOOL (*PIBT)(const INTERVAL_VECTOR&);
double MaxDiam (const INTERVAL_VECTOR&, int&);
void Sivia (PIBT, const INTERVAL_VECTOR&, double);
//-------------------------------------------------------------
```

`sivia.h` начинается с определения двух новых типов переменных (с использованием ключевого слово `typedef`). Первый новый тип INTERVAL_BOOL соответствует интервальным булевским переменным (см. параграф 2.5.1, с. 61), т.е. переменным, которые могут принимать свои значения из множества, определенного после ключевого слова `enum`,

а именно, из IB_TRUE в случае истинности, IB_FALSE в случае ложности и IB_INDET в случае неопределенности. Префикс IB_ был введен для избежания конфликта с обозначениями TRUE и FALSE библиотеки PROFIL/BIAS. Второй новый тип PIBT соответствует указателям на функции интервальных булевских проверок. Он позволяет передавать функцию в качестве параметра другой функции. Таким образом, интервальная булевская проверка может быть передана в алгоритм Sivia как параметр. Это делает алгоритм Sivia гибким, позволяя ему работать с различными проверками.

Синтаксис определения типа указателя на функцию имеет вид:

```
typedef return_type (*имя типа)(типы параметров);
```

Следовательно, указатель типа PIBT указывает на функцию, имеющую в качестве параметра ссылку на неизменяемый INTERVAL_VECTOR, как на параметр, и возвращающую INTERVAL_BOOL. Заголовок функции, реализующей проверку, должен выглядеть следующим образом:

```
INTERVAL_BOOL имя проверки (const INTERVAL_VECTOR&);
```

Как только проверка определена, алгоритм Sivia может быть вызван следующим образом:

```
Sivia(имя проверки, X, eps);
```

где X — предварительно заданный параллелотоп поиска, а eps — предварительно заданный параметр точности (см. параграф 3.4.1, с. 81). Чтобы описать множество, найденное по другой проверке, достаточно создать другую функцию, подобную функции "имя проверки".

Упражнение 11.18. *Постройте интервальную булевскую проверку* IvalBoolTest, *которая будет использоваться алгоритмом* Sivia *для описания множества*

$$S = \left\{(x,y) \in \mathbb{R}^2 \;\middle|\; x^4 - 4x^2 + 4y^2 \in [-0{,}1, 0{,}1]\right\}.$$

Эта проверка должна выдавать IB_TRUE, если $[x]^4 - 4[x]^2 + 4[y]^2 \subset [-0{,}1, 0{,}1]$, IB_FALSE, если $[x]^4 - 4[x]^2 + 4[y]^2 \cap [-0{,}1, 0{,}1] = \varnothing$ *и,* IB_INDET *в противном случае.* ■

Результат следующего упражнения будет использоваться алгоритмом Sivia для выбора оси (направления), по которому параллелотопы с неопределенностью будут подвергаться бисекции.

Упражнение 11.19. *Составить программу* MaxDiam, *использующую функцию* Diam, *доступную в модуле* ival. *Функция* MaxDiam *должна выдавать число типа* double, *содержащее ширину параллелотопа, переданного как параметр. Индекс первой компоненты параллелотопа с максимальной шириной по ней должен быть передан обратно в вызывающую функцию ссылкой на* int. ■

Упражнение 11.20. *Составить программу* Sivia, *имеющую рекурсивную структуру (т. е. подпрограмма сама себя вызывает). Вначале должна быть вызвана* IvalBoolTest. *Если ее результат есть истина или ложь, то он должен быть отображен, и далее должен следовать возврат из функции. Иначе этот параллелотоп является неопределенным, и его ширина должна быть вычислена с использованием* MaxDiam. *Если обнаруживается, что ширина параллелотопа меньше параметра точности* eps, *то должно быть выдано соответствующее сообщение, и далее должен следовать возврат из функции. Иначе текущий параллелотоп должен подвергнуться бисекции с использованием* Lower *и* Upper, *после чего алгоритм* Sivia *должен быть вызван для обработки каждого из полученных подпараллелотопов.* ■

Решения Упражнений 11.18–11.20 приводятся ниже:

```
//-------------------------------------------------------------
// Файл:     sivia.cpp
// Назначение:  первая версия алгоритма Sivia
#include "sivia.h"
#include "util.h"          // чтобы использовать Lower и Upper

int MaxDiam(const INTERVAL_VECTOR& x)
{
  int mdcomp = 1;                                  // инициализация
  double diam = Diam(x(mdcomp));
  // нужно сравнить ширину всех компонент между собой
  for (int i = 2; i <= Dimension(x); i++)
    if (Diam(x(i)) > diam)                  // компонента a больше
      { diam = Diam(x(i)); mdcomp = i; } // найденная ширина
  return mdcomp;
}

void Sivia(PIBT IBTest, const INTERVAL_VECTOR& x,
                                                  double eps)
{
   int test = IBTest(x);
   if (test == IB_TRUE)
   { cout << x << " принадлежит S" << endl;
```

```
    return; }
  if (test == IB_FALSE)
  { cout << x << " не пересекается с S" << endl;
    return; }

  int maxdiamcomp;
  if (test == IB_INDET)
  { if (MaxDiam(x, maxdiamcomp) < eps)
      { cout << x << " результат неопределен" << endl;
        return; }                    // x~--- слишком мал, чтобы
                                     // подвергаться бисекции
   // x разбивается на  xlow и xup
     INTERVAL_VECTOR xlow = Lower(x, maxdiamcomp);
     INTERVAL_VECTOR xup = Upper(x, maxdiamcomp);
   // и рекурсивно вызывается Sivia
     Sivia(IBTest, xlow, eps);
     Sivia(IBTest, xup, eps);
  }
}
//-------------------------------------------------------------
```

Чтобы использовать SIVIA, теперь достаточно определить обратную проверку и написать функцию `main()`, которая вызовет `Sivia()`:

```
//-------------------------------------------------------------
// Файл:      siviademo.cpp
// Назначение:  первая проверка Sivia
#include "sivia.h"
#include "func.h"
// Определение проверки
INTERVAL_BOOL IvalBoolTest (const INTERVAL_VECTOR& x) {
  INTERVAL z = Sqr(x(1)) * (Sqr(x(1)) - INTERVAL(4))
                         + Sqr(x(2)) * INTERVAL(4);
  INTERVAL r;                // вспомогательный интервал,
                             // который будет содержать
                             // пересечение z и [-0.1, 0.1]
  if (z <= INTERVAL(-0.1, 0.1)) return IB_TRUE;
  if (Intersection(r, z, INTERVAL(-0.1,0.1)) == 0)
    return IB_FALSE;
  return IB_INDET;
}

void main()
{
  INTERVAL_VECTOR x(2);                  // параллелотоп поиска
  x(1) = INTERVAL(-5,5);
```



```
  x(2) = INTERVAL(-5,5);
  Sivia(IvalBoolTest, x, 0,05);   // eps задано равным 0,05
}
//-------------------------------------------------------------
```

Эта версия SIVIA вырабатывает только текстовую информацию. Можно добавить графический вывод, но это лежит вне круга вопросов данной книги. На сайте

http://www.lss.supelec.fr/books/intervals

читатель найдет версию программы SIVIA, записывающей множество решения в файл, который можно импортировать для рисования, например, библиотекой MATLAB. Далее могут быть получены рисунки, подобные рис. 3.12 (с. 90).

Упражнение 11.21. *Составьте программу интервальной булевской проверки* `IBTAnnular` *для проверки принадлежности некоторого параллелотопа из* $\mathbb{IR}^2$ *области* $\mathcal{S}_c$, *лежащей между окружностями с центром в начале координат и с радиусами* 1 *и* 2. *Описать область* $\mathcal{S}_c$, *используя* `Sivia`. ■

Упражнение 11.22. *Составьте программу интервальной булевской проверки* `IBTUpRight` *для проверки принадлежности некоторого параллелотопа из* $\mathbb{IR}^2$ *множеству* $\mathcal{S}_q$, *соответствующему верхнему правому квадранту вещественной плоскости* $\mathbb{R}^2$. *Опишите множество* $\mathcal{S}_q$, *используя* `Sivia`. ■

Упражнение 11.23. *Составьте программу булевской проверки* `IBTInt` *для проверки включения параллелотопа из* $\mathbb{IR}^2$ *в множество* $\mathcal{S}_i = \mathcal{S}_c \cap \cap \mathcal{S}_q$, *где* $\mathcal{S}_c$ *и* $\mathcal{S}_q$ *определены как в* Упражнениях 11.21 *и* 11.22. *Опишите множество* $\mathcal{S}_i$, *используя* `Sivia`. ■

Упражнение 11.24. *Оцените вещественную и мнимую части полинома* $P(j\omega, \mathbf{p})$, *описанного в* Примере 7.9 (с. 263). *Используя* `Sivia`, *докажите, что задача выполнения ограничений*

$$\mathcal{H} : (P(j\omega, \mathbf{p}) = 0,\ \mathbf{p} \in [\mathbf{p}],\ \omega \in [0\,\text{рад/с}, 24\,\text{рад/с}])$$

не имеет решения и что полином $P(s, \mathbf{p})$ *робастно устойчив.* ■

11.9. Интервальные матрицы

Построение полного класса `INTERVAL_MATRIX` является сложной задачей, потому что велико число функций-друзей, необходимых для

реализации всех возможных взаимодействий с другими классами. Вот почему мы ограничимся описанием только некоторых идей того, как это сделано в библиотеке PROFIL/BIAS, перед тем, как представить некоторые свойства этой библиотеки, которые будут полезны для реализации алгоритмов, описываемых в этой книге. Примеры таких реализаций составят предмет упражнений.

Размер матрицы описывается числом ее строк и столбцов, представляемыми здесь двумя положительными целыми числами `nRows` и `nCols`. Для запоминания матрица преобразуется в одномерный массив по строкам. Элемент матрицы, расположенный на пересечении `r`-й строки и `c`-го столбца, запоминается в элементе массива с номером `(r-1)*nCols+(c-1)`. Обращение к этому массиву идет через указатель `theElements`. Элемент памяти, указываемый `theElements`, используется, таким образом, для хранения левого верхнего элемента матрицы. Заголовочный файл интервальной матричной библиотеки `ivalmat.h` может иметь вид:

```
//-------------------------------------------------------------
// Файл:      ivalmat.h
// Назначение:  определение класса INTERVAL_MATRIX
#include "ivalvec.h"    // чтобы использовать INTERVAL_VECTOR

class INTERVAL_MATRIX{
  private:
    int nRows, nCols;
    INTERVAL *theElements;
  public:
// конструкторы
    INTERVAL_MATRIX()                  // конструктор по умолчанию
      {nRows = 0; nCols = 0; theElements = NULL;}
    INTERVAL_MATRIX(int r, int c) // конструктор инициализации
      {nRows = r; nCols = c;
       theElements = new INTERVAL[r * c];}
     //...
// деструктор
    ~INTERVAL_MATRIX() {delete[] theElements;}
// другие члены-функции
  // оператор вызова функции
    INTERVAL& operator()(int r, int c)
    { return (theElements[(r-1) * nCols + (c-1)]); }
  //...
// функции-друзья
  // функции чтения
    friend int RowDimension(const INTERVAL_MATRIX& x)
      { return x.nRows; }
    friend int ColDimension(const INTERVAL_MATRIX& x)
      { return x.nCols; }
//...
};
//-------------------------------------------------------------
```


Конструктор по умолчанию строит пустую матрицу. Конструктор инициализации создает массив из `r*c` интервалов, начиная с места положения памяти `theElements`. Деструктор освобождает отведенную память.

Так же как и в случае интервальных векторов, оператор вызова функции используется для доступа к заданному элементу этой матрицы. Выдается ссылка на этот элемент, чтобы позволить изменять его значения с учетом того способа, каким эта матрица была сохранена. Присваивание, сложение и вычитание матриц, их умножение и деление на скаляры выполняется так же, как в классе `INTERVAL_VECTOR`.

Классы для матриц с вещественными или целочисленными элементами строятся в соответствии с той же моделью. Начиная с этого места, читателю настоятельно предлагается использовать матричные классы, представляемые библиотекой PROFIL/BIAS.

11.10. Матрицы с использованием библиотеки PROFIL/BIAS

Три матричных класса обеспечиваются библиотекой PROFIL/BIAS.

Имя модуля	Имя класса	Описание
`matrix`	`MATRIX`	матрица с элементами `double`
`intmat`	`INTEGER_MATRIX`	матрица с элементами `int`
`ivalmat`	`INTERVAL_MATRIX`	интервальная матрица

Использование класса `INTERVAL_MATRIX` требует, чтобы модули `bias0`, `bias1` и `bias2` (или библиотека, их содержащая) были включены в общий проект. Следующий пример иллюстрирует некоторые свойства класса `INTERVAL_MATRIX`.

```
//-------------------------------------------------------------
// Файл:     matxdemo.cpp
// Назначение: использование матриц
#include "ivalvec.h"           // библиотека INTERVAL_VECTOR
#include "ivalmat.h"           // библиотека INTERVAL_MATRIX

void main()
```

```
{
  INTERVAL_MATRIX a(4,3);      // a имеет 4 строки и 3 столбца
  INTERVAL_VECTOR b(3);
  INTERVAL_VECTOR c;                    // c не инициализирован

  Initialize(a,INTERVAL(-1,1));   // все элементы есть [-1,1]
  a(2,3) = INTERVAL(-2,3);       // элемент теперь есть [-2,3]
  a(3,3) = -2;                  // элемент теперь есть [-2,-2]
  c = Row(a,2);                    // доступ ко второй строке
  c = Col(a,3);                  // доступ к третьему столбцу
  b(1) = 1; b(2) = INTERVAL(1,2); b(3) = INTERVAL(4,5);
  c = a * b;          // результат умножения матрицы на вектор

  // выводится на экран
  cout << a << " * " << b << " = " << c << endl;
}
//-------------------------------------------------------------
```

Доступны и многие другие функции. Так функции `Inf`, `Sup`, `Mid`, `Diam` вычисляют, соответственно, нижнюю и верхнюю границы, центр и ширину всех элементов интервальной матрицы `INTERVAL_MATRIX`. Результаты запоминаются в `MATRIX`. Модулем `util` обеспечиваются также такие полезные функции, как:

- `MATRIX Inverse(MATRIX&);`, которая выдает матрицу, обратную к матрице из `double`;
- `MATRIX Transpose(MATRIX&);`, которая выдает транспонированную матрицу к матрице из `double`;
- `MATRIX Id(INT n);`, которая выдает единичную $\mathtt{n} \times \mathtt{n}$ матрицу.

Эти функции недоступны для класса `INTERVAL_MATRIX`. Читателю предлагается взглянуть на заголовочный файл `ivalmat.h`, который достаточно понятен сам по себе, или заглянуть в документацию библиотеки PROFIL/BIAS.

11.11. Упражнения на интервальные матрицы

В этом параграфе библиотека PROFIL/BIAS будет использована для реализации некоторых средств интервальных вычислений, представленных в параграфе 4.2 для решения задач выполнения ограничений. Эти инструменты будут применены в новом модуле `contractors` (сжимающие операторы).

Упражнение 11.25. (Интервализация метода исключения Гаусса) *Задача состоит в нахождении параллелотопа, содержащего множество решения задачи выполнения ограничений*

$$\mathcal{H} : \left(\begin{array}{l} \mathbf{A} \in [\mathbf{A}], \mathbf{b} \in [\mathbf{b}], \mathbf{p} \in [\mathbf{p}] \\ \mathbf{Ap} - \mathbf{b} = \mathbf{0} \end{array} \right). \tag{11.4}$$

Сжимающий оператор на основе метода исключения Гаусса, представленный в параграфе 4.2.2 (с. 98), *пытается уменьшить область значений вектора параметров* $\mathbf{p}$. *Он использует только основные операции над интервальными векторами и матрицами. Заголовочный файл модуля* `contractors` *может включать*

```
//-------------------------------------------------------------
// Файл:     contractors.h
// Назначение:  реализация некоторых операторов сжатия
#include "ivalmat.h"                    // чтобы использовать
                                       // интервальные матрицы

// сжимающий оператор на основе метода исключения Гаусса
INTERVAL_VECTOR GaussElimination(INTERVAL_MATRIX A,
        INTERVAL_VECTOR b, INTERVAL_VECTOR p, INT& err);
//...
//-------------------------------------------------------------
```

1. *Реализовать* `GaussElimination`, *опираясь на* табл. 4.3 (с. 101). *Заметим, что этот сжимающий оператор изменяет параметры матрицы* `A` *и вектора* `b`. *Для предотвращения этого эти объекты могут быть переданы по значению. Флаг ошибки* `err` *может быть использован для выдачи состояния ошибки данной процедуры. Если возникла проблема, то этому флагу может быть приписано любое значение, кроме* 0, *а* 0 *будет означать, что ошибок не обнаружено.*
2. *Примените* `GaussElimination` *для областей*

$$[\mathbf{A}] = \begin{pmatrix} [4,5] & [-1,1] & [1{,}5,2{,}5] \\ [-0{,}5,0{,}5] & [-7,-5] & [1,2] \\ [-1{,}5,-0{,}5] & [-0{,}7,-0{,}5] & [2,3] \end{pmatrix}, \tag{11.5}$$

$$[\mathbf{b}] = \begin{pmatrix} [3,4] \\ [0,2] \\ [3,4] \end{pmatrix} \text{ и } [\mathbf{p}] = \begin{pmatrix} [-10,10] \\ [-10,10] \\ [-10,10] \end{pmatrix}. \tag{11.6}$$

3. *Проверьте идемпотентность данного сжимающего оператора.* ■

Методы с неподвижной точкой, представленные в параграфе 4.23 (с. 101), стараются решить задачу выполнения ограничений

$$\mathcal{H} : (\mathbf{f}(\mathbf{x}) = \mathbf{0}, \mathbf{x} \in [\mathbf{x}]) \tag{11.7}$$

с использованием алгоритма ψ, такого, что

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \Leftrightarrow \mathbf{x} = \psi(\mathbf{x}). \tag{11.8}$$

Два следующих упражнения посвящены построению сжимающих операторов, основанных на этой идее.

Упражнение 11.26. (Сжимающий оператор Гаусса–Зейделя) *Сжимающий оператор Гаусса-Зейделя, описанный в* параграфе 4.2.3 (с. 103), *использует преобразование задачи выполнения ограничений* (11.4) *и оценивает*

$$\operatorname{diag}([\mathbf{A}])^{-1}([\mathbf{b}] - \operatorname{extdiag}([\mathbf{A}])[\mathbf{p}]).$$

Создаваемый алгоритм должен находить матрицу $\operatorname{diag}([\mathbf{A}])^{-1}$, *используя обращение каждого диагонального элемента матрицы* $[\mathbf{A}]$; *а* $\operatorname{extdiag}([\mathbf{A}])$ *получается путем сохранения недиагональных элементов матрицы* $[\mathbf{A}]$ *и сведением диагональных элементов к нулю. Остальные вычисления содержат только простые операции над векторами и матрицами. Заголовок модуля сжимающих операторов* `contractors` *может быть снабжен заголовком*

```
// сжимающий оператор, основанный на методе исключения
// Гаусса-Зейделя
INTERVAL_VECTOR GaussSeidelIteration(INTERVAL_MATRIX A,
        INTERVAL_VECTOR b, INTERVAL_VECTOR p, INT& err);
```

Опять флаг ошибки `err` *может использоваться, чтобы сигнализировать о проблемах, возникающих при построении* $\operatorname{diag}([\mathbf{A}])^{-1}$ *из-за наличия на диагонали матрицы* $[\mathbf{A}]$ *интервала, содержащего нуль.*

1. *Реализуйте сжимающий оператор Гаусса–Зейделя.*
2. *Примените его к областям, определенным соотношениями* (11.5) *и* (11.6).
3. *Примените итерационно: сходится ли сжимающий оператор?* ■

Упражнение 11.27. (Сжимающий оператор Кравчика) *Сжимающий оператор Кравчика был описан в* параграфе 4.2.3 (с. 105). *Рассмотрим задачу выполнения ограничений*

$$\mathcal{H} : (\mathbf{f}(\mathbf{x}) = \mathbf{0}, \mathbf{x} \in [\mathbf{x}]), \tag{11.9}$$

при $\mathbf{f} : \mathbb{R}^2 \to \mathbb{R}^2$,

$$\begin{cases} f_1(x_1, x_2) = x_1^2 - 4x_2, \\ f_2(x_1, x_2) = x_2^2 - 2x_1 + 4x_2, \end{cases} \tag{11.10}$$

и

$$[\mathbf{x}] = [-0{,}1, 0{,}1] \times [-0{,}1, 0{,}3]. \tag{11.11}$$

Матрица Якоби для функции $\mathbf{f}$ *имеет вид*:

$$\mathbf{J}_\mathbf{f} = \begin{pmatrix} 2x_1 & -4 \\ -2 & 2x_2 + 4 \end{pmatrix}. \tag{11.12}$$

Реализуйте алгоритм табл. 4.5 (с. 105) *для этого примера. Здесь может оказаться полезной функция обращения* `Inverse` *из модуля утилит* `util` *библиотеки* PROFIL/BIAS. *Заметим, что построение универсального сжимающего оператора Кравчика потребовало бы использования указателей на функции, принимающие интервальный вектор как параметр и возвращающие интервальный вектор (для параллелотопа* $[\mathbf{f}]$ *) и интервальной матрицы (для интервальной матрицы* $[\mathbf{J}_\mathbf{f}]$*). Реализация и использование таких указателей подобны использованию указателей на интервальные булевские проверки, описанные в* параграфе 11.8. ■

Результаты, полученные применением сжимающего оператора на основе метода исключения Гаусса и сжимающего оператора Гаусса–Зейделя, могут быть улучшены соответствующим улучшением обусловленности системы уравнений (см. параграф 4.3.2, с. 105). Это иллюстрируется следующим упражнением.

Упражнение 11.28. (Улучшение обусловленности) *В* (11.4) *матрица* $[\mathbf{A}]$ *и вектор* $[\mathbf{b}]$ *умножаются на обусловленную матрицу* $(\operatorname{mid}([\mathbf{A}]))^{-1}$ *перед применением сжимающего оператора.*

1. *Добавьте следующие сжимающие операторы в модуль сжимающих операторов* `contractors`:

```
// операторы сжатия, использующие улучшение обусловленности
INTERVAL_VECTOR PrecGaussElimination(INTERVAL_MATRIX A,
```

```
        INTERVAL_VECTOR b, INTERVAL_VECTOR p, INT& err);
INTERVAL_VECTOR PrecGaussSeidelIteration(INTERVAL_MATRIX A,
        INTERVAL_VECTOR b, INTERVAL_VECTOR p, INT& err);
```

Функция обращения `Inverse` *из модуля утилит* `util` *библиотеки* PROFIL/BIAS *может быть использована для вычисления матрицы* $(\mathrm{mid}([\mathbf{A}]))^{-1}$.

2. *Сравните результаты работы полученных операторов сжатия с результатами, полученными без улучшения обусловленности.*

■

11.12. Регулярные покрытия с использованием библиотеки PROFIL/BIAS

Регулярные покрытия (и подпокрытия), описанные в Главе 3, не поддерживаются библиотекой PROFIL/BIAS. Целью настоящего параграфа является, таким образом, реализовать регулярные покрытия на основе объектов для обращения множеств и оценивания образа.

Покрытия (и подпокрытия) запоминаются с использованием двоичных деревьев. Хотя такие деревья классически используются в алгоритмах сортировки и символьных вычислениях, их реализация, как, например, в стандартных библиотеках на шаблонах, не адекватна нашим целям, и многие из необходимых функций не реализованы. Вот почему будет разработана их специальная реализация.

11.12.1. Класс NODE

Реализация покрытий основывается в классе NODE. Каждый узел бинарного дерева, представляющего покрытие, ассоциирован с некоторым параллелотопом этого покрытия. Этот узел содержит также указатели на узлы, соответствующие левому и правому отросткам этого паралелотопа. Класс NODE, таким образом, заключает в себе указатель на класс интервальных векторов INTERVAL_VECTOR и два указателя на узлы NODE. SUBPAVING будет представлять собой указатель на класс NODE. Заголовок модуля, реализующего класс NODE (и, следовательно, SUBPAVING) может, иметь вид:

```
//-------------------------------------------------------------
// Файл:      subpaving.h
// Назначение:  определение класса NODE и SUBPAVING
#include "ivalvec.h"                        // чтобы использовать
                                    // класс INTERVAL_VECTOR
```



```
#include <iostream.h>                                  // для базового
                                  // неформатированного ввода-вывода

class NODE;
typedef NODE* SUBPAVING;              // объявляет SUBPAVING как
                                          // альтернативное имя
                                      // указателя на класс NODE

class NODE
{
  private:
    INTERVAL_VECTOR *theBox;
    SUBPAVING leftChild;
    SUBPAVING rightChild;
  public:
// конструкторы
    NODE()                                // конструктор по умолчанию
    { theBox = NULL;
      leftChild = NULL; rightChild = NULL; }
    NODE(const INTERVAL_VECTOR& v)                    // конструктор
                                                     // инициализации
    { theBox = new INTERVAL_VECTOR(v);
      leftChild = NULL; rightChild = NULL; }
    NODE(const NODE& n);              // конструктор копирования
// деструктор
    ~NODE()
    { delete theBox;
      delete leftChild; delete rightChild; }
// функции-друзья
    friend INTERVAL_VECTOR Box(SUBPAVING a)
    { return (*a->theBox); }
    friend SUBPAVING LeftChild(SUBPAVING a)
    { return a->leftChild; }
    friend SUBPAVING RightChild(SUBPAVING a)
    { return a->rightChild; }
    friend bool IsEmpty(SUBPAVING a)
    { return ((a == NULL) || (a->theBox == NULL)); }
    friend bool IsLeaf(SUBPAVING a)
    { return (!IsEmpty(a) && IsEmpty(a->leftChild)
                          && IsEmpty(a->rightChild)); }
    friend ostream& operator<<(ostream&, SUBPAVING);
};
//-------------------------------------------------------------
```

Конструктор по умолчанию класса NODE устанавливает все его свойства в NULL. SUBPAVING, указывающий на такой узел и инициализирован-

ный, например, выражением

```
SUBPAVING A = new NODE();
```

рассматривается как пустой. Еще одним полезным представлением пустого покрытия является SUBPAVING, имеющий значение NULL и инициализированный, например, выражением

```
SUBPAVING B = NULL;
```

Хотя объекты A и B созданы по-разному, они будут вести себя одинаково в представляемом алгоритме. Конструктор инициализации определяет класс NODE, который редуцируется до листа и, таким образом, содержит только класс INTERVAL_VECTOR и два указателя NULL. Наконец, конструктор копирования, представленный ниже, имеет рекурсивную структуру, подобную уже использованной в алгоритме SIVIA. Такая рекурсивная структура является типичной в вычислениях на двоичных деревьях.

```
//-------------------------------------------------------------
// Файл:      subpaving.cpp
// Назначение:  реализация класса NODE и SUBPAVING
#include "subpaving.h"          // чтобы использовать SUBPAVING

NODE::NODE(const NODE& n)              // конструктор копирования
{ theBox = new INTERVAL_VECTOR(*n.theBox);
  // рекурсивное применение на отростках
  if (n.leftChild)
    leftChild = new NODE(*n.leftChild);
  else leftChild = NULL;
  if (n.rightChild)
    rightChild = new NODE(*n.rightChild);
  else rightChild = NULL; }
//-------------------------------------------------------------
```

Конструктор копирования INTERVAL_VECTOR вызывается первым для копирования theBox. Конструктор копирования NODE, таким образом, вызывается рекурсивно на узлы, ассоциированные с левым и правым подпокрытиями, если они существуют. Деструктор, реализованный в subpaving.h, обладает такой же рекурсивной структурой: выражения delete leftChild и delete rightChild неявно вызывают этот деструктор на левое и правое подпокрытия после освобождения памяти, отведенной для theBox.

Функции-друзья, занесенные в subpaving.h, обеспечивают доступ к некоторым свойствам классов SUBPAVING. Функции Box, LeftChild и RightChild обеспечивают доступ к свойствам покрытия (или подпокрытия), переданного в аргументе. Функции IsEmpty и IsLeaf, соответственно, проверяют, является ли покрытие (или подпокрытие) пустым или оно соответствует листу.

Упражнение 11.29. *Реализуйте для* SUBPAVING *оператор* operator« *простого вывода. Могут выводиться только параллелотопы, соответствующие листьям.* ■

11.12.2. Обращение множеств с покрытиями

Алгоритм SIVIA (параграф 3.4.1, с. 81) может рассматриваться как основная функция, работающая с покрытиями; вот почему она будет реализована дружественной функцией к классу NODE. Заголовок subpaving.h, следовательно, может быть дополнен следующим образом:

```
//-------------------------------------------------------------
// Файл:      subpaving.h (continued)
// Назначение:  определение класса NODE и SUBPAVING
//            и функций, работающих с покрытиями
//...                          в ранее включенных модулях
#include "util.h"           // чтобы использовать Lower и Upper

// определить тип "интервальные булевские переменные"
typedef enum{IB_TRUE, IB_FALSE, IB_INDET} INTERVAL_BOOL;
// определить тип "указатель на интервальную
// булевскую проверку"
typedef INTERVAL_BOOL (*PIBT)(const INTERVAL_VECTOR&);
//...
class NODE
{
//...
    friend SUBPAVING Sivia (PIBT, SUBPAVING, double);
};
// сервисная функция
  double MaxDiam (INTERVAL_VECTOR&, int&);

//-------------------------------------------------------------
```

Эта версия SIVIA выдает покрытие, отличное от версии этого алгоритма на с. 401. Эти две версии могут сосуществовать, так как язык C++ допускает перегрузку функции. В зависимости от типов аргументов будет вызываться та или иная функция. Пример обращения к алгоритму SIVIA для работы с покрытиями имеет вид:

```
SUBPAVING X = Sivia(Имя~проверки, S, eps);
```

где S — некоторое заданное поисковое покрытие, Имя проверки — интервальная булевская проверка, описывающая искомое множество (см. параграф 11.8, с.401), а eps — допуск на точность поиска. В качестве решения выдается покрытие X.

Функция MaxDiam может быть реализована в файле subpaving.cpp таким же образом, как и в параграфе 11.8 (другим является только алгоритм обращения множества):

```
//-------------------------------------------------------------
// Файл:      subpaving.cpp (продолжение)
//...
SUBPAVING Sivia (PIBT IBTest, SUBPAVING S, double eps)
{
  if (IsEmpty(S)) return NULL;
  int test = IBTest(Box(S));
  int maxdiamcomp;

  if (test == IB_FALSE)
    return NULL;
  if ((test == IB_TRUE)
            || (MaxDiam(Box(S), \ maxdiamcomp) < eps))
    return new NODE(*S);
  if (IsLeaf(S))
    Expand(S, maxdiamcomp);
  return Reunite(Sivia(IBTest, LeftChild(S), eps),
            Sivia(IBTest, RightChild(S), eps), Box(S));
// Функции Expand и Reunite для расширения и объединения
// покрытий будут показаны в последующих параграфах
}
//-------------------------------------------------------------
```

Тип SUBPAVING позволяет реализацию алгоритма SIVIA, которая строго соответствует его описанию из параграфа 3.4.1 (с. 81). Однако заметим, что здесь рассчитывается только аппроксимация сверху и что для улучшения эффективности программы были добавлены некоторые выражения.

Если искомое покрытие S пусто, то программа Sivia выдает пустое покрытие. В противном случае интервальная булевская проверка, заданная пользователем, оценивается на покрытии Box(S) — на параллелотопе, соответствующем корню S. Если верна проверка IB_FALSE, то выдается пустое покрытие. Если верна проверка IB_TRUE, или корневой параллелотоп является достаточно малым, то выдается покрытие, соответствующее копии S. Если покрытие S является листом, то функция Expand преобразует его в два неминимальных покрытия, содержащие два параллелотопа,

возникающих при бисекции параллелотопа Box(S) по его первой компоненте максимальной ширины. Далее Sivia рекурсивно вызывается на левый и правый отростки покрытия S. В конце два результирующих подпокрытия объединяются (функцией Reunite), чтобы получить минимальное покрытие.

В показанной реализации Sivia модифицирует покрытие S. Если необходимо, этого можно избежать. Вернемся теперь к функциям Expand и Reunite, которые используются в Sivia.

Расширение покрытия. Функция Expand образует на узле два родственных листа, показывая, что этот узел — вырожденный и, следовательно, соответствует листу. Так как это изменяет свойства NODE, то функция Expand реализуется как дружественная классу NODE, что требует введения выражения

```
friend void Expand(SUBPAVING S, int comp);
```

в subpaving.h. Expand выглядит следующим образом:

```
//-------------------------------------------------------------
// Файл:      subpaving.cpp (продолжение)
//...
void Expand(SUBPAVING S, int comp)
{
  if (!IsLeaf(S)) return;
  S->leftChild = new NODE(Lower(Box(S), comp));
  S->rightChild = new NODE(Upper(Box(S), comp));
}
//-------------------------------------------------------------
```

Если покрытие S, которое надо расширить, не является листом, то добавляются два отростка. Эти подпараллелотопы возникают после бисекции параллелотопа Box(S) по его координате comp.

Объединение покрытий. Функция Reunite вычисляет минимальное покрытие по двум родственным подпокрытиями. Это может быть сделано следующим образом:

```
//-------------------------------------------------------------
// Файл:      subpaving.cpp (продолжение)
//...
SUBPAVING Reunite(SUBPAVING lChild, SUBPAVING rChild,
                                    INTERVAL_VECTOR& x)
{
  if (IsEmpty(lChild) && IsEmpty(rChild)) return NULL;
```

```
  SUBPAVING result = new NODE(x); // результирующее покрытие

  if (IsLeaf(lChild) && IsLeaf(rChild))
  { delete lChild; delete rChild; return result; }

  result->leftChild = lChild;
  result->rightChild = rChild;
  return result;
}
//-------------------------------------------------------------
```

Эта реализация строго следует алгоритму, представленному в параграфе 3.3.3 (с. 77). Покрытие выдается пустым, если оба объединяемых подпокрытия пусты. Если оба подпокрытия являются листьями, то они уничтожаются, и выдается порождающий узел (который, следовательно, становится листом). Во всех других случаях оба подпокрытия присоединяются к порождающему их узлу.

Упражнение 11.30. *Составьте функцию* `NbLeaves`, *вычисляющую число листьев в* SUBPAVING. ■

Упражнение 11.31. *Составьте функцию* `Volume`, *суммирующую величины всех листьев в* SUBPAVING. ■

Целью следующего упражнения является построение аппроксимации сверху множества $\mathbf{f}^{-1}\left(\overline{\mathbb{Y}}\right)$ для любого регулярного покрытия $\overline{\mathbb{Y}}$ с помощью алгоритма SIVIA. Такая процедура включает проверку вложенности параллелотопа $[\mathbf{f}]([\mathbf{x}]) \subset \overline{\mathbb{Y}}$, т. е. принадлежности этого параллелотопа покрытию.

Упражнение 11.32. *Используя алгоритм* INSIDE *из* параграфа 3.3.3 (с. 77), *реализуйте функцию*

```
friend int operator<=(const INTERVAL_VECTOR&, SUBPAVING);
```

для проверки принадлежности вектора `INTERVAL_VECTOR` *покрытию* SUBPAVING. ■

Упражнение 11.33. *Это упражнение предназначено для иллюстрации возможности использования алгоритма* SIVIA *для вычисления прямого образа множества, даваемого некоторой функцией с учетом того, что эта функция обратима в обычном смысле.*

1. *Рассчитайте покрытие* $\overline{\mathbb{S}}_c$, *содержащее множество* $\mathcal{S}_c$ Упражнения 11.21. *Оцените размер этого покрытия.*

2. *Используйте* `Sivia` *для вычисления покрытия* $\overline{\mathbb{S}}_{c1}$, *содержащего множество* $\mathbf{f}(\overline{\mathbb{S}}_c)$, *где*

$$\mathbf{f} : (x_1, x_2)^{\mathrm{T}} \mapsto (2x_1 - x_2 - x_1 + 2x_2)^{\mathrm{T}}.$$

Заметим, что $\mathbf{f}$ *обратима в обычном смысле, и обратная функция имеет вид*

$$\mathbf{f}^{-1}\!: (x_1, x_2)^{\mathrm{T}} \mapsto \left(\tfrac{2}{3}x_1 + \tfrac{1}{3}x_2,\ \tfrac{1}{3}x_1 + \tfrac{2}{3}x_2\right)^{\mathrm{T}},$$

поэтому

$$\left\{\mathbf{f}\,(x_1, x_2) \;\middle|\; (x_1, x_2) \in \overline{\mathbb{S}}_c\right\} = \left\{(x_1, x_2) \;\middle|\; \mathbf{f}^{-1}\,(x_1, x_2) \in \overline{\mathbb{S}}_c\right\}.$$

Следовательно, для того, чтобы вычислить аппроксимацию $\overline{\mathbb{S}}_{c1}$, *достаточно выполнить обращение множества* $\overline{\mathbb{S}}_c$ *с помощью обратной функции* $\mathbf{f}^{-1}$.

3. *Рассчитайте покрытие* $\overline{\mathbb{S}}_{c2}$, *содержащее* $\mathbf{f}^{-1}\left(\overline{\mathbb{S}}_{c1}\right)$. *Это покрытие было бы эквивалентно покрытию* $\overline{\mathbb{S}}_c$, *если бы не было внесено ухудшение оценки. Оцените величину* $\overline{\mathbb{S}}_{c2}$ *и сравните ее с величиной* $\overline{\mathbb{S}}_c$. *Оцените влияние параметра* `eps` *на ухудшение оценивания.*

■

11.12.3. Оценивание образов с помощью покрытий

Программа IMAGESP (параграф 3.4.2, с. 85) выполняет основные действия с покрытиями. Таким образом, она будет реализована функцией, дружественной классу `NODE`. Заголовок модуля `subpaving`, следовательно, должен быть дополнен следующим образом:

```
//-------------------------------------------------------------
// Файл:     subpaving.h (continued)
// Назначение:  описание NODE и SUBPAVING
//           и функций, работающих с покрытиями
//...                        в ранее включенных модулях
#include "imlist.h"                 // для управления списками
                                    // INTERVAL_VECTORS
//...
// описание типа "указатель на функцию,
// возвращающую INTERVAL_VECTOR"
typedef INTERVAL_VECTOR (*PIVF)(const INTERVAL_VECTOR&);
//...
class NODE
```

```
{
//...
    // функции-друзья для базовых шагов алгоритма ImageSp
    friend void Mince(SUBPAVING, double);
    friend void Evaluate(PIVF, SUBPAVING, IMAGELIST&,
                                        INTERVAL_VECTOR&);
    friend SUBPAVING Regularize(INTERVAL_VECTOR&,
                                      IMAGELIST&, double);
    // алгоритм ImageSp
    friend SUBPAVING ImageSp(PIVF, SUBPAVING, double);
};
//...
//-------------------------------------------------------------
```

Этот модифицированный заголовок реализует указатель PIVF — новый тип указателя для интервальной векторной функции, который будет применяться для указания на векторную функцию включения, используемую в IMAGESP. Заголовок этой функции должен выглядеть следующим образом:

```
INTERVAL_VECTOR Имя функции(const INTERVAL_VECTOR&);
```

Как только эта функция правильно определена, то для образа регулярного покрытия X его аппроксимация Y сверху может быть получена следующим образом:

```
SUBPAVING Y = ImageSp(Name_of_function, X, eps);
```

где eps — заданный параметр точности.

Упражнение 11.34. *Реализуйте интервальную векторную функцию* f, *соответствующую функции* $\mathbf{f} : \mathbb{R}^2 \to \mathbb{R}^2$, *определенную соотношениями*

$$f_1(x_1, x_2) = x_1^2 + x_2 - x_2^2,$$
$$f_2(x_1, x_2) = x_1^2 + x_2^2.$$

Реализация может содержать естественную функцию включения или любую ее центрированную форму. ■

Модифицированный заголовочный файл содержит теперь дружественные функции для трех базовых шагов алгоритма IMAGESP, а именно для измельчения, оценивания и регуляризации покрытия. Прежде чем перейти к рассмотрению реализации этих шагов, мы поясним, как может быть реализован список параллелотопов, используемый в IMAGESP.

Управление списком. Мы разработали модуль imlist, основанный на модуле veclist библиотеки PROFIL/BIAS,для создания нового класса IMAGELIST, чтобы запоминать и обновлять списки параллелотопов, используемых в IMAGESP. Любой экземпляр в классе IMAGELIST содержит массивы типа IMAGELIST_ELEMENT. Свойствами каждого из элементов этого массива являются: параллелотоп Box, инетервальный вектор INTERVAL_VECTOR, а также размер Volume и число с двойной точностью, соответствующее размеру параллелотопа Box. Элементы IMAGELIST_ELEMENT отсортированы по убыванию размера их параллелотопов Box.

Чтобы использовать класс IMAGELIST в алгоритме IMAGESP, необходимо только одно свойство и несколько функций. Вот почему мы ограничимся здесь только их перечислением и отошлем читателя за подробностями информации о модуле veclist к описанию библиотеки PROFIL/BIAS.

Конструктор инициализации списка элементов имеет вид:

```
IMAGELIST_ELEMENT(INTERVAL_VECTOR&);
```

Свойство

```
IMAGELIST_ELEMENT* current;
```

указывает на элемент списка, который в настоящее время является активным.

Параллелотоп может быть вставлен в список с помощью функции

```
IMAGELIST& operator+=(INTERVAL_VECTOR&);
```

Доступ к первому элементу этого списка осуществляется с помощью функции

```
IMAGELIST_ELEMENT& First(IMAGELIST&);
```

Эта функция обновляет текущее свойство current, делая активным первый элемент. Функция

```
IMAGELIST_ELEMENT& Next(IMAGELIST&);
```

дает доступ к элементу, хранимому за текущим активным элементом списка, и делает его активным, изменяя свойство current. Функция

```
INT IsEmpty(IMAGELIST&);
```

проверяет пустоту списка. Функция

```
INT Finished(IMAGELIST&);
```

проверяет, достигнут ли последний элемент списка.

ЗАМЕЧАНИЕ 11.10. Модуль list, предоставляемый стандартной библиотекой на шаблонах, не может быть использован вследствие его несовместимости с классом INTERVAL_VECTOR. ■

Далее будут выполняться измельчение, оценивание и регуляризация покрытия.

Измельчение. Первый шаг алгоритма измельчения будет выполняться функцией Mince, которая преобразует минимальное покрытие в неминимальное, содержащее параллелотопы с шириной, меньшей заданного параметра точности eps. Функция Mince может быть реализована следующим образом:

```
//-------------------------------------------------------------
// Файл:      subpaving.cpp (продолжение)
//...
void Mince(SUBPAVING A, double eps)
{
  if (IsEmpty(A)) return;
  if (IsLeaf(A))
  { int comp;
    if (MaxDiam(Box(A), comp) < eps) return;
    else Expand(A, comp);
  }

  Mince(A->leftChild, eps);
  Mince(A->rightChild, eps);
}
//...
//-------------------------------------------------------------
```

Если покрытие, которое надо дробить, пусто, то ничего не делается. Если оно является листом, и соответствующий параллелотоп больше, чем eps, то этот лист расширяется. Если покрытие не является листом, то функция Mince рекурсивно применяется к его левому и правому отросткам.

Оценивание. Теперь должны быть оценены образы всех параллелотопов, созданных функцией Mince. Функция Evaluate рассчитывает список этих образов и интервальную оболочку (hull) объединения всех параллелотопов, занесенных в окончательный список образов. Реализация этих операций может иметь вид:

```
//-------------------------------------------------------------
// Файл:      subpaving.cpp (продолжение)
//...
void Evaluate(PIVF f, SUBPAVING A, IMAGELIST& list,
```



```
                                    INTERVAL_VECTOR& hull)
{
  if (IsEmpty(A)) return;
  if (IsLeaf(A))
  {
    INTERVAL_VECTOR image(f(Box(A)));
  // вычисление интервальной оболочки
    if (IsEmpty(list)) hull = image;
    else hull = Hull(hull, image);
  // образы занесены в список
    list += IMAGELIST_ELEMENT(image);
    return;
  }

  Evaluate(f, A->leftChild, list, hull);
  Evaluate(f, A->rightChild, list, hull);
}
//...
//-------------------------------------------------------------
```

Регуляризация. Все параллелотопы образа должны теперь быть объединены в единое покрытие. Это можно выполнить следующими действиями:

```
//-------------------------------------------------------------
// File:      subpaving.cpp (продолжение)
//...
SUBPAVING Regularize (INTERVAL_VECTOR& hull,
                              IMAGELIST& list, double eps)
{
  if (IsEmpty(list)) return NULL;
  if (hull == First(list).Box) return new NODE(hull);
  int maxdcomp;
  if (MaxDiam(hull,maxdcomp) < eps)
    return new NODE(hull);

  INTERVAL_VECTOR lefthull = Lower(hull, maxdcomp);
  INTERVAL_VECTOR righthull = Upper(hull, maxdcomp);
  IMAGELIST leftlist,rightlist;
  INTERVAL_VECTOR inter;
  while (!Finished(list))
  {
    if (Intersection(inter,Current(list).Box,lefthull))
       leftlist += IMAGELIST_ELEMENT(inter);
    if (Intersection(inter,Current(list).Box,righthull))
       rightlist += IMAGELIST_ELEMENT(inter);
```

```
    Next(list);
  }

  return Reunite(Regularize(lefthull, leftlist, eps),
          Regularize(righthull, rightlist, eps), hull);
}
//-------------------------------------------------------------
```

Параллелотоп `hull` есть интервальная оболочка объединения всех параллелотопов, занесенных в список `list`. Эти параллелотопы могут быть соединены в покрытие, корень которого будет соответствовать параллелотопу `hull`. Если список `list` пуст, то выдается пустое покрытие. Если наибольший параллелотоп списка, т. е. параллелотоп, хранимый в качестве первого элемента этого списка, равен интервальной оболочке `hull`, то покрытие сводится к этому параллелотопу. Если ширина параллелотопа `hull` меньше допуска точности `eps`, то список `list` содержит только параллелотопы с шириной, меньшей допуска точности `eps`, следовательно, регуляризованное покрытие сводится к оболочке `hull`. Это позволяет создать покрытие после конечного числа итераций. При этом покрытие содержит только параллелотопы шириной, меньшей допуска точности `eps`.

Если ни одна из этих проверок не удовлетворяется, то оболочка `hull` разбивается на левую `lefthull` и правую `righthull` части, и вырабатываются соответствующие левый (`leftlist`) и правый (`rightlist`) списки, которые содержат все элементы исходного списка `list`, связанные, соответственно, с левой `lefthull` и правой `righthull` оболочками. Далее функция `Regularize` рекурсивно применяется к этим двум параллелотопам и соответствующим спискам.

Алгоритм IMAGESP может быть записан следующим образом:

```
//-------------------------------------------------------------
// Файл:      subpaving.cpp (продолжение)
//...
SUBPAVING ImageSp(PIVF f, SUBPAVING A, double eps)
{
  IMAGELIST images;
  INTERVAL_VECTOR hull;
  Mince(A,eps);
  Evaluate(A, f, images, hull);
  return (Regularize(hull, images, eps));
}
//-------------------------------------------------------------
```

Упражнение 11.35. *С помощью* `ImageSp` *оцените образ параллелотопа* $[-2,2]^2$, *даваемый функцией, заданной в* Упражнении 11.34. ■

11.12.4. Моделирование системы и оценивание ее состояния с помощью покрытий

Четыре следующих упражнения пояснят читателю вопросы моделирования системы и оценивания ее параметров и состояния.

Упражнение 11.36. *Рассмотрим систему дискретного времени*

$$\begin{pmatrix} x_1(k+1) \\ x_2(k+1) \end{pmatrix} = \begin{pmatrix} \rho\cos(\pi/4) & -\rho\sin(\pi/4) \\ \rho\sin(\pi/4) & \rho\cos(\pi/4) \end{pmatrix} \begin{pmatrix} x_1(k) \\ x_2(k) \end{pmatrix}, \tag{11.13}$$

где $\rho = 0{,}85$. *На шаге* $k = 0$ *известно только, что вектор* $(x_1(0), x_2(0))^{\mathrm{T}}$ *принадлежит параллелотопу* $\mathbb{X}_0 = [4,5] \times [-1,1]$. *Задача состоит в том, чтобы оценить множество* $\mathbb{X}_k$, *содержащее все возможные значения вектора* $(x_1(k), x_2(k))^{\mathrm{T}}$ *в любой заданный момент времени при* $k > 0$. *С помощью* IMAGESP, *составьте программу оценивания покрытия* $\overline{\mathbb{X}}_k$, *гарантированно содержащего множество* $\mathbb{X}_k$ *для* $k = 1, \ldots, 10$. *Точность описания будет регулироваться величиной параметра точности* `eps`, *задаваемой пользователем. Оцените влияние параметра* `eps` *на время вычислений и точность описания искомого покрытия и сравните размеры вычисляемых покрытий. Все покрытия должны записываться в соответствующий файл (например,* `output.spv`). *Двумерные проекции этих покрытий могут визуализироваться с помощью программы*

DrawSpaving.exe,

доступной на сайте:

http://www.lss.supelec.fr/books/intervals ■

Упражнение 11.37. *Модифицируйте программу, написанную для* Упражнения 11.36, *чтобы ответить на такие же вопросы, предполагая теперь, что параметр* ρ *в передаточной матрице системы* (11.13) *содержит неопределенность и принадлежит интервалу* $[\rho] = [0{,}8,\ 0{,}9]$, *так что система* (11.13) *принимает вид*:

$$\begin{pmatrix} x_1(k+1) \\ x_2(k+1) \end{pmatrix} = \begin{pmatrix} [\rho]\cos(\pi/4) & -[\rho]\sin(\pi/4) \\ [\rho]\sin(\pi/4) & [\rho]\cos(\pi/4) \end{pmatrix} \begin{pmatrix} x_1(k) \\ x_2(k) \end{pmatrix}. \tag{11.14}$$

■

Упражнение 11.38. *Предположим теперь, что последовательность замеров* $y(k)$, $k = 0, \ldots, 10$, *дает информацию о состоянии системы* (11.14), *в соответствии с уравнением наблюдения*

$$y(k) = x_1(k) + \xi_k, \tag{11.15}$$

где ξ_k — ограниченная по величине помеха, принадлежащая интервалу $[-0{,}1, 0{,}1]$. Целью данного упражнения является реализация алгоритма оценки состояния для этой системы, выполняющего шаги прогнозирования и коррекции, как пояснялось в параграфе 6.4 (с. 216).

1. *Реализуйте шаг прогнозирования, используя функцию* `ImageSp`.
2. *Реализуйте шаг коррекции, используя* `Sivia`.
3. *Создайте данные наблюдения $y(k)$, $k = 0, \ldots, 10$, моделированием движения системы (11.14), заменяя на каждом такте k времени символ $[\rho]$ на значение параметра ρ_k, генерируя его случайным образом из интервала $[0{,}8, 0{,}9]$ и моделируя значение помехи ξ_k случайным образом из интервала $[-0{,}1, 0{,}1]$ при истинном начальном состоянии системы $x_1(0) = 4{,}5$ и $x_2(0) = 0$.*
4. *Постройте покрытие $\overline{\mathbb{X}}_k$, содержащее множество $\mathbb{X}_k$ для тактов времени $k = 0, \ldots, 10$, предполагая, что начальное состояние известно неточно и принадлежит множеству $\mathbb{X}_0 = [4, 5] \times [-1, 1]$, а данные наблюдения те же, что и в предыдущем вопросе.*

■

Упражнение 11.39. *Рассмотрим ту же задачу, что и в* Упражнении 11.38, *но предположим, что неизвестный параметр ρ является константой и может быть добавлен в вектор состояния. Целью данного упражнения является построение алгоритма оценивания этого расширенного вектора состояний для выполнения совместного оценивания как состояния, так и параметра.*

1. *Напишите дискретное уравнение состояния, соответствующее расширенному вектору состояния.*
2. *Создайте данные наблюдения $y(k)$, $k = 0, \ldots, 10$, моделированием движения системы (11.14), заменяя на каждом такте k времени символ $[\rho]$ на его значение $\rho^* = 0{,}85$, и моделируя значение помехи ξ_k случайным образом из интервала $[-0{,}1, 0{,}1]$ при истинном начальном состоянии системы $x_1(0) = 4{,}5$ и $x_2(0) = 0$.*
3. *Примените тот же метод, что и в* Упражнении 11.38, *для оценивания расширенного вектора состояния по сформированным данным наблюдения.*
4. *Модифицируйте программу для моделирования данных наблюдения и оценивания расширенного вектора состояния, разрешая неизвестному параметру ρ изменяться по уравнению*

$$\rho_k = \rho_{k-1} + \delta_k,$$

где δ_k принадлежит интервалу $[-0{,}01, 0{,}01]$, а ρ_0 принадлежит интервалу $[0{,}8, 0{,}9]$.

■

11.13. Обработка ошибок

При наличии ошибок в работе программы можно рассматривать различные правила поведения: от игнорирования этих ошибок до выбрасывания программы, в которой ошибка появилась. Наилучшее решение обычно лежит между этими двумя крайностями, как иллюстрируется следующими примерами.

Пример 11.1. *Рассмотрим интервальную ньютоновскую процедуру поиска всех нулей уравнения $f(x) = \mathrm{tg}(x) - x$ на интервале $[-10\pi, 10\pi]$. Функция* tg *определена не на всем интервале поиска. Но это не причина отказываться от запуска алгоритма. В таком случае можно было бы взять интервал значений функции в виде $[f]([-10\pi, 10\pi]) = [-\infty, +\infty]$, чтобы указать, что интервал $[-10\pi, 10\pi]$ может содержать нули функции $f(x)$.* ■

Пример 11.2. *Предположим теперь, что функция f из предыдущего примера является некоторым промежуточным результатом в длинных и дорогих вычислениях и что получение интервала $[-\infty, +\infty]$ в этих вычислениях приведет к бесполезным результатам. Тогда лучше выбросить эту программу как можно скорее с целью экономии времени и денег и упростить обнаружение такой ситуации.* ■

Следовательно, меры, которые надо предпринять, когда данная ошибка обнаружена, зависят от существа задачи. Только те ошибки, которые вызывают процедуры операционной системы, могут быть обработаны внутри библиотеки. Другие ошибки могут быть только обнаружены, и ответственность за предпринимаемые меры лежит на пользователе. Поэтому мы ограничимся рассмотрением простых вариантов.

11.13.1. Использование оператора Exit

Это самый простой способ обработки ошибок, но наименее гибкий. Функция `exit()` принадлежит модулю `stdlib`. Она была уже использована в параграфе 11.6.2 (с. 394). В следующей программе функция `exit()` используется всякий раз, когда неудачей заканчивается операция сложения двух векторов разной размерности:

```
//-------------------------------------------------------------
// Файл:      ivalvec.cpp (продолжение)
//...
INTERVAL_VECTOR operator+(const INTERVAL_VECTOR& a,
                          const INTERVAL_VECTOR& b)
{
  if (a.nElements != b.nElements)
  // программа прервана
  { cout << "Попытка сложения векторов с разной размерностью"
         << endl;
    exit(EXIT_FAILURE); }
  // ошибки не обнаружено, сложение выполнено
  INTERVAL_VECTOR res(a.nElements);
  for (int i = 1; i <= a.nElements; i++)
    res(i) = a(i) + b(i);
  return (res);
}
//...
//-------------------------------------------------------------
```

В стандартный выходной поток выдается сообщение (программа может быть модифицирована, чтобы выдавать сообщения, например, журнальный `.log`-файл), и программа прерывается. Параметр `EXIT_FAILURE` сигнализирует операционной системе, что выполнение программы завершено с ошибкой. Многие библиотеки языка **C++** обрабатывают ошибки таким образом. Иногда флаги могут быть установлены при компиляции, чтобы настроить поведение программы при ошибках. Например, если флаг `__BIASRAISEDIVIDEBYZERO__` установлен, то программы библиотеки PROFIL/BIAS прерывают вычисление, когда выполняется деление на интервал, содержащий 0. Если же этот флаг не поставлен, то в качестве результата выдается вся числовая ось.

11.13.2. Обработка исключений

В языке **C++** *исключение* приводит к выходу из функции, при выполнении которой появляется ошибка, но при этом разрешается продолжение работы программы, в отличие от функции `exit`. Рассмотрим снова защиту от сложения векторов с разной размерностью. Предыдущая программа может быть заменена на следующую, где *выбрасывается* исключение, когда эта ошибка появляется.

```
//-------------------------------------------------------------
// Файл:      ivalvec.cpp (продолжение)
//...
```



```
INTERVAL_VECTOR operator+(INTERVAL_VECTOR& a,
                          INTERVAL_VECTOR& b)
{
  if (a.nElements != b.nElements)
  // исключение выбрасывается
    throw "Попытка сложения векторов с разной размерностью";
  // исключение не возникло, поэтому сложение
  // может быть выполнено
  INTERVAL_VECTOR res(a.nElements);
  for (int i = 1; i <= a.nElements; i++)
    res(i) = a(i) + b(i);
  return (res);
}
//...
//-------------------------------------------------------------
```

Аргументом функции `throw` может быть строка, целое число или даже класс, который может содержать значительно больше информации, чем строка. Как только исключение создается, выполнение функции прекращается. Вызывающая функция должна *уловить* это исключение использованием блоков `try` и `catch`, как показано в следующем фрагменте:

```
//-------------------------------------------------------------
// Файл:      exceptdemo.cpp
// Назначение:  демонстрация того, как ловится исключение
#include "ivalvec.h"   // чтобы использовать INTERVAL_VECTOR
#include <iostream.h>              // стандартный ввод-вывод

void main()
{
  INTERVAL_VECTOR x(3),y(2),z;

  x(1) = INTERVAL(2,3); x(2) = INTERVAL(-3,4);
  x(3) = INTERVAL(-6,-5);
  y(1) = INTERVAL(2,3); y(2) = INTERVAL(-3,4);

  try {       // если в этом блоке возникло исключение,
    z = x + y;
  }                                  // то оно будет обработано
  catch (char *msg) {                          // msg содержит
                                   // строку, описанную в throw
    cout << msg << endl;
    return;
  }
```

```
  // другие функции
//...
//-------------------------------------------------------------
```

Функции, которые могут создавать исключения, помещены в блок `try`. Если возникло исключение, то выполняется блок `catch`. В этом простом иллюстрационном примере вычисление заканчивается точно так же, как бы оно было выполнено при использовании функции `exit()`, но ничто не запрещает включить в блок `catch()` какой-нибудь прием для более утонченной обработки.

11.13.3. Обработка ошибок математических вычислений

Возможно, этот тип ошибки является самым распространенным типом ошибок в рассматриваемом контексте. Как должна процедура реагировать, например, когда ее просят рассчитать логарифм интервала с отрицательной нижней границей? Большинство программных библиотек по интервальным вычислениям используют одну из двух представленных опций диагностики, а именно, выход из программы или игнорирование обнаруженной особенности; эти опции весьма далеки от того, чтобы всегда быть удовлетворительными.

Представление интервалов, предложенное в параграфе 10.3, делает возможным уменьшить число случаев, когда численная ошибка требует специальной обработки, причем это достигается введением бесконечных и пустых интервалов, и использованием теоретико-множественного понятия точки при интервальных вычислениях.

Литература

[Ackermann (1992)] Ackermann J. Does it suffice to check a subset of multilinear parameters in robustness analysis? // *IEEE Transactions on Automatic Control*. 1992. **37**(4). 487–488.

[Ackermann *et al.* (1990)] Ackermann J., Hu H. and Kaesbauer D. Robustness analysis: a case study, *IEEE Transactions on Automatic Control*. 1990. **35**(3). P. 352–356.

[Adrot (2000)] Adrot O. *Diagnostic à base de modèles incertains utilisant l'analyse par intervalles: l'approche bornante*, PhD dissertation, Institut National Polytechnique de Lorraine, Nancy, France, 2000.

[Amato *et al.* (1995)] Amato F., Garofalo F., Glielmo L. and Pironti A. Robust and quadratic stability via polytopic set covering, *International Journal of Robust and Nonlinear Control*. 1995. **5**(8). P. 745–756.

[Anderson *et al.* (1987)] Anderson B. D. O., Jury E. I. and Mansour M. On robust Hurwitz polynomials // *IEEE Transactions on Automatic Control*. 1987. **32**(10). P. 909–913.

[Anderson and Anderson (1998)] Anderson P. and Anderson G. *Navigating C++ and Object-Oriented Design*. Prentice Hall, Upper Saddle River, NJ, 1998.

[Arsouze *et al.* (2000)] Arsouze J., Ferrand G. and Lallouet A. (2000) Chaotic iterations for constraint propagation and labeling, *Research Report RR-LIFO-2000-04*, LIFO, Laboratoire d'Informatique Fondamentale d'Orléans, Université d'Orléans, BP 6759, F-45067 Orléans Cedex 2, France. Available at: ftp://ftp-lifo.univ-orleans.fr/pub/RR/RR2000/RR2000-04.ps.

[Balakrishnan *et al.* (1991a)] Balakrishnan V., Boyd S. and Balemi S. Branch and bound algorithm for computing the minimum stability degree of parameter-dependent linear systems // *International Journal of Robust and Nonlinear Control*. 1991a. **1**(4). P. 295–317.

[Balakrishnan *et al.* (1991b)] Balakrishnan V., Boyd S. and Balemi S. Computing the minimum stability degree of parameter-dependent linear systems // *Control of Uncertain Dynamic Systems* / Eds. S. P. Bhattacharyya and L. H. Keel, Boca Raton, FL: CRC Press, 1991b. P. 359–378.

[Barmish (1984)] Barmish B. R. Invariance of the strict Hurwitz property for polynomials with perturbed coefficients // *IEEE Transactions on Automatic Control*. 1984. **29**(10). P. 935–936.

[Barmish (1988)] Barmish B. R. New tools for robustness analysis, *Proceedings of the 27th IEEE Conference on Decision and Control*. 1988. Austin, TX. P. 399–408.

[Barmish (1994)] Barmish B. R. *New Tools for Robustness of Linear Systems*. MacMillan, N.Y.: 1994.

[Barmish and Tempo (1995)] Barmish B. R. and Tempo R. On mappable nonlinearities in robustness analysis // *Proceedings of the 3rd European Control Conference*. Rome, Italy, 1995. P. 1430–1435.

[Bartlett *et al.* (1988)] Bartlett A. C., Hollot C. V. and Huang L. Root locations of an entire polytope of polynomials: It suffices to check the edges // *Mathematics of Control, Signals and Systems*. 1988. **1**(1). P. 61–71.

[Belforte *et al.* (1990)] Belforte G., Bona B. and Cerone V. Parameter estimation algorithms for a set-membership description of uncertainty // *Automatica*. 1990. **26**(5). P. 887–898.

[Benhamou and Goualard (2000)] Benhamou F. and Goualard F. Universally quantified interval constraints // (ed.), *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming / Ed. R. Dechter CP, 2000*, Vol. 1894 of *Lecture Notes in Computer Science*. Springer-Verlag, Heidelberg, Germany.

[Benhamou *et al.* (1999)] Benhamou F., Goualard F., Granvilliers L. and Puget J. F. Revising hull and box consistency // *Proceedings of the International Conference on Logic Programming*. Las Cruces, NM, 1999. P. 230–244.

[Benhamou and Granvilliers (1997)] Benhamou F. and Granvilliers L. Automatic generation of numerical redundancies for nonlinear constraint solving/!/ *Reliable Computing*. 1997. **3**(3). P. 335–344.

[Benhamou *et al.* (1994)] Benhamou F., McAllester D. and van Hentenryck P. CLP (intervals) revisited // *Proceedings of the International Logic Programming Symposium*/ Ed. M. Bruynooghe. Ithaca, NY, 1994. P. 124–138.

[Berger (1985)] Berger J. *Statistical Decision Theory and Bayesian Analysis*, 2nd edition. Springer-Verlag, N.Y., 1985.

[Berger (1979)] Berger M. *Espaces euclidiens, triangles, cercles et sphères*. Vol. 2 of *Géométrie*. Cedic/Fernand Nathan, Paris, France, 1979.

[Bertsekas and Rhodes (1971)] Bertsekas D. P. and Rhodes I. B. Recursive state estimation for a set-membership description of uncertainty // *IEEE Transactions on Automatic Control*. 1971. **16**(2). P. 117–128.

[Berz and Makino (1998)] Berz M. and Makino K. Verified integration of ODEs and flows using differential algebraic methods on high-order Taylor models // *Reliable Computing*. 1998. **4**(4). P. 361–369.

[Bialas (1983)] Bialas S. A necessary and sufficient condition for the stability of interval matrices // *International Journal of Control*. 1983. **37**(4). P. 717–722.

[Bialas (1985)] Bialas S. A necessary and sufficient condition for the stability of convex combinations of stable polynomials or matrices // *Bulletin of the Polish Academy of Sciences*. 1985. **33**. P. 473–480.

[Bialas and Garloff (1985)] Bialas S. and Garloff J. Convex combinations of stable polynomials // *Journal of the Franklin Institute*. 1985. **319**(3). P. 373–377.

[Bischof *et al.* (1992)] Bischof C., Carle A., Corliss G. F., Griewank A. and Hovland P. ADIFOR — Generating derivative codes from Fortran programs // *Scientific Programming*. 1992. **1**(1). P. 11–29.

[Bischof (1991)] Bischof C. H. Issues in parallel automatic differentiation // *Automatic Differentiation of Algorithms: Theory, Implementation, and Application* / Eds. A. Griewank and G. F. Corliss, SIAM, Philadelphia, PA, 1992. P. 100–113.

[Blondel and Tsitsiklis (1995)] Blondel V. and Tsitsiklis J. NP-hardness of some linear control design problems // *Proceedings of the 34th IEEE Conference on Decision and Control*. New Orleans, LA, 1995. P. 2910–2915.

[Borenstein *et al.* (1996)] Borenstein J., Everett H. and Feng L. *Navigating Mobile Robots*, Wellesley, MA: A. K. Peters Ltd., 1996.

[Boyd *et al.* (1994)] Boyd S., Ghaoui L. E., Feron E. and Balakrishnan V. *Linear Matrix Inequalities in System and Control Theory*, Vol. 15 of *Studies in Applied Mathematics*, SIAM, Philadelphia, PA, 1994.

[Braatz *et al.* (1994)] Braatz R., Young P., Doyle J. and Morari M. Computational complexity of μ calculation // *IEEE Transactions on Automatic Control*. 1994. **39**(5). P. 1000–1002.

[Braems *et al.* (2001)] Braems I., Berthier F., Jaulin L., Kieffer M. and Walter E. Guaranteed estimation of electrochemical parameters by set inversion using interval analysis // *Journal of Electroanalytical Chemistry*. 2001. **495**(1). P. 1–9.

[Brayton *et al.* (1979)] Brayton R. K., Director S. W., Hachtel G. D. and Vidigal L. M. A new algorithm for statistical circuit design based on quasi-Newton methods and function splitting // *IEEE Transactions on Circuits and Systems*. 1979. **26**(9). P. 784–794.

[Brooks and Lozano-Pérez (1985)] Brooks R. A. and Lozano-Pérez T. A subdivision algorithm in configuration space for findpath with rotation // *IEEE Transactions on Systems Man and Cybernetics*. 1985. **15**(2). P. 224–233.

[Capper (1994)] Capper D. M. *C++ for Scientists, Engineers and Mathematicians*. Springer-Verlag, London, UK, 1994.

[Castellanos *et al.* (1999)] Castellanos J. A., Monteil J., Neira J. and Tardos J. The SMAP: A probabilistic framework for simultaneous localization and map building // *IEEE Transactions on Robotics and Automation*. 1999. **15**(5). P. 948–952.

[Caviness and Johnson (1998)] *Quantifier Elimination and Cylindrical Algebraic Decomposition* / Eds. B. F. Caviness and J. R. Johnson. Springer-Verlag, Vienna, Austria, 1998.

[Cerone (1991)] Cerone V. Parameter bounds for models with bounded errors in all variables. *Preprints of the 9th IFAC/IFORS Symposium on Identification and System Parameter Estimation*. Budapest, Hungary, 1991. P. 1518–1523.

[Cerone (1996)] Cerone V. Errors-in-variables models in parameter bounding // *Bounding Approaches to System Identification*. Eds. M. Milanese, J. Norton, H. Piet-Lahanier and E. Walter. Plenum, New York, N.Y., 1996. P. 289–306.

[Chernousko (1994)] Chernousko F. L. *State Estimation for Dynamic Systems*. Boca Raton, FL: CRC Press, 1994. (русск.) [Черноусько (1988)] Черноусько Ф. Л. *Оценивание фазового состояния динамических систем*. — М.: Наука. 1988.

[Chiriaev and Walster (1998)] Chiriaev D. and Walster G. W. (1998) Interval arithmetic specification. Available at: http://www.mscs.edu/globsol/walster-papers.html.

[Cleary (1987)] Cleary J. C. Logical arithmetic // *Future Computing Systems*. 1987. **2**(2). P. 125–149.

[Clément and Gentil (1990)] Clément T. and Gentil S. Recursive membership set estimation for output-error models // *Mathematics and Computers in Simulation*. 1990. **32**(5–6). P. 505–513.

[Clowes (1971)] Clowes M. B. On seeing things // *Artificial Intelligence*. 1971. **2**. P. 179–185.

[Collin *et al.* (1991)] Collin Z., Dechter R. and Katz S. On the feasibility of distributed constraint satisfaction // *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, Sydney, Australia, 1991. P. 318–324.

[Combettes (1993)] Combettes P. L. The foundations of set theoretic estimation // *Proceedings of the IEEE*. 1993. **81**(2). P. 182–208.

[Connell and Corless (1993)] Connell A. and Corless R. An experimental interval arithmetic package in Maple // *Interval Computation*. 1993. **2**. P. 120–134.

[Corliss (1988)] Corliss G. F. Applications of differentiation arithmetic, *Reliability in Computing*, / Ed. R. E. Moore. London, UK: Academic Press, 1988. P. 127–148.

[Corliss (1991)] Corliss G. F. (1991) Proposal for a basic interval arithmetic subroutines library (BIAS), Technical Report, Department of Mathematics, Statistics, and Computer Science, Marquette University, Milwaukee, WI.

[Corliss (1992)] Corliss G. F. (1992) Automatic differentiation bibliography, Technical Memorandum ANL/MCS–TM–167, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL.

[Crowley (1989)] Crowley J. World modeling and position estimation for a mobile robot using ultrasonic ranging, *Proceedings of the IEEE International Conference on Robotics and Automation*, Scottsdale, AZ, 1989. P. 674–680.

[Crowley *et al.* (1998)] Crowley J. L., Wallner F. and Schiele B. Position estimation using principal components of range data // *Proceedings of the IEEE International Conference on Robotics and Automation*, Leuven, Belgium, 1998. P. 3121–3128.

[Daumas *et al.* (1995)] Daumas M., Mazenc C., Merrheim X. and Muller J. M. Modular range reduction: A new algorithm for fast and accurate computation of the elementary functions // *Journal of Universal Computer Science*. 1995. **1**(3). P. 162–175.

[Davis (1987)] Davis E. Constraint propagation with interval labels // *Artificial Intelligence*. 1995. **32**(3). P. 281–331.

[Dechter and Dechter (1987)] Dechter A. and Dechter R. Removing redundancies in constraint networks // *Proceedings of the 6th AAAI National Conference on Artificial Intelligence*, Vol. 1, Seattle, WA, 1987. P. 105–109.

[Dechter and Pearl (1989)] Dechter R. and Pearl J. Tree-clustering for constraint networks // *Artificial Intelligence*. 1989. **38**(3). P. 353–366.

[Deller *et al.* (1993)] Deller J., Nayeri M. and Odeh S. Least-square identification with error bounds for real-time signal processing and control // *Proceedings of the IEEE*. 1993. **81**(6). P. 815–849.

[Deo (1974)] Deo N. *Graph Theory with Applications to Engineering and Computer Science*. Prentice-Hall, Englewood Cliffs, N.J., 1974.

[Deville *et al.* (1997)] Deville Y., Barette O. and van Hentenryck P. Constraint satisfaction over connected row convex constraints // *Proceedings of the International Joint Conference on Artificial Intelligence*. Vol. 1. Nagoya, Japan, 1997. P. 403–410.

[Didrit (1997)] Didrit O. *Analyse par intervalles pour l'automatique; résolution globale et garantie de problèmes non linéaires en robotique et commande robuste*. PhD dissertation, Université Paris-Sud, Orsay, France, 1997.

[Didrit *et al.* (1997)] Didrit O., Jaulin L. and Walter E. Guaranteed analysis and optimization of parametric systems, with application to their stability degree, *European Journal of Control*. 1997. **3**(1). P. 68–80.

[Didrit *et al.* (1998)] Didrit O., Petitot M. and Walter E. Guaranteed solution of direct kinematic problems for general configurations of parallel manipulators // *IEEE Transactions on Robotics and Automation*. 1998. **14**(2). P. 259–266.

[Dijkstra (1959)] Dijkstra E. A note on two problems in connection with graphs // *Numerische Mathematik*. 1959. **1**. P. 269–271.

[Dorato (2000)] Dorato P. Quantified multivariate polynomial inequalities, *IEEE Control Systems Magazine*. 2000. **20**(5). P. 48–58.

[Dorato *et al.* (1993)] Dorato P., Tempo R. and Muscato G. Bibliography on robust control, *Automatica*. 1993. **29**(1). P. 201–213.

[Dorato *et al.* (1997)] Dorato P., Yang W. and Abdallah C. Robust multi-objective feedback design by quantifier elimination // *Journal of Symbolic Computation*. 1997. **24**. P. 153–159.

[Doyle (1982)] Doyle J. Analysis of feedback systems with structured uncertainties // *IEE Proceedings*. 1982. **129D**(6). P. 242–250.

[Drumheller (1987)] Drumheller M. Mobile robot localization using sonar, *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1987. **9**(2). P. 325–332.

[Du (1995)] Du D.-Z. Minmax and its application // *Handbook of Global Optimization*/ Eds. R. Horst and P. M. Pardalos. Kluwer, Dordrecht, the Netherlands, 1995. P. 339–367.

[Durieu *et al.* (1996)] Durieu C., Polyak B. and Walter E. Trace versus determinant in ellipsoidal outer bounding with application to state estimation, *Proceedings of the 13th IFAC World Congress*, Vol. I. San Francisco, CA, 1996. P. 43–48.

[El Kahoui and Weber (2000)] El Kahoui M. and Weber A. Deciding Hopf bifurcations by quantifier elimination in a software-component architecture // *Journal of Symbolic Computation*. 2000. **30**(2). P. 161–179.

[Evtushenko (1991)] Evtushenko Y. G. Automatic differentiation viewed from optimal control, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, Eds. A. Griewank and G. F. Corliss, SIAM, Philadelphia, PA, 1991. P. 25–30.

[Falk (1973)] Falk J. E. Global solutions for signomial programs, Technical Report T-274, Washington, DC: George Washington University, 1973.

[Faugère and Lazard (1995)] Faugère J.-C. and Lazard D. The combinatorial classes of parallel manipulators // *Mechanism and Machine Theory*. 1995. **30**(6). P. 765–776.

[Ferreres and Magni (1996)] Ferreres G. and Magni, J.-F. Robustness analysis using the mapping theorem without frequency gridding // *Proceedings of the 13th IFAC Triennial World Congress*, San Francisco, 1996. P. 7–12.

[Fogel and Huang (1982)] Fogel E. and Huang Y. F. On the value of information in system identification bounded noise case, *Automatica*. 1982. **18**(2). P. 229–238.

[Francis and Khargonekar (1995)] *Robust Control Theory*, / Eds. B. A. Francis and P. P. Khargonekar. Vol. 66 of *IMA Volumes in Mathematics and Its Applications*, Springer-Verlag, N.Y., 1995.

[Frazer and Duncan (1929)] Frazer R. A. and Duncan W. J. On the criteria for the stability of small motion// *Proceedings of the Royal Society of London*. 1929. **124**. P. 642–654.

[Freuder (1978)] Freuder E. C. Synthesizing constraint expressions // *Communications of the ACM*. 1978. **21**(11). P. 958–966.

[Gardenes *et al.* (1985)] Gardenes E., Mielgo H. and Trepat A. Modal intervals: Reasons and ground semantics / Eds. K. Nickel, *Interval Mathematics 1985*, Vol. 212 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 1985. P. 27–35.

[Garloff (2000)] Garloff J. Application of Bernstein expansion to the solution of control problems, *Reliable Computing*. 2000. **6**(3). P. 303–320.

[Garloff and Walter (2000)] Special Issue on Applications to Control / Eds. J. Garloff and E. Walter. Signals and Systems. *Reliable Computing*. 2000. **6**(3). P. 229-362.

[Gelb (1974)] Gelb A. *Applied Optimal Estimation*, Cambridge, MA: MIT Press, 1974.

[Goldberg (1991)] Goldberg D. What every computer scientist should know about floating-point arithmetic, *ACM Computing Surveys*. 1991. **23**(1). P. 5–47.

[Gough (1956)] Gough V. E. Contribution to discussion of papers on research in automobile stability, control and tyre performance, by Cornell staff, *Proceedings of the Automotive Division of the Institution of Mechanical Engineers*, 1956. P. 392–395.

[Granvilliers (1998)] Granvilliers L. *Consistances locales et transformations symboliques de contraintes d'intervalles*, PhD dissertation, Université d'Orléans, France, 1998.

[Grimson and Lozano-Pérez (1987)] Grimson W. E. and Lozano-Pérez, T. Localizing overlapping parts by searching the interpretation tree, *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1987. **9**(4). P. 469–482.

[Gyssens *et al.* (1994)] Gyssens M., Jeavons P. G. and Cohen D. A. Decomposing constraint satisfaction problems using database techniques, *Artificial Intelligence*. 1994. **66**(1). P. 57–89.

[Halbwachs and Meizel (1997)] Halbwachs E. and Meizel D. Multiple hypothesis management for mobile vehicule localization. *CD-ROM of the European Control Conference*, Louvain, Belgium, 1997.

[Hammer *et al.* (1995)] Hammer R., Hocks M., Kulish U. and Ratz D. *C++ Toolbox for Verified Computing*, Springer-Verlag, Berlin, Germany, 1995.

[Hanebeck and Schmidt (1996)] Hanebeck U. and Schmidt G. Set theoretic localization of fast mobile robots using an angle measurement technique // *Proceedings of the IEEE International Conference on Robotics and Automation*, Minneapolis, MN, 1996. P. 1387–1394.

[Hansen (1965)] Hansen E. R. Interval arithmetic in matrix computations part I // *SIAM Journal on Numerical Analysis: Series B* **2**(2). 1965. P. 308–320.

[Hansen (1968)] Hansen E. R. On solving systems of equations using interval arithmetic // *Mathematical Computing*. 1968. **22**. P. 374–384.

[Hansen (1992a)] Hansen E. R. Bounding the solution of interval linear equations // *SIAM Journal on Numerical Analysis* 1992a. **29**(5). P. 1493–1503.

[Hansen (1992b)] Hansen E. R. *Global Optimization using Interval Analysis*. Marcel Dekker, New York, N.Y., 1992b.

[Hansen and Sengupta (1980)] Hansen E. R. and Sengupta S. Global constrained optimization using interval analysis, / Ed. K. Nickel, *Interval Mathematics 1980*, New York, 1980. N.Y.: Academic Press, 1980. P. 25–47.

[Hanson (1968)] Hanson R. J. Interval arithmetic as a closed arithmetic system on a computer, Technical Memorandum 197, Jet Propulsion Laboratory, Section 314, California Institute of Technology, Pasadena, CA, 1968.

[Happel (1986)] Happel J. *Isotopic Assessment of Heterogeneous Catalysis*, Orlando, FL: Academic Press, 1986.

[Haroud *et al.* (1995)] Haroud D., Boulanger S., Gelle E. M. and Smith I. F. C. Management of conflicts for preliminary engineering design tasks // *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* **9**(4). 1995. P. 313–323.

[Hecht (1987)] Hecht E. *Optics*. 2nd edition. Addison-Wesley, Reading, MA, 1987.

[Hong *et al.* (1997)] Hong H., Liska R. and Steinberg S. Testing stability by quantifier elimination // *Journal of Symbolic Computation*. 1997. **24**(2). P. 161–187.

[Horowitz (1963)] Horowitz I. M. *Synthesis of Feedback Systems*, N.Y.: Academic Press, 1963.

[Husty (1996)] Husty M. L. An algorithm for solving the direct kinematics of general Stewart–Gough platforms, *Mechanism and Machine Theory*. 1996. **31**(4). P. 365–379.

[Hyvönen (1992)] Hyvönen, E. Constraint reasoning based on interval arithmetic; the tolerance propagation approach // *Artificial Intelligence*. 1992. **58**(1-3). P. 71–112.

[IEEE Computer Society (1985)] IEEE Computer Society (1985) IEEE standard for binary floating-point arithmetic, *Technical Report IEEE Std. 754-1985*, American National Standards Institute. Available at: http://standards.ieee.org/.

[Innocenti and Parenti-Castelli (1991)] Innocenti C. and Parenti-Castelli V. Direct kinematics of the reverse Stewart platform mechanism // *Proceedings of the 3rd IFAC/IFIP/IMACS Symposium on Robot Control* Vienna, Austria, 1991. P. 75–80.

[Ioakimidis (1997)] Ioakimidis N. I. Quantifier elimination in applied mechanics problems with cylindrical algebraic decomposition // *International Journal of Solids and Structures*. 1997. **34**(30). P. 4037–4070.

[Iri (1991)] Iri M. History of automatic differentiation and rounding estimation, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application* / Eds. A. Griewank and G. F. Corliss, SIAM, Philadelphia, PA, 1991. P. 1–16.

[Jacquez (1972)] Jacquez J. A. *Compartmental Analysis in Biology and Medecine*. Elsevier, Amsterdam, the Netherlands, 1972.

[Jansson and Knüppel (1995)] Jansson C. and Knüppel O. A branch and bound algorithm for bound constrained optimization problems without derivatives // *Journal of Global Optimization*. 1995. **7**. P. 297–331.

[Jaulin (1994)] Jaulin L. (1994) *Solution globale et garantie de problèmes ensemblistes; application à l'estimation non linéaire et à la commande robuste*, PhD dissertation, Université Paris-Sud, Orsay, France. Available at: http://www.istia.univ-angers.fr/~jaulin/thesejaulin.zip.

[Jaulin (2000a)] Jaulin L. Interval constraint propagation with application to bounded-error estimation // *Automatica*. 2002a. **36**(10). P. 1547–1552.

[Jaulin (2000b)] Jaulin L. (2000b) *Le calcul ensembliste par analyse par intervalles*, Habilitation à diriger des recherches, Université Paris-Sud, Orsay, France. Available at: http://www.istia.univ-angers.fr/~jaulin/hdrjaulin.zip.

[Jaulin (2001a)] Jaulin L. Path planning using intervals and graphs // *Reliable Computing*. 2001a. **7**(1). P. 1–15.

[Jaulin (2001b)] Jaulin L. Reliable minimax parameter estimation // *Reliable Computing*. 2001b. **7**(3). P. 231–246.

[Jaulin *et al.* (2001a)] Jaulin L., Braems I., Kieffer M. and Walter E. (2001a) Nonlinear state estimation using forward-backward propagation of intervals, to appear in *Proceedings of SCAN 2000.*

[Jaulin and Godon (1999)] Jaulin L. and Godon A. Motion planning using interval analysis, *Proceedings of the MISC'99 Workshop on Applications of Interval Analysis to Systems and Control.* Girona, Spain. 1999. P. 335–346.

[Jaulin *et al.* (2001b)] Jaulin L., Kieffer M., Braems I. and Walter E. (2001b) Guaranteed nonlinear estimation using constraint propagation on sets, to appear in *International Journal of Control.*

[Jaulin and Walter (1993a)] Jaulin L. and Walter E. Guaranteed nonlinear parameter estimation from bounded-error data via interval analysis // *Mathematics and Computers in Simulation.* 1993a. **35**(2). P. 123–137.

[Jaulin and Walter (1993b)] Jaulin L. and Walter E. Guaranteed nonlinear parameter estimation via interval computations // *Interval Computations.* 1993a. **3**. P. 61–75.

[Jaulin and Walter (1993c)] Jaulin L. and Walter E. Set inversion via interval analysis for nonlinear bounded-error estimation // *Automatica.* 1993c. **29**(4). P. 1053–1064.

[Jaulin and Walter (1996)] Jaulin L. and Walter E. Guaranteed tuning, with application to robust control and motion planning // *Automatica.* 1996. **32**(8).P. 1217–1221.

[Jaulin and Walter (1997)] Jaulin L. and Walter E. Global numerical approach to nonlinear discrete-time control // *IEEE Transactions on Automatic Control.* 1997. **42**(6). P. 872–875.

[Jaulin and Walter (1999)] Jaulin L. and Walter E. Guaranteed bounded-error parameter estimation for nonlinear models with uncertain experimental factors // *Automatica.* 1999. **35**(5). P. 849–856.

[Jaulin *et al.* (1996)] Jaulin L., Walter E. and Didrit O. Guaranteed robust nonlinear parameter bounding // *Proceedings of CESA'96 IMACS Multiconference (Symposium on Modelling, Analysis and Simulation)*, Lille, France, 1996. P. 1156–1161.

[Jaulin *et al.* (2000)] Jaulin L., Walter E., Lévêque O. and Meizel D. Set inversion for χ-algorithms, with application to guaranteed robot localization, *Mathematics and Computers in Simulation.* 2000. **52**(3-4). P. 197–210.

[Jirstrand (1997)] Jirstrand M. Nonlinear control system design by quantifier elimination // *Journal of Symbolic Computation.* 1997. **24**(2). P. 137–152.

[Kahan (1968)] Kahan W. A more complete interval arithmetic: Lecture notes for a summer course. Canada: University of Toronto, 1968

[Kahan (1996)] Kahan W. (1996) Lecture notes on the status of IEEE-754. Available at: http.cs.berkeley.edu/~wkahan/ieee754status/ieee754.ps.

[Kalman (1960)] Kalman R. E. A new approach to linear filtering and prediction problems // *Transactions of the AMSE. Part D. Journal of Basic Engineering.* 1960. **82**. P. 35–45.

[Kam *et al.* (1997)] Kam M., Zhu X. and Kalata P. Sensor fusion for mobile robot navigation // *Proceeedings of the IEEE.* 1997. **85**(1). P. 108–119.

[Kang and Krener (1998)] Kang W. and Krener A. Nonlinear observer design: A backstepping approach // *Proceedings of the 13th International Symposium on the Mathematical Theory of Networks and Systems.* Padova, Italy, 1998. P. 245–248.

[Kearfott (1989a)] Kearfott R. B. Interval arithmetic methods for nonlinear systems and nonlinear optimization: an introductory review // *Impact of Recent Computer Advances on Operations Research* / Eds. R. Sharda, B. L. Golden, E. Wasil, O. Balci and W. Stewart, N.Y.: North-Holland, 1989a. P. 533–542.

[Kearfott (1989b)] Kearfott R. B. Interval mathematics techniques for control theory computations // *Computation and Control. Proceedings of the Bozeman Conference*/ /Eds. K. Bowers and J. Lund, Vol. 20 of *Progress in Systems and Control Theory*, Birkhäuser, Boston, MA, 1989b. P. 169–178.

[Kearfott (1996a)] Kearfott R. B. Interval extensions of non-smooth functions for global optimization and nonlinear system solvers // *Computing.* 1996a. **57**(2). P. 149–162.

[Kearfott (1996b)] Kearfott R. B. *Rigorous Global Search: Continuous Problems.* Kluwer, Dordrecht, the Netherlands, 1996b.

[Kearfott *et al.* (1992)] Kearfott R. B., Dawande M., Du K. S. and Hu C. INTLIB: a portable FORTRAN 77 elementary function library // *Interval Computations.* 1992. **3**(5). P. 96–105.

[Kharitonov (1978)] Kharitonov V. L. Asymptotic stability of an equilibrium position of a family of systems of linear differential equations // *Differentsial'nye Uravnenya.* 1978. **14**(11). P. 2086–2088. (русск.) [Хритонов (1978)] Хритонов В. Л. *Асимптотическая устойчивость положения равновесия семейства систем линейных дифференциальных уравнений* // Дифференциальные уравнения, 1978. **14**(11). С.2086–2088.

[Khatib (1986)] Khatib O. Real-time obstacle avoidance for manipulators and mobile robots // *International Journal of Robotics Research.* 1986. **5**(1). P. 90–98.

[Khlebalin (1992)] Khlebalin N. A. Interval automatic systems theory, computer-aided design and applications // *Interval Computations.*1992. **3**. P. 106–115.

[Kieffer (1999)] Kieffer M. *Estimation ensembliste par analyse par intervalles, application à la localisation d'un véhicule*, PhD dissertation, Orsay, France: Université Paris-Sud, 1999.

[Kieffer *et al.* (2001a)] Kieffer M., Jaulin L., Braems I. and Walter E. (2001a) Guaranteed set computation with subpavings, to appear in *Proceedings of SCAN 2000.*

[Kieffer *et al.* (1998)] Kieffer M., Jaulin L. and Walter E. Guaranteed recursive nonlinear state estimation using interval analysis // *Proceedings of the 37th IEEE Conference on Decision and Control.* Tampa, FL, P. 3966–3971.

[Kieffer *et al.* (2001b)] Kieffer M., Jaulin L. and Walter E. (2001b) Guaranteed recursive nonlinear state bounding using interval analysis, to appear in *International Journal of Adaptative Control and Signal Processing.*

[Kieffer *et al.* (1999)] Kieffer M., Jaulin L., Walter E. and Meizel D. Guaranteed mobile robot tracking using interval analysis // *Proceedings of the MISC'99 Workshop on Applications of Interval Analysis to Systems and Control.* Girona, Spain, 1999. P. 347–359.

[Kieffer *et al.* (2000)] Kieffer M., Jaulin L., Walter E. and Meizel D. Robust autonomous robot localization using interval analysis // *Reliable Computing.* 2000. **6**(3). P. 337–362.

[Kieffer *et al.* (2001c)] Kieffer M., Jaulin L., Walter E. and Meizel D. Localisation et suivi robustes d'un robot mobile grâce à l'analyse par intervalles // *Traitement du Signal.* 2001c. **17**(3). P. 207–219.

[Kieffer and Walter (1998)] Kieffer M. and Walter E. .1998. Interval analysis for guaranteed nonlinear parameter estimation // *MODA 5-Advances in Model-Oriented Data Analysis and Experiment Design* / Eds. A. C. Atkinson, L. Pronzato and H. P. Wynn, Physica-Verlag, Heidelberg, Germany. P. 115–125.

[Kiendl and Michalske (1992)] Kiendl H. and Michalske A. Robustness analysis of linear control systems with uncertain parameters by the method of convex decomposition // *Robustness of Dynamic Systems with Parameter Uncertainties*/ Eds. M. Mansour, S. Balemi and W. Truol, Birkhäuser, Boston, MA, 1992. P. 189–198.

[Klatte *et al.* (1992)] Klatte R., Kulisch U., Neaga M., Ratz D. and Ullrich C. PASCAL-XSC — *Language References with Examples*, N.Y.: Springer-Verlag, 1992.

[Klatte *et al.* (1993)] Klatte R., Kulisch U., Wieth A., Lawo C. and Rauch M. C-XSC: *A C++ Class Library For Extended Scientific Computing.* Springer-Verlag, Berlin, Germany, 1993.

[Knofel (1993)] Knofel A. Hardware kernel for scientific/engineering computations // *Scientific Computing with Automated Result Verification*, / Eds. E. Adams and U. Kulisch, Orlando, FL: Academic Press, 1993. P. 549–570.

[Knüppel (1993)] Knüppel, O. BIAS — basic interval arithmetic subroutines, Technical Report 93.3, Institut für Informatik III. Technische Universität Hamburg, Germany: Hamburg-Harburg, 1993.

[Knüppel (1994)] Knüppel O. PROFIL/BIAS — A fast interval library // *Computing.* 1994. **53**. P. 277–287.

[Kokame and Mori (1992)] Kokame H. and Mori T. A branch and bound method to check the stability of a polytope of matrices // *Robustness of Dynamic Systems with Parameter Uncertainties* / Eds. M. Mansour, S. Balemi and W. Truöl, Birkhäuser, Boston, MA, 1992. P. 125–137.

[Kolev (1993)] Kolev L. An interval first-order method for robustness analysis // *Procedings of the IEEE International Symposium on Circuits and Systems.* Chicago, IL, 1993. P. 2522–2524.

[Kolev *et al.* (1988)] Kolev L. V., Mladenov V. M. and Vladov S. S. Interval mathematics algorithms for tolerance analysis // *IEEE Transactions on Circuits and Systems.* 1988. **35**(8). P. 967–974.

[Kolla *et al.* (1999)] Kolla R., Vodopivec A. and Wolff von Gudenberg J. The IAX architecture interval arithmetic extension, Technical Report 225, Institut für Informatik, Universität Würzburg. Germany, 1999.

[Kuc and Siegel (1987)] Kuc R. and Siegel M. W. Physically based simulation model for acoustic sensor robot navigation // *IEEE Transactions on Pattern Analysis and Machine Intelligence.* 1987. **9**(6). P. 766–778.

[Kühn (1998)] Kühn W. Rigorously computed orbits of dynamical systems without the wrapping effect // *Computing.* 1998. **61**. P. 47–67.

[Kulisch and Miranker (1981)] Kulisch U. and Miranker W. L. *Computer Arithmetic in Theory and Practice.* N.Y.: Academic Press, 1981.

[Kurzhanski and Valyi (1997)] Kurzhanski A. and Valyi I. *Ellipsoidal Calculus for Estimation and Control.* Birkhäuser, Boston, MA, 1997.

[Kwakernaak (1993)] Special Issue on Robust Control / Ed. H. Kwakernaak *Automatica.* 1993. **29**(1). P. 1–252.

[Lahanier *et al.* (1987)] Lahanier H., Walter E. and Gomeni R. OMNE: a new robust membership-set estimator for the parameters of nonlinear models, *Journal of Pharmacokinetics and Biopharmaceutics.* 1987. **15**. P. 203–219.

[Laumond (1986)] Laumond J. P. Feasible trajectories for mobile robots with kinematic and environment constraints // *Proceedings of the International Conference on Intelligent Autonomous Systems.* Amsterdam, the Netherlands, 1986. P. 246–354.

[Lazard (1992)] Lazard D. Stewart platform and Gröbner basis, *Proceedings of the 3rd International Workshop on Advances in Robot Kinematic* / Eds. V. Parenti-Castelli and J. Lenarcic, Ferrare, Italy, 1992. P. 136–142.

[Lazard (1993)] Lazard D. Generalized Stewart platform: how to compute with rigid motions? // *Proceedings of the IMACS Symposium on Symbolic Computation*, Lille, France, 1993. P. 85–88.

[Lee and Roth (1993)] Lee H. and Roth B. A closed-form solution of the forward displacement analysis of a class of in-parallel mechanisms // *Proceedings of the IEEE International Conference on Robotics and Automation*, Vol. 1, Atlanta, GA, 1993. P. 720–724.

[Lefèvre *et al.* (1998)] Lefèvre V., Muller J. and Tisserand A. Toward correctly rounded transcendentals // *IEEE Transactions on Computers*. 1998. **47**(11). P. 1235–1243.

[Leonard and Durrant-Whyte (1991)] Leonard J. J. and Durrant-Whyte H. F. Mobile robot localization by tracking geometric beacons // *IEEE Transactions on Robotics and Automation*. 1991. **7**(3). P. 376–382.

[Lerch and Wolff von Gudenberg (2000)] Lerch M. and Wolff von Gudenberg J. `fi_lib++` : Specification, implementation and test of a library for extended interval arithmetic, *Proceedings of the 4th Conference on Real Numbers and Computers*, Dagstuhl, Germany, 2000. Available at: http://www.imada.sdu.dk/~kornerup/RNC4/papers/p03.ps.

[Lévêque (1998)] Lévêque O. *Méthodes ensemblistes pour la localisation de véhicules*. PhD dissertation. Université de Technologie, Compiègne, France, 1998.

[Levine (1996)] *The Control Handbook* / Ed. W. Levine CRC Press, Boca Raton, FL, 1996.

[Lhomme (1993)] Lhomme O. Consistency techniques for numeric CSPs // *Proceedings of the International Joint Conference on Artificial Intelligence*. Chambéry, France, 1993. P. 232–238.

[Lhomme and Rueher (1997)] Lhomme O. and Rueher M. Application des techniques CSP au raisonnement sur les intervalles, *Revue d'Intelligence Artificielle*. 1997. **11**(3). P. 283–311.

[Liska and Steinberg (1996)] Liska R. and Steinberg S. Solving stability problems using quantifier elimination // *Stability Theory: Hurwitz Centenary Conference* / Eds. R. Jeltsch and M. Mansour, Vol. 121 of *International Series on Numerical Mathematics*, Birkhäuser, Basel, Switzerland, 1996. P. 205–210.

[Lohner (1987)] Lohner R. Enclosing the solutions of ordinary initial and boundary value problems // *Computer Arithmetic: Scientific Computation and Programming Languages* / Eds. E. Kaucher, U. Kulisch and C. Ullrich (eds), BG Teubner, Stuttgart, Germany, 1987. P. 255–286.

[Lottaz (2000)] Lottaz C. *Collaborative design using solution spaces*, PhD dissertation 2119. Swiss Federal Institute of Technology in Lausanne, Switzerland, 2000.

[Lottaz *et al.* (1998)] Lottaz C., Sam-Haroud D., Faltings B. V. and Smith I. F. C. Constraint techniques for collaborative design // *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence*, Taipei, Taiwan, ROC, 1998. P. 34–41.

[Lozano-Pérez (1981)] Lozano-Pérez T. Automatic planning of manipulator transfer movements // *IEEE Transactions on Systems Man and Cybernetics*. 1981. **11**(10). P. 681–698.

[Lozano-Pérez (1983)] Lozano-Pérez T. Spatial planning: A configuration space approach // *IEEE Transactions on Computers*. 1983. **32**(2). P. 108–120.

[Lozano-Pérez and Wesley (1979)] Lozano-Pérez T. and Wesley M. A. An algorithm for planning collision-free paths among polyhedral obstacles // *Communications of the ACM*. 1979. **22**(10). P. 560–570.

[Luenberger (1966)] Luenberger D. Observers for multivariable systems // *IEEE Transactions on Automatic Control*. 1966. **11**(2). P. 190–197.

[Mackworth (1977a)] Mackworth A. K. Consistency in networks of relations // *Artificial Intelligence*. 1977a. **8**(1). P. 99–118.

[Mackworth (1977b)] Mackworth A. K. On reading sketch maps // *Proceedings of the International Joint Conference on Artificial Intelligence*, Vol. 2, Cambridge, MA, 1977b. P. 598–606.

[Mackworth and Freuder (1985)] Mackworth A. K. and Freuder E. C. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems // *Artificial Intelligence*. 1985. **25**(1). P. 65–74.

[Maksarov and Norton (1996)] Maksarov D. and Norton J. P. State bounding with ellipsoidal set description of the uncertainty // *International Journal of Control*. 1996. **65**(5). P. 847–866.

[Malan *et al.* (1997)] Malan S. A., Milanese M. and Taragna M. Robust analysis and design of control systems using interval arithmetics // *Automatica*. 1997. **33**(7). P. 1363–1372.

[Malan *et al.* (1992)] Malan S. A., Milanese M., Taragna M. and Garloff J. B^3 algorithm for robust performance analysis in presence of mixed parametric and dynamic pertubations // *Proceedings of the 31st IEEE Conference on Decision and Control*, Tucson, AZ, 1992. P. 128–133.

[Marden (1966)] Marden M. *Geometry of Polynomials* // American Mathematical Society, Providence, RI, 1966.

[Marti and Rueher (1995)] Marti P. and Rueher M. A distributed cooperating constraint solving system // *International Journal of Artificial Intelligence Tools*. 1995. **4**(1-2). P. 93–113.

[Matula and Kornerup (1985)] Matula D. W. and Kornerup P. Finite precision rational arithmetic: Slash number system // *IEEE Transactions on Computers*. 1985. **34**(1). P. 3–18.

[McKendall (1990)] McKendall R. *Minimax estimation of a discrete location parameter for a continuous distribution*. PhD dissertation. Computer and Information Science Department, University of Pennsylvania, Philadelphia, MA, 1990. Available at: http://www.cis.upenn.edu/~mcken/pub/om.ps.gz.

[McKendall and Mintz (1992)] McKendall R. and Mintz M. Robust sensor fusion with statistical decision theory // *Data Fusion in Robotics and Machine Intelligence* / Eds. M. A. Abidi and R. C. Gonzalez, Boston, MA: Academic Press, 1992. P. 211–244.

[Meiri *et al.* (1990)] Meiri I., Pearl J. and Dechter R. Tree decomposition with applications to constraint processing // *Proceedings of the 8th AAAI National Conference on Artificial Intelligence* / Eds. T. Dietterich and W. Swartout, Boston, MA, 1990. P. 10–16.

[Meizel *et al.* (1996)] Meizel, D., Preciado-Ruiz, A. and Halbwachs, E. Estimation of mobile robot localization: geometric approaches // *Bounding Approaches to System Identification*, Plenum Press, New York / Eds. M. Milanese, J. Norton, H. Piet-Lahanier and E. Walter, N.Y., 1996. P. 463–489.

[Merlet (1990)] Merlet J. P. *Les robots parallèles*. Hermes, Paris, France, 1990.

[Milanese and Belforte (1982)] Milanese M. and Belforte G. Estimation theory and uncertainty intervals evaluation in presence of unknown but bounded errors: Linear families of models and estimators // *IEEE Transactions on Automatic Control*. 1982. **27**(2). P. 408–414.

[Milanese *et al.* (1991)] Milanese M., Fioro G., Malan S. and Taragna M. Robust performance design of nonlinearly perturbed control systems, *Proceedings of the International Workshop on Robust Control*, San Antonio, TX. 1991. P. 113–124.

[Milanese *et al.* (1996)] / Eds. Milanese M., Norton J., Piet-Lahanier H. and Walter E. *Bounding Approaches to System Identification*. N.Y.: Plenum Press, 1996.

[Milanese and Vicino (1991)] Milanese M. and Vicino A. Estimation theory for nonlinear models and set membership uncertainty, *Automatica*. 1991. **27**(2): 403–408.

[Mishra (1993)] Mishra B. *Algorithmic Algebra*, N.Y.: Springer-Verlag, 1993.

[Montanari and Rossi (1991)] Montanari U. and Rossi F. Constraint relaxation may be perfect // *Artificial Intelligence*. 1991. **48**(2). P. 143–170.

[Moore (1959)] Moore R. E. Automatic error analysis in digital computation. Technical Report LMSD-48421 Lockheed Missiles and Space Co, Palo Alto, CA, 1959.

[Moore (1966)] Moore R. E. *Interval Analysis*. Prentice-Hall, Englewood Cliffs, NJ, 1966.

[Moore (1976)] Moore R. E. On computing the range of values of a rational function of n variables over a bounded region // *Computing*. 1976. **16**. P. 1–15.

[Moore (1979)] Moore R. E. *Methods and Applications of Interval Analysis* SIAM, Philadelphia, PA, 1979.

[Moore (1992)] Moore R. E. Parameter sets for bounded-error data, *Mathematics and Computers in Simulation*. 1992. **34**(2). P. 113–119.

[Moore and Ratschek (1988)] Moore R. E. and Ratschek H. Inclusion function and global optimization II // *Mathematical Programing*. 1988. **41**(3). P. 341–356.

[Mourrain (1993)] Mourrain B. The 40 generic positions of a parallel robot // *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, Kiev, Ukraine, 1993. P. 173–182.

[Muller (1997)] Muller J. M. *Elementary Functions, Algorithms and Implementation*. Birkhäuser, Boston, MA, 1997.

[Murdock *et al.* (1991)] Murdock T. M., Schmitendorf W. E. and Forrest S. Use of a genetic algorithm to analyse robust stability problems, *Proceedings of the American Control Conference*, Boston, MA, 1991. P. 886–889.

[Nanua *et al.* (1990)] Nanua P., Waldron K. J. and Murthy V. Direct kinematic solution of a Stewart platform // *IEEE Transactions on Robotics and Automation*. 1990. **6**(4). P. 438–443.

[Nemirovskii (1993)] Nemirovskii A. Several NP-hard problems arising in robust stability analysis // *Mathematics of Control, Signals and Systems*. 1990. **6**(2). P. 99–105.

[Neubacher (1997)] Neubacher A. *Parametric robust stability by quantifier elimination*. PhD dissertation, Research Institute for Symbolic Computation – Universität Linz, Austria, 1990.

[Neumaier (1985)] Neumaier A. Interval iteration for zeros of systems of equations, *BIT*. 1985. **25**(1). P. 256–273.

[Neumaier (1990)] Neumaier A. *Interval Methods for Systems of Equations*, Cambridge, UK: Cambridge University Press, 1990.

[Nickel (1966)] Nickel K. Über die Notwendigkeit einer Fehlerschranken-Arithmetik für Rechenautomaten // *Numerische Mathematik*. 1966. **9**. P. 69–79.

[Nilsson (1969)] Nilsson N. A mobile automaton: an application of artificial intelligence techniques // *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, Washington, DC, 1969. P. 509–520.

[Norton (1987)] Norton J. P. Identification of parameter bounds for ARMAX models from records with bounded noise // *International Journal of Control*. 1969. **45**(2). P. 375–390.

[Norton (1996)] Norton J. P. Roles for deterministic bounding in environmental modeling // *Ecological Modelling*. 1996. **86**. P. 157–161.

[Norton (1994)] Norton J. P. (ed.) Special Issue on Bounded-Error Estimation: Issue 1 // *International Journal of Adaptive Control and Signal Processing*. 1994. **8**(1):1–118.

[Norton (1995)] Norton J. P. (ed.) Special Issue on Bounded-Error Estimation: Issue 2 // *International Journal of Adaptive Control and Signal Processing*. 1995. **9**(1). P. 1–132.

[O'Dunlaing and Yap (1982)] O'Dunlaing C. and Yap C. K. A retraction method for planning the motion of a disc // *Journal of Algorithms*. 1982. **6**(1): 104–111.

[Olson (2000)] Olson C. Probabilistic self-localization for mobile robots // *IEEE Transactions on Robotics and Automation*. 2000. **16**(1). P. 55–66.

[Pardalos and Rosen (1987)] Pardalos P. and Rosen J. *Constrained Global Optimization: Algorithms and Applications*, Vol. 268 of *Lecture Notes in Computer Science*, Berlin, Germany: Springer-Verlag, 1987.

[Piazzi and Visioli (1998)] Piazzi A. and Visioli A. Global minimum-time trajectory planning of mechanical manipulators using interval analysis // *International Journal of Control*. 1998. **71**(4). P. 631–652.

[Poljak and Rohn (1993)] Poljak S. and Rohn J. Checking robust nonsingularity is NP-hard // *Mathematics of Control, Signals and Systems*. 1998. **6**. P. 1–9.

[Pronzato and Walter (1994)] Pronzato L. and Walter E. Minimal volume ellipsoids // *International Journal of Adaptative Control and Signal Processing*. 1994. **8**(1). P. 15–30.

[Pruski (1996)] Pruski A. *La robotique mobile, planification de trajectoire*. Hermes, Paris, France, 1996.

[Pruski and Rohmer (1997)] Pruski A. and RohmerS. Robust path planning for non-holonomic robots // *Journal of Intelligent and Robotic Systems*. 1996. **18**. P. 329–350.

[Psarris and Floudas (1995)] Psarris P. and Floudas C. A. Robust stability analysis of systems with real parametric uncertainty: a global optimization approach // *International Journal of Robust and Nonlinear Control*. 1995. **5**(8). P. 699–717.

[Raghavan (1993)] Raghavan M. The Stewart platform of general geometry has 40 configurations // *ASME Journal of Mechanical Design*. 1993. **115**(2). P. 277–282.

[Raghavan and Roth (1995)] Raghavan M. and Roth B. Solving polynomial systems for the kinematic analysis and synthesis of mechanisms and robot manipulators // *ASME Journal of Mechanical Design*. 1995. **117**. P. 71–79.

[Rall (1980)] Rall L. B. Applications of software for automatic differentiation in numerical computation // *Fundamentals of Numerical Computation (Computer Oriented Numerical Analysis)* / Eds. G. Alefeld and R. D. Grigorieff. Computing Supplement No. 2, Berlin, Germany: Springer-Verlag, 1980. P. 141–156.

[Rall (1981)] Rall L. B. *Automatic Differentiation: Techniques and Applications*. Vol. 120 of *Lecture Notes in Computer Science*, Berlin, Germany: Springer-Verlag, 1981.

[Rall and Corliss (1999)] Rall L. B. and Corliss G. F. Automatic differentiation: Point and interval AD // *Encyclopedia of Optimization* / Eds. P. M. Pardalos and C. A. Floudas, Kluwer, Dordrecht, the Netherlands, 1999.

[Ratschan (2000a)] Ratschan S. (2000a) Approximate quantified constraint solving (AQCS). Available at: http://www.risc.uni-linz.ac.at/research/software/AQCS.

[Ratschan (2000b)] Ratschan S. Uncertainty propagation in heterogeneous algebras for approximate quantified constraint solving, *Journal of Universal Computer Science*. 2000b. **6**(9). P. 861–880.

[Ratschek and Rokne (1984)] Ratschek H. and Rokne J. *Computer Methods for the Range of Functions*. Ellis Horwood, Chichester, UK, 1984.

[Ratschek and Rokne (1988)] Ratschek H. and Rokne J. *New Computer Methods for Global Optimization*, Ellis Horwood, Chichester, UK, 1988.

[Ratschek and Rokne (1995)] Ratschek H. and Rokne J. Interval methods // *Handbook of Global Optimization* / Eds. R. Horst and P. Pardalos, Kluwer, Dordrecht, the Netherlands, 1995. P. 751–828.

[Ratz and Csendes (1995)] Ratz D. and Csendes T. On the selection of subdivision directions in interval branch-and-bound methods for global optimization // *Journal of Global Optimization*. 1995. **7**(2). P. 183–207.

[Reboulet (1988)] Reboulet C. Modélisation des robots parallèles, // *Techniques de la robotique, architecture et commande*, / Eds. J.-D. Boissonat, B. Faverjon and J.-P. Merlet Hermes, Paris, France, 1988. P. 257–284.

[Rimon and Koditschek (1992)] Rimon E. and Koditschek D. E. Exact robot navigation using artificial potential fields // *IEEE Transactions on Robotics and Automation*. 1992. **8**(5). P. 501–518.

[Rohn (1994)] Rohn J. NP-hardness results for linear algebraic problems with interval data, // *Topics in Validated Computations*, Elsevier / Ed. J. Herzberger, Amsterdam, the Netherlands, 1994. P. 463–471.

[Rump (1999)] Rump S. M. INTLAB–INTerval LABoratory // *Developments in Reliable Computing* / Ed. T. Csendes, Kluwer, Dordrecht, the Netherlands, P. 77–104.

[Rump (2001)] Rump S. M. INTLAB–INTerval LABoratory // *Handbook of Computer Algebra: Foundations, Applications, Systems* / Eds. J. Grabmeier, E. Kaltofen and V. Weispfennig, Heidelberg, Germany: Springer-Verlag, 2001.

[Saeki (1986)] Saeki M. A method of robust stability analysis with highly structured uncertainties // *IEEE Transactions on Automatic Control*. 1986. **31**(10). P. 935–940.

[Safonov and Athans (1981)] Safonov M. G. and Athans M. A multiloop generalisation of the circle criterion for stability margin analysis // *IEEE Transactions on Automatic Control*. 1981. **26**(2). P. 415–422.

[Sam-Haroud (1995)] Sam-Haroud D. *Constraint consistency techniques for continuous domains*. PhD dissertation 1423. Swiss Federal Institute of Technology in Lausanne, Switzerland, 1995.

[Sam-Haroud and Faltings (1996)] Sam-Haroud D. and Faltings B. Consistency techniques for continuous constraints // *Constraints*. 1996. **1**(1-2). P. 85–118.

[Samet (1982)] Samet H. Neighbor finding techniques for images represented by quadtrees // *Computer Graphics and Image Processing*. 1982. **18**(1). P. 37–57.

[Samet (1990)] Samet H. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.

[Schulte and Swartzlander, Jr. (2000)] Schulte M. J. and Swartzlander Jr. E. A family of variable-precision, interval arithmetic processors // *IEEE Transactions on Computers*. 2000. **49**(5): 387–397.

[Schweppe (1968)] Schweppe F. C. Recursive state estimation: unknown but bounded errors and system inputs // *IEEE Transactions on Automatic Control*. 1968. **13**(1). P. 22–28.

[Schwetlick and Tiller (1985)] Schwetlick H. and Tiller V. Numerical methods for estimating parameters in nonlinear models with errors in the variables, *Technometrics*. 1985. **27**(1). P. 17–24.

[Severance (1998)] Severance C. IEEE 754: An interview with William Kahan, *IEEE Computer*. 1998. **31**(3). P. 114–115.

[Seybold *et al.* (1998)] Seybold B., Metzger F., Ogan G. and Simon K. Using blocks for constraint satisfaction // *Principles and Practice of Constraint Programming — Proceedings of CP98* / Eds. M. J. Maher and J. Puget. Vol. 1520 of *Lecture Notes in Computer Science*, Heidelberg, Germany: Springer-Verlag, 1998. P. 474.

[Sideris (1991)] Sideris A. An efficient algorithm for checking the robust stability of a polytope of polynomials // *Mathematics of Control, Signals, and Systems*. 1991. **4**(3). P. 315–337.

[Skelboe (1974)] Skelboe S. Computation of rational interval functions // *BIT*. 1974. **14**. P. 87–95.

[Sondergeld (1983)] Sondergeld K. P. A generalization of the Routh–Hurwitz stability criteria and an application to a problem in robust controller design // *IEEE Transactions on Automatic Control*. 1983. **28**(10). P. 965–970.

[Sorenson (1983)] Sorenson H. (ed.) Special Issue on Applications of Kalman Filtering // *IEEE Transactions on Automatic Control*. 1983. **28**(3). P. 253–434.

[Steinberg and Liska (1996)] Steinberg S. and Liska R. Stability analysis by quantifier elimination // *Mathematics and Computers in Simulation*. 1996. **42**(4-6). P. 629–638.

[Stewart (1965)] Stewart D. A platform with six degrees of freedom // *Proceedings of the Institut of Mechanical Engineering*. 1965. **180**(1). P. 371–386.

[Stine and Schulte (1998a)] Stine J. E. and Schulte M.-J. A combined interval and floating-point divider // *Proceedings of the 32nd Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, 1998a. P. 218–222.

[Stine and Schulte (1998b)] Stine J. E. and Schulte M. J. A combined interval and floating-point multiplier // *Proceedings of the 8th Great Lakes Symposium on VLSI*, Lafayette, LA, 1998b. P. 208–213.

[Stroustrup (1991)] Stroustrup B. *The C++ Programming Language*. 2nd edition, Reading, MA: Addison-Wesley, 1991.

[Swartzlander and Alexopoulos (1975)] Swartzlander E. E. and Alexopoulos G. A. The sign/logarithm number system // *IEEE Transactions on Computers*. 1975. **24**(12). P. 1238–1242.

[Tesi and Vicino (1989)] Tesi A. and Vicino A. A new fast algorithm for robust stability analysis of control systems with linearly dependent parametric uncertainties // *Systems and Control Letters*. 1989. **13**(4). P. 321–329.

[van Hentenryck *et al.* (1997)] van Hentenryck P., Deville Y. and Michel L. *Numerica: A Modeling Language for Global Optimization*. Boston, MA: MIT Press, 1997.

[van Hentenryck *et al.* (1998)] van Hentenryck P., Michel L. and Benhamou F. `Newton`: Constraint programming over nonlinear constraints, *Science of Computer Programming*. 1998. **30**(1–2). P. 83–118.

[Veres and Norton (1996)] Veres S. M. and Norton J. P. Parameter-bounding algorithms for linear errors-in-variables models, // *Bounding Approaches to System Identification*, / Eds. M. Milanese, J. Norton, H. Piet-Lahanier and E. Walter, Plenum, N.Y., 1996. P. 275–288.

[Vicino *et al.* (1990)] Vicino A., Tesi A. and Milanese M. Computation of nonconservative stability perturbation bounds for systems with nonlinearly correlated uncertainties, *IEEE Transactions on Automatic Control* 1990. **35**(7). P. 835–841.

[Walster (1998)] Walster G. W. (1998) The extended real interval system. Available at: http://www.mscs.edu/globsol/walster-papers.html.

[Walster *et al.* (1985)] Walster G. W., Hansen E. R. and Sengupta S. Test results for a global optimization algorithm // *Numerical Optimization 1984*, / Eds. P. Boggs, R. H. Byrd and R. B. Schnaubel, SIAM, Philadelphia, PA, pp. 272–287.

[Walter (1990)] Walter E. (ed.) Special Issue on Parameter Identification with Error Bounds // *Mathematics and Computers in Simulation*. 1990. **32**(5–6). P. 447–607.

[Walter and Jaulin (1994)] Walter E. and Jaulin L. Guaranteed characterization of stability domains via set inversion // *IEEE Transactions on Automatic Control*. 1994. **39**(4). P. 886–889.

[Walter and Piet-Lahanier (1989)] Walter, E. and Piet-Lahanier, H. Exact recursive polyhedral description of the feasible parameter set for bounded-error models, *IEEE Transactions on Automatic Control*. 1989. **34**(8): 911–915.

[Walter and Pronzato (1997)] Walter E. and Pronzato L. *Identification of Parametric Models from Experimental Data*, London, UK: Springer-Verlag, 1997.

[Waltz (1975)] Waltz D. Generating semantic descriptions from drawings of scenes with shadows // *The Psychology of Computer Vision* / Ed. P. H. Winston, McGraw-Hill, N.Y., 1975. P. 19–91.

[Wampler (1996)] Wampler C. W. Forward displacement analysis of general six-in-parallel SPS (Stewart) platform manipulators using soma coordinates // *Mechanism and Machine Theory*. 1996. **31**(3). P. 331–337.

[Wang and Chen (1993)] Wang L.-C. T. and Chen C. C. On the numerical kinematic analysis of general parallel robotic manipulators, *IEEE Transactions on Robotics and Automation*. 1993. **9**(3): 272–285.

[Wei and Yedavalli (1989)] Wei K. H. and Yedavalli R. K. Robust stabilizability for linear systems with both parameter variation and unstructured uncertainty, *IEEE Transactions on Automatic Control*. 1989. **34**(2). P. 149–156.

[Willems (1986a)] Willems J. C. From time series to linear systems, part 1, finite dimensional linear time invariant systems // *Automatica*. 1986a. **22**(5): 561–580.

[Willems (1986b)] Willems J. C. From time series to linear systems, part 2, exact modelling // *Automatica*. 1986b. **22**(6): 675–694.

[Witsenhausen (1968)] Witsenhausen H. S. Sets of possible states of linear systems given perturbed observations // *IEEE Transactions on Automatic Control*. 1968. **13**(5). P. 556–558.

[Wolfe (1996)] Wolfe M. A. Interval methods for global optimization // *Applied Mathematics and Computation*. 1996. **75**. P. 179–206.

[Wolfe (1999)] Wolfe M. A. On discrete minimax problems in the set of real numbers using interval arithmetic // *Reliable Computing*. 1999. **5**(4). P. 371–383.

[Wolff von Gudenberg (1994)] Wolff von Gudenberg J. Comparison of accurate dot product algorithms. *Technical Report 881*. IRISA, Rennes, France, 1994. Available at: http://www.irisa.fr/bibli/publi/pi/1994/881/881.html.

[Wolff von Gudenberg (1996)] Wolff von Gudenberg J. Hardware support for interval computation // *Scientific Computing and Validated Numerics* / Eds. G. Alefeld, A. Frommer and B. Lang, Akademie-Verlag, Berlin, Germany, 1996. P. 32–37.

[Zhou (1996)] Zhou J. A permutation-based approach for solving the job-shop problem // *Constraints*. 1996. **1**: 1–30.

[Zuhe *et al.* (1990)] Zuhe S., Neumaier A. and Eiermann M. Solving minimax problems by interval methods // *BIT*. 1990. **30**: 742–751.

Дополнительная литература

[Добавлено переводчиком и научным редактором перевода]

[Алефельд Г. и Херцбергер Ю. (1987)] Алефельд Г., Херцбергер Ю. Введение в интервальные вычисления. М.: Мир, 1987.

[Добронец Б. С. и Рощина Е. Л. (2002)] Добронец Б. С., Рощина Е. Л. Приложения интервального анализа чувствительности // *Вычислительные Технологии*. 2002. Т. 7, № 1. С. 75–82.

[Добронец Б. С. и Шайдуров В. В. (1990)] Добронец Б. С., Шайдуров В. В. Двусторонние численные методы. Новосибирск: Наука, 1990.

[Дугарова И. В. и Смагина Е. М. (1990)] Дугарова И. В., Смагина Е. М. Обеспечение устойчивости системы с неопределенными параметрами // *Автоматика и Телемеханика*. 1990. № 11. С. 176–181.

[Вощинин А. П. (2002)] Вощинин А. П. Интервальный анализ данных: развитие и перспективы // *Заводская лаборатория*. 2002. № 1. С. 118–126.

[Ивлев Р. С. (2003)] Ивлев Р. С. Асимптотическая устойчивость и положительная определённость интервальной матрицы со связями // *Вычислительные Технологии*. 2003. Т. 8. № 5. С. 63–77.

[Калмыков С. А. и др. (1986)] Калмыков С. А., Шокин Ю. И., Юлдашев З. Х. Методы интервального анализа. Новосибирск: Наука, 1986.

[Канторович Л. В. (1962)] Канторович Л. В. О некоторых новых подходах к вычислительным методам и обработке наблюдений // *Сибирский Математический Журнал*, 1962. Т. 3. № 5. С. 701–709.

[Клатте Р. и др. (2000)] Клатте Р., Кулиш У., Неага М., Рац Д., Ульрих Х. PASCAL-XSC — язык численного программирования. М.: ДМК-пресс, 2000.

[Корноушенко Е. К. (1980)] Корноушенко Е. К. Интервальные покоординатные оценки для множества достижимых состояний линейной стационарной системы // *Автоматика и Телемеханика*. 1980–1983: I. 1980. № 5, С. 12–22; II. 1980. № 12, С. 10–17; III. 1982. № 10, С. 47–52; IV. 1983. № 3, С. 81–87.

[Костоусова Е. К. (1997)] Костоусова Е. К. О полиэдральном оценивании областей достижимости линейных многошаговых систем // *Автоматика и Телемеханика*. 1997. № 3. С. 57–68.

[Костоусова Е. К. (1998)] Костоусова Е. К. Внешнее и внутреннее оценивание областей достижимости при помощи параллелотопов // *Вычислительные Технологии*. 1998. Т. 3. № 2. С. 11–20.

[Костоусова Е. К. и Куржанский А. Б. (1997)] Костоусова Е. К., Куржанский А. Б. Гарантированные оценки точности вычислений в задачах управления и оценивания // *Вычислительные Технологии*. 1997. Т. 2. № 1. С. 19–27.

[Куржанский А. Б. (1991)] Куржанский А. Б. Задача идентификации — теория гарантированных оценок // *Автоматика и Телемеханика*. 1991. № 4. С. 3–26.

[Куржанский А. Б. и Фурасов В. Д. (1999)] Куржанский А. Б., Фурасов В. Д. Идентификация нелинейных процессов — гарантированные оценки // *Автоматика и Телемеханика*. 1999. № 6. С. 70–87.

[Куржанский (1977)] Куржанский А. Б. Управление и наблюдение в условиях неопределенности — М.: Наука, 1977.

[Лакеев А. В. и Носков С. И. (1994)] Лакеев А. В., Носков С. И. О множестве решений линейного уравнения с интервально заданными оператором и правой частью // *Сибирский Математический Журнал*. 1994. Т. 35. № 5. С. 1074–1084.

[Назаренко Т. И. и Марченко Л. В. (1982)] Назаренко Т. И., Марченко Л. В. Введение в интервальные методы вычислительной математики. Иркутск: Издательство Иркутского университета, 1982.

[Нариньяни А. С. (1986)] Нариньяни А. С. Недоопределенность в системах представления и обработки знаний // *Известия АН СССР. Техническая Кибернетика*. 1986. № 5. С. 3–28.

[Поляк Б. Т. и Щербаков П. С. (2002)] Поляк Б. Т., Щербаков П. С. Робастная устойчивость и управление. М.: Наука, 2002.

[Пылаев Н. К. и Ядыкин И. Б. (1989)] Пылаев Н. К., Ядыкин И. Б. Интервальные алгоритмы адаптивного управления с неявной эталонной моделью // *Автоматика и Телемеханика*. 1989. № 6. С. 63–72.

[Хлебалин Н. А. и Шокин Ю. И. (1991)] Хлебалин Н. А. и Шокин Ю. И. Интервальный вариант метода модального управления // *Доклады Академии Наук*. 1991. Т. 316. № 4. С. 846–850.

[Шарый С. П. (1997)] Шарый С. П. Алгебраический подход к анализу линейных статических систем с интервальной неопределенностью // *Известия РАН. Теория и системы управления*. 1997. № 3. С. 51–61.

[Шарый С. П. (1998)] Шарый С. П. Алгебраический подход во «внешней задаче» для интервальных линейных систем // *Вычислительные Технологии*. 1998. Т. 3. № 2. С. 67–114.

[Шарый С. П. (2002)] Шарый С. П. Оптимальное внешнее оценивание множеств решений интервальных систем уравнений. Часть 1 // *Вычислительные технологии*. 2002. Т. 7. № 6. С. 90–113.

[Шарый С. П. (2003)] Шарый С. П. Оптимальное внешнее оценивание множеств решений интервальных систем уравнений. Часть 2 // *Вычислительные Технологии*. 2003. Т. 8. № 1. С. 84–110.

[Шокин Ю.И. (1981)] Шокин Ю. И. Интервальный анализ. Новосибирск: Наука, 1981.

[Ackermann (2002)] Ackermann J. Robust control: the parametric space approach. London, Springer, 2002.

[Bhattacharyya *et. al.* (1995)] Bhattacharyya S. P., Chapellat H., Keel L. H. Robust control: the parametric approach. Upper Saddle River, NJ, Prentice Hall, 1995.

[Cope *et. al.* (1979)] Cope J.E., Rust B. W. Bounds on solutions of linear systems with inaccurate data // SIAM J. Numer. Anal. 1979. **16**. P. 950–963.

[Durieu *et. al.* (1995)] Durieu C., Walter E., Polyak B. T. Multi-input multi-output ellipsoidal state bounding // J.Opt.Theory and Appl. 2001. **111**. P. 273–303.

[Higham (2001)] Higham N. J. Accuracy and stability of numerical algorithms. Philadelphia, SIAM, 1996.

[Neumaier (2002)] Neumaier A. Grand challenges and scientific standards in interval analysis // Reliable Computing. 2002. **8**. P. 313–320.

[Polyak и Nazin (2004)] Polyak B. T., Nazin S. A. Interval solutions for interval algebraic equations // Math. and Comp. in Simulation. 2004. **66**. P. 207–217.

[Polyak *et. al.* (2004)] Polyak B. T., Nazin S. A., Durieu C., Walter E. Ellipsoidal parameter or state estimation under model uncertainty // Automatica. 2004. **40**. P. 1171–1179.

[Rohn (1989)] Rohn J. Systems of linear interval equations // Linear Algebra and Its Appl. 1989. **129**. P. 39–78.

[Schichl и Neumaier (2004)] Schichl H., Neumaier A. Exlusion regions for systems of equations // SIAM J. Numer. Anal. 2004. **42**. P. 383–408.

[Shary (2001)] Shary S. P. A surprising approach in interval global optimization // Reliable Computing. 2001. **7**, No. 6. P. 497–505.

[Shary (1995)] Shary S. P. Outer estimation of generalized solution sets to interval linear systems // SIAM J. Numer. Anal. 1995. **32**. P. 610–630.

[Shweppe (1973)] Shweppe F. C. Uncertain dynamic systems. Englewood Cliffs, NJ, Prentice-Hall, 1973.

Предметный указатель

Люк Жолен, Мишель Кифер, Оливье Дидри, Эрик Вальтер

Прикладной интервальный анализ

Дизайнер М. В. Ботя
Технический редактор А. В. Широбоков
Компьютерная верстка Д. П. Вакуленко
Корректор А. В. Соколова

Подписано в печать 25.02.2005. Формат $60 \times 84^{1}/_{16}$.
Печать офсетная. Усл. печ. л. 27,20. Уч. изд. л. 27,57.
Гарнитура Таймс. Бумага офсетная №1. Заказ №8.
АНО «Институт компьютерных исследований»
426034, г. Ижевск, ул. Университетская, 1.
http://rcd.ru E-mail: borisov@rcd.ru Тел./факс: (+73412) 500–295