

Глава 1

Введение

Данная книга может служить одновременно учебным пособием и справочным руководством по численным методам с автоматической верификацией результатов. Представленные здесь методы удобны в практическом отношении, надежны и изящны. Мы излагаем теоретические основы, описываем алгоритмы и программную реализацию методов, предназначенных для надежного решения ряда основных задач численного анализа. Кроме того, эта книга поможет читателю научиться создавать такие методы самостоятельно и перейти к решению задач, оставшихся за ее пределами.

Мы указываем на потенциальные вычислительные трудности при использовании стандартных подпрограмм, но не утверждаем, что наши собственные советы непогрешимы. Мы просто делимся своим практическим опытом решения ряда базовых численных задач с автоматической верификацией результатов. Читатель сможет составить свое мнение о надежности предлагаемых процедур на основании собственного опыта.

Мы предполагаем, что читатель может понимать компьютерные программы на языке Паскаль или его расширенном варианте PASCAL-XSC (PASCAL eXtension for Scientific Computation – расширение языка Паскаль для научных вычислений). Именно на PASCAL-XSC написаны все программы в настоящей книге. Обширный набор новых средств для выполнения научных и инженерных вычислений, реализованный в языке PASCAL-XSC, делает его особенно удобным для программирования численных методов с автоматической верификацией результатов (подробнее о языке PASCAL-XSC рассказывается в гл. 2).

В настоящем Введении разъясняются методологические предпосылки, лежащие в основе выбранного нами подхода, говорится о структуре всей книги, описывается система соглашений, принятых в основном тексте и при записи алгоритмов, сообщается о принципах компьютерной реализации представленных в книге численных методов с верификацией результатов. В заключительном разделе обсуждается, чем вызвана необходимость в подобных методах.

1.1 Краткий обзор

Книга делится на три части. Собранные в первой из них предварительные сведения включают обзор дополнительных, по сравнению со стандартным Паскалем, возможностей, включенных в язык программирования PASCAL-XSC, а также рассказ об особенностях и ценных свойствах интервальной арифметики. Читатели, не знакомые с интервальным анализом и методами численной верификации в целом, должны внимательно прочитать разделы, в которых содержится введение в интервальную арифметику и обсуждается, как эта усовершенствованная арифметика повышает надежность результатов вычислений.

Во второй и третьей частях рассмотрены основные одномерные и многомерные задачи. Читателям, знакомящимся с методами автоматической верификации впервые, следует обратить особое внимание на следующие главы:

- вычисление полиномов (гл. 4),
- автоматическое дифференцирование (гл. 5),
- нелинейные уравнения с одним неизвестным (гл. 6),
- системы линейных уравнений (гл. 10).

В других главах, например посвященных оптимизации, требуется определенное знакомство с концепцией автоматической верификации и методами интервальных вычислений. Представленные в этих главах методы включают в себя использование основных алгоритмов из глав 4–6 и 10 (например, алгоритмов для решения систем линейных уравнений).

1.2 Структура книги

Чтобы облегчить эффективное использование книги, разные главы следуют определенному общему стандарту. Это поможет читателю

найти необходимую информацию по любой теме (например, теоретические сведения или описание реализации алгоритма), не читая соответствующую главу целиком.

Каждая глава начинается с введения, в котором задача описывается с математической точки зрения, обсуждаются ее вычислительные аспекты и обосновывается стратегия решения. Теория, необходимая для решения задачи и верификации результатов вычислений, излагается в разделе «Теоретические основы». Под заголовком «Алгоритмическое описание» читатель найдет краткое, ясное и исчерпывающее формальное описание алгоритма.

Очень важный раздел каждой главы озаглавлен «Реализация и примеры». Здесь обсуждается практическая программная реализация численных методов, составляющих основную тему главы. Все процедуры реализованы в виде самостоятельных «инструментов», которые можно использовать в различных сочетаниях для решения конкретных вычислительных задач, интересующих читателя. Модульная организация процедур позволяет легко встраивать их в имеющиеся прикладные программы пользователя (например, для того чтобы обойти численную «ловушку», обнаруженную внутри сложного алгоритма решения конкретной задачи). Все программы, представленные в этой книге, находятся на ftp-сервере (см. <ftp://ftp.iam.uni-karlsruhe.de/pub>), что позволяет незамедлительно начать использовать их для тестирования или встраивания в уже существующие прикладные программы. Применение каждого алгоритма иллюстрируется учебными примерами. Мы демонстрируем превосходство методов вычислений с автоматической верификацией и использования строго определенной машинной арифметики над большинством стандартных подходов к решению численных задач.

Выполняя включенные в большинство глав упражнения, читатель может разобраться в особенностях поведения предлагаемых нами алгоритмов и способах их усовершенствования. В разделе «Основная и дополнительная литература», помещаемом в конце каждой главы, содержатся указания на аналогичные и/или более сложные и специализированные алгоритмы. В Приложение вынесены утилиты, используемые различными модулями.

1.3 Шрифтовые выделения

Для выделения некоторых терминов, имен и фрагментов текста мы применяем следующие шрифты:

- курсив** для выделения некоторых слов в основном тексте;
- полужирный** для зарезервированных слов языка PASCAL-XSC (например **repeat** или **return**), используемых при записи алгоритмов в псевдокоде или упоминаемых в основном тексте;
- наклонный** для предопределенных идентификаторов языка программирования PASCAL-XSC (например *integer* или *real*), имен процедур, функций и операций в алгоритмах и других идентификаторов, упоминаемых в основном тексте;
- машинописный** для передачи листингов и выдач программ (с некоторым дополнительным форматированием для повышения читабельности — см. разд. 1.5).

Отсылки на литературу всегда оформляются в виде [N], где N соответствует библиографической ссылке в списке литературы, помещенном в конце книги.

1.4 Запись алгоритмов

Алгоритмы, представленные во второй и третьей частях книги, записаны на псевдокоде, который напоминает язык Паскаль с добавлением некоторых математических обозначений (в том числе вводимых в гл. 3). Следующий алгоритм служит простым примером принятой нами нотации.

Алгоритм 1.1: *DoSomething*(x, y, z, Err)

- ```

1) $a := x + y; \quad b := x - y$ {Инициализация}
2) {Производим вычисления}
 if ($a > 0$) and ($b > 1$) then
 $c := \sqrt{a}; \quad d := \ln(a);$
 $a := -c; \quad b := -d;$
 else if ($b = 1$) then
 return $Err :=$ «Произошла ошибка 1!»;
 else
 return $Err :=$ «Произошла другая ошибка!»;
3) return $z := a \cdot b \cdot c/d;$

```

Для большей ясности отдельные этапы алгоритмов нумеруются, а операторы завершаются точкой с запятой. Отступы и нумерация используются для того, чтобы по мере возможности избегать

операторных скобок **begin–end**. Внутри отдельных этапов могут использоваться подалгоритмы, оформленные как процедуры или функции:

**ИмяПроцедуры** (СписокАргументов)

Переменная := **ИмяФункции** (СписокАргументов)

Операторы

**return** или **return** СписокАргументов

вызывают немедленный возврат из выполняемого алгоритма, причем во втором случае возвращаются новые значения аргументов из указанного списка. Циклы записываются одним из следующих трех способов:

**for** Индекс := НачЗначение **to** КонЗначение **do** Оператор(ы)

**repeat** Оператор(ы) **until** ЛогическоеВыражение

**while** ЛогическоеВыражение **do** Оператор(ы)

Операторы

**exit**<sub>Индекс-цикл</sub>, **exit**<sub>while-цикл</sub>, **exit**<sub>repeat-цикл</sub>

прекращают выполняемый цикл и передают управление на оператор, следующий за ним. Оператор

**next**<sub>Индекс</sub>

завершает текущую итерацию, передавая управление на конец цикла, соответствующего указанной индексной переменной. После этого, в зависимости от величины приращения и условия выхода, может быть начато выполнение следующей итерации цикла. Если индексированные выражения входят в состав включенных в псевдокод математических конструкций, то цикл может обозначаться последовательностью значений индекса, заключенной в скобки: ( $k = 1, \dots, n$ ).

Используется условный оператор

**if** ЛогическоеВыражение **then** Оператор(ы) **else** Оператор(ы),

причем **else**-ветвь может отсутствовать.

Все комментарии заключаются в фигурные скобки.

## 1.5 Реализация

По твердому убеждению авторов, читатель должен иметь гарантию того, что все программы, модули и фрагменты кода в этой книге будут компилироваться и выполняться именно так, как предполагается в тексте. Поэтому мы откомпилировали и выполнили каждый модуль, пример программы и фрагмент кода (фрагменты включались в состав какой-либо полноценной исполняемой программы). Аналогично, приводимые результаты были действительно получены при выполнении данных примеров программ, причем и ввод с клавиатуры, и выдача программы показаны в точности так, как они отображались на экране компьютера. Как код, так и результаты выполнения программ были включены непосредственно в рукопись книги с помощью форматизирующей программы, так что читателю доступен в точности тот код, который выполнялся у нас. Модули и программы включаются в текст в следующем виде:

```
{ Краткое описание экспортируемых подпрограмм... }
MODULE dosome;

CONST
 NoError = 0;
 ErrorOne = 1;
 AnotherError = 2;

GLOBAL FUNCTION DoSomethingErrMsg(ErrCode: integer) : string;
VAR
 Msg : string;
BEGIN
 CASE ErrCode OF
 NoError : Msg := '';
 ErrorOne : Msg := 'Произошла ошибка 1';
 AnotherError : Msg := 'Произошла другая ошибка';
 ELSE : Msg := 'Код ошибки не определен';
 END;
 IF (ErrCode <> NoError) THEN Msg := 'Ошибка: ' + Msg + '!';
 DoSomethingErrMsg := Msg;
END;

GLOBAL PROCEDURE
 DoSomething(x, y: real, VAR z: real; VAR Err: integer);
VAR
 a, b, c, d : real;
BEGIN
 a := x + y; b := x - y; Err := NoError; { Инициализация }
```

```
IF (a > 0) AND (b > 1) THEN { Производим вычисления }
 BEGIN
 c := sqrt(a); d := ln(b); a := -c; b := d;
 END;

ELSE IF (b = 1) THEN
 Err := ErrorOne

ELSE
 Err := AnotherError;

IF (Err = NoError) THEN z := a * b * c / d;
 { Иначе значение z остается неопределенным }
END;

{ Раздел инициализации модуля }
BEGIN
 { Инициализация не требуется }
END.
```

Функция *DoSomethingErrMsg* обрабатывает все исключительные ситуации. Она связывает диагностическое сообщение с кодом исключительной ситуации, возвращаемым подпрограммой *DoSomething*.

## 1.6 Вычислительное окружение

Мы стремились обеспечить максимально широкую применимость программ из этой книги не только с точки зрения общности решаемых ими задач, но и в смысле их переносимости между совершенно разными платформами. Это означает, что все программы должны выполняться как на персональных компьютерах и рабочих станциях, так и на машинах класса mainframe и суперкомпьютерах. В качестве средства реализации наших алгоритмов был выбран язык PASCAL-XSC [44, 45], поскольку он разрабатывался в расчете именно на такую степень переносимости. Система программирования на PASCAL-XSC доступна на ftp-сервере Института прикладной математики Университета г. Карлсруэ.<sup>1</sup>

Все программы в этой книге выполнялись и тестировались с использованием компилятора PASCAL-XSC на персональных компьютерах типа IBM PC и рабочих станциях фирм Sun и Hewlett-Packard. Все найденные ошибки были устранены, но мы

---

<sup>1</sup>В настоящее время эта система может быть также бесплатно загружена с Интернет-сайта, расположенного по адресу [www.xsc.de](http://www.xsc.de).— Прим. перев.

не утверждаем, что предлагаемые программы полностью свободны от ошибок или пригодны для любого конкретного приложения без дополнительной модификации. Тем не менее следует сказать, что эта книга — плод десятилетнего опыта разработок в области научных и инженерных вычислений с верификацией результатов, и все опубликованные в ней программы писались и тестировались весьма тщательно. В то же время мы приветствуем любые предложения, направленные на совершенствование текста книги и приведенных в ней программ.

## **1.7 Зачем нужна верификация результатов вычислений**

Общепринятым средством для выполнения научных и инженерных вычислений на компьютерах является арифметика чисел с плавающей точкой. На большинстве компьютеров каждая отдельная операция с плавающей точкой обеспечивает результат максимальной точности в том смысле, что получаемый округленный результат отличается от точного вещественного значения не более, чем на единицу последнего разряда мантиссы. Однако результат двух или нескольких последовательных операций может не содержать уже ни одной верной цифры. Современные компьютеры выполняют в секунду до  $7 \cdot 10^{12}$  операций с плавающей точкой, и поэтому достоверность вычисляемых результатов становится предметом особого внимания.

В последние годы для многих задач и приложений численного анализа были разработаны методы решения, дающие возможность автоматически проверять корректность результатов вычислений. Более того, нередко такие методы позволяют использовать компьютер для автоматического доказательства существования и единственности решения численной задачи. Например, верифицированное решение системы обыкновенных дифференциальных уравнений является не менее надежным, чем аналитическое решение, полученное с помощью средств компьютерной алгебры; к тому же численные значения последнего сами еще должны быть корректно найдены. Кроме того, численная подпрограмма может использоваться и тогда, когда задача не имеет аналитического решения.

В этом разделе мы кратко остановимся на истории, математическом контексте и целесообразности применения методов, представленных в книге.

### 1.7.1 Краткая история методов вычислений

На протяжении всей истории математики существовавшие методы вычислений оставляли свой отпечаток на этой науке. Прогресс в области работы с числами означал прогресс и в самой математике. Механические счетные машины Паскаля, Лейбница и других изобретателей, таблицы логарифмов, счетная логарифмическая линейка и прочие механические и аналоговые вычислительные устройства стали важными вехами в истории научных и инженерных вычислений.

Большой скачок в развитии вычислительной техники произошел в середине XX века, когда были построены первые электронные компьютеры. Новая технология сразу же позволила выполнять арифметические операции примерно в 1000 раз быстрее, чем при помощи существовавших до того механических или электромеханических устройств. Великие технические достижения XX века были бы невозможны без современных вычислительных средств. Сегодняшние автомобили, авиация, космические полеты, современные радио и телевидение и, не в последнюю очередь, дальнейшее быстрое развитие самой компьютерной техники стали возможными благодаря огромной вычислительной мощи новой технологии. С другой стороны, прогресс аппаратного обеспечения дал сильный импульс дальнейшему развитию алгоритмов и вычислительной математики.

Создание компьютеров обеспечило и дальнейшее развитие самой схемотехники, повысившее вычислительную мощность процессоров еще приблизительно в  $10^9$  раз по сравнению с первыми ламповыми вычислительными машинами. Сравнивая числа  $10^3$  и  $10^9$ , легко понять, что настоящая компьютерная революция произошла после создания электронных компьютеров. Достоинно удивления, что ничего подобного не случилось в арифметико-математической области. С учетом колоссального технического прогресса компьютеров попытка снабдить их более совершенными арифметическими средствами напрашивалась сама собой. Тем не менее математики практически не оказали здесь никакого влияния. В интересующей нас области научных и инженерных расчетов компьютеры и сейчас используются в большинстве случаев примерно так же, как это было в середине 50-х годов прошлого века. По-прежнему, хотя и гораздо быстрее, чем раньше, выполняются те же четыре операции сложения, вычитания, умножения и деления с плавающей точкой.

С развитием вычислительной техники всё большие объемы вычислений становилось возможным выполнять за всё меньшее время. Поэтому постоянно возрастали масштабы решаемых численных задач, а вместе с ними и требования к надежности результатов вычислений. Стандартная арифметика с плавающей точкой уже не удовлетворяла этим новым требованиям. Поиск возможностей, направленных на повышение надежности вычислений, положил начало развитию обширной области исследований (см., например, большие библиографии в [28, 64]). Однако методы надежных вычислений все еще ожидают своего широкого признания.

### 1.7.2 Арифметические вычисления на компьютере

Обычно аппаратное обеспечение компьютеров поддерживает две числовые системы: целые числа и числа с плавающей точкой. Целочисленная арифметика оперирует конечным подмножеством множества целых чисел. Если аппаратная часть исправна, а программы не содержат ошибок, то целочисленная арифметика и программные надстройки над ней работают без погрешностей. Например, на основе целочисленной арифметики можно построить арифметику систематических дробей произвольной разрядности и обыкновенных дробей. В идеале компьютер получает возможность сохранять столько разрядов, сколько необходимо для точного представления результатов операций, что является наиболее предпочтительным способом вычислений в теории чисел и компьютерной алгебре. Кроме того, целочисленная арифметика позволяет безошибочно осуществлять компиляцию и другие формы трансляции, а также реализовать алгоритмы поиска и сортировки.

С другой стороны, в численном анализе и естественных науках операции производятся над вещественными числами. Произвольное вещественное число представляется бесконечной систематической (например десятичной или двоичной) дробью. На практике в научных и инженерных вычислениях вещественные числа приходится представлять в компьютере конечными дробями, чаще всего числами с плавающей точкой. Арифметика чисел с плавающей точкой поддерживается аппаратным обеспечением компьютеров и поэтому выполняется очень быстро, но каждая операция с плавающей точкой может вносить погрешность. Хотя, как мы уже сказали, на современных компьютерах всякая отдельная операция с плавающей точкой выполняется с максимально возможной точностью, результат нескольких последовательных операций может оказаться совершенно неверным. Приведем два простых примера.

**Пример 1.1.** Пусть даны два вещественных вектора  $x$  и  $y$ :

$$x = (10^{20}, 1223, 10^{18}, 10^{15}, 3, -10^{12}), \quad y = (10^{20}, 2, -10^{22}, 10^{13}, 2111, 10^{16}).$$

Обозначим скалярное произведение  $x$  и  $y$  через  $x \cdot y$  (соответственные элементы перемножаются и эти произведения складываются). В точной целочисленной арифметике имеем:

$$x \cdot y = 10^{40} + 2446 - 10^{40} + 10^{28} + 6333 - 10^{28} = 8779.$$

Однако арифметика чисел с плавающей точкой на любом современном компьютере (включая те, на которых арифметика реализована в соответствии со стандартом IEEE<sup>1</sup>) даст для такого скалярного произведения нулевое значение. Причина этого в столь большой разнице порядков слагаемых, что обычное представление чисел с плавающей точкой не позволяет корректно выполнить вычисление. Эта катастрофическая погрешность возникает несмотря на то, что данные (элементы векторов) используют менее 5% диапазона значений порядка, доступного на большинстве компьютеров!

**Пример 1.2.** Рассмотрим систему чисел с плавающей точкой по основанию 10 с мантиссой длиной в 5 разрядов. Мы хотим вычислить разность двух чисел  $x$  и  $y$ , которые к моменту вычисления имели значения  $x = 0.10005 \cdot 10^5$  и  $y = 0.9973 \cdot 10^4$ . На сей раз порядок величины обоих операндов одинаков, и компьютер дает абсолютно правильный результат:  $x - y = 0.77000 \cdot 10^1$ . Но допустим теперь, что каждое из чисел  $x$  и  $y$  получено на предыдущем шаге в результате умножения. Поскольку мы работаем с арифметикой длиной 5 разрядов, неокругленные произведения чисел, конечно, должны иметь длину 10 разрядов. Предположим, что эти произведения в нашем случае таковы:

$$x_1 \cdot x_2 = 0.1000548241 \cdot 10^5, \quad y_1 \cdot y_2 = 0.9997342213 \cdot 10^4.$$

Вычитая второе число из первого, нормализуя результат и округляя его до 5 цифр, получим число  $0.81402 \cdot 10^1$ , в котором нет ни одной общей цифры с результатом, вычисленным в арифметике с плавающей точкой. Во втором случае значение выражения  $x_1 \cdot x_2 - y_1 \cdot y_2$  было вычислено точно и затем округлено до ближайшего числа с плавающей точкой и мантиссой длиной 5 разрядов. Напротив, стандартная арифметика с плавающей точкой, в которой округление

<sup>1</sup>Краткую информацию о стандартах IEEE на арифметику чисел с плавающей точкой можно найти в настоящей книге в Предисловии к русскому изданию. — *Прим. ред.*

производится после каждой отдельной операции, дала результат, не имеющий ни одной верной цифры.

Тем, кто считает, что в их собственных расчетах «ничего подобного не происходит», следует быть очень, очень осторожными. Крайне трудно проследить за всеми данными, появляющимися на промежуточных стадиях при прогоне задачи в течение нескольких часов на рабочей станции или суперкомпьютере. Многие их пользователи похожи на того резчика, который продолжает орудовать тупым ножом, хотя нож так легко заточить!

В классическом анализе погрешностей принято оценивать погрешности, возникающие при каждой отдельной операции численных алгоритмов. Конечно, для такого алгоритма, в котором за час выполняется  $10^{15}$  операций с плавающей точкой, это уже практически неосуществимо. Поэтому реалистичный анализ погрешностей обычно и не производится. В действительности, сам факт, что вычисленный результат может вообще не иметь верных цифр, редко принимается во внимание.

Проблема математической корректности вычислений становится в настоящее время центральной. Она весьма значима для многих приложений, например таких, как имитационное или математическое моделирование. Точность и достоверность требуются для того, чтобы погрешности вычислений не искажали истинные свойства модели. Математическая модель будет адекватной и работоспособной лишь в том случае, если погрешности вычислений контролируются.

### **1.7.3 Расширение стандартной арифметики чисел с плавающей точкой**

В научных и инженерных вычислениях прошлого использовались логарифмические линейки или таблицы логарифмов и стандартных функций. Применение логарифмов, как и счетной линейки, позволяет не следить за выравниванием порядков, но в любом случае результат имеет ограниченную точность. Аппаратную реализацию математических операций в современных процессорах можно рассматривать как «автоматические таблицы логарифмов» (в том числе сумм и разностей) с большим, но ограниченным числом знаков. Эти процессоры, однако, работают столь быстро, что проследить за кумулятивным эффектом пренебрежения малыми величинами и погрешностей потери значащих разрядов при вычитании близких чисел невозможно.

В прошлом научные и инженерные вычисления характеризовались интенсивным использованием операций умножения и деления. Напротив, счетоводы и бухгалтеры имели дело со сложением и вычитанием длинных рядов чисел различной величины. Погрешности в этих вычислениях не допускались, и потому были изобретены различные методы проверки (например, двойная бухгалтерия), которые позволяли убедиться, что ответ правилен с точностью до копейки. В современных вычислениях потенциально необходимо осуществлять миллионы сложений и вычитаний; ясно, что для обеспечения точности таких расчетов «молниеносная» скорость, унаследованная арифметикой с плавающей точкой от счетной линейки, должна быть дополнена дотошностью опытного счетовода. При современном состоянии техники для этого вполне достаточно одной микросхемы.

Компьютеры были изобретены для того, чтобы освободить людей от трудоемкой работы. Очевидный разрыв между производительностью компьютеров и возможностями контроля погрешностей наводит на мысль переложить сам процесс оценки погрешностей на компьютер. К настоящему времени это уже успешно выполнено для всех основных задач численного анализа и многих приложений. Чтобы достичь такого результата, компьютерную арифметику надо было сделать более «интеллектуальной», чем обычно. Будем исходить из следующего наблюдения: большая часть погрешностей в арифметике с плавающей точкой возникает при осуществлении последовательности сложений и вычитаний. С другой стороны, умножение и деление в этой арифметике относительно устойчивы. В арифметике чисел с фиксированной точкой, наоборот, без погрешностей выполняются аддитивные операции. Поэтому в идеале нужно лишь создать в арифметическом устройстве регистр с фиксированной точкой, покрывающий своей разрядностью весь диапазон чисел с плавающей точкой. Если на аппаратном уровне такого регистра не существует, то его можно моделировать с помощью программных средств. Возрастание надежности во многих случаях оказывается важнее происходящей при этом потери скорости.

Если такой регистр сделать еще вдвое длиннее, то скалярные произведения любой конечной размерности можно будет вычислять точно. Действительно, для точного представления отдельных слагаемых скалярного произведения требуется удвоить как длину мантиссы, так и диапазон возможных значений порядка. Надо заметить, что стандарт IEEE на машинную арифметику с плавающей

точкой допускает подобное изменение формата. Реализация даже такого относительно длинного регистра с фиксированной точкой не представляет серьезной проблемы для современной технологии создания программно-аппаратных компьютерных комплексов. Достижимая же возможность точного вычисления скалярного произведения векторов с плавающей точкой любой конечной размерности открывает перед численным анализом новые перспективы. В частности, такое *оптимальное (по точности) скалярное произведение* само по себе оказывается важным средством повышения общей точности вычислений.

При описанной реализации скалярного произведения все операции с плавающей точкой, операции над машинными комплексными числами, и, в частности, операции над машинно-представимыми векторами и матрицами с вещественными или комплексными элементами могут производиться *с максимальной точностью*. Это означает, что для каждого элемента вектора или матрицы после выполнения всех операций в регистре с фиксированной точкой осуществляется только одно округление, и затем вычисленное значение записывается в память в соответствующем формате с плавающей точкой. До разработки концепции длинного регистра вычисление с максимальной точностью произведения двух матриц, состоящих из чисел с плавающей точкой, считалось более сложной задачей, чем вычисление собственных значений симметричной матрицы! С нашей точки зрения, оптимальное скалярное произведение является базовой арифметической операцией. Кроме того, на оптимальном скалярном произведении может быть основана арифметика повышенной разрядности с плавающей точкой.

Чтобы обеспечить возможность автоматического контроля погрешностей, машинную арифметику надо дополнить еще одним элементом. Все операции над числами с плавающей точкой (сложение, вычитание, умножение, деление и скалярное произведение векторов, составленных из чисел с плавающей точкой) надо обеспечить возможностью направленного округления результата, т. е. округления до ближайшего машинного числа с недостатком или избытком. Такие операции над машинно-представимыми вещественными и комплексными числами, а также составленными из них векторами и матрицами позволяют построить машинную интервальную арифметику.

Интервалы представляют в компьютере континуальные объекты и открывают для численного анализа совершенно новую перспективу. В памяти компьютера интервал записывается парой чисел

с плавающей точкой. Он фиксирует весь континуум вещественных чисел, заключенных между этими двумя машинными числами. Арифметические операции над интервалами выражаются через операции над парами чисел, задающими их границы. Округление при вычислении нижней границы интервала-результата осуществляется с недостатком, а при вычислении верхней границы — с избытком. В итоге получается интервал, который гарантированно содержит все результаты применения данной арифметической операции к любым парам чисел, взятых из первого и второго интервалов-операндов. Интервальное вычисление арифметического выражения, такого как полином, требует примерно вдвое большего числа операций по сравнению с вычислением такого же выражения в стандартной арифметике с плавающей точкой. Результатом интервального вычисления является числовой интервал, гарантированно содержащий множество значений этого выражения на заданном интервале значений входящих в него переменных.

#### 1.7.4 Научные и инженерные вычисления с автоматической верификацией результатов

В совокупности интервальная арифметика и оптимальное скалярное произведение позволяют проводить научные и инженерные вычисления с автоматической верификацией результатов. Проиллюстрируем это утверждение на двух простых примерах.

**Пример 1.3.** Пусть необходимо выяснить, обращается ли в нуль на заданном интервале  $[x]$  вещественная функция, определенная некоторым арифметическим выражением. Используя стандартную арифметику с плавающей точкой, математически строго ответить на этот вопрос затруднительно. В самом деле, данную функцию можно вычислить, скажем, в 1000 разных точек интервала  $[x]$ . Если все вычисленные значения окажутся положительными, то с большой вероятностью можно будет сделать вывод об отсутствии на интервале  $[x]$  нулей данной функции. Однако этот вывод, конечно, не является абсолютно надежным. Погрешности округления могут привести к тому, что вычисление значения функции, в действительности отрицательного, даст положительный результат. Функция также может принимать отрицательное значение в промежутке между точками вычисления. Напротив, одна-единственная интервальная оценка множества значений этой функции может решить нашу задачу с полной математической строгостью. Если вычислен-

ный интервал не будет содержать нуля, то множество значений функции тем более не может содержать нуля, так как оно является подмножеством этого интервала. Поэтому с достоверностью можно будет сделать вывод о том, что на интервале  $[x]$  наша функция в нуль не обращается. Если окажется, что вычисленный интервал все же содержит нуль, это может случиться либо потому, что функция действительно имеет корень, либо из-за переоценки диапазона ее значений. Тем не менее во многих случаях вычисление единственной интервальной оценки (требующее примерно столько же операций с плавающей точкой, сколько и обычное вычисление в двух точках) дает информацию, которую не могут гарантировать обычные вычисления в сотнях, тысячах и миллионах точек (см. рис. 1.1).

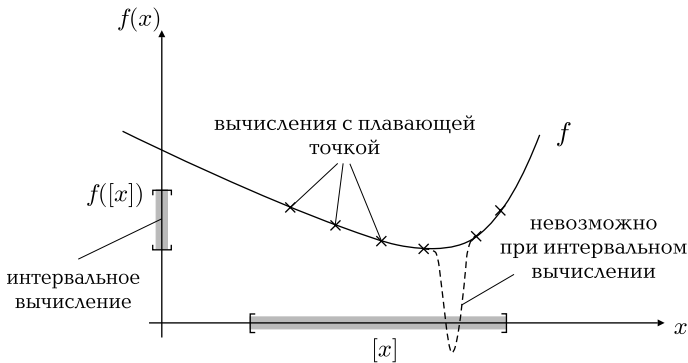


Рис. 1.1: Одного интервального вычисления достаточно, чтобы гарантировать отсутствие отрицательных значений функции  $f$  (как на пунктирном участке графика)

**Пример 1.4.** Еще одним примером может служить метод, позволяющий верифицировать правильность результата при решении системы линейных уравнений  $A \cdot x = b$ . Сначала вычислим приближенное решение (например, методом исключения Гаусса). Преобразуем полученный вектор в интервальный, добавляя к каждому его элементу небольшое число  $\pm\epsilon$ , что даст многомерный куб  $[x]$  со стороной  $2\epsilon$  и центром в вычисленной точке. Этот куб вдоль каждого координатного направления задает свой числовой интервал. Теперь приведем систему уравнений к рекуррентной форме:  $x = B \cdot x + c$ . Например, можно воспользоваться тем, что предполагаемое приближенное решение уже вычислено, и получить  $B$

и  $c$  из уравнения для невязки, соответствующего исходной системе  $A \cdot x = b$  (см. гл. 10). Теперь положим  $[y] := B \cdot [x] + c$ . Если  $[y]$  содержится в  $[x]$  (для проверки чего достаточно просто вычислить крайние точки этих интервалов), то по теореме Брауэра о неподвижной точке (см. теорему 3.1) интервал  $[y]$  содержит решение уравнения  $x = B \cdot x + c$ . Если, сверх того, границы интервальных векторов  $[x]$  и  $[y]$  не имеют общих точек, то это решение будет единственным. Иначе говоря, в данном случае обратимость матрицы исходной системы уравнений  $A$  может быть доказана автоматически и притом с полной строгостью. После этого решение может быть найдено с любой необходимой степенью точности стандартным итерационным методом с использованием оптимального скалярного произведения. Из данного примера видно, что доказательное вычисление и получение результата с высокой точностью могут проводиться по отдельности. В нашем случае первая из этих задач сводится к доказательству, что система  $A \cdot x = b$  имеет единственное решение в  $[y]$ , а вторая — к итерированию  $[x] := B \cdot [x] + c$ , пока ширина интервала  $[x]$  не станет достаточно малой. С помощью такого метода можно получить достоверный и математически строгий результат.

Приведенные нами примеры 1.3 и 1.4 не дают ответа на вопрос, что делать, если достоверный результат вычислить не удастся. Для решения нелинейных уравнений (пример 1.3), систем линейных уравнений (пример 1.4) и многих других задач были разработаны улучшенные методы, обеспечивающие строгое соответствие вычисленных результатов постановке задачи. Большим преимуществом методов вычислений с автоматической верификацией результатов является возможность быстро убедиться в том, что произведенное вычисление не позволило получить корректный и пригодный для использования результат. В подобном случае можно применить альтернативный алгоритм или повторить вычисление с увеличенной разрядностью.

Аналогичная техника автоматической верификации результатов может применяться при решении многих других задач, таких как нахождение корней систем нелинейных уравнений и нулей комплексных полиномов, вычисление собственных значений и собственных векторов матриц, оптимизация и т. д. В частности, в данной книге рассматриваются верификационные методы компьютерного вычисления с высокой точностью арифметических выражений и функций. Подпрограммы, реализующие такие методы, могут принимать на входе неточные или изначально интервальные данные.

Возможность вычисления высокоточных включающих интервалов для значений заданной функции существенно облегчает научные и инженерные вычисления. Например, при решении систем уравнений методом Ньютона часто происходит потеря устойчивости в окрестности корня. В частности, так бывает, если обращение в нуль достигается благодаря взаимному сокращению слагаемых разных знаков. В непосредственной окрестности корня эти слагаемые весьма близки друг к другу по величине, и в арифметике с плавающей точкой начинает происходить потеря значащих разрядов. В то же время при недостаточно точном вычислении функции в методе Ньютона легко может произойти «перелет» с попаданием в область притяжения другого корня. Итерационный процесс, сходящийся к корню  $x$  в точной вещественной арифметике, может не сходиться к округленному значению  $x$  в арифметике с плавающей точкой — итерации способны «перескочить» к совершенно другому корню. Таким образом, завершающий этап верификации также требуется и для итерационных процессов.

Описанные здесь методы вычислений с автоматической верификацией результатов наряду с обычной машинной арифметикой существенно используют машинную интервальную арифметику и оптимальное скалярное произведение. При решении численных задач машинная интервальная арифметика обеспечивает получение гарантированных двусторонних оценок, а с помощью оптимального скалярного произведения достигается высокая точность вычислений. Применение этих двух средств в совокупности выводит численный анализ на качественно новый уровень.

Даже простая, непосредственная замена операций машинной арифметики на их интервальные варианты всегда обеспечивает надежные двусторонние оценки. Вычисляемые интервалы, однако, могут быть настолько широкими, что лишаются всякой практической ценности. Это явление было замечено еще в конце 60-х годов прошлого века и заставило многих специалистов по численному анализу без достаточных оснований отвергнуть эту полезную и необходимую технологию вычислений. Тем не менее даже такая интервальная арифметика успешно применяется в ряде приложений, в частности в методах глобальной оптимизации. С другой стороны, применяемая изолированно, высокая точность вычислений неоправданно расточительна. Однако ее объединение с интервальной арифметикой позволяет получать конечные результаты с *высокой и гарантированной точностью*.

Мы уже видели, что интервальная арифметика позволяет получить оценку множества значений, принимаемых данной функцией на континуальном множестве значений аргумента, в том числе и в тех точках, которые не имеют конечного представления. Поэтому интервальный анализ необходим при решении таких задач, как гарантированное нахождение всех нулей нелинейной функции или глобальная оптимизация с подтверждением правильности определения точек глобального минимума и его величины.

Другим полезным методом является автоматическое дифференцирование, использовавшееся разными исследователями при разработке методов решения целого ряда задач численного анализа с автоматической верификацией результатов. Среди таких задач можно назвать вычисление квадратур и кубатур, решение интегральных уравнений, а также дифференциальных уравнений в обыкновенных и частных производных. Эксперименты с верифицированным решением дифференциальных уравнений оказались очень интересными. Например, «решение», вычисленное традиционным методом, может описывать совершенно неправильную траекторию, если в критической точке ему не хватает всего двух верных знаков. Обычные приближенные методы не распознают такую ситуацию; в случае верификационного метода ширина включающего интервала в критической точке резко возрастает, указывая, что точность должна быть увеличена.

В последние годы для целого ряда инженерных и научных задач надежные решения были вычислены при помощи решающих процедур с автоматической верификацией результатов. Нередко удавалось решить задачи, для которых обычные подпрограммы и методы решения не работают. С 1980 года GAMM (Gesellschaft für Angewandte Mathematik und Mechanik – Общество прикладной математики и механики) и IMACS (International Association for Mathematics and Computers in Simulation – Международная ассоциация специалистов по математическим и компьютерным методам моделирования) регулярно проводят симпозиумы по теме «Компьютерная арифметика и научные вычисления с автоматической верификацией результата». Труды многих из этих симпозиумов были опубликованы [6, 37, 53, 55–57, 63, 86, 87]. Приложения охватывают разнообразные области науки и техники, такие как высокоскоростные турбины, расчет фильтров, потоки плазмы в прямоугольном канале, вибрации механизмов, высокотемпературная сверхпроводимость, проникновение загрязнений в подземные воды, индуктивные двигатели для автобусов, исследование пери-

одических решений орегонатора в химической кинетике, геометрическое моделирование, расчет размеров дренажных каналов, верифицированные квадратуры в технологии химического переноса, нелинейное управление ядерными реакторами, опровержение существования некоторых «хаотических» решений задачи трех тел в небесной механике, решение уравнения Шредингера для волновых функций, расчет магнитогидродинамических потоков при больших числах Гартмана, установление оптических свойств ячеек жидких кристаллов, численное моделирование полупроводниковых диодов, оптимизация СБИС, геометрические методы для САПР, разрушение балок и робототехника. Документацию по этим приложениям и обширную библиографию можно найти в [1].

### **1.7.5 Проверка корректности программ и верификация результатов вычислений**

Заметим в заключение, что проверка правильности алгоритма в общепринятом смысле вовсе не гарантирует корректность вычисленных с его помощью результатов. Чтобы убедиться в этом, достаточно вспомнить приведенные выше примеры. В таких ситуациях численные методы с автоматической верификацией результатов являются простым, но важным дополнительным средством обеспечения корректности производимых вычислений.

Естественно, сами по себе интервальные методы тоже не обеспечивают абсолютной гарантии, поскольку этап верификации может завершиться с неверным результатом из-за программной ошибки. Интервальная арифметика никогда не сможет избавить нас от необходимости проверять корректность алгоритма, компилятора, операционной системы и тестировать аппаратную часть. С другой стороны, интервальный метод может помочь выявить программную ошибку, если на этапе верификации будет достоверно установлено, что вычисленный результат явно ошибочен. Возникновение такой ситуации свидетельствует о необходимости дальнейшей отладки.

Если верификация завершается успешно, то с высокой вероятностью можно утверждать, что комплекс из самой программы и среды ее выполнения обеспечил вычисление корректного результата независимо от того, проводилось ли доказательство правильности алгоритма.

Допустим, наконец, что с помощью верификации достоверность полученного решения не может быть ни подтверждена, ни опровергнута. Даже такой отрицательный результат приносит определенную пользу. Во-первых, он устанавливается автоматиче-

ски, без необходимости для пользователя проводить какой-либо дополнительный анализ. Во-вторых, при написании программы на такой случай можно заранее предусмотреть использование альтернативного алгоритма или повтор вычислений с повышенной разрядностью.

**Дополнение к русскому изданию.** Примеры применения интервальных вычислений можно найти в работах [92, 111, 133, 136, 138, 144, 150] и в журнале *Reliable Computing*.

Ряд книг посвящен вопросам интервальных вычислений в анализе электрических цепей [14.5], экономике [14.5], инженерном проектировании [97, 163] робастном управлении [95, 101], идентификации систем [160, 202] и анализе их надежности [14.5].

Использование пакета GlobSol в финансовых и медицинских приложениях, для обработки сигналов и прочего описывается в работах [14.5] и [14.5].

Помимо применения в научных и инженерных расчетах, интервальные методы оказали помощь в проверке различных математических гипотез, представляющих существенный интерес [14.5]:

- гипотеза Кеплера о наиболее компактном расположении шаров в трехмерном пространстве [117–118];
- гипотеза двойного пузырька, состоящая в том, что поверхность наименьшей площади, заключающая в себе два равных объема, имеет вид двойного пузырька [14.5];
- гипотеза Дирака-Швингера об асимптотике атомов с большим атомным весом [14.5];
- различные гипотезы теории хаоса.

В частности, с помощью интервальных методов было доказано [14.5], что в фазовом пространстве системы уравнений Лоренца — исторически первого примера хаотической системы, исследованной численно, — существует «странный аттрактор», т. е. структура, на которой задаваемая уравнениями динамика обладает весьма сильной стохастичностью. Вопрос о существовании этого странного аттрактора оставался открытым с 1963 года и был отнесен С. Смейлом к числу самых сложных математических вопросов, не решенных перед наступлением XXI века.