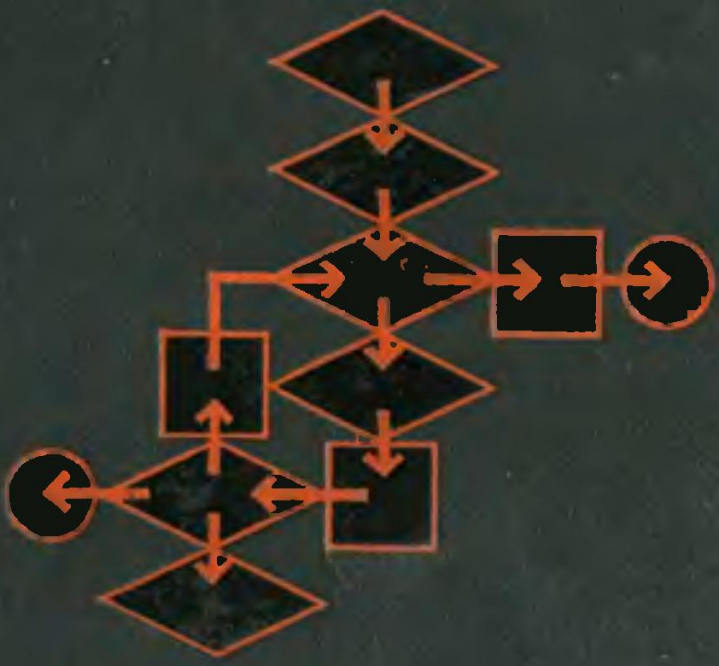
Д. МАК-КРАКЕН
У. ДОРН

ЧИСЛЕННЫЕ МЕТОДЫ И ПРОГРАММИРОВАНИЕ НА ФОРТРАНЕ



NUMERICAL METHODS AND FORTRAN PROGRAMMING

with application in engineering and science

Daniel D. McCracken
McCracken associates, Inc.
William S. Dorn
International Business Machines Corporation

John Wiley and Sons, Inc., New York · London · Sydney
WILEY INTERNATIONAL EDITION

Second printing
1965

Д. МАК-КРАКЕН,
У. ДОРН

ЧИСЛЕННЫЕ МЕТОДЫ И ПРОГРАММИРОВАНИЕ на ФОРТРАНе

ПЕРЕВОД С АНГЛИЙСКОГО
Б. Н. КАЗАКА

ПОД РЕДАКЦИЕЙ И С ДОПОЛНЕНИЕМ
Б. М. НАЙМАРКА

ИЗДАТЕЛЬСТВО «МИР»
Москва 1969

УДК 681.142.2

Книга является руководством по структуре и использованию алгоритмического языка ФОРТРАН при решении вычислительных задач на современных электронных цифровых машинах.

Специфика и простота трансляторов для ФОРТРАНа, эффективность оттранслированных программ и методика выявления и оценки ошибок выгодно отличают этот язык от других алгоритмических языков. Все это обусловило широкое внедрение ФОРТРАНа в технику программирования за рубежом.

Одновременно в книге подробно излагаются тщательно отобранные численные методы, применение которых иллюстрируется на многочисленных практических примерах.

Объединение численных методов и основ программирования на ФОРТРАНе делает эту книгу полезной для широкого круга читателей, как для студентов и аспирантов втузов, так и для инженеров и специалистов по теории программирования.

Редакция литературы по математическим наукам

Инд. 2-2-3.
11—69

ПРЕДИСЛОВИЕ РЕДАКТОРА ПЕРЕВОДА

В последнее время в связи с широчайшим развитием вычислительной техники активно разрабатываются новые алгоритмические языки для программирования. Однако среди многих десятков таких языков, разработанных за рубежом, ФОРТРАН остается до сих пор самым распространенным и употребительным: подавляющее большинство программ для вычислительных задач в различных областях науки и техники составлено и составляется на ФОРТРАНе. Разработаны и накоплены обширнейшие математические программные библиотеки на этом языке, включающие как стандартные (часто используемые) программы, так и множество специальных программ, применяемых для решения специфических задач. Поэтому ни одна из достаточно крупных зарубежных ЭЦВМ не выпускается в широкое пользование без соответствующего программного обеспечения.

Такое повсеместное внедрение ФОРТРАНа в технику программирования происходит благодаря его качествам, из которых следует отметить, во-первых, его простоту по сравнению с другими алгоритмическими языками (например, АЛГОЛом). Во-вторых, благодаря отсутствию слишком сложных конструкций, оттранслированные программы получаются более эффективными по сравнению с другими языками; в то же время ФОРТРАН подходит для программирования подавляющего большинства вычислительных алгоритмов. В-третьих, в ФОРТРАНе имеются очень мощные средства для связи человека с машиной: выдаваемая ЭЦВМ информация представляется в виде, привычном для ученых и инженеров. Наконец, ФОРТРАН хорошо приспособлен для эффективного использования внешних устройств ЭЦВМ.

В литературе, опубликованной на русском языке, до сих пор отсутствовало сколь-либо систематическое описание ФОРТРАНа. Поэтому данная книга может оказаться очень

полезной для инженерных и научных расчетов. В ней хорошо изложены программирование на ФОРТРАНе и основы приближенных вычислений. Эти две области, обычно отделяемые друг от друга, в данной книге тесно переплетаются. При этом благодаря отсутствию скрупулезности все доказательства становятся прозрачными, а программы— понятными.

Со времени публикации американского издания этой книги ФОРТРАН претерпел некоторые изменения и стал полнее. Поэтому в конце дается дополнение, где кратко описаны средства ФОРТРАНа, чаще всего встречающиеся в настоящее время. Этим дополнением можно пользоваться и как кратким руководством.

Язык ФОРТРАН не определяется точным алгоритмическим способом (как АЛГОЛ с помощью металингвистических формул). Это сильно упрощает ФОРТРАН, но иногда приводит к тому, что программы, составленные на одном варианте этого языка, не совместимы с другими его вариантами. В тексте даются указания относительно разных вариантов ФОРТРАНа.

Б. М. Наймарк

ПРЕДИСЛОВИЕ

Эта книга должна дать читателю, специализирующемуся в области техники или науки, основные понятия о численном решении различных задач на современных электронных цифровых вычислительных машинах.

Основное внимание в книге уделено тщательно подобранным и весьма полезным практическим методам исследования различных численных задач, встречающихся в современной науке и технике. Изложенные в книге методы полезны не только сами по себе, но они к тому же подготавливают читателя для изучения более сложных методов и помогают понять их во всей глубине. Точно так же, как физику необходимо иметь интуитивное понятие об инерции до того, как он сможет глубоко изучить механику, так и изучение численных методов должно предшествовать тщательному изучению численного анализа. Цель книги, таким образом, двойная: дать читателю необходимые знания для практических вычислений и заложить основы для дальнейшего изучения предмета.

Объединение численных методов и программирования на ФОРТРАНЕ — это гораздо больше, нежели просто издание двух обычных книг в общей обложке. Сведения по программированию на ФОРТРАНЕ появляются в книге в том порядке, в котором это требуется для различных классов задач и для соответствующих им методов численного решения. Основные примеры программирования также в большинстве случаев иллюстрируют те или иные численные методы. Очень важно также, что во всей книге используются современные понятия. Например, весьма важный вопрос об ошибках округления трактуется в терминах арифметики с плавающей запятой (арифметики нормализованных чисел с постоянным количеством значащих цифр в мантиссе), введенные там же графические представления о вычислительных процессах облегчают полное понимание.

Численные методы, включенные в книгу, были тщательно проанализированы с точки зрения их пригодности для использования на цифровых вычислительных машинах.

Поскольку изучение интерполяции не имело для этой книги особой методической ценности и поскольку интерполяция при машинных вычислениях гораздо менее важна, чем при ручных, вопросы, связанные с ней, излагаются кратко. Еще одна традиционная тема также не нашла отражения на страницах этой книги — операции с матрицами. Причиной этого является то, что многие из предполагаемых читателей книги не владеют теорией матриц и линейных преобразований.

Читатель, изучивший настоящую книгу, конечно, не может считаться специалистом по численному анализу, отчасти именно из-за отсутствия этих традиционных тем. Тем не менее мы убеждены, что имеет большой смысл изучить материал этой книги до того, как начать детальное изучение более полных курсов численного анализа: материал книги дает для этого достаточную основу.

Главы, посвященные численным методам, в большинстве случаев начинаются с геометрической трактовки задачи. Как правило, такая трактовка позволяет наглядно представить смысл задачи и наметить пути ее решения. Затем основные результаты подтверждаются аналитически и, наконец, метод конкретизируется с помощью практических примеров. Мы надеемся, что такой подход приводит к наиболее быстрому и глубокому пониманию материала.

Тринадцать практических примеров включены в книгу по трем причинам. Во-первых, необходимо было проиллюстрировать тот материал, который относится к численным методам и к программированию. Во-вторых, нужно было показать, как различные аспекты численных методов и программирования взаимодействуют и сочетаются между собой. (В связи с этим в нескольких практических примерах показаны последствия легкомысленного подхода к ошибкам вычисления.) В-третьих, необходимо было дать понятие о диапазоне областей науки и техники, в которых могут использоваться вычислительные машины. Содержание практических примеров, однако, таково, что понимание материала не зависит от степени знакомства читателя с данной областью науки или техники. Во многих случаях

в книге воспроизведена печать на выходном бланке вычислительной машины.

В книге имеется более 300 упражнений, и ответы ко многим из них приведены в конце книги. В некоторых упражнениях требуется применить только что изученные численные методы, в других применяются и развиваются методы программирования, в третьих требуется доказать некоторые положения, связанные с материалом книги, или развить эти положения. Многие упражнения дают любознательному читателю возможность самому получить результаты, раскрываемые в полной мере лишь в последующих главах.

На основе одного из ранних вариантов этой книги читался курс лекций группам студентов, начиная от студентов младших курсов технических факультетов до выпускников математических факультетов. Естественно, преподавателю необходимо отбирать темы и видоизменять примеры для приведения курса в соответствие с интересами и возможностями группы. Весьма сильно может помочь в этом отбор упражнений, тем более что количество и разнообразие упражнений вполне позволяют этот отбор сделать.

На основе этой книги можно построить лекционный курс на один семестр (по четыре часа занятий в неделю), если у студентов имеется возможность работать на вычислительной машине и если ожидается, что студенты выполнят достаточно большое число упражнений. С другой стороны, книгу можно использовать и в качестве пособия при гораздо более кратком курсе; для этого можно ограничиться меньшим числом упражнений или пропустить некоторые главы. Более того, главы, посвященные численным методам, не очень жестко связаны между собой. Преподаватель, который пожелает, например, уделить меньше внимания главе 3 о вычислении функций и сконцентрировать внимание аудитории на системах линейных уравнений, довольно легко может это сделать. Возможны и многие другие способы построения учебного курса на основе этой книги.

Мы с большой признательностью отмечаем сотрудничество тех, кто помогал нам в написании и подготовке этой книги: мисс Агнесы Кульке и Джеймса Грисмера из IBM, Уильяма Найбека из инженерного колледжа Милуоки, Фреда Гринбергера из РЭНД Корпорейшен, Джека Холлингсуорта и Поля Макслойна из политехнического

института Реннселера, Юджина Голуба и Гарольда Р. ван Зорена из Стэнфордского университета и Чарльза Дэвидсона из Висконсинского университета. Помощь этих специалистов была для нас очень ценной: своим участием они помогли сформировать эту книгу во многих очень важных деталях. Наконец, мы выражаем свою признательность фирме IBM, и в частности доктору Герману Х. Голдстайну, без поддержки и участия которого эта книга не была бы написана.

Даниель Д. Мак-Кракен
Уильям С. Дорн

ГЛАВА

1

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ФОРТРАНе

1.1. ПРИМЕНЕНИЕ ЦИФРОВЫХ ВЫЧИСЛИТЕЛЬНЫХ МАШИН

Электронные цифровые вычислительные машины (ЭЦВМ) широко применяются для решения научных, технических и экономических задач. Они способны производить вычисления очень быстро, выдавать очень точные результаты, запоминать большие массивы информации и производить длинные и сложные последовательности вычислений без вмешательства человека.

В этой книге будут в основном рассмотрены методы решения на ЭЦВМ научных и технических задач. Некоторое представление об этом круге задач можно получить из следующих примеров. Конструирование нового самолета требует затраты тысяч часов машинного времени для исследования взаимосвязанных требований к конструкционным материалам, аэродинамике, двигателям и системам управления при различных условиях полета. Проектирование химического завода зависит от расчетов продуктивности, условий производства и от многих других обстоятельств. При проектировании линии электропередачи необходимо изучение нагрузок, которые будут приложены к различным участкам линии при изменении потребления электроэнергии или при необычных ситуациях.

В этом введении не лишним будет отметить, что ЭЦВМ не «решает задачу». Она помогает нам лишь исследовать различные варианты. Мы не задаем машине вопрос: «Как нужно построить это устройство?», мы задаем вопрос: «Как будет это устройство работать при данной системе условий, если оно построено определенным образом?» Существует множество способов построения инженерной системы, большое разнообразие условий, при которых этой системе придется работать, и различные, часто противоречивые критерии, которым эта система должна удовлетворять.

ЭЦВМ *не может* задаться исходными данными для проектирования, перечислить условия, при которых надо исследовать работу системы, определить или найти разумную степень компромисса между противоречивыми критериями. Обычно она лишь может оказать нам большую помощь, рассчитывая и предсказывая результаты *нашего* выбора в этих вопросах.

1.2. ПОСЛЕДОВАТЕЛЬНЫЕ ЭТАПЫ В «РЕШЕНИИ ЗАДАЧИ» С ПОМОЩЬЮ ЭЦВМ

Понятие «решение задачи» с помощью ЭЦВМ включает в себя гораздо больше, нежели просто вычисления на ЭЦВМ. Будет полезно представить себе основные этапы решения типичной инженерной проблемы на ЭЦВМ и установить, что делает человек и что делает машина.

Постановка задачи и определение конечных целей

Это вопрос выбора общего подхода, определения совокупности критериев, которым должна удовлетворять система и задания условий ее работы. В некоторых случаях это просто сделать, в других случаях на этот этап могут уйти месяцы работы. Как бы то ни было, на данной стадии требуется глубокое понимание существа задачи; вычислительная машина не может оказать в этой работе практически никакой помощи [1]).

Математическое описание

Как правило, существует несколько способов математического описания задачи; должен быть выбран один из них или разработан новый способ, если ни один из имеющихся не приложим. Этот этап также требует полного понимания проблемы и знания соответствующих областей математики.

[1]) Разумеется, машина не может сама ставить задачи и решать их. Однако в последнее время появились работы, относящиеся к непосредственной связи человека с машиной. Соответствующий режим работы состоит в том, что специалист непрерывно связан с машиной, может оперативно вмешиваться в ее работу, задавать вопросы и получать ответы на них в привычной ему терминологии, не теряя времени на сложное программирование и ожидание результатов. В некоторых случаях это может сильно облегчить и ускорить правильную постановку задачи. — *Прим. ред.*

Численный анализ

Математическая формулировка задачи может оказаться непереводимой непосредственно на язык ЭЦВМ, так как она (ЭЦВМ) выполняет только арифметические действия и принимает простейшие количественные решения. Такие общеизвестные математические понятия, как тригонометрические функции, дифференциальные уравнения, интегралы, квадратные корни, логарифмы, — все должны быть выражены через элементарные арифметические операции. Более того, необходимо убедиться, что никакие погрешности, содержащиеся в исходных данных или внесенные в процессе вычислений, не влияют сколько-нибудь заметно на точность результатов.

Это целая область современной математики. Одной из целей этой книги является ознакомление читателей с элементами численного анализа.

Программирование для ЭЦВМ

Численный алгоритм решения задачи теперь необходимо выразить в виде точно определенной последовательности операций вычислительной машины. Обычно эта работа производится в две стадии. На первой стадии последовательность операций изображается графически в виде *блок-схемы*. Затем алгоритм нужно изложить на «языке», который может быть «понят» вычислительной машиной непосредственно или после предварительного «перевода». Таким языком является ФОРТРАН.

Другими словами, мы обычно рисуем блок-схему метода решения; эта схема важна тем, что она дает ясную картину предстоящих операций, но эта схема не может быть «понята» машиной. Затем, используя блок-схему в качестве руководства, мы пишем программу на ФОРТРАНе, которая будет «понята» машиной после некоторого предварительного перевода, о чем будет сказано ниже.

Одна из задач этой книги состоит в том, чтобы научить читателей писать программы на языке ФОРТРАН. В книге изучаются правила языка ФОРТРАН и подробно рассматриваются способы выражения численных математических алгоритмов в виде последовательностей тщательно определенных вычислительных операций.

Отладка программы

В процессе программирования имеется столько возможностей допустить ошибку, что большинство программ работает сначала неверно. Ошибки в программе должны быть обнаружены и исправлены, а самую программу необходимо тщательно испытать, чтобы быть уверенным в достоверности результатов. На этом этапе используется ЭЦВМ.

Вычисления

Теперь, наконец, можно производить расчеты по программе, используя исходные данные задачи. Как правило, расчет делается сразу для нескольких вариантов набора исходных данных. Этот этап может занять от нескольких секунд до многих часов, в зависимости от задачи и от возможностей ЭЦВМ.

Интерпретация результатов

Как уже говорилось, результаты вычислений, выдаваемые машиной, не всегда содержат полный «ответ» к задаче. Человек, производящий расчет на ЭЦВМ, должен каким-то образом интерпретировать результаты, чтобы понять, что они означают с точки зрения критериев, которым должна удовлетворять исследуемая система. Очень часто бывает нужно частично или полностью повторить предшествующие этапы, пока задача не будет действительно решена.

Из этого краткого рассмотрения можно сделать некоторые выводы. Во-первых, ЭЦВМ сама задач не решает, она только производит заранее заданные последовательности вычислений. Во-вторых, использование ЭЦВМ не освобождает нас от необходимости тщательно осмысливать свою работу. В действительности использование ЭЦВМ заставляет уделять гораздо больше внимания осмысливанию решаемых задач. Машина может производить вычисления быстрее и точнее человека, но она не способна решать, какова должна быть программа вычислений или что делать с получаемыми результатами. В-третьих, ЭЦВМ никоим образом не снимает необходимости детального изучения исследуемой области и применяемых для решения задачи разделов математики.

Как уже было сказано, основным содержанием этой книги являются средние этапы — численные методы и про-

граммирование. Постановка задачи, определение конечных целей и математическое описание относятся к соответствующим областям науки и техники: энергетике, физике, аэродинамике, химии и т. д. Отладка программы описана кратко, сами вычисления по готовой отлаженной программе весьма просты, а интерпретация результатов снова относится к той же области науки или техники, что и исходная задача.

Авторы предполагают, что читатель приступает к изучению предмета этой книги с полным пониманием своих задач и методов их математического описания; мы покажем такому читателю, как перевести математическую формулу в последовательность арифметических операций и из этой последовательности в правильную программу для ЭЦВМ; после этого мы предоставим ему поступать с результатами по собственному усмотрению.

В оставшейся части этой главы будут изложены основы языка ФОРТРАН. Изучив правила написания выражений и операторов на ФОРТРАНе, мы на двух коротких практических задачах покажем, как работают простые программы для ЭЦВМ.

1.3. ПРОГРАММА НА ФОРТРАНе

Алгоритм решения задачи, записанный с помощью ФОРТРАНа, состоит из последовательности *операторов* [1]). Эти операторы могут принадлежать к нескольким различным типам. Одни из них определяют арифметические операции, являющиеся основным содержанием алгоритма, другие определяют порядок ввода и вывода информации, как, например, ввод чисел с перфокарты либо вывод чисел на печать или на перфорацию. Операторы этих двух первых типов исполняются в том порядке, в котором они написаны. Третий тип операторов изменяет *порядок выполнения* операций, так что группы операций могут выполняться повторно или в каком-либо другом порядке, отличном от того, в котором они написаны. Операторы четвертого типа содержат *информацию об* алгоритме, но никаких действий не определяют.

[1]) У авторов, как и в другой зарубежной литературе, применен термин «statement».— *Прим. перев.*

Вместе взятые, все операторы, определяющие алгоритм решения задачи, составляют *исходную программу*. После того как исходная программа написана и отперфорирована на перфокартах, она преобразуется с помощью *транслятора* ФОРТРАНа в *рабочую программу*. Рабочая программа представляет собой последовательность элементарных команд для ЭЦВМ, таких, как сложение, сравнение двух чисел, запоминание числа в памяти машины и т. д. Преобразование исходной программы в рабочую программу необходимо потому, что язык ФОРТРАН является гораздо более сложным, нежели язык команд самой ЭЦВМ. Расчеты на ЭЦВМ производятся именно по рабочей программе, которая и приводит к численным результатам.

Слово ФОРТРАН, таким образом, относится как к языку для описания вычислительных алгоритмов, так и к транслятору. Транслятор ФОРТРАН, называемый также иногда *процессором* или *компилятором*, является сам большой программой на языке машины. (Понятие трансляции повлияло на само название языка ФОРТРАН: ФОРмула и ТРАНсляция.)

Теперь мы обращаемся к изучению элементов, из которых составляются операторы: констант, переменных, операций, выражений и функций. После изучения этих основных элементов языка мы научимся составлять из них операторы и писать некоторые простые программы.

1.4. КОНСТАНТЫ

Мы начинаем с рассмотрения двух типов чисел, применяемых в ФОРТРАНе,— целых и действительных.

Целое число может быть равно нулю, может быть положительным или отрицательным, но по абсолютной величине не должно превосходить 32 768[1]. Как будет показано ниже,

[1]) Различные варианты ФОРТРАНа и различные ЭЦВМ имеют различные ограничения для величины целых чисел; здесь приведены данные для варианта ФОРТРАН-II и машины IBM-7090 или родственных ей. Различия между ФОРТРАНом, описанным в настоящей книге, и другими его вариантами невелики и не заслуживают рассмотрения в пределах этой книги. Однако перед тем, как начинать составлять программу для реальной ЭЦВМ, необходимо ознакомиться с особенностями того варианта ФОРТРАНа, который на этой ЭЦВМ применяется. Как правило, мы не будем останавливаться на подобных особенностях.

целые числа используются только в некоторых специальных ситуациях.

Большинство чисел, используемых в ФОРТРАНе, являются действительными числами. Форма представления таких чисел, называемая формой представления с *плавающей запятой*, где число записывается в виде дроби величиной между 0.1 и 1[1]), умноженной на целую степень 10, имеет сходство с формой записи чисел в научной литературе[2]). *Абсолютная величина числа*, представленного таким образом, должна лежать приблизительно в пределах от от 10^{-38} до 10^{38}.

Как легко видеть, действительные числа могут быть целыми или иметь дробную часть. Большинство ЭЦВМ производит операции с действительными числами таким образом, что программист может не заботиться о положении десятичной запятой. Выравнивание порядков чисел, например при сложении или вычитании, производится в ЭЦВМ автоматически (откуда, собственно, и произошел термин «плавающая запятая»).

Все действительные числа, применяемые в ФОРТРАНе, являются рациональными; применение иррациональных и (иногда —*Ред.*) комплексных чисел запрещено.

Однако вычисления с комплексными числами можно производить, выполняя арифметические действия отдельно с их действительными и мнимыми частями.

Любое число, которое появляется в программе в явном виде, называется *константой*, в то время как величина, которой присваивается наименование и которая может принимать в процессе вычислений различные числовые значения, называется *переменной*. Например, ниже будет показано, что следующие комбинации символов являются *арифметическими операторами*:

$$I = 2$$

$$X = A + 12.7$$

[1]) Правила языка ФОРТРАН предусматривают для отделения дробной части числа от целой не запятую, а точку. Поэтому для сохранения единства обозначений во всей книге в качестве десятичного знака употребляется точка.— *Прим. перев.*

[2]) В обычной научной записи число представляется в виде целой степени 10, умноженной на число величиной между 1.0 и 10, а не между 0.1 и 1.0. Очевидно, легко преобразовать число для того, чтобы печатать его в наиболее привычном виде.

Здесь 2 и 12.7 являются константами: *I*, *X* и *A* являются переменными.

В ФОРТРАНе целые и действительные константы различаются по отсутствию или присутствию десятичной точки. Так, 3— целая константа, в то время как 3.0 (а также 3. или 3.0000 и т. д.) — действительная константа. Два эти типа чисел *не взаимозаменяемы*, потому что способы записи их в памяти ЭЦВМ и выполнения с ними арифметических операций совершенно различны.

Если константа положительна, то знак плюс перед ней можно ставить и можно не ставить; если же константа отрицательна, то перед ней обязательно должен стоять знак минус. Ниже даны примеры правильных целых констант:

0	—1234
6	10000
+400	—20000

Примеры неправильных целых констант:

12.78 (присутствует десятичная точка; десятичную точку нельзя использовать в целой константе)

—10,000 (запятую употреблять нельзя)

1 234 567 890 (слишком велико)

Десятичная точка в действительной константе может стоять в начале числа, в его конце или между двумя его любыми цифрами. Действительное число может иметь любое количество цифр, но только восемь или девять старших из них запоминаются в памяти ЭЦВМ. Другими словами, хотя и не существует никакого ограничения на количество цифр в действительной константе, нет смысла использовать больше восьми или девяти значащих цифр, так как все остальные будут потеряны.

Чтобы упростить запись очень больших или очень малых чисел, применяется следующий прием. После действительной константы ставится буква E и одно- или двузначное целое число. Такая запись означает, что исходная константа должна быть умножена на 10 в указанной степени. Ниже приведены примеры правильных действительных констант:

0.0	+15.016
6.0	5.0E+6 ($=5.0\cdot 10^6$)

6.	$-7.E-12 (= -7.0 \cdot 10^{-12})$
$-20\,000.$	$6.205E12 (= 6.205 \cdot 10^{12})$
$-.0002784$	$-.1E7 (= -0.1 \cdot 10^{7})$

Примеры неправильных действительных констант:

12,345.6 (запятую употреблять нельзя)
+234 (нет десятичной точки)
1.6E63 (слишком велико для большинства ЭЦВМ)
1E—7 (нет десятичной точки)
5.683E2.5 (показатель степени должен быть целым)
E+5 (нет мантиссы)

Упражнения [1])

***1.** Напишите следующие числа в виде действительных констант, соблюдая правила ФОРТРАНа:

256; 2.56; —43 000; 10^{12}; 0.000000492; —10; -10^{-16}

2. То же, что и в упражнении 1:

16; 4.59016; —10 000; 10^{17}; 0.000006; —1; -10^{-10}

***3.** Все приведенные внизу обозначения неприемлемы в качестве действительных констант. Почему?

12,345.6; +234; 1.6E63; 1E—7

4. Все приведенные внизу обозначения неприемлемы в качестве действительных констант. Почему?

—100,000; 1E—55; 2.379427—E12; 2E+2.5

***5.** Определите, представляют ли в каждом случае две рядом стоящие действительные константы одно и то же число:

16.9	+16.9
23 000	2.3E4
0.000007	.7E—5
1.0	1.
.906E5	+906.E2

***6.** Некоторые из приведенных внизу обозначений неприемлемы в качестве целых констант. Определите, в чем состоят ошибки.

+234; —234.; 23,400; 1E12; +1 000 000 000 000; 10 000

7. Некоторые из приведенных внизу обозначений неприемлемы в качестве целых констант. Определите, в чем состоят ошибки.

—16.5; 16 000; 16,000; 2.E12.5; 2.E12; 0.01

---

[1]) Ответы на упражнения, отмеченные звездочкой, приведены в конце книги.

1.5. ПЕРЕМЕННЫЕ И НАИМЕНОВАНИЕ ПЕРЕМЕННЫХ

Термин «*переменная*» употребляется в ФОРТРАНе для обозначения величины, обращение к которой производится через ее наименование и которая может принимать различные значения, а не ограничена каким-либо одним.

Так же как и константы, переменные могут быть целыми и действительными. Целая переменная может принимать любое из значений, допустимых для целой константы, а именно нуль, положительное и отрицательное целое число меньше 32 768.

Наименование целой переменной может содержать от одной до шести букв и цифр, причем оно должно начинаться с одной из букв I, J, K, L, M, N. Примеры правильных наименований целых переменных: I, KLM, MATRIX, L123, I6M2K, KAPPA. Примеры неправильных наименований: J123456 (слишком много знаков), ABC (начинается с неправильной буквы), 5M (начинается не с буквы), $J78 (содержит знак, не являющийся ни буквой, ни цифрой), J34.5 (содержит знак, не являющийся ни буквой, ни цифрой).

Действительная переменная представляется в машине в той же форме, что и действительная константа, т. е. в виде правильной дроби, умноженной на целую степень 10. *Наименование действительной переменной* может содержать от одной до шести букв и цифр и начинаться с буквы, *но не с* I, J, K, L, M, N. Дело в том, что в трансляторе ФОРТРАНа первая буква наименования переменной используется для того, чтобы определить, является ли эта переменная целой или действительной. Примеры правильных наименований действительных переменных: AVAR, R51TX, FRONT, G, F0009, SVECT. Примеры неправильных наименований действительных переменных: A123456 (слишком много знаков), 8BOX (начинается не с буквы), KJL1 (начинается с неправильной буквы), *BCD (содержит знак, не являющийся ни буквой, ни цифрой), A + B (содержит знак, не являющийся ни буквой, ни цифрой), B9.43 (содержит знак, не являющийся ни буквой, ни цифрой).

Присвоение наименований переменным, появляющимся в программе, находится на полном усмотрении программиста. Необходимо только следить за соблюдением правила о различии в наименовании целых и действительных пере-

менных. Если это правило нарушается, то в большинстве случаев ошибка обнаруживается транслятором ФОРТРАНа, и исходная программа не переводится в рабочую; иногда ошибка может пройти незамеченной, и тогда результаты работы программы будут неверными.

Необходимо отметить, что в трансляторе не придается никакого значения наименованиям, за исключением того что исследуется первая буква наименования и определяется тип переменной — целая или действительная. Наименование B7 *не означает* B, умноженное на 7, B^7 или B_7. Многие программисты выбирают для переменных наименования, которые напоминают смысл этих переменных, но в трансляторе ФОРТРАНа наименованиям переменных соответствуют только адреса ячеек, в которых запоминаются численные значения этих переменных. Необходимо также отметить, что каждая комбинация букв и цифр представляет собой самостоятельное наименование. Так, наименование ABC не то же самое, что BAC, и наименования A, AB, AB8 относятся к различным переменным.

Упражнения

***1.** Укажите, какие из следующих наименований допустимы для целых переменных, какие для действительных переменных, а какие вообще не могут быть наименованиями *никаких* переменных:

G, GAMMA, GAMMA421, I, IJK, IJK*, J79-12, LARGE, R(2)19, BTO7TH, ZSQUARED, ZCUBED, 12AT7, 2N173, CDC160, DELTA, KAPPA, EPSILON, A1.4, A1P4, FORTRAN, ALGOL.

2. То же, что и в упражнении 1:

K, I12G, CAT, X+2, XP2. NEXT, 42G, LAST, MU, A*B, X1.4, (X61), GAMMA81, AI, IA, X12, 1X2, GAMMA, KAPPA, XSQUARED.

1.6. ОПЕРАЦИИ И ВЫРАЖЕНИЯ

В ФОРТРАНе существует пять основных операций: сложение, вычитание, умножение, деление и возведение в степень. Каждая из этих операций обозначается особым символом:

Сложение	+
Вычитание	—
Умножение	*
Деление	/
Возведение в степень	**

Заметим, что комбинация ** рассматривается как один символ; символы * и ** не могут быть спутаны: ниже мы увидим, что нельзя ставить два символа операций один рядом с другим. Только эти операции разрешены в ФОРТРАНе в качестве основных, все остальные математические операции должны быть сведены к пяти основным или определены с помощью функций, о которых будет сказано ниже.

Термин «*выражение*» используется в ФОРТРАНе в том же смысле, что и в обычной математической записи. Выражение может состоять из констант, переменных и функций, объединенных между собой символами операций, запятыми и скобками. Для повышения наглядности программист может при этом использовать пробелы в любом порядке и количестве по своему усмотрению. Некоторые примеры выражений и их значения приведены в табл. 1.1.

Таблица 1.1

Выражение	Значение
K	Величина целой переменной *K*
3.14159	Величина действительной константы 3.14159
A+2.1828	Сумма величин *A* и 2.1828
RHO—SIGMA	Разность величин RHO и SIGMA
X* Y	Произведение величин *X* и *Y*
OMEGA/6.2832	Частное от деления величины OMEGA на 6.2832
C** 2	Квадрат величины *C*
(A+F)/(X+2.0)	Сумма величин *A* и *F*, деленная на сумму величин *X* и 2.0
1./(X** 2+Y** 3)	Величина, обратная сумме X^2+Y^3

При написании выражений программист должен соблюдать определенные правила, чтобы составленная им программа соответствовала его намерениям.

1. Два символа операций не должны стоять рядом. Поэтому A * — B является бессмысленным выражением, но A* (—B) допустимо.

2. Скобки используются для указания очередности выполнения операций так же, как и в обычной математической записи. Поэтому $(X + Y)^3$ следует писать в виде (X + Y) **3. Точно так же оба выражения A — B + C и A — (B + C) допустимы, но означают не одно и то же.

3. Сомнительные выражения типа A^{BC} следует писать в виде (A **B) ** C или A ** (B ** C), согласно тому, что необходимо вычислить, но не в виде A ** B ** C. (Разница действительно существует. Например, $(2^2)^3 = 4^3 = 64$, но $2^{(2^3)} = 2^8 = 256$.)

4. Если очередность выполнения операций в выражении не полностью определена скобками, то операции выполняются в следующем порядке: сначала производится возведение в степень, потом все операции умножения и деления, потом операции сложения и вычитания.

Поэтому следующие два выражения эквивалентны:

A*B + C/D — E*F

(A*B) + (C/D) — (E*F)

Другой пример: выражение X* Y** 3 имеет смысл $X \cdot Y^3$, но не $(X \cdot Y)^3$. Заметим, что это правило применимо только при отсутствии скобок. Поэтому выражение (X*Y)**3 означает $(X \cdot Y)^3$ согласно правилу 2.

5. Внутри последовательности умножений и делений или сложений и вычитаний, в которой порядок операций не определен скобками, действия выполняются подряд слева направо. Так, выражение A/B *C означает $\frac{AC}{B}$, но не $\frac{A}{B \cdot C}$, а I — J + K означает $(I - J) + K$, но не $I - (J + K)$.

6. Хотя любое выражение можно возвести в целую степень, только действительное выражение может быть возведено в действительную степень. Сам показатель степени может в свою очередь быть любым выражением. Так, вполне допустимо выражение X** (I + 2).

7. Целые и действительные величины нельзя «смешивать» в одном и том же выражении, однако целые величины могут появляться в действительных выражениях в качестве показателей степени и индексов (см. гл. 7). Исключения суще-

ствуют также для некоторых функций, но в этой книге они не рассматриваются.

8. Скобки в выражении обозначают только очередность выполнения операций. (Как мы увидим ниже, они употребляются иногда и совсем для других целей.) В частности, скобки не заменяют умножения. Так, выражение (A + B) (C + D) бессмысленно; оно должно быть написано в виде (A + B)*(C + D).

В табл. 1.2 приведены примеры правильно и неправильно построенных выражений.

Таблица 1.2

Математическая формула	Правильное выражение	Неправильное выражение
$A \cdot B$	A* B	AB (нет операции)
$A \cdot (-B)$	A* (—B) или —A* B	A*—B (два символа операции стоят рядом)
$A+2$	A+2.	A+2 (смешаны целые и действительные величины)
$-(A+B)$	—(A+B)	—A+B или —+A+B
A^{I+2}	A** (I+2)	A** I+2 (перепутано с $A^{I}+2$)
$A^{B+2} \cdot C$	A** (B+2.) *C	A** B+2.*C ($=A^{B}+2.C$)
$\frac{A \cdot B}{C \cdot D}$	(A* B)/(C* D)	A* B/C* D $\left(=\frac{ABD}{C}\right)$
$\left(\frac{A+B}{C}\right)^{2.5}$	((A + +B)/C) **2.5	(A+B)/C** 2.5 $\left(=\frac{A+B}{C^{2.5}}\right)$
$A\,[X+ +B\,(X+C)]$	A* (X+ +B* (X+ C))	A (X+B (X+C)) (нет операции)

Эти правила важны по многим причинам. Во-первых, необходимо точно составить программу, чтобы были произведены именно те вычисления, которые требуются. Так же как и в обыкновенной математической записи, выражение A* (B + C) *должно* быть написано со скобками. Во-вторых, некоторые вычисления могут оказаться просто невозможными из-за способа работы вычислительной машины и транслятора. Целое число не может быть возведено в действительную степень, потому что, как правило, результат будет

иметь дробную часть и уже не будет целым. Целые и действительные числа в большинстве систем ФОРТРАН нельзя «смешивать», потому что арифметика двух этих типов чисел часто строится совершенно различным образом. В-третьих, для следования этим правилам имеется менее очевидная причина: арифметические операции с дробями конечной длины не всегда точно соответствуют обычным правилам арифметики. Чтобы продемонстрировать, что может случиться, рассмотрим следующее выражение:

$$0.40000000 + 12345678. - 12345677.$$

Правило 5 гласит, что операции будут выполняться слева направо. Результат сложения с точностью до восьми знаков будет равен 12345678. и 0.4 безвозвратно потеряны. После вычитания 12345677. окончательный результат равен 1.0000000.

Предположим, что выражение записано в виде:

$$0.40000000 + (12345678. - 12345677).$$

Скобки заставляют произвести сначала вычитание, дающее 1.0000000. После прибавления 0.40000000 окончательный результат равен 1.4000000. В гл. 2 мы вернемся к вопросу обеспечения точности вычислений путем выполнения арифметических действий в должном порядке. Это представляет существенный практический интерес, хотя эффект и не всегда так заметен, как в приведенном примере.

Неправильный порядок выполнения арифметических операций может привести не только к потере точности результата, но и к полной неудаче, как показывает следующий пример. Предположим, необходимо вычислить выражение A*B/C, в котором величины A, B и C все равны 10^{30}. Сначала будет выполнено умножение, дающее 10^{60}, что является слишком большой величиной для большинства вычислительных машин. Конечный результат вычислений, равный 10^{30}, был бы снова в допустимых пределах, но промежуточный результат выходит из этих пределов. Так как промежуточный результат не умещается в разрядной сетке машины, то машина или остановит программу, или выдаст совершенно неверный результат в зависимости от того, как она устроена.

Простейшее решение — ввести скобки, чтобы деление было выполнено раньше умножения: A*(B/C). Результат деления равен 1.0 и конечный результат получится правильным.

При делении целых чисел возникают свои особые проблемы. Когда целое число делится на целое, то результат в общем случае не будет целым. Деление целых чисел построено так, что результат *уменьшается по абсолютной величине* до ближайшего целого числа, что равносильно просто игнорированию дробной части. Так, результат деления целых чисел 5/3 будет равен 1, а не 2.

Как правило, в большинстве технических расчетов деление целых чисел не употребляется, но будет полезно отметить предосторожности, которые следует соблюдать, если оно все же потребуется. Рассмотрим выражение 5/3*6. Правило 5 гласит, что сначала будет выполнено деление; после отбрасывания дробной части результат будет равен 1, он умножается на 6 и в окончательном результате имеем 6. Результат не равен 10, как мы получили бы, если бы сначала было выполнено умножение, или 12, если бы результат деления был округлен до ближайшего целого числа. С другой стороны, если выражение записано в виде 5*6/3, 6*5/3, 5*(6/3) или (6/3)*5, результат будет равен 10.

Все это относится только к арифметике целых чисел. Любое выражение из предыдущего абзаца, если написать его в действительной форме, дало бы результат 10.000000 (может быть, с одной неправильной цифрой в младшем разряде).

Однако и с действительными числами, как это показано ниже, могут случаться неожиданности. Рассмотрим следующую сумму:

$$1.0/3.0 + 1.0/3.0 + 1.0/3.0$$

Результат деления 1.0/3.0 равен с точностью до восьми знаков 0.33333333. Результат сложения будет равен 0.99999999. Если сравнить его с ожидаемым результатом 1.00000000, то он, конечно, окажется «неверным», что может вызвать недоумение у неопытных программистов.

Трудности, связанные с правильной организацией хода вычислений, не непреодолимы, и многие из них по существу ничем не отличаются от того, что может случиться при рабо-

те с карандашом и бумагой или с настольной вычислительной машиной. С электронной машиной, однако, мы должны иногда принимать специальные меры, чтобы *заранее избежать* этих трудностей, так как в то время, когда машина выполняет программу, мы не можем оперативно вмешиваться в вычисления и являемся только наблюдателями.

Упражнения

1. Напишите выражение, соответствующее каждой из следующих математических формул:

*а. $X+Y^3$;

б. $(X+Y)^3$;

в. X^4;

*г. $A+\frac{B}{C}$;

д. $\frac{A+B}{C}$;

*е. $A+\frac{B}{C+D}$;

ж. $\frac{A+B}{C+D}$;

*з. $\left(\frac{A+B}{C+D}\right)^2+X^2$;

и. $\frac{A+B}{C+\frac{D}{F+G}}$;

*к. $1+X+\frac{X^2}{2!}+\frac{X^3}{3!}$;

*л. $\left(\frac{X}{Y}\right)^{G-1}$;

м. $\frac{\frac{A}{B}-1}{G\left(\frac{G}{D}-1\right)}$.

2. Ниже приведены несколько математических формул и соответствующих им выражений, каждое из которых содержит по крайней мере одну ошибку. Найдите ошибки и напишите правильные выражения:

а. $(X+Y)^4$ — X+Y**4

*б. $\frac{X+2}{Y+4}$ — X+2.0/Y+4.0

в. $\frac{A+B}{C+2}$ — AB/(C+2.)

г. $-\frac{(-X+Y-16)}{Y^3}$ — —(—X+Y—16)/Y**3

*д.	$\left(\frac{X+A+\pi}{2Z}\right)^2$	(X+A+3.1416)/(2.0*Z)**2
е.	$\left(\frac{X}{Y}\right)^{N-1}$	(X/Y)**N-1
*ж.	$\left(\frac{X}{Y}\right)^{R-1}$	(X/Y)**(R-1)
з.	$\frac{A}{B}+\frac{C\cdot D}{F\cdot G\cdot H}$	A/B+CD/FGH
и.	$(A+B)\,(C+D)$	A+B*C+D
*к.	$A+BX+CX^2+DX^3=$ $=A+X\,[B+X\,(C+DX)]$	A+X*(B+X*(C+DX))
л.	$\frac{1{,}600{,}042X+10^5}{4{,}309{,}992X+10^5}$	(1,600,042X+ +1E5)/(4,309,992X+1E5)
м.	$\frac{1}{A^2}\left(\frac{R}{10}\right)^A$	1/A**2*(R/10)**A

1.7. МАТЕМАТИЧЕСКИЕ ФУНКЦИИ

В ФОРТРАНе предусмотрена возможность использования некоторых элементарных математических функций, таких, как квадратный корень, логарифм, экспонента, синус, косинус, арктангенс и абсолютная величина. Конкретный набор функций, предусмотренных в трансляторе ФОРТРАНа, зависит от варианта ФОРТРАНа и в некоторой степени от самой вычислительной машины. Однако все варианты ФОРТРАНа предусматривают возможность вычисления нижеперечисленных функций.

Каждая функция имеет присвоенное ей наименование. В общем наименования различны в различных вариантах ФОРТРАНа, но наименования элементарных функций довольно стандартны. Функции, которые мы будем использовать, перечислены в табл. 1.3.

Например, необходимо вычислить косинус угла X. Угол должен быть выражен в радианах. Запись в каком-либо выражении COSF(X) приведет к вычислению косинуса угла X. В этом примере аргумент функции есть просто переменная X. В действительности аргумент может быть любым выражением с тем ограничением, что для всех

Таблица 1.3

Математическая функция	Наименование на ФОРТРАНе
Квадратный корень	SQRTF
Экспонента	EXPF
Синус от угла в радианах	SINF
Косинус от угла в радианах	COSF
Арктангенс, угол в радианах	ATANF
Натуральный логарифм	LOGF
Абсолютная величина	ABSF

функций табл. 1.3 аргумент должен быть действительной величиной и значение функции также будет действительной величиной. Если, например, нам нужен квадратный корень из $(B^2 - 4AC)$, то мы просто напишем

```
SQRTF(B**2-4.*A*C)
```

1.8. АРИФМЕТИЧЕСКИЕ ОПЕРАТОРЫ

Основные элементы языка ФОРТРАН, которые мы только что рассмотрели, широко применяются при составлении исходных программ. Наиболее важным таким применением является вычисление нового значения переменной, которое производится с помощью *арифметического оператора.* Общий вид этого оператора есть $a = b$, где a — наименование переменной, написанное без знака, а b является выражением одного из типов, описанных в разд. 1.6. Арифметический оператор является командой, исполняя которую, ЭЦВМ вычисляет величину выражения в правой части оператора и присваивает эту величину переменной в левой части.

Знак равенства в арифметическом операторе используется не совсем в том смысле, что в обычной математической записи. В ФОРТРАНе нельзя писать операторы вида Z — RHO = ALPHA + BETA, где Z было бы неизвестным, а все остальные были бы известными. Единственной допустимой формой написания арифметического оператора

является такая, где в левой части оператора стоит наименование только одной переменной. Точное значение знака равенства в этом случае будет следующим: *заменить прежнее значение переменной, стоящей в левой части, на значение выражения, стоящего в правой части оператора.* Таким образом, оператор $A = B + C$ есть не что иное, как команда вычислить сумму значений переменных B и C и заменить прежнее значение переменной A этой суммой. Прежнее значение переменной A при этом теряется, но значения переменных B и C остаются неизменными. Вполне возможно, что в каких-то других частях программы могут присваиваться новые значения переменным B и C; когда мы говорим «значение переменной B», то мы имеем в виду *последнее значение, которое было присвоено переменной B* перед началом действия данного оператора.

В другом примере арифметического оператора эти специфические особенности знака равенства проявляются предельно ярко. Оператор типа $N = N + 1$ имеет смысл: *присвоить переменной N ее старое значение плюс* 1. Этот тип оператора, явно не имеющий аналогии в обычной математической записи, широко распространен в ФОРТРАНе.

Хотя смешение арифметики целых и действительных чисел недопустимо внутри арифметического выражения, можно преобразовывать целые величины в действительные и обратно. Для этого достаточно написать выражение одного типа справа и переменную другого типа слева. Если, например, справа будет написано целое выражение, а слева действительная переменная, то вычисления будут произведены по правилам арифметики целых чисел, но результат вычислений будет преобразован в действительную форму перед тем, как присвоить это значение переменной в левой части. Этой возможностью приходится пользоваться довольно редко, но в случае необходимости она создает ощутимые удобства для программиста.

Несколько примеров помогут уяснить применение арифметических операторов. Предположим, что A, B, C, D и X уже вычислены с помощью предыдущих операторов, и теперь нам необходимо вычислить значение R из соотношения

$$R = \frac{A + BX}{C + DX}.$$

Необходимые вычисления могут быть произведены с помощью следующего оператора:

R = (A + B*X)/(C + D*X)

Ни одно из значений переменных в правой части не будет изменено после выполнения оператора, переменной R будет присвоено новое значение, старое значение переменной R будет потеряно.

Предположим, что необходимо вычислить один из корней квадратного уравнения $AX^2 + BX + C = 0$. По-прежнему A, B и C должны быть вычислены с помощью предыдущих операторов или введены в ЭЦВМ с помощью оператора READ, описанного ниже. Следующий оператор определит необходимый порядок вычислений и присвоит вычисленное значение переменной ROOT1:

ROOT1 = (−B + SQRTF (B**2 − 4.*A*C))/(2.*A)

Было бы полезно еще раз напомнить о значении скобок. Скобки, в которые заключено выражение B**2 —4.*A*C, необходимы для того, чтобы обозначить аргумент функции квадратного корня. Скобки вокруг числителя необходимы для того, чтобы все, что находится перед знаком деления, было разделено на то, что следует далее. Скобки вокруг выражения 2.*A показывают, что A находится в знаменателе: если бы эти скобки отсутствовали, то числитель был бы разделен на 2, а результат деления был бы умножен на A.

Последний пример показывает, как аргумент функции в случае необходимости может сам быть другой функцией. Предположим, что необходимо вычислить значение V из формулы

$$V = \frac{1}{\cos X} + \log \left| \operatorname{tg} \frac{X}{2} \right| .$$

Так как в ФОРТРАНе не предусмотрена специальная функция для вычисления тангенса, нам придется воспользоваться тригонометрическим равенством

$$\operatorname{tg} \theta = \frac{\sin \theta}{\cos \theta} .$$

Оператор для вычисления V запишется следующим образом:

V = 1./COSF (X) + LOGF (ABSF (SINF (X/2.)/COSF (X/2.)))

Вполне допустимо введение промежуточных переменных, что может облегчить чтение исходной программы:

```
Y = X/2.
TAN = SINF (Y)/COSF (Y)
ABSVAL = ABSF (TAN)
V = 1./COSF (X) + LOGF (ABSVAL)
```

Примеры, приведенные в табл. 1.4, показывают правильно написанные арифметические операторы и эквивалент-

Таблица 1.4

Арифметический оператор	Исходная формула
`BETA = -1./(2.*X) + A** 2/(4.*X** 2)`	$\beta = \frac{-1}{2X} + \frac{A^2}{4X^2}$
`FY = X* (X** 2 - Y** 2)/(X** 2 + Y** 2)`	$F_y = X \cdot \frac{X^2 - Y^2}{X^2 + Y^2}$
`C = 1.112* D* R1* R2/(R1 - R2)`	$C = 1.112D \frac{r_1 r_2}{r_1 - r_2}$
`Y = (1.E - 6 + A* X** 3)** (2./3.)`	$Y = (10^{-6} + AX^3)^{2/3}$
`J = 4* K - 6* K1* K2`	$J = 4K - 6k_1k_2$
`I = I + 1`	$I_{нов} = I_{стар} + 1$
`K = 12`	$K = 12$
`PI = 3.1415927`	$\pi = 3.1415927$
`M = 2* M + 10* J`	$M_{нов} = 2M_{стар} + 10J$
`R = COSF (X) + X* SINF (X)`	$R = \cos X + X \sin X$
`S = -COSF (X)** 4/4.`	$S = -\cos^4 X/4$
`T = ATANF (1.414214* SINF (X)/COSF (X))`	$T = \operatorname{arctg} (\sqrt{2} \operatorname{tg} X)$

ные им математические формулы. Наименования переменных выбраны произвольно; любые другие допустимые наименования переменных можно было бы употребить с тем же успехом. Кроме того, подразумевается, что все величины переменных в правой части известны к моменту начала действия арифметического оператора.

Примеры, приведенные в табл. 1.5, подчеркивают необходимость строгого соблюдения правил написания выраже-

ний и операторов, так как успешная работа транслятора ФОРТРАНа невозможна при нарушении этих правил. Каждый оператор табл. 1.5 содержит по крайней мере одну ошибку.

Таблица 1.5

Арифметический оператор	Ошибка
Y = 2.X + A	Отсутствует * (знак умножения)
3.14 = X — A	Слева должно стоять наименование переменной
A = ((X + Y) A** 2 + + (R — S)** 2/16.7	Неодинаковое количество левых и правых скобок, отсутствует *
X = 1,624,009 * DELTA	Запятые в константах употреблять нельзя
— J = I** 2.	Целую величину нельзя возводить в действительную степень, переменная в левой части должна быть написана без знака
BX6 = 1./ — 2.* A** 6	Нельзя ставить два символа операций рядом
DERIV = N* X** (N — 1)	Смешение арифметики целых и действительных чисел
A* X + B = Q	Слева должна стоять одна переменная, нужно написать Q = A* X + B
FNC = CUBRTF (X + Y)	Такой функции в ФОРТРАНе нет, следует написать FNC = (X + Y)** 0.3333
SQRTF (Z) = Z** 0.5	Название функции нельзя использовать в качестве переменной, в левой части должно стоять наименование переменной

Упражнения

1. Определите значение A или I, которое получится после исполнения арифметического оператора, и определите также, будет ли результат целым или действительным:

***а.** A = 2*6 + 1

***б.** A = 2/3

в. A=2.*6./4.
г. I =2*10/4
д. I =2(10/4)
е. A=2(10/4)
ж. A=2.*(10./4.)
з. A=2.0*(1.0E1/4.0)
и. A=6.0*1.0/6.0
к. A=6.0*(1.0/6.0)
*л. A=1./3. +1./3. +1./3. +1/3.
м. A=(4.0)**(3/2)
н. A=(4.0)**3./2.
*о. A=(4.0)**(3./2.)
*п. I =19/4+5/4
р. A=19/4+5/4
с. I =100*(99/100)

2. Каждый из арифметических операторов, приведенных ниже, содержит по крайней мере одну ошибку. Определите эти ошибки:

а. —V=A+B
б. 4=I
в. V—3.96=X**1.67
г. X=(A+6)**2
д. A*X**2+B*X+C
е. K6=I**A
ж. Z2=A*—B+C**4
з. X=Y+2.0=Z+9.0

3. Напишите арифметические операторы, которые производили бы следующие вычисления:

*а. Прибавить 2 к текущему значению переменной BETA, сделать сумму новым значением переменной DELTA.
б. Отнять значение переменной *B* от значения переменной *A*, возвести разность в квадрат и сделать ее новым значением переменной *W*.
*в. Возвести в квадрат величину *A*, прибавить результат к квадрату величины *B* и сделать корень квадратный из окончательного результата новым значением *C*.
*г. Заменить текущее значение переменной *R* корнем квадратным из 2.

д. Умножить THETA на π и запомнить косинус произведения в качестве нового значения переменной RHO.

е. Сложить величины F и G, результат разделить на сумму величин R и S, частное возвести в квадрат и сделать его новым значением переменной P.

*ж. Умножить косинус двойного угла X на корень квадратный из половины X, сделать Y равным этому результату.

*з. Увеличить текущее значение G на 2, прежнее значение G заменить полученной суммой.

и. Умножить текущее значение A на —0.1, прежнее значение A заменить полученным произведением.

4. Напишите арифметические операторы для вычисления значений по следующим формулам. Для наименования переменных используйте те буквы и обозначения, которые имеются в формулах:

*а. $\mathrm{AREA}=2\cdot P\cdot R\cdot\sin\left(\frac{\pi}{P}\right)$;

б. $\mathrm{CHORD}=2R\sin\left(\frac{A}{2}\right)$;

*в. $\mathrm{ARC}=2\sqrt{Y^2+\frac{4X^2}{3}}$;

г. $\mathrm{S}=-\frac{\cos^4 X}{X}$;

*д. $\mathrm{S}=-\frac{\cos^{P+1}X}{P+1}$;

*е. $\mathrm{G}=\frac{1}{2}\log\frac{1+\sin X}{1-\sin X}$;

ж. $\mathrm{R}=\frac{\sin^3 X\cos^2 X}{5}+\frac{2}{15}\sin^3 X$;

з. $\mathrm{D}=\log|\sec X+\operatorname{tg} X|$;

*и. $\mathrm{E}=X\operatorname{arctg}\frac{X}{A}-\frac{A}{2}\log(A^2+X^2)$;

к. $\mathrm{F}=-\frac{\pi}{2}\log|X|+\frac{A}{X}-\frac{A^3}{9X^3}$;

л. $\mathrm{Z}=-\frac{1}{\sqrt{X^2-A^2}}-\frac{2A^2}{3(\sqrt{X^2-A^2})^3}$;

*м. $\mathrm{Q}=\left(\frac{2}{\pi X}\right)^{1/2}\sin X$;

н. $\mathrm{B}=\frac{e^{X/\sqrt{2}}\cos\left(\sqrt{\frac{X}{2}}+\frac{\pi}{8}\right)}{\sqrt{2\pi X}}$;

*о. $\mathrm{Y}=(2\pi)^{1/2}X^{X+1}e^{-X}$;

п. $\mathrm{T}=A\cdot e^{-\sqrt{W/2P}\cdot X}$.

1.9. ВВОД И ВЫВОД

Если предстоит произвести вычисления только один раз, то все исходные величины можно ввести в программу в виде констант в операторах. Однако так поступают редко; обычно программы составляются таким образом, чтобы исходные величины вводились в ЭЦВМ с перфокарт во время выполнения программы, а константы употребляются для тех величин, которые действительно являются постоянными. В этом случае одна и та же программа может быть использована для вычислений при любом наборе исходных величин и столько раз, сколько необходимо.

Ввод информации в ЭЦВМ производится с помощью оператора READ, в котором перечисляются переменные, для которых нужно прочесть новые значения с перфокарты. (Пример оператора READ можно видеть на стр. 42.) Наименования переменных должны быть разделены между собой запятыми. Новые значения, присваиваемые переменным, должны быть отперфорированы на карте в той же самой последовательности, в какой их наименования перечислены в операторе READ. Таким образом, процесс ввода происходит в следующем порядке: первое значение с перфокарты присваивается первой переменной, второе значение присваивается второй переменной и т. д., пока не будет исчерпан список переменных и значений на перфокарте. Последовательный перебор идет слева направо, так что первой переменной будет присвоено значение, пробитое в левом краю перфокарты. Далее, следует помнить, что оператор READ *всегда* означает чтение с новой перфокарты. Если, например, на одной перфокарте пробито шесть величин, то их нельзя ввести с помощью двух операторов READ, перечислив переменные так, чтобы второй оператор начал ввод с того места, где первый остановился. Единственный способ ввести шесть величин с одной перфокарты — это написать один оператор READ и перечислить в нем наименования всех шести переменных.

Подобным же образом действует оператор PRINT: переменные, значения которых нужно напечатать, перечисляются в том порядке, в котором они должны быть расположены

в строчке — слева направо. Оператор PRINT всегда означает печатание с новой строки [1]).

Операторы READ и PRINT дают нам ответ на следующие вопросы относительно операций ввода и вывода:

1. Относится ли операция к вводу или выводу и какой при этом используется носитель информации. READ определяет ввод информации и в то же время определяет чтение с *перфокарты*; PRINT определяет вывод на печать на *бумажную ленту*.

2. Какие переменные должны получить новые значения или значения каких переменных надо напечатать: это определяется перечислением наименований переменных в операторе READ или PRINT.

3. В каком порядке расположены значения на перфокарте или должны расположиться при печати на бумаге: это определяется порядком, в котором наименования переменных появляются в операторе READ или PRINT.

Однако для выполнения операций ввода и вывода необходима еще информация другого характера. В каком *формате* пробита или отпечатана информация? Сколько колонок на перфокарте занимает каждая входная величина? Сколько знаков займет каждая выходная величина при печати? Являются ли вводимые и выводимые величины целыми или действительными?

Эти сведения содержатся в операторе FORMAT, который должен сопутствовать каждому оператору READ или PRINT[2]). Порядок использования оператора FORMAT следующий. Каждый оператор READ или PRINT должен содержать сразу же после слова READ или PRINT номер соответствующего оператора FORMAT. (Номер оператора — это просто метка, которую можно присвоить любому оператору в программе.) Номер оператора состоит из деся-

---

[1]) Существуют также и другие операторы ввода и вывода. Вкратце они описываются в приложении 1, некоторые примеры их применения имеются в гл. 10 и 11.

[2]) Некоторые варианты ФОРТРАНа имеют стандартизованный формат для операций ввода и вывода. В этих случаях оператор FORMAT либо вовсе не нужен, либо может быть применен в упрощенной форме. Если ЭЦВМ, с которой намерен работать читатель, имеет такой стандартизованный формат, то читатель вполне может пропустить остаток этого раздела и вернуться к нему позднее.

тичных цифр — от одной до пяти. В операторе FORMAT должна присутствовать *спецификация поля* для каждой переменной из списка в операторах READ и PRINT. (Полем называется группа колонок на перфокарте или группа позиций для печати на бумажной ленте.) Процесс последовательного перебора имеет место также и для оператора FORMAT. При вводе, например, первое значение на перфокарте соответствует первому наименованию переменной в операторе READ и первой спецификации поля в операторе FORMAT. Полный список спецификаций поля в операторе FORMAT заключается в скобки, а отдельные спецификации отделяются друг от друга запятыми.

В большинстве вариантов ФОРТРАНа существует около полдюжины различных спецификаций поля. Сочетание этих спецификаций обеспечивает бóльшую гибкость в операциях ввода и вывода, гораздо бóльшую, чем может потребоваться начинающему программисту. Поэтому мы сначала рассмотрим только три из них, и то не во всех деталях. Более полное описание можно найти в приложении 1.

Самая простая спецификация поля используется при вводе и выводе целых чисел. Спецификация поля для целых чисел состоит из буквы I, после которой стоит цифра, означающая количество колонок, занятое полем. Если, например, поле для вводимой информации на карте занимает восемь колонок, то спецификация поля будет I8.

Подлежащие вводу целые числа необходимо перфорировать без десятичной точки; при выдаче их на печать десятичная точка также не печатается. Если входная величина имеет меньше значащих цифр, чем предусмотрено в спецификации, число должно быть отперфорировано в правой части поля. Если количество значащих цифр при выводе меньше количества позиций, предусмотренного в спецификации, число будет напечатано в правом краю поля. При вводе перед положительным числом знак плюс можно ставить и можно не ставить, перед отрицательным числом обязательно ставить знак минус перед первой значащей цифрой. При выводе в большинстве случаев знак плюс не печатается. При подсчете числа в спецификации поля необходимо предусматривать место для печати или для перфорации знака.

При вводе и выводе действительных чисел мы чаще всего будем пользоваться спецификацией поля, обозначаемой буквой F. Эта спецификация немного сложнее спецификации типа I, так как в ней необходимо определять положение десятичной точки.

Общая форма спецификации типа F есть F $w.d$, где F — символ типа спецификации, w — целое число, определяющее общее количество позиций, включая колонки для знака и десятичной точки, d — количество цифр после десятичной точки. При вводе можно пробивать на перфокарте десятичную точку или не пробивать по усмотрению программиста. Если десятичная точка пробита на перфокарте, то число d не играет никакой роли, если же десятичная точка отсутствует, d определяет положение *подразумеваемой* десятичной точки. Если, например, спецификация имеет вид F 10.3, то подразумевается, что число занимает на перфокарте 10 колонок, и если на перфокарте *не пробита* десятичная точка, то число будет введено в ЭЦВМ так, как будто десятичная точка стоит между третьим и четвертым знаками справа.

При выводе информации спецификация F определяет, что на ленте будет напечатана десятичная точка, причем позицию, занимаемую этой точкой, необходимо учитывать при подсчете общего количества позиций в спецификации поля. Например, если необходимо напечатать действительное число, которое по абсолютной величине может доходить до 99 999, и если необходимо напечатать еще два знака после десятичной точки, то при подсчете общего количества позиций учитывается позиция для знака числа, пять позиций перед десятичной точкой, сама десятичная точка и две позиции после десятичной точки. Таким образом, поле должно состоять по крайней мере из девяти позиций, или, что то же самое, спецификация поля должна быть по крайней мере F 9.2. Если же программист хочет иметь некоторое количество лишних позиций, например для того, чтобы число можно было легче прочесть, то следует написать что-либо вроде F 12.2. При этом три позиции перед числом окажутся пустыми.

При употреблении спецификации F часто возникает одна трудность. Что делать, если заранее неизвестно, каков порядок выходной величины? В этом случае программист

не может знать, какое количество позиций он должен предусмотреть для печати. Эта трудность снимается в случае употребления спецификации типа E. Она называется обычно спецификацией нормализованных чисел и определяет печать выходной величины с заранее заданным количеством значащих цифр в мантиссе и со степенью десятки (порядком). Общая форма спецификации типа E есть E $w \cdot d$, где E обозначает тип спецификации, а w и d имеют тот же смысл, что и для спецификации типа F. Выходная величина в этом случае будет напечатана в виде

$$\pm\ 0.\text{nnn} \ldots \text{E} \pm \text{ee}$$

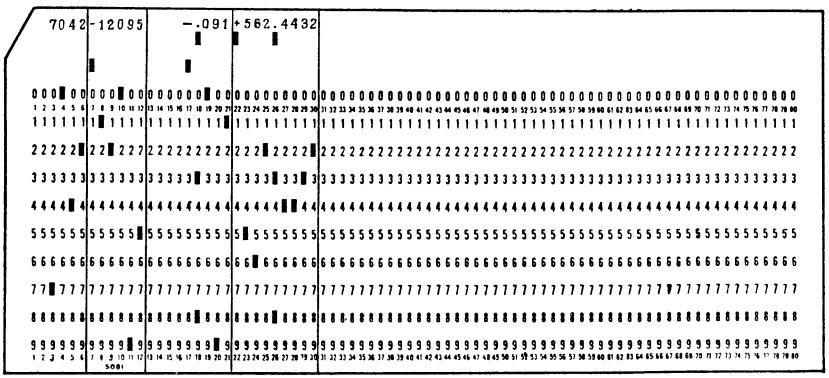
причем количество знаков после десятичной точки определяется числом d. Так же как и в спецификациях I и F, знаки плюс обычно не печатаются. При определении числа w необходимо учитывать позиции, занимаемые знаками, десятичной точкой, буквой E и двумя цифрами порядка.

Заметим, что если число выводится на печать с помощью спецификации поля типа E, то мантисса его представляет собой правильную дробь величиной между 0.1 и 1.0, умноженную на степень десяти. Как уже указывалось, общепринятая запись в научной литературе отличается от этой формы: там принято выбирать величину мантиссы между 1.0 и 10.0. Для того чтобы выходная информация была отпечатана в более привычном виде, достаточно поставить 1P перед буковой E в спецификации поля. При этом десятичная точка будет напечатана между первой и второй значащими цифрами мантиссы, считая слева, а порядок автоматически будет уменьшен на единицу. Символ 1P называется *масштабным коэффициентом*. Такой способ применяется очень часто, в этой книге его можно найти в нескольких примерах.

Входную информацию также можно перфорировать в нормализованном виде, и иногда это оказывается целесообразным. В данной книге, однако, мы не будем пользоваться спецификацией типа E для ввода. Читатель может найти необходимые подробности в приложении 1.

Рассмотрим пример. Мы хотим прочесть с перфокарты четыре числа. Первые два числа — целые, они должны стать значениями переменных с наименованиями J и K. Третье и четвертое числа — действительные, они должны стать

значениями переменных с наименованиями X и Y. Первые два числа могут иметь каждое до пяти цифр и могут быть отрицательными. Третье и четвертое числа могут иметь до семи цифр, могут быть отрицательными и всегда будут перфорироваться с десятичной точкой. Образец перфокарты с входной информацией приведен на рис. 1.1. Вертикальные линии на рисунке проведены для удобства читателей. Эти линии не играют никакой роли для ЭЦВМ, поскольку при



Р и с. 1.1. Пример перфокарты с исходными данными.

вводе в ЭЦВМ имеют значение только отверстия на перфокарте. Заметим, что знак плюс можно пробивать или не пробивать по усмотрению программиста. Перфокарта, изображенная на рис. 1.1, была пробита на перфораторе, который печатает содержание каждой колонки над этой колонкой.

Для того чтобы прочесть эту карту, необходим оператор READ и оператор FORMAT. В операторе READ указывается номер оператора FORMAT и перечисляются наименования переменных в том порядке, в котором на перфокарте расположены соответствующие значения. Оператор FORMAT определяет спецификации четырех полей на карте, опять в том же порядке, в котором они расположены. Два целых числа занимают по шести колонок каждое — пять колонок для цифр и одну для знака. Действительные числа занимают по девять колонок — семь для цифр, одну для знака и одну для десятичной точки.

Операторы для ввода чисел с этой карты могут выглядеть, например, следующим образом:

```
READ 523, J, K, X, Y
523 FORMAT (I6, I6, F9.0, F9.0)
```

Номер 523 для оператора FORMAT выбран произвольно. Единственное, что может вызвать некоторое недоумение при рассмотрении оператора FORMAT, это нуль в F9.0. Было сказано, что этот нуль представляет собой количество цифр, расположенных после десятичной точки; однако если десятичная точка отперфорирована на карте, то эта часть спецификации не играет никакой роли. Так как мы намеревались отперфорировать точку, то в эту часть спецификации можно подставить *любое* число.

Предположим, что теперь мы хотим напечатать эти же четыре числа. Обычно рекомендуется оставлять некоторое пустое пространство между числами для того, чтобы их было легче прочесть; например, отведем 10 позиций под каждое целое и 15 позиций под каждое действительное число. Печатать действительные числа можно с помощью спецификации типа F, предусматривая, скажем, четыре цифры после десятичной точки. Тогда операторы для печати этих чисел будут выглядеть так:

```
PRINT 609, J, K, X, Y
609 FORMAT (I10, I10, F15.4, F15.4)
```

Выходная информация будет напечатана так, как показано в первой строке рис. 1.2.

Для того чтобы напечатать выходную информацию в нормализованном виде, можно воспользоваться спецификацией типа E. Если каждое действительное число занимает по 20 позиций и имеет восемь значащих цифр после десятичной точки, то операторы будут выглядеть следующим образом:

```
PRINT 781, J, K, X, Y
781 FORMAT (I10, I10, E20.8, E20.8)
```

Выходная информация будет напечатана так, как показано во второй строке рис. 1.2.

Если две или более последовательных спецификаций поля одинаковы, то можно перед спецификацией поля указывать число ее повторений. В нашем примере четыре числа

```
7042    -12095        -0.0910          562.4432
7042     12095     -0.91000000E-01      0.56244320E 03
7042     12095     -9.1000000E-02       5.6244320E 02
```

Р и с. 1.2. Пример печати на выходном бланке ЭЦВМ.
Здесь показано, как можно тремя различными способами напечатать числа, отперфорированные на перфокарте рис. 1.1.

можно напечатать точно так же, как на первой строке рис. 1.2, написав операторы:

```
PRINT 439, J, K, X, Y
439 FORMAT (2I10, 2F15.4)
```

Наконец, предположим, что мы хотим напечатать действительные числа так, как это принято в научной литературе, т. е. с десятичной точкой, стоящей после первой значащей цифры. Для этого достаточно написать оператор FORMAT следующим образом:

```
49 FORMAT (2I10, 1P2E20.7)
```

или, что то же самое:

```
49 FORMAT (I10, I10, 1PE20.7, 1PE20.7)
```

Выходная информация будет напечатана так, как показано на третьей строке рис. 1.2.

Примеры использования оператора FORMAT будут неоднократно встречаться на протяжении данной книги.

Упражнения

В каждом из следующих упражнений необходимо прочесть с перфокарты исходную информацию, использовать считанные значения для вычислений и напечатать результат. Исходную информацию необходимо печатать вместе с результатом вычислений для того, чтобы легче было сопоставлять входные и выходные величины. Предположите, что каждое входное число занимает 10 колонок

на перфокарте, используйте спецификацию F10.0 для ввода и спецификацию E20.8 для вывода. Если у читателя есть возможность производить расчеты на ЭЦВМ, то можно рекомендовать произвести вычисления по этим программам непосредственно на ЭЦВМ.

*1. Ввести с карт A, B, C.
Напечатать A, B, C, X1, X2.
Вычислить:

$$X1=\frac{-B+\sqrt{B^2-4AC}}{2A},$$

$$X2=\frac{-B-\sqrt{B^2-4AC}}{2A}.$$

2. Ввести с карт A, B, C, X.
Напечатать A, B, C, X, R.
Вычислить:

$$R=\frac{B\cdot C}{12}\left[6X^2\left(1-\frac{X}{A}\right)^2+B^2\left(1-\frac{X}{A}\right)^4\right].$$

*3. Ввести с карт A, E, H, P.
Напечатать A, E, H, P, X.
Вычислить:

$$X=\frac{E\cdot H\cdot P}{(\sin A)\cdot\left(\frac{H^4}{16}+H^2P^2\right)}.$$

4. Ввести с карт HO, HF, HR, HI, ZG.
Напечатать HO, HF, HR, HI, ZG, ADMTNC.
Вычислить:

$$ADMTNC=HO-\frac{HF\cdot HR}{HI+ZG}.$$

5. Ввести с карт A, X, S.
Напечатать A, X, S, Y, Z.
Вычислить:

$$Y=\sqrt{X^2-A^2},$$

$$Z=\frac{X\cdot S}{2}-\frac{A^2}{2}\log|X+S|.$$

6. Ввести с карт ET, ES, RG, ROPT, RIN.
Напечатать ET, ES, RG, ROPT, RIN, F.
Вычислить:

$$F=\frac{1}{1-\frac{1+\left(\frac{RG}{ROPT}\right)^2}{\frac{ET^2}{ES^2}\left(1+\frac{RG}{RIN}\right)^2}}.$$

1.10. ПЕРЕДАЧА УПРАВЛЕНИЯ. ОПЕРАТОРЫ GO TO И IF

Элементы языка ФОРТРАН, рассмотренные в предшествующих параграфах, позволяют производить весьма сложные вычисления, ввод и вывод информации, но при использовании их остается не снятым серьезное ограничение. Все операторы, рассмотренные в разд. 1.8 и 1.9, могут выполняться только в той последовательности, в которой они написаны, и ни в какой другой. Операторы GO TO и IF, называемые операторами *передачи управления* или операторами *перехода*, снимают это ограничение.

Оператор GO TO определяет передачу управления какому-либо оператору, отличному от следующего. Этот оператор записывается в виде «GO TO n», где n — номер оператора, находящегося где-либо в программе. (Номер оператора пишется перед оператором и может содержать от одной до пяти десятичных цифр; номер является меткой, которую можно присвоить любому оператору в программе, см. разд. 1.9.) Когда в ходе выполнения программы очередь доходит до оператора GO TO n, то следующим будет исполнен оператор с номером n, т. е. управление «передается» оператору с номером n. Этот оператор может быть расположен как до, так и после оператора GO TO.

Оператор GO TO является оператором *безусловного* перехода. Однако часто требуется изменять последовательность выполнения операторов *в зависимости от того, что происходит в процессе выполнения программы*. Другими словами, необходим оператор *условного* перехода, передающий управление в зависимости от результатов вычислений.

Оператор IF записывается в виде

$$\text{IF}\,(e)\,n_1,\ n_2,\ n_3$$

Здесь e означает некоторое выражение, а n_1, n_2 и n_3 представляют собой номера операторов. Оператор IF выполняется следующим образом: если выражение в скобках отрицательно, то следующим выполняется оператор с номером n_1, если это выражение равно нулю, то следующим выполняется оператор n_2, если это выражение положительно, то следующим выполняется оператор n_3.

Применение обоих операторов перехода можно проиллюстрировать на следующем примере. Предположим, что

по ходу выполнения программы необходимо вычислить Y как функцию X по одной из следующих двух формул:

$$Y = 0.5X + 0.95 \text{ если } X \leqslant 2.1$$
$$Y = 0.7X + 0.53 \text{ если } X > 2.1$$

Значение X вычислено предшествующими операторами, в последующих операторах будет использовано значение Y. Другими словами, мы должны сейчас написать *участок* некоторой более обширной программы.

Основная идея заключается в том, чтобы использовать два различных арифметических оператора и с помощью оператора IF производить выбор между ними. Соответствующий участок программы изображен на рис. 1.3. Какой

```
      IF (X — 2.1) 40, 40, 30
40  Y = 0.5*X + 0.95
      GO TO 45
30  Y = 0.7*X + 0.53
45  Продолжение программы
```

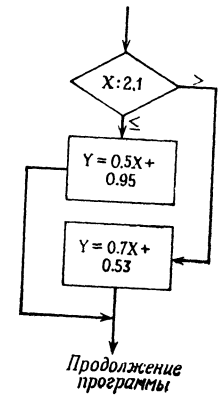
Рис. 1.3. Участок программы, позволяющий вычислить Y по одной из двух формул.

из арифметических операторов, упомянутых в операторе IF, будет выполнен, зависит от значения выражения X — 2.1. Если X меньше 2.1, то X — 2.1 отрицательно и будет выполнен оператор 40, который вычислит Y согласно формуле, соответствующей этому случаю. Если X равно 2.1, то X — 2.1 равно нулю и вычисления по-прежнему производятся с помощью оператора 40, как и требуется в этом случае. Если же X больше 2.1, то X — 2.1 положительно и вычисления будут произведены с помощью оператора 30 по формуле, соответствующей этому случаю. Каким бы ни был оператор 45, этот участок программы, вычисляющий Y как функцию от X, всегда даст к моменту начала действия оператора 45 значение Y, подсчитанное по нужной формуле.

Почему необходим оператор GO TO 45? Потому что без него значение Y было бы всегда вычислено по второй формуле! Рассмотрим, что произошло бы при отсутствии GO TO. В случае, когда X меньше или равно 2.1, мы перешли бы

к оператору 40 и вычислили бы Y по первой формуле, как и требуется, но немедленно вслед за тем был бы исполнен оператор 30, значение Y было бы вычислено по второй формуле, а результат вычисления по первой формуле был бы безвозвратно потерян. Поэтому оператор GO TO необходим для того, чтобы *обойти* оператор 30, если вычисления делались с помощью оператора 40.

Эта короткая программа достаточно проста для того, чтобы ее можно было понять, читая ее непосредственно. В дальнейшем нам придется встретиться с программами, которые гораздо легче составлять и анализировать, если ход основных вычислений изображен вначале в виде блок-схемы (см. рис. 1.4). В этой книге ромбом обозначается любое принятие решения. Двоеточие внутри ромба обозначает сравнение, а возможные последствия решения обозначаются стрелками, выходящими из ромба. Прямоугольник обозначает вычисления: в нашем случае два арифметических оператора, соответствующие двум формулам. Позднее мы будем применять в блок-схемах еще один символ — овал для операций ввода и вывода.



Р и с. 1.4. Блок схема участка программы, приведенного на рис. 1.3.

В следующих упражнениях читатель может потренироваться в различных способах применения операторов GO TO и IF.

Упражнения

В каждом из следующих упражнений начертите блок-схему и напишите участок программы, который должен соответствовать предложенной задаче. Подразумевается, что каждый такой участок является частью большей программы, и в этих упражнениях не требуется писать полную программу с операторами ввода и вывода.

***1.** Определите, какая из двух переменных, X и Y, больше другой алгебраически, и присвойте это большее значение переменной BIG. (Для тренировки не используйте существующую в некоторых системах функцию, которая позволяет делать это автоматически.) Если $X=Y$, присвойте переменной BIG значение *любой* из переменных X или Y.

2. Присвойте переменной BIG 3 значение той из переменных X, Y или Z, которая больше остальных алгебраически. (Это можно

сделать с помощью всего двух операторов IF, определив сначала наибольшую из двух переменных X и Y, присвоив это наибольшее из двух значение промежуточной переменной, название которой читатель может выбрать по своему усмотрению, а затем сравнив эту промежуточную переменную с Z, для того чтобы найти наибольшую из трех.)

3. Переменные R и S могут быть положительными или отрицательными. Присвойте переменной BIGAB значение той из переменных, которая больше по *абсолютной величине*. (Значение абсолютной величины дается функцией ABSF: так, ABSF (R) есть абсолютная величина R.)

***4.** Угол THETA положителен и меньше 30 радиан. Вычитайте 2π из THETA столько раз, сколько необходимо для того, чтобы остаток был меньше 2π. Сделайте остаток новым значением THETA.

***5.** XREAL и XIMAG есть действительная и мнимая части комплексного числа. Если *обе* эти величины меньше 1 по абсолютной величине, перейдите к оператору 81, в противном случае перейдите к оператору 82.

***6.** Если корень квадратный из суммы квадратов XREAL и XIMAG меньше 1, перейдите к оператору 187, в противном случае перейдите к оператору 459.

7. Если XREAL и XIMAG представляют точку, лежащую внутри квадрата со стороной, равной $\sqrt{2}$, с центром в начале координат и с вершинами, лежащими на осях координат, перейдите к оператору 12, в противном случае перейдите к оператору 99.

8. Y1, Y2 и Y3 суть ординаты трех точек кривой. Если Y2 является относительным максимумом, т. е. $Y2 > Y1$ и $Y2 > Y3$, перейдите к оператору 456, в противном случае перейдите к оператору 567.

9. Если DENOM по абсолютной величине меньше 10^{-5}, перейдите к оператору 50, если же больше или равен 10^{-5}, перейдите к оператору 51. Напишите соответствующие участки программы:

а. Не применяя функцию ABSF.

б. Применяя функцию ABSF.

***10.** Если $0.999 \leqslant x \leqslant 1.001$, перейдите к оператору 63, в противном случае перейдите к оператору 67. Напишите соответствующие участки программы:

а. Используя два оператора IF и не используя функцию ABSF.

б. Используя один оператор IF, содержащий в своем выражении функцию ABSF.

1.11. ОПЕРАТОРЫ PAUSE, STOP И END

Операторы PAUSE и STOP могут быть написаны в любом случае, когда необходимо остановить выполнение программы. Как правило, оператор STOP или какой-либо другой эквивалентный оператор должен стоять в конце каждой программы, когда вычисления кончаются, но у

этих операторов существуют также и другие полезные приложения. Очень часто в программу вводятся многочисленные операции контроля, как за ходом вычислений, так и за входной и выходной информацией. Если контрольные операции обнаруживают какую-либо ошибку, то выполнение программы может быть приостановлено с помощью какого-нибудь из этих операторов.

Оба оператора, PAUSE и STOP, работают сходным образом. Оба приостанавливают исполнение *только* рабочей программы, другими словами, они *не вызывают остановки* работы программы-транслятора. Оба оператора могут иметь после слова PAUSE или STOP некоторое целое *восьмеричное* число (составленное из цифр от нуля до семи), причем это число будет изображено на пульте управления ЭЦВМ в случае останова программы по данному оператору. Последнее очень полезно в тех случаях, когда в программе имеется много операторов PAUSE или STOP и программист желает знать, какой из них вызвал останов программы.

Разница между операторами STOP и PAUSE сводится к следующему: после оператора STOP нельзя быстро продолжить программу, в то время как при использовании оператора PAUSE программист может просто нажать кнопку на пульте управления и продолжить выполнение программы с оператора, следующего за оператором PAUSE.

Использование операторов PAUSE и STOP может существенно различаться в различных вариантах ФОРТРАНа и в различных ЭЦВМ. В частности, на больших вычислительных системах иногда операторы PAUSE и STOP являются недопустимыми, так как они вызывают большие потери дорогостоящего машинного времени. Иногда в таких случаях владельцы вычислительных систем видоизменяют программу-транслятор с тем, чтобы эта программа отвергала операторы PAUSE и STOP.

Следует заметить, что работа больших вычислительных систем очень часто происходит под контролем управляющей программы (*программа-резидент* или *программа-монитор*). В этих случаях программа, написанная на ФОРТРАНе, *не должна* иметь оператора STOP в конце программы, а вместо этого должна после окончания вычислений перейти на управляющую программу. Это делается различными

способами. Часто встречается способ, согласно которому необходимо в конце исходной программы на ФОРТРАНе писать оператор CALL EXIT.

До сих пор рассматривались способы окончания выполнения *рабочей* программы. Оператор END имеет другое назначение: этот оператор содержит информацию для программы-транслятора, что достигнут конец *исходной* программы и больше в этой исходной программе никаких операторов нет. Оператор END должен быть последним по порядку оператором в каждой программе, написанной на ФОРТРАНе.

Иначе говоря, оператор STOP может стоять где угодно внутри программы, и их (операторов STOP) может быть несколько, в то время как оператор END должен быть всего один и он обязан стоять на самой последней перфокарте, когда программа пробита на перфокартах.

Упражнения

В следующих упражнениях необходимо прочитать исходную информацию, произвести некоторые вычисления и напечатать результаты. Для каждого упражнения начертите блок-схему и напишите полную программу. Для ввода используйте спецификацию F10.0, для вывода — спецификацию E20.8.

*1. Прочитайте значение ANNERN, вычислите TAX, согласно нижеследующей таблице, и напечатайте ANNERN и TAX:

ANNERN (годовой заработок) . .	TAX (налог)
Меньше 2000.00 долларов	Нуль
Больше 2000.00 долларов, но меньше 5000.00 долларов	2% от суммы сверх 2000.00 долларов
Больше 5000.00 долларов	60.00 долларов плюс 5% от суммы сверх 5000.00 долларов

2. Федеральный подоходный налог США на недельную зарплату можно подсчитать следующим образом: из общей суммы недельной зарплаты вычитается 12 долларов, умноженное на количество иждивенцев данного человека; налог составляет 14% от разности. Очевидно, что существуют случаи, когда нельзя взимать налог, так как заработок человека может и не выйти из тех пределов,

которые освобождаются от налога. Прочитайте значения GROSS (общей суммы зарплаты) и DEPEND (количества иждивенцев), подсчитайте TAX (налог) и напечатайте GROSS, DEPEND и TAX.

*3. Необходимо вычислить Y как функцию X, согласно формуле

$$Y = 16.7X + 9.2X^2 - 1.02X^3$$

Ввод информации в этом упражнении не требуется; вычислите и напечатайте X и Y для значений X от 1.0 до 9.9 с шагом 0.1. В этом упражнении предположите, что вы работаете с ЭЦВМ, производящей вычисления в десятичной системе, так что последнее значение X будет равно *в точности* 9.9.

*4. То же, что и в упражнении 3, но на этот раз ЭЦВМ работает в двоичной системе. Поскольку 0.1 в двоичной системе представляется в виде бесконечной дроби, нет никакой гарантии того, что, прибавляя каждый раз к величине X двоичную дробь, являющуюся *приблизительным* представлением 0.1, мы в конце концов получим *точно* 9.9. Поэтому если начать вычисления с $X = 1.0$, прибавлять каждый раз по 0.1 и сравнивать X на равенство с 9.9, то почти наверняка точного равенства получить не удастся и не удастся выйти из этого цикла. В числе прочих возможны два следующих решения. Или необходимо сравнивать X на *приблизительное* равенство с 9.9 с точностью, например, 10^{-4}, или подсчитать число прохождений цикла с помощью дополнительной целой переменной.

5. Необходимо вычислить Y как функцию X согласно формуле

$$Y = \sqrt{1+X} + \frac{\cos 2X}{1+\sqrt{X}}$$

для набора равноотстоящих значений X. С перфокарты требуется прочитать три величины: XINIT, XINC и XFIN. (XINIT меньше, чем XFIN, XINC положительно.) Вначале подсчитывается и печатается Y для значения $X = $ XINIT. Затем X увеличивается на величину XINC, снова подсчитывается и печатается Y и т. д., пока Y не будет вычислено и напечатано для самого большого значения X, *не превосходящего* XFIN. (При такой постановке задачи вопрос окончания цикла, возникающий в двух предыдущих упражнениях, решается очень просто.)

1.12. НАПИСАНИЕ ПРОГРАММЫ, ЕЕ ПЕРФОРАЦИЯ НА ПЕРФОКАРТАХ И ПОСТАНОВКА ЕЕ НА ЭЦВМ

Программа на ФОРТРАНе обычно пишется на бланке наподобие того, что показан на рис. 1.7 в разд. 1.13. Оператор при такой записи может занимать одну или несколько строчек, причем каждый оператор начинается с новой строчки. Информация с каждой строчки затем пробивается

на перфокарте в таком виде, как это показано на рис. 1.8. Каждая строчка записи на бланке переносится на отдельную перфокарту.

Перфоратор для пробивки на картах операторов ФОРТРАНа по внешнему виду и по способу работы на нем напоминает больше всего пишущую машинку. На его клавиатуре имеются заглавные буквы латинского алфавита, цифры, точка, запятая, скобки и символы арифметических операций. Стандартные 80-колонные перфокарты проходят в перфорирующем устройстве справа налево, так же как бумага в пишущей машинке. Каждый символ изображается на перфокарте комбинацией отверстий в колонке, например буква S обозначается отверстиями в нулевой и второй строчках, а левая скобка — отверстиями в нулевой, четвертой и восьмой строчках (см. рис. 1.8).

Вместе взятые перфокарты с пробитыми на них операторами составляют *исходную программу*. Эта программа должна быть введена в машину и переработана с помощью программы-транслятора в *рабочую программу*, состоящую из команд ЭЦВМ.

Для того чтобы записывать на бланках программы, необходимо познакомиться с назначением различных частей бланка и соответствующих им частей перфокарты.

Номера, написанные над верхней строчкой бланка, соответствуют номерам колонок на перфокарте, в которых будут пробиты те или иные символы. Первые 5 колонок, с первой по пятую, содержат номер оператора (в том случае, если данному оператору присваивается номер). Примеры использования номеров встречались нам в разд. 1.9 и 1.10 в связи с операторами FORMAT, GO TO и IF.

Первая колонка выполняет другую роль. Если в этой колонке пробита буква C, то транслятором ФОРТРАНа эта карта будет игнорирована. Информация, содержащаяся на этой карте, не будет обработана при составлении рабочей программы, но будет напечатана при выдаче текста исходной программы на печать. Эта возможность позволяет вводить в программу комментарии, облегчающие ее чтение. Комментарии помогают программисту понять ранее составленную им программу, детали которой он забыл.

Шестая колонка используется для того, чтобы обозначить продолжение предыдущей карты. Если оператор умещается

целиком на одной перфокарте, то колонку 6 можно оставить пустой или отперфорировать на ней нуль. Если же для записи оператора требуется более одной карты, то все карты после первой должны иметь в колонке 6 какой-либо ненулевой символ. Рекомендуется перфорировать нуль в первой перфокарте этого «длинного» оператора и все дальнейшие перфокарты, на которых продолжается этот оператор, нумеровать в колонке 6 последовательно возрастающими цифрами. Следует заметить, что количество перфокарт, которые может занимать оператор, ограничено десятью (первая карта и девять карт продолжения).

Собственно оператор пробивается в колонках 7—72. Пробелы в этом поле игнорируются программой-транслятором; их можно свободно оставлять в любых местах для увеличения наглядности программы. Оператор вовсе не должен начинаться точно с 7-й колонки: напротив, некоторые программисты предпочитают оставлять пробел перед началом записи оператора, чтобы было ясно видно символ продолжения в 6-й колонке. Некоторые программисты оставляют пробелы по обеим сторонам символов арифметических операций, что облегчает чтение операторов. Все эти условности не играют никакой роли для работы программы-транслятора и находятся на полном усмотрении программиста[1]).

Все же необходимо всегда точно указывать тому человеку, который будет переносить вашу программу на перфокарты, какие пробелы и где вы желаете оставить. Это позволяет впоследствии проверить точность пробивки перфокарт путем сравнения их с исходным бланком. Поэтому большинство бланков для программирования на ФОРТРАНе имеет клетки или черточки для отделения различных символов друг от друга.

Колонки 73—80 не обрабатываются транслятором ФОРТРАНа и могут быть использованы для любой информации, например в этих колонках может быть пробито название программы или последовательные номера перфокарт в программе.

---

[1]) Исключение составляет оператор FORMAT: в некоторых случаях количество пробелов в его спецификации должно быть строго определено.— *Прим. ред.*

Очень важно, чтобы бланки программирования заполнялись весьма тщательно. Операторы должны быть написаны точно в установленном формате; если хотя бы одна запятая будет пропущена или поставлена не на свое место, то исходную программу нельзя будет преобразовать в рабочую. *Требуется также проявлять большое внимание, чтобы не спутать* некоторые сходные символы, такие, как, например, нуль и буква «О». Один из возможных способов написания этих символов показан на рис. 1.5.

Ноль	*0*	*Буква О*	*Ø*
Один	*1*	*Буква I*	*I*
Два	*2*	*Буква Z*	*Z*

Рис. 1.5. Пример того, как можно различать при записи близкие по написанию символы.

1.13. ПРАКТИЧЕСКИЙ ПРИМЕР 1: ПЛОЩАДЬ ТРЕУГОЛЬНИКА

Для того чтобы проиллюстрировать некоторые возможности применения методики, описанной в этой главе, возьмем какую-нибудь простую задачу и посмотрим, как можно составить на ФОРТРАНе полную программу ее решения.

Если длины сторон треугольника заданы величинами переменных A, B и C, то площадь треугольника можно вычислить по формуле

$$\mathrm{AREA} = \sqrt{S(S-A)(S-B)(S-C)},$$

где

$$S = \frac{A+B+C}{2}.$$

Значения A, B и C должны быть введены с перфокарты. Значение A пробито в колонках 1—10 с десятичной точкой, но без порядка. Значение B пробито в колонках 11—20, а значение C —в колонках 21—30 в той же форме, что и значение A. На рис. 1.6 показана примерная перфокарта с входной информацией, причем значения A, B и C равны 300.0, 400.0 и 500.0 соответственно.

После окончания вычислений необходимо напечатать длины всех трех сторон и затем площадь треугольника. Каждая выходная величина займет при печати 16 позиций, будет иметь восемь значащих цифр и порядок.

Программа для этого вычисления показана на рис. 1.7. Можно заметить, что номер первого оператора FORMAT

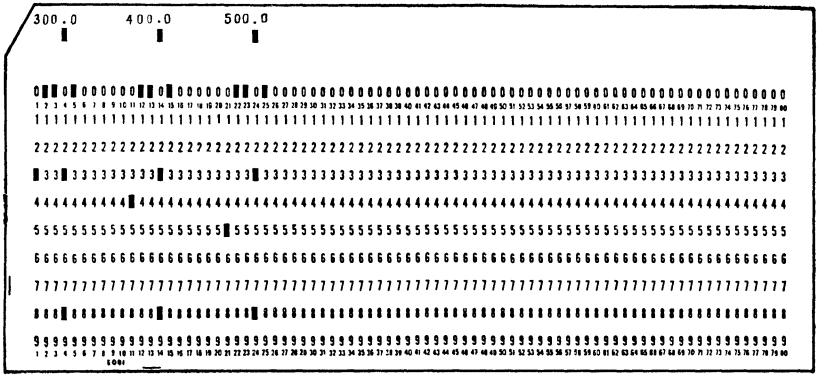


Рис. 1.6. Перфокарта с исходными данными для программы рис. 1.7 (практический пример 1).

IBM

FORTRAN CODING FORM

Program Area of triangle
Coded By D.D McCracken
Checked By

Identification TRIANGLE (73–80)

Date 4 Nov. 63
Page 1 of 1

1 5	6 7 FORTRAN STATEMENT 72
	READ 23, A, B, C
23	FORMAT (3F10.0)
	S = (A + B + C) / 2.0
	AREA = SQRTF (S * (S - A) * (S - B) * (S - C))
	PRINT 17, A, B, C, AREA
17	FORMAT (4E16.8)
	STOP
	END

Рис. 1.7. Программа для вычисления площади треугольника по длинам трех его сторон (практический пример 1).

написан в колонках 4 и 5, в то время как номер второго оператора FORMAT написан в колонках 2 и 3. Этим иллюстрируется тот факт, что номер оператора можно писать где угодно в колонках с 1-й по 5-ю, если в нем содержится меньше пяти цифр. В обоих операторах FORMAT применены коэффициенты повторения спецификации поля, так как три числа с перфокарты вводятся в одном и том же формате и четыре числа выдаются на печать также в одном и том же формате. Операторы FORMAT написаны сразу же после соответствующих операторов READ и PRINT, эта практика общепринята, хотя и не обязательна, в действительности операторы FORMAT могут быть расположены в любом месте программы. Они будут найдены по их номерам, даже если они и не следуют непосредственно за операторами READ или PRINT.

В первом операторе FORMAT применена спецификация поля F 10.0. Уже говорилось, что спецификация F применяется для ввода действительных переменных; 10 означает, что каждое число занимает на перфокарте 10 колонок, нуль означает, что если бы на перфокарте не была пробита десятичная точка, то число было бы введено в машину так, как будто все десять цифр стояли бы перед десятичной точкой. Поскольку на перфокарте десятичная точка пробита, эта часть спецификации не играет роли. Во втором операторе FORMAT буква E означает, что действительные числа будут напечатаны в нормализованном виде, 16 означает, что каждое число займет при печати 16 позиций, 8 означает, что будут напечатаны 8 десятичных значащих цифр.

Можно заметить, что знаки равенства и символы арифметических операций написаны с пробелами перед символом и после него, пробелы также оставлены после запятых в операторах READ и PRINT. Это было сделано для того, чтобы программу было легче читать; вообще говоря, этих пробелов можно было бы и не оставлять.

На рис. 1.8 показана перфокарта исходной программы, пробитая на перфораторе, описанном в разд. 1.12. На этой перфокарте записан арифметический оператор для вычисления площади треугольника.

Когда все перфокарты готовы, колода карт с исходной программой помещается в читающее устройство ЭЦВМ, и начинает работать программа-транслятор. Транслятор

перерабатывает исходную программу в рабочую, состоящую из команд ЭЦВМ. Перфокарта с входной информацией во время этой операции остается непрочитанной.

В некоторых вычислительных системах рабочая программа перфорируется на картах по мере работы транслятора и затем должна быть снова введена в машину. В других же системах программа, полученная в результате работы транслятора, остается записанной в памяти машины и может

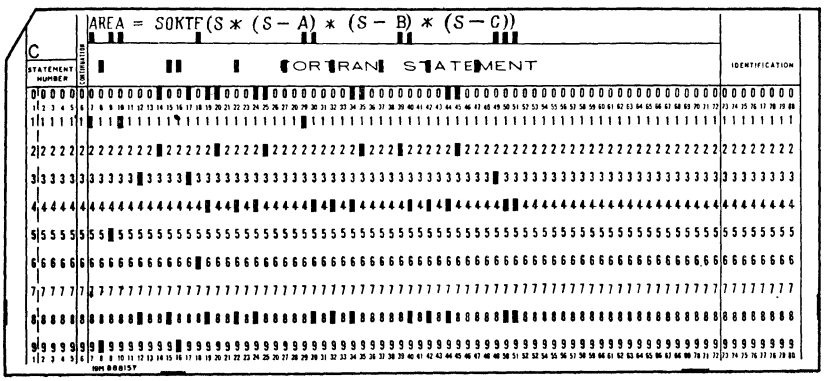


Рис. 1.8. Перфокарта из исходной программы, приведенной на рис. 1.7 (практический пример 1).

быть немедленно выполнена. (Второй метод известен под названием «непосредственных вычислений после ввода» («load and go» operation).)

0.30000000E 03 0.40000000E 03 0.50000000E 03 0.6000000E 06

Рис. 1.9. Строка печати на выходном бланке для программы рис. 1.7.

В качестве исходных данных взяты числа, отперфорированные на перфокарте рис. 1.6 (практический пример 1).

Так или иначе, начинает действовать рабочая программа. Прочитывается карта с вводимой информацией, производятся необходимые вычисления по рабочей программе, их результаты выводятся на печать, и программа останавли-

вается. На рис. 1.9 можно видеть строку напечатанных машиной выходных чисел для разобранного выше примера.

1.14. ПРАКТИЧЕСКИЙ ПРИМЕР 2: РАСЧЕТ ЦЕПИ ПЕРЕМЕННОГО ТОКА

В этом примере будет показано, как можно применить операторы GO TO и IF, если необходимо произвести серию одинаковых вычислений для нескольких вариантов набора исходных данных.

Предположим, что необходимо рассчитать ток в цепи, состоящей из последовательно соединенных сопротивления, индуктивности и емкости[1]). Ток в такой цепи определяется по формуле

$$I = \frac{E}{\sqrt{R^2 + \left(2\pi FL - \frac{1}{2\pi FC}\right)^2}},$$

где I — ток в амперах, E — напряжение в вольтах, R — сопротивление в омах, L — индуктивность в генри, C — емкость в фарадах, F — частота в герцах.

Предположим, что цель вычислений состоит в получении данных для построения графика зависимости тока в этой цепи от частоты. Поэтому напряжение, сопротивление, индуктивность и емкость будут считаться известными, а в программе будет предусмотрено считывание с перфокарт последовательных значений частоты и печать этой частоты и вычисленного для этой частоты значения тока.

Необходимо так составить программу, чтобы возможно было произвести необходимые вычисления для любого неизвестного заранее количества перфокарт со значениями частоты. Поэтому мы условимся, что последняя перфокарта

[1]) Приведенные здесь и далее практические примеры взяты из различных областей науки и техники. Как уже указывалось в начале книги, для полного решения задачи необходимы детальные знания в соответствующей области. Мы предполагаем, однако, что все предварительные этапы решения задачи уже выполнены. Таким образом, читатель не обязан знать, каким образом выведены соответствующие формулы или даже что они означают. Здесь рассматриваются в основном лишь численные методы и программирование; читатель не должен смущаться, если приводится пример из области, с которой он незнаком.

с данными будет содержать отрицательную частоту, являясь таким образом *пробной*. Все вычисления будут проведены для положительных частот, поэтому такую пробную карту можно смело употребить, не опасаясь спутать ее со всеми прочими входными картами. После чтения очередного значения частоты остается только применить оператор IF, чтобы определить, не дошла ли очередь до пробной карты. Если очередь еще не дошла, значит, с перфокарты введено очередное значение частоты и необходимо произвести вычисления. После печати очередного результата будет прочитана очередная входная перфокарта и так до тех пор, пока наконец не будет введена с пробной карты отрицательная частота и оператор IF не передаст управление оператору STOP для остановки вычислений.

При составлении программы возникает следующее несущественное затруднение. Символы, стоящие в формуле, не пригодны в качестве обозначений соответствующих переменных, так как I и L могут быть наименованиями только целых переменных, а в нашем примере все они должны быть действительными. Эта трудность возникает довольно часто, и иногда неправильная работа программы может объясняться тем, что переменным были присвоены неподходящие наименования (это приводит, как правило, к смешению арифметики целых и действительных чисел). Наиболее простое решение состоит в том, что перед неподходящими наименованиями ставится какая-нибудь буква, с которой должны начинаться правильные наименования действительных переменных. В этом примере мы воспользуемся тем, что английские названия единиц, которыми измеряются переменные, начинаются все с правильных букв, так что их можно сделать наименованиями действительных переменных.

Последовательность операций можно более наглядно изобразить с помощью блок-схемы, показанной на рис. 1.10. Сначала прочитываются значения четырех параметров, которые не будут меняться: напряжение, сопротивление, емкость и индуктивность. На блок-схеме эти величины обозначены такими же символами, как и в программе, так что довольно легко сравнивать блок-схему и программу на ФОРТРАНе.

Чтение с перфокарты величин, которые не будут меняться в ходе вычислений, производится всего один раз; все

остальные стадии придется повторять всякий раз после ввода нового значения частоты. Чтобы можно было вводить по очереди последовательные значения частоты независимо от ввода остальных параметров, используется отдельный блок. Сразу же после ввода очередного значения частоты задается вопрос (с помощью оператора IF): не является ли частота отрицательной? Отрицательная частота означает, что очередь дошла до пробной карты и необходимо остановить выполнение программы. Естественно, это не может случиться при первом прохождении цикла, и может показаться, что нецелесообразно проверять знак частоты для первой карты. Однако необходимо помнить, что цикл вычислений может повторяться *неизвестное заранее* количество раз: может оказаться одна перфокарта со значением частоты и могут оказаться сто таких карт. Поэтому проверку на пробную карту необходимо производить каждый раз, кроме первого. Если строить программу таким образом, чтобы не делать проверку первый раз и делать ее каждый следующий раз, то программа усложнится, а затраты машинного времени на вычисления будут гораздо больше, чем если делать проверку также и для первой карты.

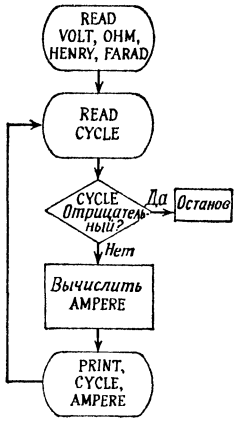


Рис. 1.10. Блок-схема программы для расчета цепи переменного тока (практический пример 2).

Если перфокарта со значением частоты не является пробной, то мы переходим к блоку вычисления тока в цепи (AMPERE). В программе этому вычислению соответствует длинный арифметический оператор; на блок-схеме формула не написана, хотя иногда программисты пишут на блок-схеме все формулы. После вычисления тока мы печатаем частоту и соответствующий ей ток и возвращаемся, чтобы прочесть с перфокарты новое значение частоты.

Рассмотренная здесь программа будет продолжать считывание с перфокарты очередного значения частоты, вычисление и печать значения тока и возвращение для считывания

новой карты данных, пока программа не будет «выведена из цикла» с помощью пробной карты.

После того как процесс вычислений детально определен блок-схемой, написать программу на ФОРТРАНе довольно

```
C     1   5 6 7  FORTRAN STATEMENT
      READ 200, VOLT, OHM, HENRY, FARAD
  200 FORMAT (4F10.0)
   23 READ 200, CYCLE
      IF (CYCLE) 600, 601, 601
  600 STOP
  601 AMPERE = VOLT/SQRT(F(OHM**2 + (6.2832*CYCLE*
     1 HENRY - 1./(6.2832*CYCLE*FARAD))**2)
      PRINT 201, CYCLE, AMPERE
  201 FORMAT (1P2E16.6)
      GO TO 23
      END
```

Р и с. 1.11. Программа для расчета цепи переменного тока (практический пример 2).

просто. Программа приведена на рис. 1.11. Номера присвоены операторам FORMAT и трем операторам, к которым возможны переходы от операторов IF и GO TO. Несколько слов об операторе IF. Если частота, введенная с перфокарты, отрицательна, то управление передается на оператор STOP, который стоит сразу же после IF. (Оператор STOP можно поместить и в другое место программы, например поставить его в конце программы, непосредственно перед END.) Если же частота равна нулю или положительна, то будут произведены вычисления. Совершенно ясно, что вычисление тока по этой программе бессмысленно проводить для частоты, равной нулю, но в операторе IF необходимо предусматривать случай, когда выражение в скобках равно нулю, потому что правила ФОРТРАНа требуют указать оператор, которому будет передано управление при *каждом из трех* возможных значений выражения в скобках оператора IF: отрицательном, нулевом и положительном.

Арифметический оператор не уместился на одной перфокарте, и поэтому использована вторая для продолжения.

Оператор GO TO определяет, что после печати значения частоты и соответствующего ей тока управление будет передано второму оператору READ, которому присвоен номер 23.

```
1.000000E 03    3.644963E-03
1.100000E 03    4.134031E-03
1.200000E 03    4.660168E-03
1.300000E 03    5.225302E-03
1.400000E 03    5.828942E-03
1.500000E 03    6.466553E-03
1.600000E 03    7.127432E-03
1.700000E 03    7.792458E-03
1.800000E 03    8.432623E-03
1.900000E 03    9.009940E-03
2.000000E 03    9.482350E-03
2.100000E 03    9.812753E-03
2.200000E 03    9.979228E-03
2.300000E 03    9.981335E-03
2.400000E 03    9.838939E-03
2.500000E 03    9.584666E-03
2.600000E 03    9.254577E-03
2.700000E 03    8.891089E-03
2.800000E 03    8.489593E-03
2.900000E 03    8.097989E-03
3.000000E 03    7.717787E-03
3.100000E 03    7.355687E-03
3.200000E 03    7.015050E-03
3.300000E 03    6.697064E-03
3.400000E 03    6.401578E-03
3.500000E 03    6.127667E-03
```

Р и с. 1.12. Печать на выходном бланке при вычислениях по программе рис. 1.11 с теми значениями параметров, которые приведены в тексте (практический пример 2).

Во втором операторе FORMAT использован коэффициент шкалы 1P, чтобы десятичная точка была напечатана между первой и второй значащими цифрами мантиссы. Порядок автоматически уменьшается при этом на 1.

Для вычислений по этой программе была составлена входная перфокарта, содержавшая следующие значения параметров: 10 *в*, 1000 *ом*, 0.1 *гн* и 0.00000005 *ф* (=0.05 *мкф*). Частота менялась от 1000 до 3500 *гц* с шагом 100 *гц*. Результаты вычислений изображены на рис. 1.12.

Когда $2\pi FL = 1/(2\pi FC)$, то выражение в скобках, стоящее под радикалом, обращается в нуль и принято говорить, что в цепи имеет место *резонанс*. На таблице выходных значений тока (рис. 1.12) хорошо прослеживается широкий резонансный пик с вершиной около 2250 *гц*.

ГЛАВА

2

ОШИБКИ

2.1. ВВЕДЕНИЕ

Анализ ошибок в численном результате должен являться непременной составной частью любого серьезного вычисления, независимо от того, производится это вычисление вручную или с помощью ЭЦВМ. Исходная информация очень редко является точной, так как часто исходные величины являются экспериментальными данными или основаны на приблизительных оценках. Кроме того, сами процессы вычислений могут вносить в результат различного рода ошибки. Поэтому, прежде чем начать систематическое изучение ошибок, убедимся на нескольких примерах в важности такого систематического изучения.

В упражнении 18 в конце этой главы предлагается найти один из корней уравнения $x^2 + 0.4002\,x + 0.00008 = 0$, используя вычисления с точностью до 4-й значащей цифры. Используя обычную формулу

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a},$$

мы получаем ответ —0.00015. Эта формула обычно приводится в курсах алгебры без всяких оценок точности, однако ограничение точности вычислений четырьмя значащими цифрами приводит к тому, что результат имеет ошибку 25%; правильный корень, найденный при вычислениях с восемью значащими цифрами, равен —0.0002.

В данном случае камнем преткновения оказалась четырехзначная арифметика, но пусть читатель не думает, что восемь значащих цифр решат все проблемы. Рассмотрим ряд Тейлора для синуса:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots .$$

Считается, что этот ряд годится для любого конечного угла, а ошибка, происходящая от ограничения ряда конечным числом членов, не превосходит по модулю первого отброшенного члена ряда. Все это было бы справедливо, *если бы существовал способ производить арифметические вычисления с бесконечным числом значащих цифр*. В практическом примере 3 в гл. 3 мы увидим, что для больших углов этот ряд совершенно бесполезен. Предположим, что нужно вычислить синус 1470° (приблизительно 25.7 радиана), и мы намерены производить вычисления с восемью значащими цифрами и остановиться тогда, когда очередной член ряда будет по абсолютной величине меньше 10^{-8}. Выданный машиной результат будет равен 24.25401855, с большим числом десятичных знаков и совершенно без всякого смысла. Даже если производить вычисления с 16 значащими цифрами, то вместо синуса 2550° машина выдаст число 29.5.

В приведенных выше примерах трудности были обусловлены тем, что вычисления производились с конечным числом значащих цифр. Существуют и другого рода источники ошибок. Например, рассмотрим следующую систему уравнений:

$$5x - 331y = 3.5$$

$$6x - 397y = 5.2$$

Довольно легко найти «точное» решение этой системы: $x = 331.7$, $y = 5.000$, причем решение может быть найдено вручную с любым нужным количеством значащих цифр. Попытаемся оценить количество достоверных значащих цифр в этом решении. Проверим сначала, что случится, если константу в правой части второго уравнения изменить с 5.2 до 5.1, т. е. приблизительно на 2%. При этом решение системы становится равным $x = 298.6$, $y = 4.5$. Этот факт может вызвать беспокойство: изменение исходной величины на 2% приводит к изменению результата на 10%? Еще более поразительно то, что если подставить в уравнения $x = 358.173$, $y = 5.4$, то при округлении вычисленные значения левых частей дадут те же самые правые части, что и в исходной системе уравнений. По-видимому, в этом случае можно считать, что величины x и y имеют не больше одной достоверной значащей цифры.

В данном случае неточность результата не зависит от количества значащих цифр, все вычисления были проделаны точно. Причина неточности в данном случае — *малая величина определителя системы уравнений*. Геометрически это означает, что две прямые линии, представленные указанными уравнениями, почти параллельны.

Наконец, рассмотрим интеграл

$$\int_{e^{-4}}^{1} \frac{dx}{x}.$$

Этот интеграл легко вычислить точно; его значение равно 4.00. Если же произвести численное интегрирование, воспользовавшись формулой трапеций и разбив интервал интегрирования на 10 частей, то результат будет равен 5.3. Даже разбивая интервал на 40 частей, получаем 4.13, т. е. ошибку приблизительно в 3%.

На этот раз причиной трудностей является поведение подинтегральной функции, сильно возрастающей при уменьшении x, а также неточная формула для численного интегрирования. В данном случае даже точные исходные данные и точные вычисления приводят к ошибке, обусловленной численным методом.

Из приведенных примеров становится ясно, что без анализа ошибок вычисления нельзя сказать что-либо определенное относительно точности результата. Иногда, конечно, можно уже при составлении программы быть уверенным, что никаких специфических проблем, связанных с точностью результата, в задаче не встретится, например в первых двух практических примерах в гл. 4 будет коротко показано, что в этих случаях вопросы точности не возникают. Вообще говоря, такие случаи являются скорее исключением, чем правилом.

Материал, изложенный в этой главе, представляет интерес при анализе результатов простых арифметических вычислений. На основе этого материала в дальнейшем будет проводиться анализ ошибок различных численных методов, описанных в последующих главах. Анализ ошибок — это как раз тот раздел, с которого мы начнем изучение численных методов.

2.2. ОТНОСИТЕЛЬНЫЕ И АБСОЛЮТНЫЕ ОШИБКИ

Относительные и абсолютные ошибки определяются следующим образом. *Абсолютная ошибка* есть разность между истинным значением величины (считая это истинное значение известным) и ее приближенным значением. Обычно приближенное значение некоторой величины, или *приближение*, обозначается тем же символом, что и точное значение, только сверху этого символа ставится черта; ошибка же обозначается буквой e с символом приближаемой величины вместо индекса. Таким образом, если точное значение равно x, то мы должны написать

$$x = \bar{x} + e_x.$$

Здесь e_x есть абсолютная ошибка, определяемая как разность между точным значением и приближением:

$$e_x = x - \bar{x}.$$

Относительная ошибка определяется, как отношение абсолютной ошибки к приближению. Казалось бы, что более естественно определить ее, как отношение абсолютной ошибки к точному значению, но обычно точное значение нам неизвестно. Все, что обычно бывает известно,— это приближенное значение величины и *оценка* ошибки или *границы* максимально возможной величины ошибки. Если ошибка мала, то разница в определениях не скажется на численной величине относительной ошибки.

Для величин, близких по значению к единице, абсолютная и относительная ошибки почти одинаковы. Для очень больших или для очень малых величин относительная и абсолютная ошибки представляются совершенно разными числами. Так, если точное значение некоторой величины равно 0.00006, а приближенное значение равно 0.00005, то абсолютная ошибка составляет всего 10^{-5}, в то время как относительная ошибка составляет 0.2, или 20%. С другой стороны, если точное значение равно 100 500, а приближенное значение равно 100 000, то абсолютная ошибка составляет 500, хотя относительная ошибка составляет всего 0.005, или 0.5%.

Необходимо всегда указывать, какая ошибка имеется в виду — абсолютная или относительная, —если это не ясно из условий задачи или из контекста.

2.3. ОШИБКИ, СОДЕРЖАЩИЕСЯ В ИСХОДНОЙ ИНФОРМАЦИИ

В процессе численного решения некоторой задачи приходится иметь дело с тремя основными видами ошибок: ошибками, содержащимися в исходной информации, ошибками, возникающими при ограничении бесконечного математического процесса конечным числом операций (ошибки ограничения), и ошибками, возникающими в результате необходимости представлять число в виде конечной последовательности цифр (ошибки округления). Каждую из этих ошибок можно представить в абсолютной и относительной форме.

Ошибки в исходной информации возникают в результате неточности измерений, грубых просмотров или из-за невозможности представить необходимую величину конечной дробью.

Всякое физическое измерение, будь то измерение расстояния, напряжения или интервала времени, не может быть выполнено абсолютно точно. Если, например, указано, что величина напряжения составляет 6.4837569 *в*, то можно с уверенностью сказать, что по меньшей мере несколько младших значащих цифр недостоверны, потому что невозможно измерить напряжение с такой точностью. Если же экспериментальный результат содержит небольшое количество значащих цифр (например, промежуток времени в 2.3 *сек*), то можно быть абсолютно уверенным в том, что эта величина дана с некоторой ошибкой, так как лишь случайно величина интервала времени может составить в точности 2.3 *сек*. В таких случаях подразумеваются некоторые границы, внутри которых эта величина должна находиться, что-либо вроде 2.3 ± 0.1 *сек*.

Иногда подразумевается, что если для экспериментального результата не указаны его возможные границы, то результат имеет точность половины единицы младшего разряда. Поэтому если дано, что некая длина равна 5.63 *см*, то это следует понимать так, что эта длина не меньше 5.625 и не больше 5.635 *см*. Однако это правило не всегда соблю-

дается; поэтому, когда границы точности результата важны, их следует указать в явном виде, например 5.63 ± 0.005 *см*.

Независимо от количества значащих цифр в какой-либо величине, в ней может содержаться порой какая-нибудь грубая ошибка. Грубые ошибки могут возникнуть от опечаток, от ошибочного отсчета показаний прибора, хотя встречаются иногда и такие ошибки, возникновение которых связано с некорректной постановкой задачи или с неполным пониманием некоторых физических законов.

Многие числа нельзя представить точно ограниченным числом значащих цифр. Если в вычислениях используется число π, то оно может быть представлено в виде 3.14, или 3.14159265, или 3.141592653589793, в зависимости от того, какая точность требуется в данном вычислении. В любом случае, однако, невозможно представить π *точно*, так как π является иррациональным числом и не может быть представлено конечным числом знаков. Даже обыкновенные дроби очень часто нельзя представить с помощью конечного числа десятичных знаков, как 1/3, которую можно представить только в виде периодической дроби.

Часто случается также, что дроби, которые являются конечными в одной системе счисления, становятся бесконечными в другой системе счисления. Например, дробь 1/10 явно имеет конечное десятичное представление 0.1, но, будучи переведена в двоичную систему счисления, становится бесконечной дробью 0.000110011001100.... Вычисляя сумму десяти чисел, каждое из которых будет представлять собой двоичное приближение к десятичной 0.1, мы в сумме *не получим* точно 1. Известно, что начинающие программисты иногда приходят в замешательство, сталкиваясь впервые с такими «чудесами». Трудность эта неизбежна, но легко преодолима, как мы убедимся ниже на некоторых практических примерах.

2.4. ОШИБКИ ОГРАНИЧЕНИЯ

Те ошибки, которые содержатся в исходной информации, определяют точность результата вычислений *независимо* от того, каким методом эти вычисления проводятся. Два других типа ошибок — ошибки ограничения и ошибки

округления — определяются теми численными методами, которые были использованы для решения задачи.

Общеизвестный ряд Тейлора для синуса

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

может использоваться для вычисления синуса любого угла x, выраженного в радианах [1]). Конечно, невозможно использовать все члены ряда для вычислений, так как ряд бесконечен; вычисления ограничиваются конечным числом членов: например, до x^7 или x^9. Отброшенные члены ряда (а их число бесконечно) вносят некоторую ошибку в результат вычислений. Эта ошибка называется ошибкой ограничения, так как она возникает в результате ограничения бесконечного математического процесса.

Очень многие процессы, используемые при вычислениях, являются бесконечными, так что анализ ошибок ограничения очень важен. Этот анализ будет подробно проводиться в последующих главах в связи с теми или иными численными методами.

2.5. ОШИБКИ ОКРУГЛЕНИЯ

Даже если предположить, что исходная информация не содержит никаких ошибок и все вычислительные процессы конечны и не приводят к ошибкам ограничения, то все равно в этом случае присутствует третий тип ошибок — ошибки округления. Предположим, что вычисления производятся на машине, в которой каждое число представляется пятью значащими цифрами, и что необходимо сложить два числа 9.2654 и 7.1625, причем эти два числа являются точными. Сумма их равна 16.4279, она содержит шесть значащих цифр и не помещается в разрядной сетке нашей гипотетической машины. Поэтому шестизначный результат будет *округлен* до 16.428, и при этом возникает *ошибка округления*. Так как вычислительные машины всегда работают с конечным количеством значащих цифр, то потребность в округлении возникает довольно часто.

[1]) Однако, как мы увидим в гл. 3, это далеко не всегда лучший способ вычисления синуса угла.

Вопросы округления относятся только к действительным числам. При выполнении арифметических операций с целыми числами потребность в округлении не возникает. Сумма, разность и произведение целых чисел сами являются целыми числами; если результат слишком велик, то это свидетельствует об ошибке в программе и результат при этом не округляется. Частное от деления двух целых чисел не всегда является целым числом, но, как об этом уже говорилось в гл. 1, деление двух целых чисел всегда выполняется таким образом, что дробная часть просто игнорируется. Кроме того, в подавляющем большинстве научных и инженерных расчетов деление целых чисел вообще не используется.

Поскольку будет рассматриваться округление действительных чисел, необходимо вспомнить форму их записи и договориться о некоторых условностях. Напомним, что действительное число можно представить в виде дроби, которую часто также называют мантиссой, умноженной на целую степень 10, называемую порядком. Например:

$$0.7392 \cdot 10^4 \quad (=7392.)$$
$$0.3246 \cdot 10^2 \quad (=32.46)$$
$$0.1627 \cdot 10^{-3} \, (=0.0001627)$$

Действительное число называется *нормализованным*, если первая значащая цифра мантиссы не равна нулю. В дальнейшем мы будем предполагать, что все действительные числа нормализованы.

Если обозначить мантиссу действительного числа x через f, а его порядок через e, то в общей форме число запишется следующим образом:

$$x = f \cdot 10^e.$$

Величина f не может быть меньше 1/10, так как все числа должны быть нормализованными; эта величина не может также быть больше 1, так как мантисса должна быть правильной дробью.

Результат выполнения арифметической операции в общем случае состоит из двух частей: старшей и младшей. Например, предположим, что необходимо сложить два действительных числа и что в ячейке памяти вычислительной машины мантисса может представляться четырьмя

значащими цифрами, а порядок — одной цифрой:

$$0.1624 \cdot 10^3 \,(=162.4),$$
$$0.1769 \cdot 10^1 \,(=1.769).$$

Выше уже говорилось (см. гл. 1), что ЭЦВМ автоматически выравнивает порядки действительных чисел при сложении и вычитании. Это означает, что порядки сначала сравниваются, а затем мантисса меньшего по абсолютной величине числа сдвигается вправо на столько разрядов, сколько необходимо для того, чтобы порядки стали одинаковыми. В нашем случае получается следующий результат:

$$0.1624 \cdot 10^3$$
$$0.001769 \cdot 10^3$$

Другими словами, мантисса числа с меньшим порядком сдвигается вправо на число разрядов, равное разности между порядками. Теперь можно непосредственно сложить две мантиссы.

Очевидно, что сумма будет иметь больше четырех значащих цифр в мантиссе. Результат до округления можно представить в виде суммы двух действительных чисел:

$$\begin{array}{l} 0.1624 \cdot 10^3 \\ 0.001769 \cdot 10^3 \\ \hline 0.164169 \cdot 10^3 = 0.1641 \cdot 10^3 + 0.6900 \cdot 10^{-1} \end{array}$$

Любая из четырех арифметических операций даст в результате число, которое можно представить в виде двух аналогичных слагаемых. В общем виде перед округлением результат арифметической операции можно записать так:

$$y = f_y \cdot 10^e + g_y \cdot 10^{e-t}$$

Здесь f_y имеет t значащих цифр. Диапазон возможных значений f_y, как мы условились, лежит между 0.1 и 1.0 ($0.1 \leqslant f_y < 1.0$). Для g_y это не так, потому что g_y может и не быть нормализованным числом; в частности, g_y может оказаться равным нулю. Для g_y диапазон возможных значений составляет $0 \leqslant g_y < 1.0$.

Теперь мы подходим к двум вопросам первостепенной важности: согласно каким правилам следует учитывать

величину g_y при изменении f_y и какова максимально возможная ошибка округления величины $\bar{y}$ для каждого из таких правил?

Обычно «округление» означает, что с величиной f_y производится какое-то действие, зависящее от величины g_y. В общем случае необходимо рассматривать и такую возможность, когда никакого действия не производится, т. е. когда g_y просто *отбрасывается*. Соответствующий метод округления называется отбрасыванием младших разрядов.

Очень большое число трансляторов ФОРТРАН, находящихся в употреблении в момент написания данной книги, *организовано таким образом, что рабочие программы, составленные с их помощью, используют правило отбрасывания* g_y. Этот способ округления вносит бóльшие ошибки, чем обычное округление (так называемое симметричное округление), как мы убедимся ниже. С другой стороны, если использовать обычное правило округления при выполнении каждой арифметической операции даже там, где в нем нет никакой необходимости, то это приводит к неоправданным затратам машинного времени. Поэтому многие составители программ-трансляторов резонно решили, что уменьшение точности, вызванное отбрасыванием, достаточно мало и не перевешивает тех выгод, которые приобретаются в результате упрощения рабочих программ и экономии машинного времени.

Довольно легко определить максимально возможную относительную ошибку для случая отбрасывания g_y. Максимальная ошибка возникает тогда, когда g_y велико, а f_y мало. Максимально возможное значение g_y равно 1.0, минимально возможное значение f_y равно 0.1. Величина относительной ошибки (без учета знака) выразится так:

$$\left|\frac{e_y}{\bar{y}}\right| = \left|\frac{g_y \cdot 10^{e-t}}{f_y \cdot 10^{e}}\right| \leqslant \frac{1 \cdot 10^{e-t}}{0.1 \cdot 10^{e}} = 10^{-t+1}$$

Поскольку мы условились, что t есть число значащих цифр в любом действительном числе, то этот результат можно трактовать следующим образом: при использовании правила отбрасывания максимальная относительная ошибка округления действительного числа не зависит от величины этого числа, а зависит только от количества значащих цифр в ячейке памяти ЭЦВМ. Таким образом, получается твер-

дая оценка величины относительной ошибки для вычислений с действительными числами.

Более общепринятый способ округления, так называемое симметричное округление, можно описать следующим образом: если две части арифметического результата обозначены так же, как и раньше, то округленный результат записывается так:

$$|\bar{y}| = \begin{cases} |f_y| \cdot 10^e, & \text{если } |g_y| < \frac{1}{2}, \\ |f_y| \cdot 10^e + 10^{e-t}, & \text{если } |g_y| \geqslant \frac{1}{2}, \end{cases}$$

где $\bar{y}$ имеет тот же знак, что и f_y. Прибавление 10^{e-t} во второй строке формулы соответствует прибавлению 1 к самому младшему разряду, если отброшенное число начинается с цифры 5 или с большей.

Если $|g_y| < \frac{1}{2}$, то абсолютная ошибка равна

$$|e_y| = |g_y| \cdot 10^{e-t}.$$

Если $|g_y| \geqslant \frac{1}{2}$, то абсолютная ошибка равна

$$|e_y| = |1 - g_y| \cdot 10^{e-t}.$$

В обоих случаях 10^{e-t} умножается на число, не большее 1/2 по абсолютной величине. Максимально возможное значение модуля абсолютной ошибки для этого способа округления составляет

$$|e_y| \leqslant \frac{1}{2} \cdot 10^{e-t}.$$

Теперь легко найти максимально возможную относительную ошибку:

$$\left|\frac{e_y}{\bar{y}}\right| \leqslant \left|\frac{\frac{1}{2} \cdot 10^{e-t}}{f_y \cdot 10^e}\right| \leqslant \left|\frac{\frac{1}{2} \cdot 10^{e-t}}{0.1 \cdot 10^e}\right| = 5. \cdot 10^{-t} = \frac{1}{2} \cdot 10^{-t+1}.$$

Иногда применяется более точное правило округления, учитывающее случай $g = 1/2$: f_y не изменяется, если ее последняя цифра четная, и округляется, если эта цифра нечетная. Это несколько усложняет систему и редко используется. Таким образом, мы предположим, что округление производится обычным образом, без учета случая $g_y = 1/2$.

Чтобы проиллюстрировать разницу между двумя способами округления, рассмотрим следующую арифметическую

операцию:

$$y = 0.7324 \cdot 10^3 + 0.8261 \cdot 10^{-1}$$

Для случая отбрасывания

$$\bar{y} = 0.7324 \cdot 10^3$$

и

$$\left| \frac{e_y}{\bar{y}} \right| = \frac{0.8261 \cdot 10^{-1}}{0.7324 \cdot 10^3} \approx 1.1 \cdot 10^{-4}$$

В случае симметричного округления

$$\bar{y} = 0.7325 \cdot 10^3; \quad e_y = -0.1739 \cdot 10^{-1}$$

и

$$\left| \frac{e_y}{\bar{y}} \right| = \frac{0.1739 \cdot 10^{-1}}{0.7325 \cdot 10^3} \approx 0.24 \cdot 10^{-4}$$

Ошибка симметричного округления в этом примере существенно меньше ошибки отбрасывания. Вообще, ошибка симметричного округления никогда не превосходит ошибки отбрасывания и в среднем вдвое меньше последней.

И та и другая ошибки гораздо меньше своих максимально возможных значений (10^{-3} для отбрасывания и $5 \cdot 10^{-4}$ для симметричного округления). Может случиться даже так, что ошибка округления будет равна нулю. Вообще говоря, обычно бывает известен только верхний предел возможной ошибки, но не она сама. Для полной уверенности всегда необходимо предполагать самое худшее, т. е. ситуацию, когда ошибка равна своему верхнему пределу. Более близкий к истине результат можно получить, введя некую «среднюю» ошибку и пользуясь методами математической статистики для определения наиболее вероятной величины ошибки в вычислениях. В этой книге, однако, такие методы рассматриваться не будут.

Рассмотрение ошибок округления проведено здесь для действительных десятичных чисел. Многие ЭЦВМ работают с действительными двоичными числами, т. е. с числами, основанием которых является число 2, а не число 10. В таком случае каждое действительное число будет представлено в виде двоичной дроби, умноженной на целую степень 2:

$$\bar{x} = f \cdot 2^e, \quad \frac{1}{2} \leqslant |f| < 1.$$

Анализ, подобный тому, который был проведен для десятичных чисел, позволяет показать, что максимально возможные значения ошибок округления составляют $2 \cdot 2^{-t}$ для отбрасывания и 2^{-t} для симметричного округления.

Некоторые ЭЦВМ работают с шестнадцатиричными числами, т. е. с числами, построенными по основанию 16. В этом случае максимальные ошибки составляют $16 \cdot 16^{-t}$ для отбрасывания и $8 \cdot 16^{-t}$ для симметричного округления.

В дальнейшем все вопросы, касающиеся округления действительных чисел, будут рассматриваться для случая десятичных чисел, но все сказанное будет с несущественными изменениями применимо к числам, выраженным в другой системе счисления.

2.6. РАСПРОСТРАНЕНИЕ ОШИБОК

Одним из наиболее важных вопросов в численном анализе является вопрос о том, как ошибка, возникшая в определенном месте в ходе вычислений, распространяется дальше, т. е. становится ли ее влияние больше или меньше по мере того, как производятся последующие операции. Крайним случаем является вычитание двух почти равных чисел: даже при очень маленьких ошибках обоих этих чисел относительная ошибка разности может оказаться очень большой. Эта большая относительная ошибка будет распространяться дальше при выполнении всех последующих арифметических операций.

В качестве первого шага при рассмотрении этого важного вопроса необходимо найти выражения для абсолютной и относительной ошибок результата каждого из четырех арифметических действий как функции величин, участвующих в операции, и их ошибок.

Сложение

Имеются два приближения $\bar{x}$ и $\bar{y}$ к двум величинам x и y, а также соответствующие абсолютные ошибки e_x и e_y. Тогда в результате сложения имеем

$$x + y = \bar{x} + e_x + \bar{y} + e_y = (\bar{x} + \bar{y}) + (e_x + e_y).$$

Ошибка суммы, которую мы обозначим через e_{x+y}, будет равна

$$e_{x+y} = e_x + e_y.$$

Вычитание

Тем же путем получаем

$$e_{x-y} = e_x - e_y.$$

Умножение

При умножении мы имеем

$$x \cdot y = (\bar{x} + e_x) \cdot (\bar{y} + e_y) = \bar{x}\bar{y} + \bar{x}e_y + \bar{y}e_x + e_x e_y.$$

Поскольку ошибки обычно гораздо меньше самих величин, пренебрегаем произведением ошибок:

$$x \cdot y \approx \bar{x}\bar{y} + \bar{x}e_y + \bar{y}e_x.$$

Ошибка произведения будет равна

$$e_{xy} \approx \bar{x}e_y + \bar{y}e_x.$$

Деление

Имеем

$$\frac{x}{y} = \frac{\bar{x} + e_x}{\bar{y} + e_y}.$$

Преобразовываем это выражение к виду

$$\frac{x}{y} = \frac{\bar{x} + e_x}{\bar{y}} \cdot \left(\frac{1}{1 + \frac{e_y}{\bar{y}}} \right).$$

Множитель, стоящий в скобках, при $\left| \frac{e_y}{\bar{y}} \right| \ll 1$ можно разложить в ряд

$$\frac{x}{y} = \frac{\bar{x} + e_x}{\bar{y}} \cdot \left(1 - \frac{e_y}{\bar{y}} + \left(\frac{e_y}{\bar{y}} \right)^2 - \dots \right).$$

Перемножая и пренебрегая всеми членами, которые содержат произведения ошибок или степени ошибок выше первой, имеем

$$\frac{x}{y} \approx \frac{\bar{x}}{\bar{y}} + \frac{e_x}{\bar{y}} - \frac{\bar{x}}{\bar{y}^2} e_y.$$

Следовательно,

$$e_{x/y} \approx \frac{1}{\bar{y}} e_x - \frac{\bar{x}}{\bar{y}^2} e_y.$$

В качестве иллюстрации, которая может помочь наглядно представить себе смысл этих формул, рассмотрим сложение двух четырехзначных логарифмов. Так как оба логарифма точны до четвертого знака, то ошибка каждого из них не более 0.00005. Ошибка суммы не может быть больше 0.0001. Конечно, это не означает, что ошибка действительно столь велика, но означает только то, что ошибка *может* достигать такой величины.

Необходимо четко представлять себе, что знак ошибки бывает известен только в очень редких случаях. Не следует думать, например, что ошибка увеличивается при сложении и уменьшается при вычитании потому, что в формуле для сложения стоит плюс, а для вычитания — минус. Если, например, ошибки двух чисел имеют противоположные знаки, то дело будет обстоять как раз наоборот, т. е. ошибка уменьшится при сложении и увеличится при вычитании этих чисел.

После того как мы вывели формулы для распространения абсолютных ошибок при четырех арифметических действиях, довольно просто вывести соответствующие формулы для относительных ошибок. Для сложения и вычитания формулы были преобразованы с тем, чтобы в них входила в явном виде относительная ошибка каждого исходного числа.

Сложение

$$\frac{e_{x+y}}{\bar{x}+\bar{y}} = \frac{\bar{x}}{\bar{x}+\bar{y}}\left(\frac{e_x}{\bar{x}}\right) + \frac{\bar{y}}{\bar{x}+\bar{y}}\left(\frac{e_y}{\bar{y}}\right).$$

Вычитание

$$\frac{e_{x-y}}{\bar{x}-\bar{y}} = \frac{\bar{x}}{\bar{x}-\bar{y}}\left(\frac{e_x}{\bar{x}}\right) - \frac{\bar{y}}{\bar{x}-\bar{y}}\left(\frac{e_y}{\bar{y}}\right).$$

Умножение

$$\frac{e_{x\cdot y}}{\bar{x}\cdot\bar{y}} = \frac{e_x}{\bar{x}} + \frac{e_y}{\bar{y}}.$$

Деление

$$\frac{e_{x/y}}{\bar{x}/\bar{y}} = \frac{e_x}{\bar{x}} - \frac{e_y}{\bar{y}}.$$

Очень важно четко понимать смысл этих формул распространения ошибок. Мы начинаем арифметическую операцию, имея в своем распоряжении два приближенных значения $\bar{x}$ и $\bar{y}$ с соответствующими ошибками e_x и e_y. Ошибки эти могут быть любого происхождения. Величины $\bar{x}$ и $\bar{y}$ могут быть экспериментальными результатами, содержащими ошибки; они могут быть результатами предварительного вычисления согласно какому-либо бесконечному процессу и поэтому могут содержать ошибки ограничения; они могут быть результатами предшествующих арифметических операций и могут содержать ошибки округления. Естественно, они могут также содержать в различных комбинациях и все три вида ошибок.

Вышеприведенные формулы дают выражение ошибки результата каждого из четырех арифметических действий как функции от $\bar{x}$, $\bar{y}$, e_x, e_y; ошибка округления в данном арифметическом действии при этом *не учитывается*. Если же в дальнейшем необходимо будет подсчитать, как распространяется в последующих арифметических операциях ошибка этого результата, то необходимо к вычисленной по одной из четырех формул ошибке результата *прибавить отдельно ошибку округления*.

Мы часто будем в дальнейшем писать x без черточки сверху, хотя для полной строгости следовало бы писать эту черточку. Дело в том, что из контекста всегда в этих случаях будет ясно, что мы имеем дело с приближениями, а не с точными величинами.

Проиллюстрируем распространение ошибки на конкретном примере. Предположим, что перед началом вычислений имеются три числа x, y и z, и для упрощения вопроса предположим, что исходные числа не содержат никаких ошибок. Необходимо вычислить

$$u = (x + y) \cdot z.$$

Согласно тому, как написано выражение, сначала должно быть выполнено сложение. Оба слагаемых не содержат никаких ошибок, поэтому ошибка, распространяющаяся через сложение, равна нулю; однако при выполнении операции сложения будет введена ошибка округления. Эта ошибка округления уже будет содержаться в сумме перед началом выполнения операции умножения. Обозначим через

e_{x+y} общую ошибку суммы. Тогда

$$\left|\frac{e_{x+y}}{x+y}\right| \leqslant 5\cdot 10^{-t}.$$

Число в правой части этой формулы является просто верхним пределом ошибки округления для любой арифметической операции при симметричном округлении. В данном случае по-прежнему предполагается, что действительное число в ЭЦВМ представляется в виде десятичной дроби с t значащими цифрами.

Мы знаем, что относительная ошибка произведения равна сумме относительных ошибок сомножителей плюс ошибка округления при умножении. Так как результат умножения равен $\bar{u}$, нашему приближению к u, то можно написать

$$\frac{e_u}{\bar{u}} = \frac{e_{x+y}}{x+y} + \frac{e_z}{z} + r_m,$$

где e_z/z равно относительной ошибке z и r_m равно ошибке округления при умножении. Но мы предполагали, что ошибка z равна нулю, а поэтому

$$\left|\frac{e_u}{\bar{u}}\right| = \left|\frac{e_{x+y}}{x+y} + r_m\right| \leqslant \left|\frac{e_{x+y}}{x+y}\right| + |r_m|.$$

(Последнее неравенство называется неравенством треугольника: равенство имеет место, когда $e_{x+y}/(x+y)$ и r_m имеют одинаковые знаки, а неравенство— тогда, когда они имеют разные знаки.) Поэтому можно написать

$$\left|\frac{e_u}{\bar{u}}\right| \leqslant 5\cdot 10^{-t} + 5\cdot 10^{-t}.$$

Так как по окончании вычислений нам известно $\bar{u}$, то легко можно найти максимально возможную абсолютную ошибку:

$$|e_u| \leqslant |\bar{u}|\cdot 10^{-t+1}.$$

2.7. ГРАФЫ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ

Выше были выведены формулы для распространения ошибок при выполнении арифметических операций и был рассмотрен пример того, как можно оценить общую ошибку в вычислении. Теперь рассмотрим более удобный способ

подсчета распространения ошибки в каком-либо арифметическом вычислении.

С этой целью мы будем изображать последовательность операций в вычислении с помощью так называемого *графа* и будем писать около стрелок графа коэффициенты, которые позволят нам сравнительно легко определить общую ошибку окончательного результата. Метод этот удобен еще и тем, что позволяет легко определить вклад любой ошибки, возникшей в процессе вычислений, в общую ошибку.

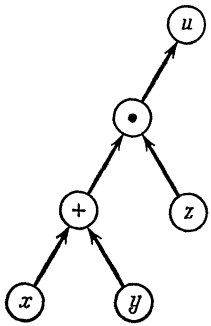


Рис. 2.1. Граф вычислительного процесса $u = (x + y) \cdot z$.

На рис. 2.1 изображен граф вычислительного процесса, соответствующий примеру из предыдущего раздела, $u = (x + y) \cdot z$. Граф следует читать снизу вверх, следуя стрелкам. Сначала выполняются операции, расположенные на каком-либо горизонтальном уровне, после этого— операции, расположенные на более высоком уровне, и т. д. Из рис. 2.1, например, ясно, что x и y сначала складываются, а потом умножаются на z. Граф, изображенный на рис. 2.1, является только изображением самого вычислительного процесса. Для подсчета общей ошибки результата необходимо дополнить этот граф коэффициентами, которые пишутся около стрелок согласно следующим правилам.

Сложение

Пусть две стрелки, которые входят в кружок сложения, выходят из двух кружков с величинами a_1 и a_2. Эти величины могут быть как исходными, так и результатами предыдущих вычислений. Тогда стрелка, ведущая от a_1 к знаку $+$ в кружке, получает коэффициент $a_1/(a_1 + a_2)$, стрелка же, ведущая от a_2 к знаку $+$ в кружке, получает коэффициент $a_2/(a_1 + a_2)$.

Вычитание

Если выполняется операция $a_1 - a_2$, то соответствующие стрелки получают коэффициенты $a_1/(a_1 - a_2)$ и $-a_2/(a_1 - a_2)$.

Умножение

Обе стрелки, входящие в кружок умножения, получают коэффициент $+1$.

Деление

Если выполняется деление a_1/a_2, то стрелка от a_1 к косой черте в кружке получает коэффициент $+1$, а стрелка от a_2 к косой черте в кружке получает коэффициент -1.

Смысл всех этих коэффициентов следующий: *относительная ошибка результата любой операции (кружка) входит в результат следующей операции, умножаясь на коэффициент у стрелки, соединяющей эти две операции.*

В качестве примера можно рассмотреть рис. 2.2, который отличается от 2.1 только тем, что около стрелок расставлены соответствующие коэффициенты.

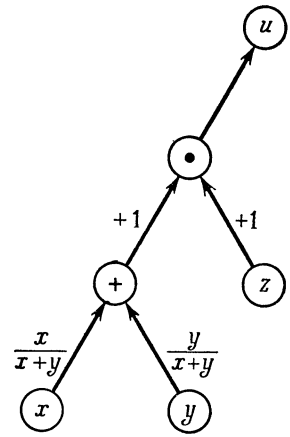


Рис. 2.2. Граф вычислительного процесса, отличающийся от графа рис. 2.1 тем, что около стрелок поставлены коэффициенты распространения ошибок.

Предположим, что три исходные величины на рис. 2.1 имеют относительные ошибки округления, равные соответственно i_x, i_y и i_z, и посмотрим, как применяется правило подсчета ошибки. Сначала рассмотрим сложение. Относительная ошибка величины x составляет i_x; эта ошибка войдет в результат следующей операции (сложения) умноженной на коэффициент у стрелки, соединяющей x в кружке со знаком $+$ в кружке:

$$\frac{x}{x+y}\,i_x.$$

В последнем выражении были опущены черточки над x и y; тем не менее подразумевается, что эти величины являются приближенными. Аналогично, относительная ошибка y, равная i_y, войдет в результат операции сложения умноженной на коэффициент при стрелке, соединяющей y в кружке

со знаком + в кружке:

$$\frac{y}{x+y}\,i_y.$$

Наконец, при выполнении операции сложения появляется ошибка округления, которую мы обозначим через r_1. Таким образом, полная относительная ошибка результата сложения равна следующей сумме:

$$\frac{e_{x+y}}{x+y} = \frac{x}{x+y}\,i_x + \frac{y}{x+y}\,i_y + r_1.$$

Теперь можно применить то же правило к умножению. Один из сомножителей есть сумма x и y, ошибку которой мы только что вычислили; эта ошибка, согласно изложенным выше правилам, войдет в результат умножения умноженной на $+1$. Относительная ошибка сомножителя z, равная i_z, также войдет в результат умножения умноженной на $+1$. При выполнении операции умножения появляется ошибка округления, равная r_2. Полная ошибка результата операции умножения выразится следующим образом:

$$\frac{e_u}{u} = \frac{x}{x+y}\,i_x\cdot 1 + \frac{y}{x+y}\,i_y\cdot 1 + r_1\cdot 1 + i_z\cdot 1 + r_2.$$

Если все результаты соответствующим образом округлены (имеется в виду симметричное округление), то ни одна из ошибок округления не превзойдет $5\cdot 10^{-t}$. Поэтому

$$\left|\frac{e_u}{u}\right| \leqslant \left(\left|\frac{x}{x+y}\right| + \left|\frac{y}{x+y}\right| + 3\right)\cdot 5\cdot 10^{-t}.$$

Если x и y оба неотрицательны, то сумма

$$\left|\frac{x}{x+y}\right| + \left|\frac{y}{x+y}\right|$$

не может быть больше 1, и окончательно мы имеем

$$\left|\frac{e_u}{u}\right| \leqslant 20\cdot 10^{-t} = 2\cdot 10^{-t+1}.$$

2.8. ПРИМЕРЫ

Применим теперь методику графов к трем примерам и проиллюстрируем, что означает распространение ошибки в практических вычислениях. Те выводы, которые будут

сделаны в этом параграфе, найдут применение в последующих главах книги. Эти примеры отлично иллюстрируют специфику вычислений на цифровой вычислительной машине; в частности, первые два результата будут противоречить тому, что кажется само собой разумеющимся в классической математике.

Пример 1

Сложение положительных чисел, расположенных в порядке возрастания их величин.

Рассмотрим задачу сложения четырех положительных чисел:

$$y = x_1 + x_2 + x_3 + x_4,$$

где

$$0 < x_1 < x_2 < x_3 < x_4.$$

Граф этого процесса изображен на рис. 2.3. Предположим, что все исходные величины заданы точно и не имеют ошибок, и пусть r_1, r_2 и r_3 являются относительными ошибками округления после каждой следующей операции сложения. Последовательное применение правила для подсчета полной ошибки окончательного результата приводит к формуле

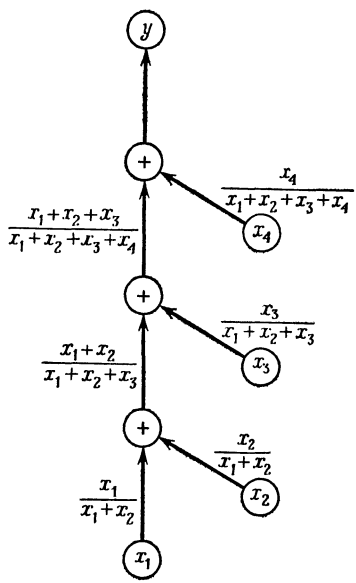


Рис. 2.3. Граф вычислительного процесса для сложения $y = x_1 + x_2 + x_3 + x_4$, причем $0 < x_1 < x_2 < x_3 < x_4$.

$$\frac{e_y}{y} = r_1 \frac{x_1+x_2}{x_1+x_2+x_3} \cdot \frac{x_1+x_2+x_3}{x_1+x_2+x_3+x_4} + r_2 \frac{x_1+x_2+x_3}{x_1+x_2+x_3+x_4} + r_3.$$

Сокращая сумму $x_1+x_2+x_3$ в первом члене и умножая все выражение на $y = x_1+x_2+x_3+x_4$, получаем

$$e_y = r_1(x_1+x_2) + r_2(x_1+x_2+x_3) + r_3(x_1+x_2+x_3+x_4).$$

Учитывая, что ошибка округления равна $5 \cdot 10^{-t}$, окончательно имеем

$$|e_y| \leqslant (3x_1+3x_2+2x_3+x_4) \cdot 5 \cdot 10^{-t}.$$

Из этой формулы ясно, что максимально возможная ошибка (абсолютная или относительная), обусловленная округлением, становится меньше, если сначала складывать меньшие числа.

Результат довольно удивительный, так как вся классическая математическая подготовка инженера или ученого основана на предположении, что при перемене мест слагаемых или их группировке сумма не изменяется, причем зачастую эти свойства операции сложения просто постулируются. Разница, если она до сих пор не очевидна читателю, заключается в том, что ЭЦВМ не может производить вычисления с бесконечно большой точностью, а именно такие вычисления рассматриваются в классической математике. Любое число в ЭЦВМ должно быть представлено с помощью конечного набора значащих цифр, и вот это-то простое ограничение может самым неожиданным образом изменить многие «само собой разумеющиеся» математические правила.

Для сложения n чисел, не имеющих первоначально никаких ошибок, общая формула для ошибки округления запишется так:

$$|e_y| \leqslant [(n-1)x_1 + (n-1)x_2 + (n-2)x_3 + \ldots$$
$$\ldots + 2x_{n-1} + x_n] \cdot 5 \cdot 10^{-t}.$$

Рассмотрим численный пример. Предположим, что необходимо найти сумму следующих чисел:

$$0.2897 \cdot 10^0$$
$$0.4976 \cdot 10^0$$
$$0.2488 \cdot 10^1$$
$$0.7259 \cdot 10^1$$
$$0.1638 \cdot 10^2$$
$$0.6249 \cdot 10^2$$
$$0.2162 \cdot 10^3$$
$$0.5233 \cdot 10^3$$
$$0.1403 \cdot 10^4$$
$$0.5291 \cdot 10^4$$

Если сложение производится, начиная с наименьшего числа в порядке возрастания чисел, то последовательно получающиеся частичные суммы будут такими, как показа-

но ниже. (Первая частичная сумма есть сумма первых двух чисел, вторая частичная сумма есть сумма первой частичной суммы и третьего числа и т. д.) Необходимо помнить, что «применяемая» нами ЭЦВМ имеет четыре значащих цифры в мантиссе каждого числа, а все числа, мантиссы которых имеют более четырех значащих цифр, должны быть округлены. Это условие лежит в основе всех выкладок данного параграфа, хотя для реальных ЭЦВМ была бы более типичной мантисса, содержащая восемь значащих цифр.

$$0.7873 \cdot 10^{0}$$
$$0.3275 \cdot 10^{1}$$
$$0.1053 \cdot 10^{2}$$
$$0.2691 \cdot 10^{2}$$
$$0.8940 \cdot 10^{2}$$
$$0.3056 \cdot 10^{3}$$
$$0.8289 \cdot 10^{3}$$
$$0.2232 \cdot 10^{4}$$
$$0.7523 \cdot 10^{4}$$

С другой стороны, если сложить те же 10 чисел в обратном порядке, от самого большого к самому малому, то частичные суммы будут следующими:

$$0.6694 \cdot 10^{4}$$
$$0.7217 \cdot 10^{4}$$
$$0.7433 \cdot 10^{4}$$
$$0.7495 \cdot 10^{4}$$
$$0.7511 \cdot 10^{4}$$
$$0.7518 \cdot 10^{4}$$
$$0.7520 \cdot 10^{4}$$
$$0.7520 \cdot 10^{4}$$
$$0.7520 \cdot 10^{4}$$

Если при каждом сложении сохранять все значащие цифры, то можно найти точную сумму, равную $0.75229043 \cdot 10^{4}$. Ошибка при вычислении суммы от меньшего числа к большему равна $-0.1 \cdot 10^{0}$, в то время как

при обратном порядке вычислений ошибка равна $2.9.10^0$, или почти в 30 раз больше.

Максимально возможные ошибки составляют $5.5 \cdot 10^0$ для первого варианта и $33 \cdot 10^0$ для второго варианта. В обоих случаях действительные ошибки гораздо меньше максимально возможных. Максимально возможная ошибка могла бы получиться, например, если бы при округлении каждой частичной суммы пришлось бы отбрасывать величину, близкую к половине единицы младшего разряда, но так случается очень редко.

Заметим также, что если не суммировать два наименьших числа, а суммировать только восемь бóльших чисел, то сумма, вычисленная от меньших чисел к бóльшим, слегка изменяется и становится равной $0.7522 \cdot 10^4$, в то время как сумма, вычисленная от бóльших чисел к меньшим, остается неизменной и равной $0.7520 \cdot 10^4$. Дело в том, что во втором варианте оба последних числа слишком малы для того, чтобы повлиять на последнюю значащую цифру суммы, если их прибавлять по одному. В первом же варианте эти числа складываются в самом начале и их сумма уже достаточно велика, чтобы изменить последнюю значащую цифру окончательной суммы.

Пример 2

Сложение почти равных положительных чисел. Предположим, что мы снова складываем четыре положительных числа, но на этот раз они почти равны одно другому. Можно написать

$$x_i = x_0 + \delta_i, \quad i = 1, 2, 3, 4,$$

где

$$|\delta_i| \ll x_0.$$

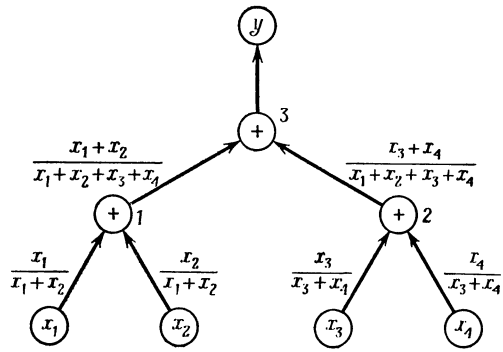
(Символ «$\ll$» означает «много меньше».) Применяя выведенную в предыдущем примере формулу для ошибки округления для сложения четырех чисел, имеем

$$|e_y| \leqslant (9x_0 + 3|\delta_1| + 3|\delta_2| + 2|\delta_3| + |\delta_4|) \cdot 5 \cdot 10^{-t}.$$

Так как $|\delta_i|$ малы по сравнению с x_0, то приблизительно можно написать

$$|e_y| \leqslant 4.5 \cdot 10^{-t+1} \cdot x_0.$$

Таков будет верхний предел для ошибки округления, если сложение ведется согласно графу, изображенному на рис. 2.3. Рассмотрим другой путь вычисления той же суммы, согласно графу, изображенному на рис. 2.4. Здесь $y = (x_1 + x_2) + (x_3 + x_4)$, и операции в скобках



Р и с. 2.4. Другой вид графа для сложения четырех чисел. Порядок сложения определяется числами, поставленными рядом с кружками сложения.

необходимо выполнить вначале. Если обозначить ошибки округления через r_1, r_2, r_3 соответственно порядку выполнения операций сложения, то получим

$$\frac{e_y}{y} = r_1 \cdot \frac{x_1+x_2}{x_1+x_2+x_3+x_4} + r_2 \cdot \frac{x_3+x_4}{x_1+x_2+x_3+x_4} + r_3.$$

Преобразуя последнюю формулу, получаем

$$|e_y| \leqslant (2x_1 + 2x_2 + 2x_3 + 2x_4) \cdot 5 \cdot 10^{-t}.$$

Подставляя $x_i = x_0 + \delta_i$ и пренебрегая $|\delta_i|$ по сравнению с x_0, окончательно имеем

$$|e_y| \leqslant 4 \cdot 10^{-t+1} \cdot x_0.$$

Сравнивая верхние пределы ошибки округления для двух способов вычисления одной и той же суммы, замечаем, что второй способ имеет несколько меньший верхний предел для ошибки. Результат вовсе не очевидный.

В общем случае если имеется n^2 положительных чисел приблизительно одинаковой величины, то общая ошибка

округления уменьшается, если числа сложить сначала группами по n чисел, а потом сложить n частичных сумм. Для больших n верхний предел ошибки округления при таком способе составляет всего $1/n$ от соответствующего предела при сложении чисел друг за другом (см. рис. 2.3).

Рассмотрим численный пример. Требуется сложить четыре числа:

$$x_1 = 0.5243 \cdot 10^0$$
$$x_2 = 0.5262 \cdot 10^0$$
$$x_3 = 0.5226 \cdot 10^0$$
$$x_4 = 0.5278 \cdot 10^0$$

Можно положить $x_0 = 0.5200$; вычисления по-прежнему производятся с четырьмя значащими цифрами. Складывая числа одно за другим и округляя, получаем $y = 0.2102 \cdot 10^1$. С другой стороны, вычисляя $x_1 + x_2 = 0.1051 \cdot 10^1$ и $x_3 + x_4 = 0.1050 \cdot 10^1$, получаем $y = 0.2101 \cdot 10^1$. Точное значение суммы равно $0.21009 \cdot 10^1$.

У неискушенного читателя может возникнуть вопрос, стóят ли эти маленькие усовершенствования того труда, который на них затрачивается. Дело в том, что в этой главе мы рассматриваем примеры, содержащие не больше десятка операций. Впоследствии нам придется рассматривать вычислительные процессы, которые требуют сотен и даже тысяч арифметических операций. В этих условиях даже небольшая ошибка может быть сильно увеличена последующими операциями. Поэтому методы уменьшения верхних пределов ошибок, разбираемые в этой главе, несомненно представляют ценность для практики.

Пример 3

Вычитание двух почти равных чисел. Предположим, что необходимо вычислить $z = x - y$. Тогда из формулы дл распространения ошибки при вычитании имеем

$$\frac{e_z}{z} = \frac{x}{x-y}\left(\frac{e_x}{x}\right) - \frac{y}{x-y}\left(\frac{e_y}{y}\right).$$

Предположим, кроме того, что x и y являются соответствующим образом округленными положительными числами,

так что

$$\left|\frac{e_x}{x}\right| \leqslant 5\cdot 10^{-t} \quad \text{и} \quad \left|\frac{e_y}{y}\right| \leqslant 5\cdot 10^{-t}.$$

Если разность $x - y$ мала, то относительная ошибка z может стать большой, даже если абсолютная ошибка мала. Так как в дальнейших вычислениях эта большая относительная ошибка будет распространяться, то она может поставить под сомнение точность окончательного результата вычислений.

Рассмотрим простой пример:

$$x = 0.5628\cdot 10^4$$

$$y = 0.5631\cdot 10^4$$

Тогда

$$z = -0.0003\cdot 10^4$$

Зная x и y, мы можем написать

$$\left|\frac{e_x}{x}\right| \leqslant \frac{0.5}{0.5628}\cdot 10^{-4} < 0.01\%$$

$$\left|\frac{e_y}{y}\right| \leqslant \frac{0.5}{0.5631}\cdot 10^{-4} < 0.01\%$$

Как видим, относительные ошибки x и y малы. Однако

$$\left|\frac{e_z}{z}\right| \leqslant \frac{1}{0.0003}\cdot 10^{-4} \approx 33\%$$

Эта относительная ошибка очень велика, и, что важнее всего, она будет распространяться в ходе последующих вычислений. Если, например, следующей операцией будет умножение полученной разности на $0.7259\cdot 10^4$, то результат составит $0.2178\cdot 10^5$; этот результат имеет четыре десятичные значащие цифры, и неискушенный программист может счесть этот результат точным до 4-го знака, хотя в нем можно доверять не более чем одному знаку.

2.9. ПАМЯТКА ПРОГРАММИСТУ

Выводы, полученные в настоящей главе, можно свести в краткий перечень рекомендаций для практической организации вычислений. Некоторые упражнения, приведенные

в конце главы, иллюстрируют эти рекомендации; на эти упражнения даны ссылки.

1. Если необходимо произвести сложение — вычитание длинной последовательности чисел, работайте сначала с наименьшими числами (упражнение 13).

2. Если возможно, избегайте вычитания двух почти равных чисел. Формулы, содержащие такое вычитание, часто можно преобразовать так, чтобы избежать подобной операции (упражнения 12, 14 и 18).

3. Выражение вида $a(b-c)$ можно написать в виде $ab-ac$, а выражение вида $(b-c)/a$ можно написать в виде $b/a-c/a$. Если числа в разности почти равны друг другу, *производите вычитание до умножения или деления*. При этом задача не будет осложнена дополнительными ошибками округления (упражнения 16 и 17).

4. В любом случае сводите к минимуму число необходимых арифметических операций (упражнения 6 и 7).

Упражнения

*1. Ток протекает по резистору 10 *ом*, известному с точностью $\pm$ 10%. Ток равен 2 ± 0.1 *а*. Согласно закону Ома, падение напряжения на резисторе равно произведению тока на сопротивление. Каковы относительная и абсолютная ошибки вычисленного значения напряжения? Ошибками округления пренебречь.

2. Средняя длина авиалинии от Нью-Йорка до Сан-Франциско равняется 2700 милям, но может быть на 200 миль короче или длиннее в результате вариаций маршрута самолета. Средняя скорость самолета на этой линии составляет 580 миль в час, но может оказаться на 60 миль в час больше или меньше из-за ветра. Каковы верхний и нижний пределы времени полета?

3. Реактивное сопротивление емкости дается формулой

$$X_c = \frac{1}{2\pi f C},$$

где X_c — реактивное сопротивление емкости в омах,
f — частота в герцах,
C — емкость в фарадах.

Указать границы возможных значений X_c для $f = 400 \pm 1$ *гц* и $C = 10^{-7}$ *ф* $\pm$ 10%.

4. Положение S свободно падающего тела в вакууме дается следующей формулой:

$$S = \frac{1}{2} g t^2,$$

где g — ускорение свободного падения в *м/сек*^{2},
t — время, прошедшее с начала падения, в *сек*.

Предположим, что $g = 9.81$ *м/сек*^{2} *точно*, но время может быть измерено только с точностью до 0.1 *сек*. Покажите, что с ростом t абсолютная ошибка вычисленного значения S увеличивается, но относительная ошибка уменьшается.

***5.** Предположим, что a есть положительное, соответствующим образом округленное число и что число 2 можно записать в ЭЦВМ точно. Начертите графы вычислительных процессов и выведите формулы для максимально возможных относительных ошибок при вычислении $u = a + a$ и $v = 2a$. Покажите, что эти ошибки одинаковы.

***6.** При тех же предположениях, что и в упражнении 5, покажите, что максимально возможная относительная ошибка для $u = a + a + a$ больше, чем для $v = 3a$. Проиллюстрируйте этот вывод на примере $a = 0.6992$, сохраняя после каждой арифметической операции четыре десятичные значащие цифры.

7. Предположим, что a и b являются положительными и соответствующим образом округленными числами. Начертите графы вычислительных процессов и выведите выражения для максимально возможных ошибок при вычислении $u = (a + a + a)\,b$ и $v = 3\,(ab)$. Покажите, что максимально возможная относительная ошибка больше для первого выражения. Проиллюстрируйте для $a = 0.4299$ и $b = 0.6824$.

***8.** Предположим, что x есть соответствующим образом округленное число. Нарисуйте графы вычислительных процессов, выведите выражения для максимально возможных ошибок и покажите, что границы ошибок одинаковы для $u = (x^2)^2$ и $v = x \cdot (x \cdot (x \cdot x))$.

9. При тех же предположениях, что и в предыдущем упражнении, начертите графы вычислительных процессов, выведите выражения для максимально возможных ошибок и покажите, что ошибки одинаковы для $u = x \cdot (x \cdot (x \cdot (x \cdot (x \cdot (x \cdot x)))))$ и для $v = ((x^2)^2)^2$.

10. Покажите, что если пользоваться десятичной арифметикой с конечным количеством значащих цифр, то $10./10. = 10.$ * $(1./10.)$ и $2./2. = 2.$ * $(1./2.)$, но $3./3. \neq 3.$ * $(1./3.)$

***11.** Предположим, что a, b и x положительны и заданы точно. Начертите графы вычислительных процессов, выведите выражения для максимально возможных ошибок и покажите, что для $u = ax + bx^2$ и $v = x\,(a + bx)$ пределы ошибок одинаковы. В качестве численного примера возьмите $a = 0.7625$, $b = 0.6947$ и $x = 0.4302$. Покажите, что хотя максимально возможные ошибки и одинаковы, но действительные ошибки, будучи меньше максимально возможных, вовсе не обязаны быть одинаковыми.

12. Предположим, что a и b положительны, заданы точно и что $a > b$. Покажите, что хотя и справедливо равенство $(a + b) = (a^2 - b^2)/(a - b)$ (при вычислениях с бесконечно большой точностью), тем не менее в результате ошибок округления вычисленные значения левой и правой частей выражения могут сильно различаться. Покажите, что наихудший случай имеет место тогда, когда ошибки округления при вычислении a^2 и b^2 максимальны, но имеют противоположные знаки. Проиллюстрируйте на примере $a = 0.3525$ и $b = 0.3411$, используя десятичную арифметику с четырьмя значащими цифрами.

13. Предположим, что a есть соответствующим образом округленное положительное число и что 1 можно представить в ЭЦВМ точно. Рассмотрим выражения $u=(1+a)^2$ и $v=1+(2a+a^2)$. Покажите, что при $a \gg 1$ максимально возможные относительные ошибки для u и v почти равны, но при $a \ll 1$ максимально возможная относительная ошибка для u почти в 3 раза больше, чем для v. Проиллюстрируйте это для $a=0.2635$.

14. Начертите граф вычислительного процесса и выведите выражения для максимально возможных относительных ошибок при вычислении $(a+b)-b$. Проиллюстрируйте для $a=0.8614\times\times 10^{-2}$ и $b=0.9949$, а также для $a=0.3204$ и $b=0.5837$.

15. Рассмотрим выражение $5a+b$. Покажите, что при вычислении этого выражения относительная ошибка величины a скажется в пять раз сильнее, чем относительная ошибка величины b.

***16.** Рассмотрим выражения $u=(a-b)/c$ и $v=a/c-b/c$. Предположим, что a, b и c положительны и заданы точно и что $a \approx b$. Покажите, что относительная ошибка округления для v может быть гораздо больше, чем для u. Проиллюстрируйте для $a=0.41$, $b=0.36$ и $c=0.70$. Используйте десятичную арифметику с двумя значащими цифрами.

17. Рассмотрим выражения $a(b-c)=u$ и $ab-ac=v$. Предположим, что a, b и c все положительны; кроме того, $b>c$ и $b \approx c$. Покажите, что при этих условиях u имеет гораздо меньшую максимально возможную относительную ошибку. Покажите, что для $a=0.9364$, $b=0.6392$ и $c=0.6375$ значение $u=0.1592\times\times 10^{-2}$, вычисленное с четырьмя значащими цифрами, равно соответствующим образом округленному точному ответу, в то время как $v=0.1500\cdot 10^{-2}$.

***18.** Дано квадратное уравнение $ax^2+bx+c=0$; предположим, что все коэффициенты положительны, заданы точно и что $b^2 \gg 4ac$. Сначала покажите, что при вычислениях с бесконечным количеством значащих цифр меньший из двух корней может быть вычислен по любой из формул:

$$x_1=\frac{-b+\sqrt{b^2-4ac}}{2a}$$

или

$$x_1'=\frac{-2c}{b+\sqrt{b^2-4ac}}.$$

Теперь покажите, что при заданных ограничениях на коэффициенты вторая формула обеспечит гораздо более высокую точность. Покажите, что для $a=0.1000\cdot 10^1$, $b=0.4002\cdot 10^0$ и $c=0.8000\times\times 10^{-4}$ $x_1=-0.1500\cdot 10^{-3}$ и $x_1'=-0.2000\cdot 10^{-3}$. Последняя величина равна точному значению корня. (Операцию извлечения квадратного корня можно представить на графе в виде кружка, к которому ведет стрелка только от одной величины. Относительная ошибка исходной величины появляется в квадратном корне умноженной на 0.5, и этот коэффициент можно поставить около стрелки. При извлечении квадратного корня появляется также дополнитель-

ная ошибка округления, которая в большинстве трансляторов ФОРТРАН не превосходит 10^{-t+1}.)

19. Рассмотрим систему уравнений

$$ax+by=c,$$
$$dx+ey=f$$

и их решение по правилу Крамера

$$x=\frac{ce-bf}{ae-bd},$$

$$y=\frac{af-cd}{ae-bd}.$$

Покажите, что если $ae-bd$ мало по абсолютной величине, то точность решения может оказаться низкой, даже если все коэффициенты заданы точно. Проиллюстрируйте на примере системы уравнений:

$$0.2038x+0.1218y=0.2014$$
$$0.4071x+0.2436y=0.4038$$

Решение этой системы, полученное при вычислениях с четырьмя значащими цифрами, равно $x=-1.714$ и $y=4.286$, в то время как точное решение, полученное при вычислениях с восемью значащими цифрами, равно $x=-2.000$ и $y=5.000$. Если коэффициенты уравнений сами содержат ошибки, как это почти всегда и бывает, то «решение» этой системы может оказаться полностью бессмысленным.

ГЛАВА 3

ПРАКТИЧЕСКОЕ ВЫЧИСЛЕНИЕ ФУНКЦИЙ

3.1. ВВЕДЕНИЕ

В гл. 1 мы узнали о том, что обычные элементарные функции — синус, косинус, логарифм — могут быть использованы в ФОРТРАНе и что для вычисления этих функций достаточно только написать в программе их наименования. Для большинства случаев практических вычислений обычно бывает вполне достаточно этих функций. Однако иногда эти стандартные функции могут оказаться неподходящими; например, может случиться так, что стандартная функция вычисляется каждый раз с излишне высокой точностью, и это приводит к неоправданным затратам машинного времени. Наконец, может просто потребоваться функция, которой нет в ассортименте стандартных функций.

Поэтому мы начинаем изучение практических методов численных расчетов с изучения того, как вычисляются в ЭЦВМ различные функции. Эти методы разбираются на примере общеизвестной функции синуса, но следует иметь в виду, что те же самые методы применимы к любой функции, которую можно разложить в ряд Тейлора. Ряды Тейлора, знакомые всем из курса математического анализа, будут являться отправной точкой для вычисления любой другой функции с помощью описанных ниже методов.

Кроме рассмотрения методов вычисления функций, в данной главе рассматривается также весьма важный вопрос о наилучших способах вычисления полиномов и развиваются некоторые идеи о методах анализа точности вычислений.

3.2. СТЕПЕННЫЕ РЯДЫ

Первое, что необходимо сделать при вычислении какой-либо функции по ее разложению в ряд, это уменьшить, если возможно, диапазон значений аргумента, для которых

требуется это вычисление. Это может значительно уменьшить ошибку округления. «Математическое» определение синуса через его разложение в степенной ряд пригодно для *всех* значений аргумента, но при этом подразумевается, что вычисление синуса необходимо производить с бесконечно большим количеством значащих цифр. На практике при вычислениях с помощью ЭЦВМ степенной ряд для синуса становится совершенно бесполезным при больших значениях аргумента и дает совершенно бессмысленные результаты.

К счастью, в случае синуса задача решается весьма просто. Вспомним, что для целого n

$$\sin(n\pi + y) = \sin n\pi \cos y + \cos n\pi \sin y = (-1)^n \sin y,$$

$$-\frac{\pi}{2} \leqslant y \leqslant \frac{\pi}{2}.$$

Таким образом, отнимая некоторое число, кратное π, мы сводим задачу нахождения синуса произвольного угла к задаче нахождения синуса угла, лежащего между $-\pi/2$ и $\pi/2$. Наконец, делая подстановку

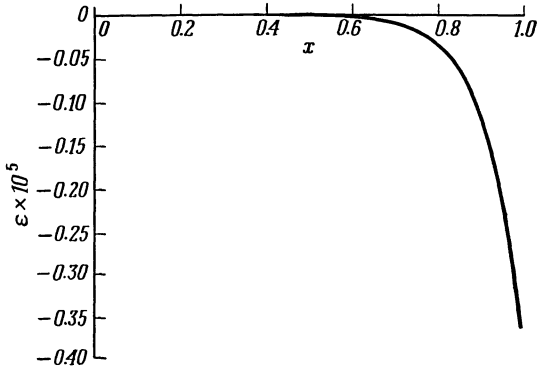
$$y = \frac{\pi x}{2}; \quad \sin y = \sin \frac{\pi x}{2},$$

мы сводим диапазон значений аргумента к $-1 \leqslant x \leqslant 1$.

На практике уменьшение аргумента не производится последовательным вычитанием. Вместо этого первоначальный угол делится на π, причем деление организовано так, что частное получается целым. Остаток от деления будет представлять собой некоторый угол, заключенный между 0 и π. Если остаток больше $\pi/2$, то еще одно вычитание дает угол между $-\pi/2$ и $\pi/2$. Целое частное от деления исходного угла на π используется для того, чтобы определить, следует ли изменить знак окончательного результата (при нечетном частном).

Эти предварительные операции с углом изменяют *его ошибку*. Значение π, используемое при делении, содержит некоторую ошибку округления, так как π является иррациональным числом и не представляется конечным количеством значащих цифр; уменьшение величины угла увеличивает его относительную ошибку, хотя абсолютная ошибка может остаться неизменной; подстановка $y = \pi x/2$ вносит дополнительную ошибку округления. При тщательном анализе ошибок необходимо учитывать все эти эффекты. На

практике, однако, все эти ошибки будут перекрыты неопределенностью исходного значения угла и ошибкой, происходящей от ограничения ряда, используемого для



Р и с. 3.1. Графическое представление ошибки для ограниченного ряда Тейлора (3.1).

вычисления синуса. Именно этот последний род ошибок — ошибки ограничения — мы сейчас и рассмотрим.

Напишем пять первых членов ряда Тейлора для синуса:

$$\sin\frac{\pi x}{2} \approx \frac{\pi x}{2} - \frac{1}{3!}\left(\frac{\pi x}{2}\right)^3 + \frac{1}{5!}\left(\frac{\pi x}{2}\right)^5 - \frac{1}{7!}\left(\frac{\pi x}{2}\right)^7 + \frac{1}{9!}\left(\frac{\pi x}{2}\right)^9 = 1.5707963x - 0.64596410x^3 + 0.079692626x^5 - 0.0046817541x^7 + 0.00016044118x^9 \quad (3.1)$$

Естественно, ряд имеет бесконечное количество членов, так что, отбросив остальные члены, мы внесли ошибку ограничения. Можно показать, что для всякого знакопеременного сходящегося ряда эта ошибка ограничения меньше первого отброшенного члена

$$|e_T| \leqslant \frac{1}{11!}\left(\frac{\pi x}{2}\right)^{11} \leqslant 0.0000035988 \approx 3.6 \cdot 10^{-6} \quad (3.2)$$

так как максимальное значение $x = 1$.

На рис. 3.1 изображен график общей ошибки, которая получается при использовании разложения (3.1). Общая ошибка составляется из ошибки ограничения и ошибки

округления, но ошибка округления в данном случае мала, а ошибка ограничения доминирует над ней. Заметим, что хотя ошибка практически равна нулю для $|x| < 1/2$, она становится довольно большой для x, близких к 1. Максимальная ошибка составляет $3.54 \cdot 10^{-6}$ в полном согласии с оценкой (3.2).

3.3. ПОЛИНОМЫ ЧЕБЫШЕВА

Интересно было бы найти какой-либо способ уменьшить ошибку в окрестности $x = 1$. Такой способ существует, но при этом ошибка уменьшается в окрестности $x = 1$ за счет того, что она увеличивается в других местах. Способ, который будет изложен в этом разделе, а именно использование полиномов Чебышева, позволяет более равномерно распределить ошибку по всему интервалу.

Полиномы Чебышева определяются следующим образом:

$$T_n(x) = \cos n\theta, \tag{3.3}$$

где $x = \cos\theta$. Другими словами,

$$T_n(x) = \cos(n \arccos x).$$

Например,

$$T_0(x) = \cos 0 = 1, \tag{3.4}$$

$$T_1(x) = \cos\theta = x, \tag{3.5}$$

$$T_2(x) = \cos 2\theta = \cos^2\theta - \sin^2\theta = x^2 - (1 - x^2) = 2x^2 - 1. \tag{3.6}$$

Можно было бы и дальше использовать тригонометрические соотношения для нахождения полиномов Чебышева любого порядка, но будет лучше установить между ними рекуррентное соотношение, связывающее $T_{n+1}(x)$, $T_n(x)$ и $T_{n-1}(x)$:

$$T_{n+1}(x) = \cos(n\theta + \theta) = \cos n\theta \cos\theta - \sin n\theta \sin\theta,$$

$$T_{n-1}(x) = \cos(n\theta - \theta) = \cos n\theta \cos\theta + \sin n\theta \sin\theta.$$

Складывая эти два равенства, получаем

$$T_{n+1}(x) + T_{n-1}(x) = 2\cos n\theta \cos\theta = 2xT_n(x),$$

и окончательно

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x). \tag{3.7}$$

Из (3.4), (3.5), (3.6) и (3.7) можно найти любой полином Чебышева. Например, приняв $n=2$ в формуле (3.7), имеем

$$T_3(x) = 2xT_2(x) - T_1(x).$$

И, подставляя значения $T_2(x)$ и $T_1(x)$ из формул (3.5) и (3.6):

$$T_3(x) = 2x(2x^2 - 1) - x = 4x^3 - 3x.$$

Первые 12 полиномов Чебышева приведены в приложении 2, часть А, вместе с формулами, выражающими первые 11 степеней x через полиномы Чебышева.

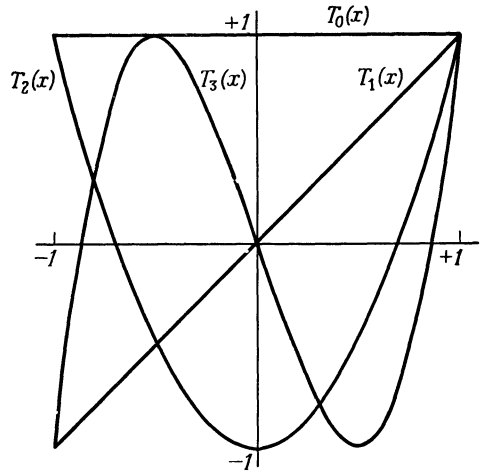
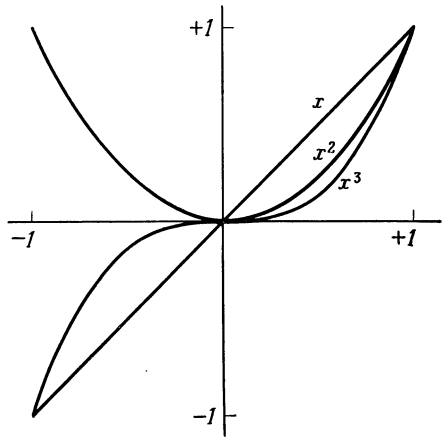


Рис. 3.2. Графическое представление полиномов Чебышева $T_0(x)$, $T_1(x)$, $T_2(x)$ и $T_3(x)$.

Графически первые четыре полинома Чебышева изображены на рис. 3.2. Последующие полиномы по-прежнему колеблются между $+1$ и -1, причем период колебаний уменьшается с ростом порядка полинома.

Для того чтобы подчеркнуть разницу, а также чтобы показать, почему мы так заинтересованы в полиномах Чебышева, на рис. 3.3 построены графики первых трех степеней x. Сравнивая два рисунка, можно видеть, что

изменение коэффициентов в ряде Тейлора, где члены ряда являются просто степенями x, повлияет на вычисленное значение функции в окрестности $x = 1$ гораздо сильнее, чем в окрестности $x = 0$, в то время как изменение коэффициентов ряда, где члены являются полиномами Чебыше-



Р и с. 3.3. Графическое представление первых трех степеней x.

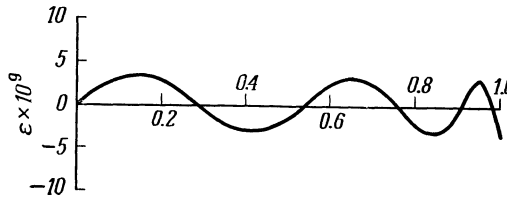
ва, даст ошибку, распределенную по всему интервалу значений аргумента (+1, —1).

Задача нахождения коэффициентов ряда Чебышева довольно сложна и в этой книге обсуждаться не будет. Результат для функции sin x, полученный при использовании полиномов Чебышева до 9-го порядка и содержащий поэтому нечетные степени x до 9-й, приведен ниже:

$$\sin\left(\frac{\pi x}{2}\right) = 1.5707963x - 0.64596336x^3 +$$
$$+ 0.079688475x^5 - 0.0046722203x^7 + 0.00015081716x^9 \quad (3.8)$$

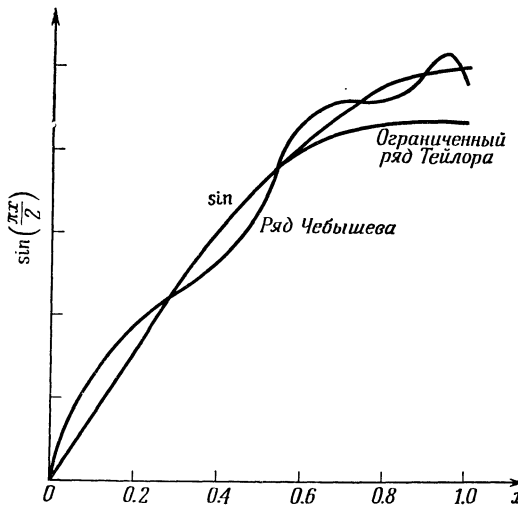
Кривая ошибок для этого приближения показана на рис. 3.4. Можно сравнить ее с кривой ошибок для ряда Тейлора на рис. 3.1. Ряд Тейлора содержит те же степени x, но с другими коэффициентами. (Обратите внимание на разницу в масштабе.) При этом сравнении ясно проявляются свойства полиномов Чебышева — максимальная ошибка меньше, чем при использовании ряда Тейлора,

ошибка знакопеременна и распределена более или менее равномерно по всему интервалу.



Р и с. 3.4. Графическое представление ошибки для приближения (3.8).
Обратите внимание на разницу в масштабах.

Более наглядно разницу между двумя разложениями можно понять из рис. 3.5, где изображена функция sin x



Р и с. 3.5. Синусоида, приближение с помощью ряда Чебышева и с помощью ограниченного ряда Тейлора.
Масштаб ошибок сильно увеличен.

и два различных разложения в ряд. Масштаб ошибки, конечно, сильно преувеличен.

Наилучшее приближение функции степенным рядом в том смысле, что максимальная ошибка при этом прибли-

жении минимальна, называется *чебышевским приближением*. (Речь идет все время об интервале (+1, —1).) Это приближение *не совпадает* с (3.8), существуют методы нахождения этого приближения, но обычно относительно малое уменьшение ошибки не стоит того труда, который приходится тратить на нахождение этого приближения. (Многие вычислительные машины снабжаются программами, позволяющими отыскивать чебышевские приближения функций. Нахождение чебышевского приближения превращается тогда в обычный вычислительный процесс.)

3.4. ЭКОНОМИЗАЦИЯ СТЕПЕННЫХ РЯДОВ

Полиномы Чебышева дают очень хорошее приближение функции в том смысле, что максимальная ошибка этого приближения мала, но эти приближения довольно сложно вычислять. Обычно имеет смысл использовать эти полиномы в стандартных программах, которые будут впоследствии применяться многими программистами, но разложение функции в ряд по полиномам Чебышева для использования в одной программе обычно невыгодно, так как требует слишком больших затрат труда на программирование.

Существует относительно простой метод корректировки разложения в ряд Тейлора. Нахождение исправленных коэффициентов не представляет большой сложности, поэтому этот метод, называемый *экономизацией* степенного ряда, может применяться для повседневного программирования.

Рассмотрим снова ряд Тейлора для синуса, но на этот раз с членами до x^{11} включительно:

$$\sin\frac{\pi x}{2} = 1.5707963x - 0.64596410x^3 + 0.079692626x^5 -$$

$$- 0.0046817541x^7 + 0.00016044118x^9 - 0.0000035988432x^{11}$$

Из приложения 2, часть Б, имеем

$$x^{11} = \frac{1}{1024}(462T_1 + 330T_3 + 165T_5 + 55T_7 + 11T_9 + T_{11}).$$

Теперь, заменяя T_1, T_3, T_5, T_7, T_9 выражениями, приведенными в приложении 2, часть А, имеем

$$x^{11} = \frac{1}{1024}(11x - 220x^3 + 1232x^5 - 2816x^7 + 2816x^9 + T_{11}).$$

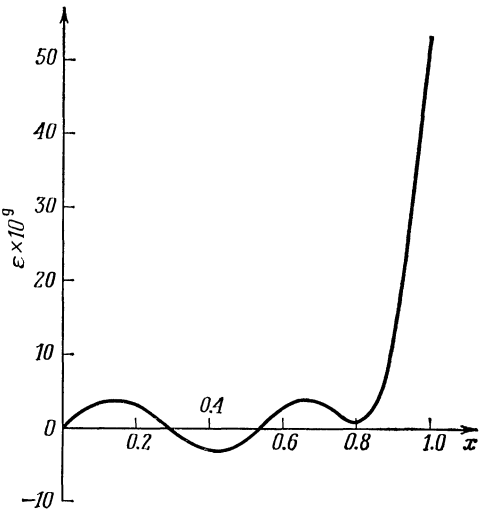
Подставляя правую часть этого равенства вместо x^{11} в ряд Тейлора, получаем

$$\sin\frac{\pi x}{2} = 1.5707962x - 0.64596332x^3 + 0.079688296x^5 - \\ - 0.0046718573x^7 + 0.00015054436x^9 - 0.00000000351T_{11} \tag{3.9}$$

Учитывая, что T_{11} не превосходит по абсолютной величине 1, оценим ошибку

$$|e_T| \leqslant 3{,}51 \cdot 10^{-9}$$

В действительности ошибка существенно больше, так как коэффициенты в (3.9) были вычислены с конечной точностью.



Р и с. 3.6. Графическое представление ошибки для экономизированного ряда (3.9).

Если произвести вычисления по формуле (3.9) для всевозможных значений x из интервала $(+1, -1)$, то максимальная ошибка составит примерно

$$\max|e_T| \approx 8.0 \cdot 10^{-9}$$

Это существенно меньше, нежели ошибка ограничения при вычислении ряда Тейлора той же степени (ср. рис. 3.2).

График ошибок при вычислении синуса по (3.9) показан на рис. 3.6. Заметим, что этот график «лучше», чем график ошибок для простого ряда Тейлора (рис. 3.1), но «хуже», чем для полиномов Чебышева (рис. 3.4), в особенности для $0.7 \leqslant |x| \leqslant 1.0$.

Процесс экономизации можно продолжить: x^9 можно заменить полиномом седьмой степени и T_9. Естественно, ошибка ограничения при этом возрастет. Процесс экономизации можно продолжать до тех пор, пока ошибка ограничения не выйдет из допустимых пределов. Формулы экономизации для всех степеней x до 11-й включительно приведены в приложении 2, часть В.

3.5. ВЫЧИСЛЕНИЕ РЯДА

Независимо от того, какая формула применена для разложения функции в ряд — формула Тейлора, Чебышева или формула экономизации,— в конце концов всегда бывает необходимо вычислить значение полинома вида

$$p(x) = a_0 + a_1 x + \ldots + a_{n-1}x^{n-1} + a_n x^n. \qquad (3.10)$$

Этот полином можно переписать в несколько отличном виде, после чего его можно вычислить не только быстрее, но и во многих случаях с большей точностью. Мы проведем вывод очень подробно, разрабатывая при этом детальную схему вычислений, хотя результат и очевиден. Этим мы заложим основы некоторых дальнейших приложений, в частности в гл. 5.

Разделим $p(x)$ на $(x - x_0)$. В результате деления получится полином степени $n - 1$ и постоянный остаток:

$$p(x) = (x - x_0)(b_1 + b_2 x + \ldots + b_{n-1}x^{n-2} + b_n x^{n-1}) + b_0. \qquad (3.11)$$

Заметим, что $p(x_0) = b_0$, так что если удастся найти метод вычисления b_0, то это будет как раз метод для вычисления $p(x_0)$. Найти этот метод довольно просто. Раскроем скобки в правой части (3.11) и приравняем коэффициенты при соответствующих степенях x:

$$\begin{aligned}
a_n &= b_n, \\
a_{n-1} &= b_{n-1} - x_0 b_n, \\
&\ldots\ldots\ldots \\
a_j &= b_j - x_0 b_{j+1}, \\
&\ldots\ldots\ldots \\
a_0 &= b_0 - x_0 b_1,
\end{aligned}$$

или, записывая в виде рекуррентной формулы:

$$b_n = a_n,$$
$$b_j = a_j + x_0 b_{j+1}, \quad j = n-1, \ldots, 0.$$

Таким образом, мы вычисляем b_n, b_{n-1}, b_{n-2},... и наконец b_0 в указанном порядке.

Для примера положим $n = 5$ и выпишем формулы для последовательных b:

$$b_5 = a_5,$$
$$b_4 = a_4 + x_0 a_5,$$
$$b_3 = a_3 + x_0 (a_4 + x_0 a_5),$$
$$b_2 = a_2 + x_0 (a_3 + x_0 (a_4 + x_0 a_5)),$$
$$b_1 = a_1 + x_0 (a_2 + x_0 (a_3 + x_0 (a_4 + x_0 a_5))),$$
$$b_0 = a_0 + x_0 (a_1 + x_0 (a_2 + x_0 (a_3 + x_0 (a_4 + x_0 a_5)))).$$

Так как величина x_0 ничем не ограничена, то индекс 0 можно опустить.

Этот метод вычисления значения полинома (3.10) известен под названием *правила Горнера* и в общем виде записывается так:

$$p(x) = a_0 + x(a_1 + x(a_2 + \ldots + x(a_{n-1} + x(a_n)) \ldots)).$$

Коэффициенты a в этой формуле те же, что и в (3.10). Подразумевается, что самые внутренние скобки должны быть раскрыты первыми; в действительности не существует никакого другого способа вычислить это выражение, не изменив форму его записи. Благодаря внешнему виду формулы правило Горнера иногда называют также *гнездовой процедурой*.

Для вычисления значения полинома по правилу Горнера требуется n умножений и n сложений. Число же умножений при вычислении по формуле (3.10) составляет $n(n+1)/2$, если каждая степень x получается путем последовательного умножения на x, т. е. $x^k = x^{k-1} \cdot x$, и т. д.

Для большинства приложений правило Горнера вполне пригодно и оно широко употребляется. Для специальных полиномов, которые приходится вычислять многократно при различных значениях аргументов, разработаны способы

позволяющие существенно сократить количество элементарных арифметических операций, но в этой книге подобные способы рассматриваться не будут[1]).

Метод вычисления полинома оказывает большое влияние на распространение ошибок вычисления (имеются в виду ошибки исходной информации и ошибки округления). Например, предположим, что необходимо вычислить полином второго порядка

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

Граф вычислительного процесса при использовании правила Горнера изображен на рис. 3.7. Коэффициенты около стрелок поставлены в соответствии с правилами, изложенными в гл. 2.

Теперь мы можем определить влияние ошибок исходной информации и ошибок округления на значение $p(x)$. Пусть m_1 и m_2 — соответственно ошибки первого и второго умножения. Аналогично пусть α_1 и α_2 — соответственно ошибки первого и второго сложения. Наконец, пусть Δ — ошибка задания величины x_0, а δ_a, δ_b и δ_c — соответственно ошибки задания a, b и c. Тогда

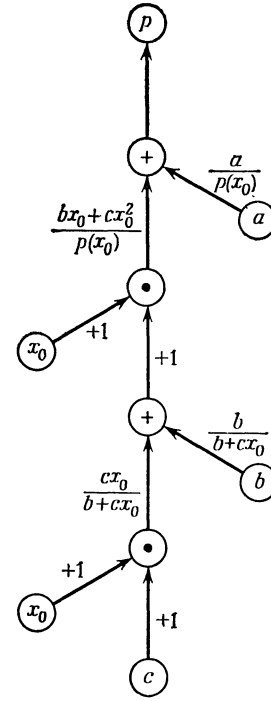


Рис. 3.7. Граф вычислительного процесса для вычисления $p(x) = a + bx + cx^2$ по правилу Горнера, $p(x) = a + x\cdot(b + x\cdot c)$.

$$e_p = \delta_c \frac{cx_0^2}{p(x_0)} + \delta_b \frac{bx_0}{p(x_0)} + \delta_a \frac{a}{p(x_0)} + \Delta\left(\frac{cx_0^2}{p(x_0)} + \frac{bx_0 + cx_0^2}{p(x_0)}\right) + \\ + m_1 \frac{cx_0^2}{p(x_0)} + m_2 \frac{bx_0 + cx_0^2}{p(x_0)} + \alpha_1 \frac{bx_0 + cx_0^2}{p(x_0)} + \alpha_2.$$

[1]) См. Knuth D. E., Evaluation of polynomials by computer, *Comm. ACM*, 5, № 12 (1962).

Абсолютная ошибка $p(x_0)$ выразится в виде

$$e_p \cdot p(x_0) = cx_0^2(\delta_c + 2\Delta + m_1 + m_2 + \alpha_1 + \alpha_2) + \\ + bx_0(\delta_b + \Delta + m_2 + \alpha_1 + \alpha_2) + a(\delta_a + \alpha_2).$$

Для ЭЦВМ с t десятичными значащими цифрами в мантиссе действительного числа и при условии $|x_0| \leqslant 1$ имеем

$$|e_p \cdot p(x_0)| \leqslant 5 \cdot 10^{-t}(7|c| + 5|b| + 2|a|) = E_H.$$

Рассмотрим теперь другой способ вычисления того же полинома, а именно прямое вычисление значения полинома в том порядке, в котором он написан. Последователь-

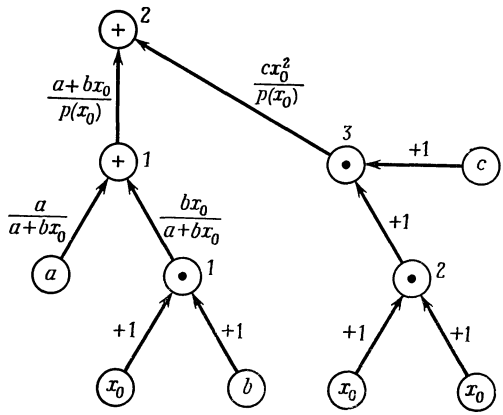


Рис. 3.8. Граф вычислительного процесса для вычисления $p(x_0) = \{[a + (b \cdot x_0)] + c \cdot (x_0 \cdot x_0)\}$.

ность операций можно задать, например, следующим образом: сначала выполняются операции в круглых скобках, затем в квадратных скобках, затем в фигурных скобках. Полином при этом запишется так:

$$p(x_0) = \{[a + (b \cdot x_0)] + c \cdot (x_0 \cdot x_0)\}.$$

Граф этого вычислительного процесса изображен на рис. 3.8. Пусть α_1 и α_2 будут ошибки двух операций сложения, а m_1, m_2, m_3 — ошибки трех операций умножения (имеются

в виду относительные ошибки округления). Тогда

$$p(x_0)\cdot e_p = cx_0^2(\delta_c + 2\Delta + m_2 + m_3 + \alpha_2) + \\ + bx_0(\delta_b + \Delta + m_1 + \alpha_1 + \alpha_2) + a(\delta_a + \alpha_1 + \alpha_2)$$

и, если учесть, что $|x_0| \leqslant 1$:

$$|e_p \cdot p(x_0)| \leqslant 5\cdot 10^{-t}(6|c| + 5|b| + 3|a|) = E_*.$$

Поэтому

$$E_* - E_H = 5\cdot 10^{-t}(|a| - |c|).$$

Если $|a| > |c|$, то вычисления по правилу Горнера приводят к меньшему верхнему пределу ошибок, проистекающих от округления и неточности задания исходных величин. В тех примерах, которые рассматриваются в этой главе, условие $|a| > |c|$ обычно выполняется: полином является частью сходящегося ряда и его коэффициенты уменьшаются при увеличении степеней x.

Поэтому во многих практических расчетах применение правила Горнера не только экономит машинное время, но и повышает точность вычислений за счет уменьшения верхнего предела ошибки округления.

Для полинома вида (3.10) ошибка округления, получающаяся при вычислениях по правилу Горнера, имеет следующий верхний предел:

$$\text{ошибка} \leqslant 5\cdot 10^{-t}\left[\sum_{j=0}^{n}(3j+2)|a_j| - |a_n|\right].$$

Для сходящегося ряда a_j уменьшается с ростом j, так что в выражении для верхнего предела ошибки бóльшие коэффициенты умножаются на меньшие числа.

Следует еще раз отметить, что все выведенные в этом параграфе формулы относятся только к верхним пределам ошибок. Фактические ошибки обычно бывают существенно меньше своих верхних пределов.

3.6. РАЦИОНАЛЬНЫЕ ПРИБЛИЖЕНИЯ И НЕПРЕРЫВНЫЕ ДРОБИ

Некоторые функции нельзя с достаточной точностью приблизить полиномами; в некоторых же случаях бывает так, что полиномиальное приближение очень медленно

сходится. Поэтому в данном разделе мы обратимся к другому методу — рациональному приближению, которое соответствует отношению двух многочленов.

Рассмотрим снова разложение функции в ряд Тейлора:

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_7 x^7 + \ldots . \qquad (3.12)$$

Теперь запишем $f(x)$ в виде частного от деления двух полиномов третьего порядка:

$$f(x) = \frac{b_0 + b_1 x + b_2 x^2 + b_3 x^3}{1 + c_1 x + c_2 x^2 + c_3 x^3} . \qquad (3.13)$$

Константа $+1$ в знаменателе не нарушает общности выражения, так как любую другую константу можно было бы сократить в числителе и в знаменателе. Приравнивая правые части (3.12) и (3.13) и освобождаясь от знаменателя, получаем

$$b_0 + b_1 x + b_2 x^2 + b_3 x^3 = \\ = (1 + c_1 x + c_2 x^2 + c_3 x^3)(a_0 + a_1 x + \ldots + a_7 x^7).$$

Раскрывая скобки и приравнивая коэффициенты при одинаковых степенях x, получаем

$$\begin{aligned} b_0 &= a_0, \\ b_1 &= a_1 + a_0 c_1, \\ b_2 &= a_2 + a_1 c_1 + a_0 c_2, \\ b_3 &= a_3 + a_2 c_1 + a_1 c_2 + a_0 c_3, \\ 0 &= a_4 + a_3 c_1 + a_2 c_2 + a_1 c_3, \\ 0 &= a_5 + a_4 c_1 + a_3 c_2 + a_2 c_3, \\ 0 &= a_6 + a_5 c_1 + a_4 c_2 + a_3 c_3. \end{aligned}$$

(Последние три уравнения отражают тот факт, что в числителе рационального приближения коэффициенты при степенях x выше третьей равны нулю.)

Таким образом, мы получили семь уравнений для семи неизвестных b_0, b_1, b_2, b_3, c_1, c_2, c_3. Решить эту систему уравнений можно, например, методами, изложенными в гл. 8.

Ошибку, которую мы допускаем, пользуясь этим приближением, можно оценить следующим образом. Вычислим,

каким был бы коэффициент b_7, если бы он был включен в это приближение, и разделим его на величину знаменателя:

$$\frac{(a_7+a_6c_1+a_5c_2+a_4c_3)\,x^7}{1+c_1x+c_2x^2+c_3x^3}\,.$$

Эта оценка годится только для ошибки ограничения и не включает ошибку округления, но обычно ошибка ограничения бывает при таких вычислениях гораздо больше ошибки округления.

Рациональные приближения обычно вычисляются не в том виде, в каком они записываются согласно (3.13), а с помощью эквивалентной *непрерывной дроби*. Для того чтобы проиллюстрировать этот метод, рассмотрим опять ту же функцию синуса:

$$\sin\left(\frac{\pi x}{2}\right)=\frac{\pi x}{2}-\frac{1}{3!}\left(\frac{\pi x}{2}\right)^3+\frac{1}{5!}\left(\frac{\pi x}{2}\right)^5-+\ldots. \qquad (3.14)$$

Будем искать рациональное приближение в виде

$$\sin\left(\frac{\pi x}{2}\right)=\frac{ax+bx^3}{1+cx^2}\,. \qquad (3.15)$$

Отсюда имеем

$$(1+cx^2)\left(\frac{\pi}{2}x-\left(\frac{\pi}{2}\right)^3\frac{1}{3!}x^3+\left(\frac{\pi}{2}\right)^5\frac{1}{5!}x^5-\left(\frac{\pi}{2}\right)^7\frac{1}{7!}x^7\right)=$$
$$=ax+bx^3.$$

Получаем систему уравнений:

$$a=\frac{\pi}{2} \qquad \text{(коэффициенты при } x\text{)},$$

$$c\cdot\frac{\pi}{2}-\left(\frac{\pi}{2}\right)^3\frac{1}{3!}=b \qquad \text{(коэффициенты при } x^3\text{)},$$

$$-c\left(\frac{\pi}{2}\right)^3\frac{1}{3!}+\left(\frac{\pi}{2}\right)^5\frac{1}{5!}=0 \qquad \text{(коэффициенты при } x^5\text{)}.$$

Решая эту систему, получаем

$$\left.\begin{aligned} c&=\frac{1}{20}\left(\frac{\pi}{2}\right)^2=0.123370055\\ b&=-\frac{7}{60}\left(\frac{\pi}{2}\right)^3=-0.452174868\\ a&=\frac{\pi}{1}=1.57079633 \end{aligned}\right\} \qquad (3.16)$$

Ошибка ограничения составляет приблизительно

$$e_T \approx \frac{\left(\frac{\pi}{2}\right)^7 \frac{1}{7!} x^7}{1+\frac{1}{20}\left(\frac{\pi}{2}\right)^2 x^2} \approx \frac{0.00468x^7}{1+0.123x^2}$$

Заметим, что (3.15) приблизительно эквивалентно степенному ряду с тремя членами вида (3.14).

Это рациональное приближение имеет тот же порядок точности, что и ряд Тейлора с членами до 5-го порядка включительно:

$$\sin\left(\frac{\pi x}{2}\right) \approx \frac{\pi x}{2} - \frac{1}{3!}\left(\frac{\pi x}{2}\right)^3 + \frac{1}{5!}\left(\frac{\pi x}{2}\right)^5.$$

Если последний полином вычислять по правилу Горнера, предварительно вычислив x^2, то потребуется четыре умножения и два сложения:

$$\sin\left(\frac{\pi x}{2}\right) = x(a - x^2(b - cx^2)).$$

Если вычислять выражение (3.15), используя правило Горнера для числителя и для знаменателя, то потребуется четыре умножения, два сложения и одно деление. По сравнению с рядом Тейлора никакой экономии машинного времени при вычислении рациональной функции не получается.

Поэтому мы преобразуем (3.15) в эквивалентную непрерывную дробь. Сначала перепишем (3.15) в виде

$$\sin\left(\frac{\pi x}{2}\right) = -\frac{7\pi}{6}\left[\frac{x^3 - \frac{60}{7}\left(\frac{2}{\pi}\right)^2 x}{x^2 + 20\left(\frac{2}{\pi}\right)^2}\right]$$

и разделим числитель на знаменатель, получая при этом частное x и остаток $-\frac{200}{7}\left(\frac{2}{\pi}\right)^2 x$:

$$\sin\left(\frac{\pi x}{2}\right) = -\frac{7\pi}{6}\left[x - \frac{\frac{200}{7}\left(\frac{2}{\pi}\right)^2 x}{x^2 + 20\left(\frac{2}{\pi}\right)^2}\right],$$

или

$$\sin\left(\frac{\pi x}{2}\right) = -\frac{7\pi}{6}\left\{x - \frac{\frac{200}{7}\left(\frac{2}{\pi}\right)^2}{\left[\frac{x^2 + 20\left(\frac{2}{\pi}\right)^2}{x}\right]}\right\}.$$

Теперь берем выражение в квадратных скобках и снова делим числитель на знаменатель, при этом получается

$$\frac{x^2 + 20\left(\frac{2}{\pi}\right)^2}{x} = x + \frac{20\left(\frac{2}{\pi}\right)^2}{x}$$

и окончательно

$$\sin\left(\frac{\pi x}{2}\right) = -\frac{7\pi}{6}\left[x - \frac{\frac{200}{7}\left(\frac{2}{\pi}\right)^2}{x + \frac{20\left(\frac{2}{\pi}\right)^2}{x}}\right],$$

или

$$\sin\left(\frac{\pi x}{2}\right) = -3.66519143\left(x - \frac{11.47221432}{x + \frac{8.03055026}{x}}\right). \quad (3.17)$$

Вычисление выражения (3.17) требует двух делений, двух сложений и одного умножения. Это гораздо меньше, чем требуется для вычисления рациональной функции (3.15). Может оказаться даже, что если время деления на ЭЦВМ составляет не более 3/2 времени умножения, то непрерывная дробь (3.17) потребует для своего вычисления меньше машинного времени, чем полином (3.14). (Это соотношение времени деления и умножения выполняется в большом числе распространенных ЭЦВМ, в некоторых же ЭЦВМ длительности деления и умножения просто одинаковы.)

Цепные дроби можно экономизировать подобно тому, как это делалось для степенных рядов.

3.7. ЭЛЕМЕНТАРНЫЕ ФУНКЦИИ

В разд. 3.1 было показано, что не обязательно иметь возможность вычислять синус любого угла, так как это можно свести к вычислению синуса угла, заключенного

в пределах $-\pi/2 \leqslant x \leqslant \pi/2$. Это один из примеров того, как можно упростить вычисление элементарных функций. Рассмотрим, какие аналогичные приемы можно применить к другим общеизвестным функциям[1]).

Косинус

Специальной программы для вычисления косинуса вообще не требуется, потому что существует тригонометрическое тождество

$$\sin\left(x+\frac{\pi}{2}\right)=\cos x.$$

Поэтому все, что требуется для вычисления косинуса,— это увеличить исходный угол на $\pi/2$ и вычислить синус получившегося угла.

Гиперболические функции

Если существует программа для вычисления экспоненты, то гиперболический синус и косинус можно найти из соотношений:

$$\operatorname{sh} x=\frac{1}{2}(e^x-e^{-x}),$$

$$\operatorname{ch} x=\frac{1}{2}(e^x+e^{-x}).$$

Заметим, однако, что для x, близких к нулю, e^x и e^{-x} почти равны, поэтому при вычислении $\operatorname{sh} x$ придется производить вычитание двух почти равных чисел, а от этого теряется точность результата, как мы уже видели в разд. 2.8. Если при вычислениях важна высокая относительная точность, то предпочтительнее воспользоваться степенным рядом

$$\operatorname{sh} x=x+\frac{x^3}{3!}+\frac{x^5}{5!}+\ldots.$$

При малых x достаточно будет взять несколько первых членов ряда.

Логарифмы

Если в исходной программе на ФОРТРАНе написана функция LOGF, то будет вычислен натуральный логарифм, т. е. логарифм по основанию e. В некоторых вариантах

[1]) См., например, Maehly H. J., Methods for fitting rational approximation, *Comm. ACM*, **7**, 150-162 (апрель 1960).

ФОРТРАНа предусмотрена возможность вычисления наряду с натуральными также и десятичных логарифмов чисел, но это сделано только для того, чтобы облегчить написание программ в тех частях, где участвуют десятичные логарифмы чисел.

Если дан логарифм некоторого числа по основанию e и требуется найти логарифм того же числа по основанию b, достаточно вспомнить формулу перевода логарифмов от одного основания к другому:

$$\log_b x = (\log_e x)(\log_b e).$$

Таким образом, чтобы получить логарифм некоторого числа по какому-либо основанию, не равному e, достаточно только умножить натуральный логарифм этого числа на логарифм e по новому основанию; этот логарифм является некоторой постоянной величиной, в частности $\log_{10} e = 0.43429448$.

Приближения для некоторых элементарных функций приведены в приложении 2, часть D.

3.8. ПРАКТИЧЕСКИЙ ПРИМЕР 3: ОШИБКИ ПРИ ПРЯМОМ ВЫЧИСЛЕНИИ СИНУСА ПО РЯДУ ТЕЙЛОРА

Ряд Тейлора для синуса имеет вид

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots .$$

Как уже отмечалось выше, теоретически этот ряд пригоден для любых значений x, однако в действительности он практически бесполезен для больших значений x. Поучительно рассмотреть, почему это так получается.

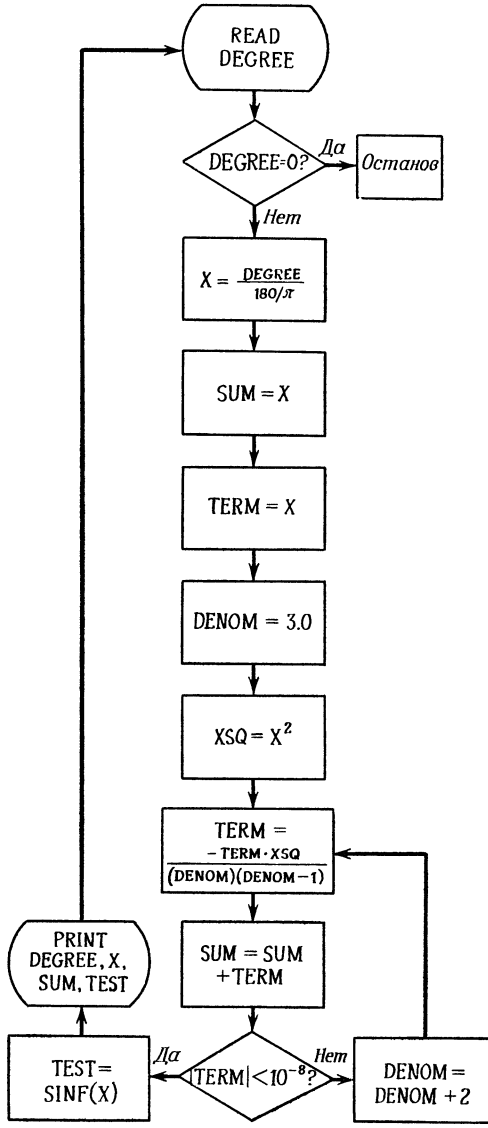
Мы составим программу, которая позволит производить прямое вычисление синуса по ряду Тейлора, т. е. начать с первого члена ряда и вычислять последующие члены, пока не будет найден член ряда, не превосходящий по абсолютной величине, например, 10^{-8}. Мы знаем, что ошибка не превосходит первого отброшенного члена ряда; поэтому, казалось бы, можно вычислить синус с точностью до 10^{-8}, просто взяв достаточное количество членов в ряде Тейлора. Ниже мы убедимся, что это невозможно из-за очень больших ошибок округления.

В программе придется применить некоторые ухищрения для того, чтобы не получить в ходе вычислений промежуточ-

ных результатов, которые были бы слишком велики для записи в памяти ЭЦВМ. Наибольший угол, который мы будем рассматривать, составит около 50 радиан; если попытаться возвести 50 в какую-либо большую степень, то результат может оказаться слишком большим для большинства известных систем ФОРТРАН. Поэтому в программе будет использован следующий прием: каждый следующий член ряда будет вычисляться по предыдущему члену ряда с помощью простого рекуррентного соотношения. Делается это весьма просто: имея первый член ряда, равный x, мы получаем второй член, умножая первый на $-x^2$ и деля полученное произведение на 2 и на 3; имея второй член, получаем третий, умножая второй на $-x^2$ и деля на 4 и на 5. Короче говоря, каждый следующий член ряда получается из предыдущего умножением на $-x^2$ и делением на произведение следующих двух целых чисел.

Блок-схема программы изображена на рис. 3.9. В программе предусмотрено чтение перфокарт, на каждой из которых отперфорировано значение угла в градусах (DEGREE); выход из цикла производится при чтении пробной карты, на которой задан угол, равный нулю. Угол, выраженный в градусах, сначала переводится в радианы делением на $180/\pi$; результат становится значением переменной X. Теперь начинается процесс вычисления последовательных членов ряда. Мы будем каждый раз прибавлять новый член ряда к предыдущей сумме; в конце концов именно эта сумма и станет значением синуса, когда будет просуммировано достаточное количество членов. Для начала счета сумме присваивается значение X; член, который необходимо вычислить, равен $-x^3/3!$, а предыдущий член равен X. Для того чтобы получать последовательные числа натурального ряда, мы присваиваем переменной DENOM значение 3.0. Для того чтобы не вычислять каждый раз снова x^2, мы вычисляем его однажды перед началом внутреннего цикла и присваиваем это значение переменной XSQ.

Вычисление нового члена ряда сводится теперь просто к умножению предыдущего члена на —XSQ и делению получившегося произведения на DENOM и на DENOM —1.0. Вычисленный таким образом новый член ряда прибавляется к сумме, и если требуется вычислить следующий за ним член, то операции вычисления нового члена ряда



Р и с. 3.9. Блок-схема программы для вычисления синуса (практический пример 3).

повторяются, причем только что вычисленный член берется в качестве «предыдущего», а величина переменной DENOM увеличивается на 2.0. Остается определить момент, когда вычислено достаточное количество членов ряда. Для этого только что вычисленный член ряда сравнивается по абсолютной величине с 10^{-8}. Если он окажется меньше или равен 10^{-8}, то необходимо напечатать результат и прочесть следующую перфокарту с входной информацией; если же он больше 10^{-8} по абсолютной величине, то необходимо вычислить следующий член ряда.

```
1    5 6 7  FORTRAN STATEMENT
      62    READ 100, DEGREE
     100    FORMAT (F10.0)
            IF (DEGREE) 150, 200, 150
     200    STOP
     150    X = DEGREE / 57.2957795
            SUM = X
            TERM = X
            DENOM = 3.0
            XSQ = X * X
      25    TERM = -TERM * XSQ / (DENOM * (DENOM - 1.0))
            SUM = SUM + TERM
            IF (ABSF(TERM) - 1.E-8) 16, 16, 12
      12    DENOM = DENOM + 2.0
            GO TO 25
      16    TEST = SINF(X)
            PRINT 30, DEGREE, X, SUM, TEST
      30    FORMAT (F10.0, F15.8, F20.8, F15.8)
            GO TO 62
            END
```

Р и с. 3.10. Программа для вычисления синуса (практический пример 3).

Наряду с печатанием величины синуса, вычисленной при помощи этого метода, интересно также сравнить ее с величиной синуса того же угла, полученной с помощью стандартной функции, имеющейся в трансляторе ФОРТРАНа.

Поэтому переменной TEST перед печатанием окончательного результата присваивается значение SINF(X) и эта переменная также печатается.

Программа, изображенная на рис. 3.10, соответствует блок-схеме. Величины полей в спецификации оператора 30 были выбраны с тем расчетом, чтобы в этих полях уместились ожидаемые результаты.

На рис. 3.11 напечатаны результаты вычислений по этой программе. Вычисления производились для углов, равных 30° плюс угол, кратный 360°. Точный результат в каждом

30.	0.52359878	0.49999999	0.49999999
390.	6.80678415	0.49999993	0.50000005
750.	13.08996952	0.50013507	0.50000010
1110.	19.37315488	0.51658490	0.50000016
1470.	25.65634012	24.25401855	0.50000010
1830.	31.93952560	14380.23767090	0.50000025
2190.	38.22271109	25902480.00000000	0.50000040
2550.	44.50589609	-130402508.00000000	0.50000013
2910.	50.78908157	-83272283.00000000	0.50000029

Рис. 3.11. Выходная печать для программы рис. 3.10 (практический пример 3).

случае должен быть равен 0.5. Значение для 30° очень точно соответствует тому, что ожидалось, ошибка составляет не более 10^{-8}. При увеличении угла ошибка возрастает. Для 390° значение вполне приемлемо, для 750° и 1110° ошибка существенно возрастает, для 1470° вычисленное значение синуса получается абсолютно бессмысленным, то же самое имеет место и для еще бóльших углов.

Попытаемся установить, что происходит при том значении аргумента, когда получается первый бессмысленный результат (при 1470°). Используя отдельную программу, не приведенную здесь, можно выводить на печать отдельные члены ряда по мере их последовательного вычисления. Первый член равен величине X, 25.656340 радиана. Второй член с точностью до восьми значащих цифр равен —2789.0484; сумма этих двух членов с точностью до восьми значащих цифр равна —2763.3921. При сложении были, таким образом, потеряны последние две значащие цифры первого члена ряда. Естественно, эта потеря является безвозвратной, эти две значащие цифры больше нигде не будут введены в окончательную сумму. Третий член равен

89849.610; в результате сложения получается сумма, равная 87086.218, при этом теряется последняя значащая цифра предыдущей суммы. Таким образом, потеряны уже три значащие цифры первого члена ряда. Четвертый член равен —1362035.9; после очередного сложения получается сумма, равная —1274949.7, и две последние значащие цифры предыдущей суммы снова потеряны. Число потерянных значащих цифр первого члена возросло уже до пяти, и картина постепенной потери точности должна стать ясной читателю. Наибольший по абсолютной величине член ряда равен $55037680 \cdot 10^2$, после прибавления которого к предыдущей сумме будут потеряны *все* значащие цифры первого члена ряда, а вместе с ними и некоторые значащие цифры последующих членов. Для следующего значения 1830° (см. рис. 3.11) наибольший член ряда составляет по величине около $2.7 \cdot 10^{12}$, что приводит к потере всех значащих цифр первого и второго членов ряда при суммировании.

Источник неприятностей в данном случае состоит в том, что сложение производится далеко не в наилучшей последовательности. Как уже указывалось в гл. 2, лучше всего начинать сложение с наименьших членов, или, что более правильно, вести сложение в таком порядке, чтобы частичные суммы были наименьшими по абсолютной величине.

Но неправильным порядком сложения неприятности не исчерпываются. Дело в том, что вычисления по программе производились на ЭЦВМ, работавшей в двоичной системе счисления, и действительные числа представлялись в этой ЭЦВМ таким количеством двоичных разрядов, которое приблизительно эквивалентно восьми десятичным значащим цифрам. Рассмотрим теперь такой член ряда Тейлора, как, например, $0.26553689 \cdot 10^{13}$. Если написать это число без порядка, то оно будет равно 2 655 368 900 000, причем нули не имеют никакого смысла, они просто поставлены вместо тех цифр, которые не уместились в памяти ЭЦВМ, так как ЭЦВМ *не может вычислить число точнее, чем с восьмью значащими цифрами*. Другими словами, это приближение может отличаться от истинного значения данного члена ряда на величину до 50 000. Естественно, что при таких громадных ошибках совершенно безнадежно ожидать, что можно более или менее точно вычислить окончательный результат, который наверняка не может быть больше 1.

Для того чтобы убедиться, что все затруднения действительно происходят от недостаточного количества значащих цифр, можно произвести вычисления с *удвоенной точностью.* Удвоенная точность — это такой метод выполнения арифметических операций в ЭЦВМ, когда каждое число занимает не одну, а две ячейки памяти и поэтому представляется вдвое большим, чем обычно, количеством значащих цифр, и все эти значащие цифры принимаются во внимание при выполнении арифметических операций. В том варианте ФОРТРАНа, который был использован в этой задаче, пере-

30.	0.52359878	0.49999999	0.49999999
390.	6.80678415	0.50000006	0.50000005
750.	13.08996952	0.50000011	0.50000010
1110.	19.37315488	0.50000016	0.50000016
1470.	25.65634012	0.50000143	0.50000010
1830.	31.93952560	0.49953845	0.50000025
2190.	38.22271109	0.79868912	0.50000040
2550.	44.50589609	29.53991437	0.50000013
2910.	50.78908157	-142982.02734375	0.50000029

Р и с. 3.12. Выходная печать для программы рис. 3.10, где вычисления были сделаны с удвоенной точностью (практический пример 3).

ход к удвоенной точности достигается тем, что перфокарты с арифметическими операторами должны иметь в первой колонке букву D. При этом все арифметические операции, обозначенные в данном операторе, будут выполнены с удвоенной точностью. Результаты работы той же программы с удвоенной точностью показаны на рис. 3.12. Нетрудно убедиться, что результаты вычислений по этой программе получаются сравнительно неплохими вплоть до 1830°, где программа с обычной точностью уже дала полную бессмыслицу. При бóльших углах, однако, даже удвоенная точность не спасает положения.

Коротко остановимся на том, как работает программа вычисления синуса, имеющаяся в трансляторе ФОРТРАНа. В любом случае эта программа дает не меньше шести достоверных значащих цифр. Уменьшение точности при переходе к большим углам вызвано увеличением ошибки при сведении исходного угла к углу меньше $\pi/2$. Например, когда необходимо перевести 2190° в радианы, то получается результат, равный 38.22271109. При выводе этого числа

было напечатано десять значащих цифр, но в действительности значащих цифр в этом числе не может быть более восьми. Если свести этот угол к углу меньше $\pi/2$, то при этом будет произведено вычитание двух почти равных чисел и относительная ошибка результата существенно возрастет. Другими словами, вычислялся синус не 2190°, а слегка отличного угла.

Синусы больших углов, естественно, не вычисляются методом, описанным в этом параграфе. Авторы надеются, что читатели получили некоторое представление о проблемах, с которыми приходится сталкиваться при вычислениях на ЭЦВМ, где все величины по необходимости являются приближенными и ограничены конечным количеством значащих цифр. Наивные вычислители часто думают, что если «гигантский мозг» печатает восемь значащих цифр, то они должны что-нибудь значить. Авторы надеются, что содержание этого параграфа предостережет некоторых вычислителей от излишней доверчивости.

Упражнения

1. Ниже приведены некоторые функции и соответствующие им ряды Тейлора. Для указанного значения x определите число членов ряда, необходимых для того, чтобы вычислить значение функции с ошибкой ограничения не более $5\cdot10^{-5}$. Определите также число членов ряда, необходимых для вычисления значения функции с ошибкой не более $5\cdot10^{-9}$. Ошибками округления пренебречь.

*а. $\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots, \quad x = 1.$

б. $\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots, \quad x = 3.$

*в. $\operatorname{arctg} x = \frac{\pi}{2} - \frac{1}{x} + \frac{1}{3x^3} - \frac{1}{5x^5} + \frac{1}{7x^7} + \ldots, \quad x = 2.$

(Ряд сходится для $x > 1$.)

г. $\log_e x = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} + \ldots, \quad x = 2.$

(Ряд сходится для $0 < x \leqslant 2$.)

д. $e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots, \quad x = 1.$

2. Покажите, что при $|x| \leqslant 1$ полиномы Чебышева удовлетворяют неравенству $|T_n(x)| \leqslant 1$.

3. Покажите, что

$$\int_{-1}^{1} \frac{T_m(x)\,T_n(x)\,dx}{\sqrt{1-x^2}} = 0, \qquad m \neq n.$$

Это свойство полиномов Чебышева называется их *ортогональностью* на интервале $(-1, +1)$ с весом $1/\sqrt{1-x^2}$. (Указание: используйте тригонометрическое определение полиномов Чебышева и замените x на θ.)

4. Покажите, что

$$\int_{-1}^{1} \frac{[T_m(x)]^2\,dx}{\sqrt{1-x^2}} = \begin{cases} \pi, & m = 0, \\ \dfrac{\pi}{2}, & m = 1, 2, \ldots . \end{cases}$$

5. Используя результаты упражнений 3 и 4, покажите, что коэффициенты a_i в разложении функции $f(x)$ в ряд по полиномам Чебышева

$$f(x) = \sum_{i=0}^{\infty} a_i T_i(x)$$

могут быть определены следующим образом:

$$a_0 = \frac{1}{\pi} \int_{-1}^{1} \frac{f(x)\,dx}{\sqrt{1-x^2}},$$

$$a_i = \frac{2}{\pi} \int_{-1}^{1} \frac{f(x)\,T_i(x)\,dx}{\sqrt{1-x^2}}, \qquad i \neq 0.$$

На практике эти формулы редко применяются для вычисления коэффициентов a_i, потому что вычисление интегралов сопряжено с большими трудностями.

***6.** Исходя из результата упражнения 5, найдите первые пять коэффициентов разложения в ряд по полиномам Чебышева функции

$$f(x) = x.$$

Можно использовать следующие определенные интегралы:

$$\int_{-1}^{1} \frac{dx}{\sqrt{1-x^2}} = \pi, \quad \int_{-1}^{1} \frac{x^2 dx}{\sqrt{1-x^2}} = \frac{\pi}{2}, \quad \int_{-1}^{1} \frac{x^4 dx}{\sqrt{1-x^2}} = \frac{3\pi}{8},$$

$$\int_{-1}^{1} \frac{x\,dx}{\sqrt{1-x^2}} = 0, \quad \int_{-1}^{1} \frac{x^3 dx}{\sqrt{1-x^2}} = 0, \quad \int_{-1}^{1} \frac{x^5 dx}{\sqrt{1-x^2}} = 0.$$

7. Исходя из того факта, что шестой коэффициент a_5 в разложении из предыдущего упражнения должен обратиться в нуль, получить

$$\int_{-1}^{1} \frac{x^6 dx}{\sqrt{1-x^2}} = \frac{5\pi}{16}.$$

Используйте интегралы из предыдущего упражнения.

*8. а. Найдите первые пять коэффициентов разложения по полиномам Чебышева функции

$$f(x) = \sqrt{1-x^2}.$$

б. Напишите первые пять членов ряда, расписав полиномы Чебышева по степеням x.

в. Вычислите значение функции по разложению в ряд Чебышева с пятью членами при $x = +0.5$; сравните вычисленное значение с истинным значением функции и со значением функции, вычисленным по разложению в ряд Тейлора с пятью членами в окрестности $x = 0$.

9. а. Используя интегралы из упражнений 6 и 7, найдите первые пять членов разложения по полиномам Чебышева для функции

$$f(x) = |x|.$$

б. Напишите первые пять членов ряда, расписав полиномы Чебышева по степеням x.

в. Вычислите значение функции по этому разложению при $x = +0.5$.

*10. Сдвинутые полиномы Чебышева можно определить следующим образом:

$$T_n^*(x) = T_n(2x-1).$$

Сдвинутые полиномы T_n^* используются для интервала $0 \leqslant x \leqslant 1$ так же, как обычные полиномы T_n используются для интервала $-1 \leqslant x \leqslant 1$. Найдите четыре первых сдвинутых полинома Чебышева.

11. Покажите, что сдвинутые полиномы Чебышева ортогональны на интервале $(0, +1)$ с весом $1/\sqrt{x-x^2}$, т. е.

$$\int_0^1 \frac{T_n^*(x)\, T_m^*(x)\, dx}{\sqrt{x-x^2}} = 0, \qquad m \neq n.$$

*12. Проведите экономизацию степенного ряда

$$e^x \approx 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \frac{x^6}{6!} + \frac{x^7}{7!}$$

и сведите его к полиному, в котором наивысшая степень x будет x^6.

13. Проведите экономизацию результата упражнения 12 и сведите его к полиному степени не выше x^5.

14. а. Выразите следующую функцию через полиномы Чебышева:

$$f(x)=6x^4-2x^3+x^2-x+4, \quad |x|<1.$$

б. Примените дважды правило экономизации и найдите приближение этой функции с помощью полинома второго порядка.

15. Покажите, что выражение

$$e^x \approx 1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\frac{x^4}{4!}+\frac{x^5}{5!}$$

можно переписать в виде

$$e^x \approx 1+x\left(1+\frac{x}{2}\left(1+\frac{x}{3}\left(1+\frac{x}{4}\left(1+\frac{x}{5}\right)\right)\right)\right).$$

16. Покажите, что выражение

$$\operatorname{arctg} x \approx x-\frac{x^3}{3}+\frac{x^5}{5}-\frac{x^7}{7}+\frac{x^9}{9}$$

можно переписать в виде

$$\operatorname{arctg} x \approx x\left(1-x^2\left(\frac{1}{3}-x^2\left(\frac{1}{5}-x^2\left(\frac{1}{7}-x^2\left(\frac{1}{9}\right)\right)\right)\right)\right).$$

17. Найдите выражение, подобное выражениям из упражнений 15 и 16, для

$$\arcsin x \approx x+\frac{x^3}{2\cdot 3}+\frac{1\cdot 3\cdot x^5}{2\cdot 4\cdot 5}+\frac{1\cdot 3\cdot 5\cdot x^7}{2\cdot 4\cdot 6\cdot 7}+\frac{1\cdot 3\cdot 5\cdot 7\cdot x^9}{2\cdot 4\cdot 6\cdot 8\cdot 9}.$$

18. Найдите выражение, подобное выражениям из упражнений 15 и 16, для

$$J_0(x) \approx 1-\frac{(x/2)^2}{1^2}+\frac{(x/2)^4}{1^2\cdot 2^2}-\frac{(x/2)^6}{1^2\cdot 2^2\cdot 3^2}+\frac{(x/2)^8}{1^2\cdot 2^2\cdot 3^2\cdot 4^2}.$$

***19.** Если число членов некоторого ограниченного сходящегося ряда известно заранее, то наилучшим способом вычисления является правило Горнера. Напишите арифметический оператор, с помощью которого можно произвести по правилу Горнера вычисление e^x с помощью ряда Тейлора, имеющего члены степени до x^6 включительно.

а. Если значения коэффициентов заданы в десятичной форме:

$$e^x \approx 1+x+0.5x^2+0.166667x^3+0.0416667x^4+$$
$$+0.00833333x^5+0.00138889x^6.$$

б. Используя в качестве констант целые числа от 1 до 6, согласно методу, разработанному в упражнении 15.

***20.** Рассмотрим следующий вычислительный процесс:

1. Присвоить переменной E значение 1.
2. Присвоить переменной D значение 5.
3. Заменить прежнее значение E на $1+(E*X/D)$.

4. Если D равно 1, прекратить вычисления, в противном случае отнять от текущего значения D единицу и вернуться к пункту 3.

Покажите, что, когда вычисления будут окончены,

$$E = 1 + x + \frac{x^2}{2} + \frac{x^3}{2 \cdot 3} + \frac{x^4}{2 \cdot 3 \cdot 4} + \frac{x^5}{2 \cdot 3 \cdot 4 \cdot 5} \approx e^x.$$

Начертите блок-схему вычислительного процесса и напишите участок программы, в котором подразумевается, что переменной X было присвоенно некоторое значение в предшествующих операторах.

21. Следуя методу, разработанному в упражнении 20, напишите участок программы, в котором производилось бы вычисление функции $\arcsin x$ по ее разложению в ряд Тейлора вплоть до члена степени x^{15}. (См. упражнение 17.)

22. Напишите участок программы для вычисления функции $\sin x$ для значений x из интервала $(-3, +3)$ с ошибкой ограничения не более $5 \cdot 10^{-9}$. При составлении программы используйте подходящий к данному случаю вариант метода, разработанного в упражнении 20. Если $|x| \leqslant 1$, то переменной D можно присвоить значение 7 перед началом вычислительного цикла, если же $|x| > 1$, то переменной D следует присвоить такое значение, которое получилось при решении упражнения 1, б.

23. Если число членов ряда, которые примут участие в вычислении, заранее неизвестно, то правило Горнера применить не удастся и необходимо вычислять ряд так, как он написан, т. е. вычисляя и суммируя отдельные члены ряда. Однако если x велико, например 10 или 20, то при возведении x в большие степени могут получиться результаты, не умещающиеся в разрядной сетке ЭЦВМ. Одно из возможных решений разобрано в разд. 3.8: деление в знаменателе очередного члена ряда производится одновременно с возведением x в очередную степень. Напишите программу для вычисления e^{-x}, причем вычисляйте по очереди все члены ряда и прекратите вычисление тогда, когда очередной член станет по абсолютной величине меньше 10^{-7}.

24. Напишите программы для вычисления следующих функций методом разд. 3.8, продолжая вычисления до тех пор, пока абсолютная величина очередного члена не станет меньше 10^{-7}:

а. $J_0(x) = 1 - \frac{(x/2)^2}{1^2} + \frac{(x/2)^4}{1^2 \cdot 2^2} - \ldots$.

б. $\operatorname{arctg} x = \frac{\pi}{2} - \frac{1}{x} + \frac{1}{3x^3} - \frac{1}{5x^5} + \ldots, \quad x > 1.$

в. $J_2(x) = \frac{x^2}{2^2 \cdot 2!} - \frac{x^4}{2^4 \cdot 1! \cdot 3!} + \frac{x^6}{2^6 \cdot 2! \cdot 4!} - \frac{x^8}{2^8 \cdot 3! \cdot 5!} + \ldots$.

25. Как было показано выше, полином степени n можно представить в виде

$$p(x) = (x - x_0)\, q(x) + b_0,$$

где

$$q(x) = b_1 + b_2 x + \ldots + b_n x^{n-1},$$

и величины b_j можно вычислить из рекуррентных соотношений

$$b_n = a_n,$$
$$b_j = a_j + x_0 b_{j+1}; \qquad j = n-1, \ldots, 0.$$

а. Покажите, что

$$\frac{dp}{dx} = q(x).$$

б. Найдите простую рекуррентную формулу, позволяющую вычислить производную $p(x)$ при $x = x_0$. (Указание: заметьте, что $q(x)$ является полиномом степени $n-1$.)

***26.** Напишите арифметический оператор, с помощью которого можно было бы произвести вычисления по формуле (3.17).

27. Напишите арифметические операторы для вычисления пяти приближений из приложения 2, часть D.

***28.** Ряд Тейлора для синуса

$$\sin\left(\frac{\pi x}{2}\right) \approx \frac{\pi}{2}x - \left(\frac{\pi}{2}\right)^3 \frac{x^3}{6} + \left(\frac{\pi}{2}\right)^5 \frac{x^5}{120} - \left(\frac{\pi}{2}\right)^7 \frac{x^7}{5040};$$

найдите коэффициенты рационального приближения вида

$$\sin\left(\frac{\pi x}{2}\right) \approx \frac{b_1 x + b_3 x^3}{1 + c_2 x^2 + c_4 x^4}.$$

Используя тот факт, что синус является нечетной функцией, т. е. $\sin(-x) = -\sin x$, объясните, почему в рассматриваемой формуле можно положить $b_0 = b_2 = c_1 = c_3 = 0$.

29. Найдите коэффициенты в рациональном приближении для $\cos(\pi x/2)$. Рассмотрите следующую формулу:

$$\cos\left(\frac{\pi x}{2}\right) \approx \frac{b_0 + b_2 x^2}{1 + c_2 x^2}.$$

30. Ряд Тейлора для тангенса

$$\operatorname{tg} x \approx x + \frac{1}{3}x^3 + \frac{2}{15}x^5 + \frac{17}{315}x^7.$$

Найдите коэффициенты в рациональном приближении вида

$$\operatorname{tg} x \approx \frac{b_1 x + b_3 x^3}{1 + c_2 x^2 + c_4 x^4}.$$

***31.** Для рационального приближения

$$f(x) = \frac{a + bx + cx^2}{1 + dx}$$

найдите соответствующую ему непрерывную дробь вида

$$f(x)=k_1+\cfrac{x}{k_2+\cfrac{x}{k_3+\cfrac{x}{k_4}}}.$$

32. Найдите непрерывную дробь вида

$$f(x)=k_1+\cfrac{x}{k_2+\cfrac{x}{k_3+\cfrac{x}{k_4+\cfrac{x}{k_5}}}}$$

для рационального приближения

$$f(x)=\frac{a+bx+cx^2}{1+dx+ex^2}.$$

33. Найдите непрерывную дробь вида

$$f(x)=\cfrac{1}{k_1+\cfrac{x}{k_2+\cfrac{x}{k_3+\cfrac{x}{k_4}}}}$$

для рационального приближения

$$f(x)=\frac{a+bx}{1+dx+ex^2}.$$

34. Из биномиальной теоремы следует:

$$(1-x)^{1/2}\approx 1-\frac{1}{2}x-\frac{1}{8}x^2-\frac{3}{24}x^3-\frac{15}{384}x^4-\frac{105}{3840}x^5.$$

а. Проведите экономизацию ряда так, чтобы в правой части остались члены степени не выше x^3.

б. Исходя из результата предыдущего пункта а, найдите рациональное приближение вида

$$(1-x)^{1/2}\approx\frac{a+bx+cx^2}{1+dx}.$$

в. Исходя из рационального приближения, найдите соответствующую ему непрерывную дробь вида

$$(1-x)^{1/2}\approx k_1+\cfrac{x}{k_2+\cfrac{x}{k_3+\cfrac{x}{k_4}}}.$$

35. Покажите, что логарифм любого конечного положительного числа может быть получен из логарифма некоторого другого числа, заключенного между 0 и 1. Таким образом, для того чтобы вычислить логарифм любого числа, достаточно иметь приближенную формулу для аргументов, меньших 1.

ГЛАВА

4

НЕКОТОРЫЕ ПРОСТЫЕ ПРОГРАММЫ

4.1. ВВЕДЕНИЕ

У читателя теперь должно быть достаточно знаний для того, чтобы начать самостоятельную работу с ЭЦВМ. В этой главе мы рассмотрим три практических примера, которые помогут читателю уяснить процесс составления программы для решения какой-либо конкретной задачи. Мы покажем всю последовательность этого процесса: анализ математических формул, анализ ошибок, что делать для отладки программы, организация ввода и вывода информации и т. д.

Таким образом, основное внимание будет уделено тому, как сопоставить и объединить в единое целое те разрозненные приемы программирования, которые были изложены выше. Примеры, с помощью которых это иллюстрируется, взяты непосредственно из инженерной практики. Они довольно типичны по тому объему работы, который приходится проделывать при составлении простых программ, хотя, конечно, во многих случаях задачи, решаемые с помощью ЭЦВМ, бывают существенно сложнее.

4.2. ПРАКТИЧЕСКИЙ ПРИМЕР 4: РАСЧЕТ КОЛОННЫ

Инженер желает получить цифровые данные для построения графика безопасной нагрузки несущей колонны как функции отношения высоты колонны к ее диаметру. В справочнике удалось найти две эмпирические формулы, позволяющие вычислить эту безопасную нагрузку:

$$S = \begin{cases} 17\,000 - 0.485R^2 & \text{для } R < 120; \\ \dfrac{18\,000}{1 + \dfrac{R^2}{18\,000}} & \text{для } R \geqslant 120. \end{cases}$$

В этих формулах:

S — безопасная нагрузка в фунтах на кв. дюйм,

R — отношение высоты колонны к диаметру.

Безопасную нагрузку следует вычислить для значений R от 20 до 200 с шагом 5.

Приступая к составлению программы, нетрудно заметить, что прежде всего придется дать ответ на три следующих вопроса:

1. Как производить выбор между двумя формулами, если задано значение R? Этот вопрос решается довольно просто: нужно использовать оператор IF для того, чтобы сравнить величину R с числом 120, после чего оператор IF передаст управление тому из двух арифметических операторов, который осуществит вычисления по соответствующей формуле.

2. Как произвести вычисления для всей последовательности значений R? Этот вопрос можно решить различными путями. Вероятно, самый простой путь заключается в том, чтобы присвоить R значение 20, вычислить S, прибавить 5 к величине R, снова вычислить S и т. д., каждый раз при этом проверяя, не достигло ли R величины 200. При этом не потребуется чтения входных перфокарт, программа существенно упрощается, но зато произвести те же вычисления для каких-либо других значений R, не предусмотренных заранее в программе, нельзя без переделки программы. Выигрывая в простоте, мы тем самым лишаем программу универсальности — ситуация довольно распространенная.

3. В каком виде представлять результаты, принимая во внимание ожидаемую точность вычислений? С точки зрения самих вычислений вопросы точности в этой задаче не возникают: вычитания двух почти равных чисел не производится, ошибки округления будут с лихвой перекрыты неточностью исходных формул и нет никакого процесса, который мог бы привести к накоплению ошибок. В действительности вся точность вычислений в данном случае целиком зависит от точности исходных формул. Справочник, из которого взяты эти формулы, хранит молчание относительно их точности (это случается довольно часто), но кое-какие выводы можно извлечь хотя бы из того факта, что числовые константы в этих формулах приведены всего с двумя-тремя значащими цифрами. Более того, формулы этого

типа всегда включают в себя определенный запас прочности, так что высокая точность в этих расчетах подразумеваться никак не может.

Ниже мы убедимся, однако, что вопросы точности возникают и в этой задаче, если требуется произвести вычисления для последовательных значений R с другим шагом, а не с тем, что указан выше. Мы впоследствии возвратимся к этому вопросу.

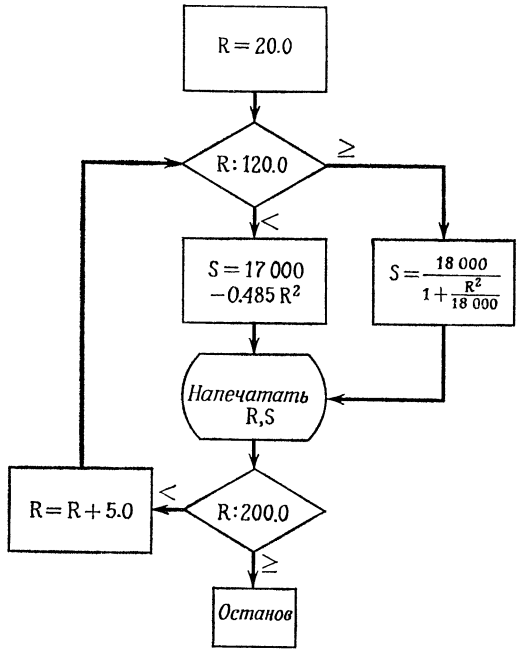
Исходя из того, что высокая точность вычислений не требуется в этой задаче, наилучшей формой представления выходной информации надо признать такую форму, где печатались бы около трех значащих цифр и десятичный порядок. Таким образом, для выводимой информации придется предусмотреть спецификацию поля типа E.

Блок-схема программы показана на рис. 4.1. Вычисления начинаются с того, что R присваивается значение 20; этот шаг в программе повторяться не будет. Затем сразу необходимо проверить, не равно ли R 120. Этот шаг кажется бессмысленным, потому что мы положили R равным 20. Однако дело в том, что с этого места начинается тот участок программы, который будет повторяться многократно, и не существует другого способа правильно произвести все вычисления (применяя каждый раз соответствующую формулу), как проверять равенство R и числа 120. Нельзя просто сказать машине: «Действуй разумно, вычисляй по первой формуле, пока R не достигнет 120». Такое сравнение *должно быть предусмотрено* заранее в программе программистом.

Результат сравнения определяет, по какой из двух формул будет вычислена величина S, и после вычисления величины R и S печатаются независимо от того, по какой формуле они были вычислены. (Между прочим, в большинстве случаев целесообразно печатать и выходной результат и аргумент. Если печатать только результат, то тому, кто захочет этот результат использовать, придется самому вписывать в бланк все значения R, соответствующие последовательным значениям S.)

Затем величина R сравнивается с числом 200. Если она равна 200, то это означает, что последняя строка результатов уже напечатана и пора остановиться. Если она меньше 200, то необходимо прибавить 5 к R и повторить вычисления.

Можно подумать, что при таком построении программы R не может стать больше 200, так как в противном случае вычисления были бы остановлены еще при предыдущем прохождении цикла, когда R стало равно 200. Это действительно верно при условии, что вычисления производятся



Р и с. 4.1. Блок-схема программы для расчета конструкции колонны (практический пример 4).

с бесконечным количеством значащих цифр, а в *данном конкретном случае с данными конкретными числами* (R меняется от 20 до 200 с шагом 5) это будет справедливо и при вычислениях с конечной точностью. К этому вопросу мы скоро вернемся.

Программа, написанная на ФОРТРАНе, приведена на рис. 4.2. Если читатель хорошо разобрался в блок-схеме, то прочесть программу будет очень легко. В операторе FORMAT использованы две наиболее распространенные

спецификации поля. Для аргумента использована спецификация F10.0, где буква F означает, что печатается действительное число в обычной форме (без порядка), 10 означает, что при печати этому числу будут отведены на выходном бланке 10 колонок, 0 означает, что число не имеет дробной части. Для печати результата вычислений использован формат 1PE 15.3, где 1P поставлено для того, чтобы десятичная точка была напечатана не перед первой значащей цифрой, а между первой и второй цифрами, что более привычно,

```
1    5 6 7  FORTRAN STATEMENT
       R = 20
  10   IF(R - 1120.) 20, 30, 30
  20   S = 1.7E4 - .485 * R * R
       GO TO 40
  30   S = 1.8E4/(1. + R * R / 1.8E4)
  40   PRINT 70, R, S
  70   FORMAT (F10.0, 1PE15.3)
       IF(R - 200.) 50, 60, 60
  50   R = R + 5.
       GO TO 10
  60   STOP
       END
```

Р и с. 4.2. Программа для расчета конструкции колонны (практический пример 4).

Е означает форму представления числа с десятичным порядком, 15 означает 15 колонок на выходном бланке и 3 соответствует трем значащим цифрам после десятичной точки.

На выходном бланке решение задачи будет напечатано так, как это показано на рис. 4.3.

Вернемся теперь снова к вопросу о том, как определять окончание цикла, т. е. как сравнивать R и 200. Предположим, что нужно вычислить S для значений R от 20 до 200, но на этот раз с шагом 0.1. Для этого достаточно изменить константу в операторе 50. Одновременно с этим изменим спецификации в операторе FORMAT, чтобы каждое число теперь печаталось с 7 значащими цифрами. Несколько последних строчек печати на выходном бланке приведены на рис. 4.4.

```
 20.       1.681E 04
 25.       1.670E 04
 30.       1.656E 04
 35.       1.641E 04
 40.       1.622E 04
 45.       1.602E 04
 50.       1.579E 04
 55.       1.553E 04
 60.       1.525E 04
 65.       1.495E 04
 70.       1.462E 04
 75.       1.427E 04
 80.       1.390E 04
 85.       1.350E 04
 90.       1.307E 04
 95.       1.262E 04
100.       1.215E 04
105.       1.165E 04
110.       1.113E 04
115.       1.059E 04
120.       1.000E 04
125.       9.636E 03
130.       9.284E 03
135.       8.944E 03
140.       8.617E 03
145.       8.302E 03
150.       8.000E 03
155.       7.710E 03
160.       7.431E 03
165.       7.164E 03
170.       6.908E 03
175.       6.663E 03
180.       6.429E 03
185.       6.204E 03
190.       5.989E 03
195.       5.783E 03
200.       5.586E 03
```

Рис. 4.3. Выходная печать для программы рис. 4.2. (практический пример 4).

Что же произошло? Почему последнее значение R не равно 200? Читателю, изучившему гл. 2, все должно стать ясным, если принять во внимание, что вычисления производились на ЭЦВМ в двоичной системе счисления. Дело в том, что в двоичной системе не существует точного эквивалента десятичной 0.1. В ЭЦВМ IBM 7094, на которой решалась эта задача, двоичное приближение к десятичному числу 0.1 отличается от своего истинного значения на величину порядка 2^{-33}, или 10^{-8}. Эта величина слишком мала для

того, чтобы повлиять на точность результата вычислений, но ошибка в величине R постепенно накапливается, когда 0.1 прибавляется к R много сотен раз, а именно так и обстоит дело в последнем варианте программы.

Если бы для окончания цикла мы потребовали *точного* равенства R и 200, то этого равенства не удалось бы получить вообще.

При вычислениях с шагом 0.1 на ЭЦВМ, работающей в десятичной системе счисления, эта проблема не возникла бы, так как в такой машине 0.1 можно представить точно. Тем не менее эта трудность является общей независимо от типа применяемой машины. Предположим, например, что вычисления необходимо произвести с шагом 1/7, а эта величина, как нетрудно убедиться, не имеет ни двоичного, ни десятичного точного эквивалента. (Вычислительных машин, работающих в семиричной системе счисления, также нет.)

Решение проблемы заключается в том, чтобы немного усложнить способ сравнения R и 200. Вместо того чтобы непосредственно сравнивать R и 200, можно определить разницу между R и 200 и сравнить ее с некоторой малой величиной. Например, можно написать

```
IF (ABSF (R — 200.) — 0.001) 60, 60, 50
```

В этом случае если абсолютное значение разности между R и 200 меньше или равно 0.001, то происходит переход к оператору STOP, потому что R очень близко к 200. В данном варианте необходимо использовать для сравнения *абсолютное значение* разности между R и 200, так как заранее неизвестно, будет ли при многократных сложениях накапливаться положительная или отрицательная ошибка. Малая величина (допуск) для сравнения должна быть меньше шага 0.1, но больше, чем ожидаемая накопленная ошибка. Если принять допуск 0.2, то вычисления явно будут остановлены преждевременно, если же принять допуск 10^{-8}, то вычисления вообще никогда не остановятся, так как условия выхода из цикла не будут выполнены и управление не будет передано оператору STOP.

Если сделать это изменение в программе и одновременно изменить спецификации в операторе FORMAT таким образом, чтобы R печаталось только с точностью до десятых,

то все вычисления пройдут гладко. Будет вычислено ровно столько значений S, сколько требуется, а накопленная ошибка будет слишком мала, чтобы изменить последнюю значащую цифру R.

```
198.0985    5.660078E 03
198.1985    5.656163E 03
198.2985    5.652250E 03
198.3985    5.648341E 03
198.4985    5.644436E 03
198.5985    5.640534E 03
198.6985    5.636635E 03
198.7985    5.632740E 03
198.8985    5.628849E 03
198.9985    5.624960E 03
199.0985    5.621075E 03
199.1985    5.617194E 03
199.2985    5.613316E 03
199.3985    5.609441E 03
199.4985    5.605570E 03
199.5985    5.601702E 03
199.6985    5.597838E 03
199.7985    5.593977E 03
199.8985    5.590119E 03
199.9985    5.586265E 03
200.0985    5.582414E 03
```

Рис. 4.4. Выходная печать для программы рис. 4.2, преобразованной с целью уменьшить шаг изменения аргумента и напечатать больше значащих цифр (практический пример 4).

```
198.0       5.664E 03
198.1       5.660E 03
198.2       5.656E 03
198.3       5.652E 03
198.4       5.648E 03
198.5       5.644E 03
198.6       5.641E 03
198.7       5.637E 03
198.8       5.633E 03
198.9       5.629E 03
199.0       5.625E 03
199.1       5.621E 03
199.2       5.617E 03
199.3       5.613E 03
199.4       5.609E 03
199.5       5.606E 03
199.6       5.602E 03
199.7       5.598E 03
199.8       5.594E 03
199.9       5.590E 03
200.0       5.586E 03
```

Рис. 4.5. Выходная печать для программы рис. 4.2, где величина шага взята равной 0.1 и применен метод проверки окончания цикла, описанный в тексте (практический пример 4).

Несколько последних строчек выходного бланка для этого окончательного варианта программы, где шаг по-прежнему равен 0.1, приведены на рис. 4.5.

В гл. 7 (разд. 7.7) будет рассмотрен другой способ вычисления последовательности значений с малым шагом аргумента.

4.3. ЧАСТОТНАЯ ХАРАКТЕРИСТИКА СЕРВОМЕХАНИЗМА. ОТЛАДКА ПРОГРАММЫ

Передаточная функция некоторого сервомеханизма задается формулой

$$T(j\omega) = \frac{K}{j\omega(1+jT_1\omega)(1+jT_2\omega)},$$

где ω — круговая частота, *рад/сек*,

$j = \sqrt{-1}$ [1]),

T — передаточная функция,

K — коэффициент усиления,

T_1, T_2 — постоянные времени, *сек*.

В рамках этой книги нет возможности излагать сколько-нибудь подробно теорию сервомеханизмов, однако общее представление о частотной характеристике и о передаточной функции можно получить из следующего примера. Имеется «черный ящик», к входу которого приложен синусоидальный сигнал известной частоты и амплитуды; сигнал появляется на выходе «ящика», причем амплитуда его умножается на абсолютную величину передаточной функции (на ее модуль, если сохранять терминологию, принятую для комплексных чисел), а фаза сдвигается на фазовый угол (аргумент) передаточной функции.

В общем случае передаточная функция будет комплексной. Наиболее простой способ производить вычисления с комплексными числами в ЭЦВМ — это преобразовать формулу таким образом, чтобы разделить действительную и мнимую части. После этого все арифметические операции будут производиться над вещественными числами (как правило, ЭЦВМ не могут работать непосредственно с комплексными числами)[2]).

Преобразованная формула передаточной функции приобретает следующий вид:

$$T(j\omega) = \frac{-K(T_1+T_2)}{(1+T_1^2\omega^2)(1+T_2^2\omega^2)} - \frac{K(1-T_1T_2\omega^2)}{(1+T_1^2\omega^2)(1+T_2^2\omega^2)\omega}\, j.$$

Если теперь написать эту форму передаточной функции в виде $T(j\omega) = a + bj$, то легко понять, что амплитуда

[1]) В математике $\sqrt{-1}$ обозначается через i, в электротехнике и в теории автоматического регулирования i употребляется для обозначения тока, а $\sqrt{-1}$ обозначается через j.

[2]) Некоторые варианты ФОРТРАНа допускают арифметические операции с комплексными числами, при этом число занимает в машине две ячейки памяти: одну — для действительной части и одну — для мнимой части числа. Если в исходной программе определены арифметические действия с комплексными числами, то в трансляторе ФОРТРАНа эти действия преобразуются в раздельные действия над действительными и мнимыми частями комплексных чисел.

выходного сигнала равна амплитуде входного сигнала, умноженной на $\sqrt{a^2 + b^2}$, а его фаза сдвигается на угол $\theta = \text{arctg}\,(b/a)$.

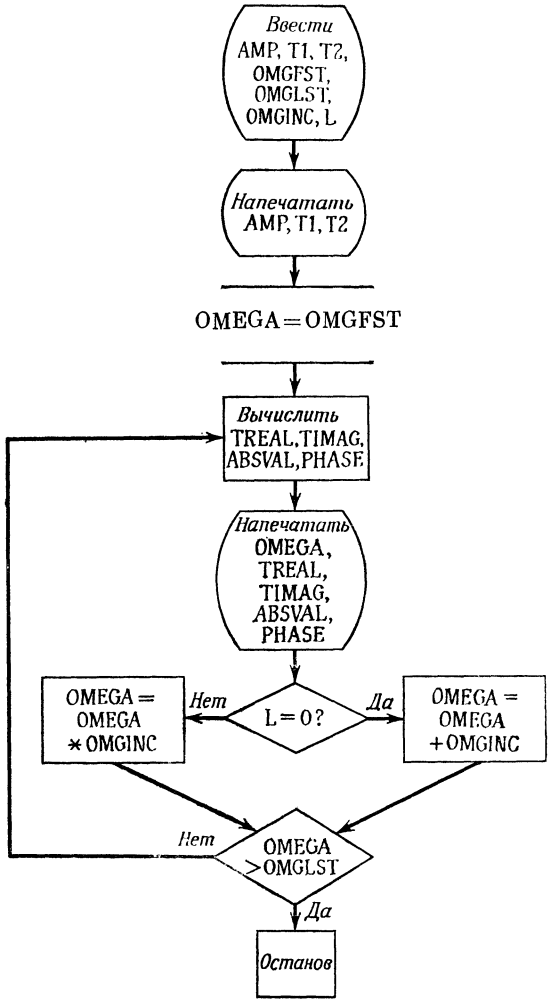
Вычисления должны быть организованы следующим образом. Необходимо ввести в машину значения K, T_1 и T_2 и вычислить передаточную функцию для некоторой последовательности значений ω. Для вычислений также необходимо ввести начальное и конечное значения ω, приращение ω, обозначаемое далее через OMGINC, и целое число L. Если L равно нулю, то OMGINC нужно *прибавлять* к ω перед каждым новым прохождением цикла, но если L не равно нулю, то перед каждым новым прохождением цикла ω нужно *умножать* на OMGINC. Так или иначе, вычисления продолжаются до тех пор, пока T не будет вычислено для наибольшего значения ω, не превосходящего OMGLST.

Для каждого значения ω и для данных K, T_1 и T_2 необходимо печатать действительную и мнимую части передаточной функции, модуль и фазовый угол в градусах.

Дальнейшее рассмотрение станет проще, если мы сейчас договоримся о наименованиях переменных, которые будут использованы в программе.

AMP — коэффициент усиления K.
OMEGA — ω.
T1, T2 — постоянные времени T_1 и T_2.
OMGFST — начальное значение ω.
OMGLST — конечное значение ω.
OMGINC — приращение ω.
L — прибавить OMGINC (L = 0) или умножить на OMGINC (L ≠ 0).
TREAL — действительная часть передаточной функции.
TIMAG — мнимая часть передаточной функции.
ABSVAL — модуль передаточной функции.
PHASE — фазовый угол передаточной функции в градусах.
DENOM — промежуточная переменная, знаменатель в выражениях для действительной и мнимой частей.
OMGSQ — промежуточная переменная ω^2.

Обратите внимание, что наименования переменных выбраны так, чтобы не присвоить действительным перемен-



Р и с. 4.6. Блок-схема программы для вычисления частотной характеристики сервомеханизма (практический пример 5).

ным неправильных наименований. Например, K нельзя использовать для обозначения коэффициента усиления и MODULS — для обозначения модуля. Переменная L названа правильно, так как она будет введена и использована в вычислениях как целая переменная.

Блок-схема программы изображена на рис. 4.6. Программа начинается с ввода информации, и непосредственно вслед за этим печатаются три параметра K, T_1 и T_2; так как программа должна быть использована для последовательности значений ω при заданных K, T_1 и T_2, то очень желательно, чтобы эти параметры являлись как бы заголовком перед печатанием последовательности выходных величин. После этого переменной OMEGA присваивается значение OMGFST, вычисляются четыре выходных величины и печатаются на выходном бланке. После этого значение OMEGA изменяется согласно тому правилу, которое определяется числом L, и производится сравнение нового значения OMEGA с OMGLST. Для того чтобы точно выполнить требования задачи, нужно проверять OMEGA на равенство OMGLST *после того*, как значение OMEGA изменено, в противном случае есть опасность произвести вычисления на один раз больше, чем требуется, и выйти из цикла уже тогда, когда OMEGA стало больше, чем OMGLST.

Программа, приведенная на рис. 4.7, содержит большое количество преднамеренных ошибок. После того как мы поясним основные черты программы, мы подробно рассмотрим, каким образом можно обнаружить и исправить различные виды ошибок. Для того чтобы облегчить читателю понимание дальнейшего материала, всем операторам в программе присвоены номера.

Печатание коэффициента усиления и двух постоянных времени производится с помощью спецификации поля типа 1P3E14.5; эта спецификация весьма произвольна, потому что нет никаких доводов в пользу того или другого количества значащих цифр для этих параметров. Для второго оператора FORMAT использована спецификация типа 1P7E12.4; эти величины выбраны с таким расчетом, чтобы все семь выходных значений можно было напечатать в одну строку на широкоформатной ленте, используемой в качестве выходного бланка. По этой причине пришлось пойти на некоторое уменьшение количества значащих цифр в каждом

выходном числе. Предусмотрена также печать двух промежуточных переменных: OMGSQ и DENOM, чтобы облегчить отыскание ошибок, вероятно допущенных при программировании. В остальном программа весьма точно воспроизводит блок-схему (конечно, за исключением ошибок).

```
1    5 6 7   FORTRAN STATEMENT
     1       READ 2, AMP, T1, T2, OMGFST, OMGLST, OMGINC, L
     2       FORMAT (6F10.0, I1)
     3       PRINT 4, AMP, T1, T2
     4       FORMAT (1P3E14.5)
     5       OMEGA = OMGFST
     6       OMGSQ = OMEGA * OMEGA
     7       DENOM = (1. + T1**2 * OMGSQ) *(1. + T2**2 * OMGSQ)
     8       TREAL = -AMP*(T1 + T2)/DENOM
     9       TIMAG = -AMP*(1. - T1*T2*OMGSQ)/DENOM*OMEGA
     10      ABSVAL = SQRTF(TREAL**2 + TIMAG**2)
     11      PHASE = ATANF(TIMAG/TREAL)
     12      PRINT 13, OMEGA, OMGSQ, DENOM, TREAL, TIMAG,
            1      ABSVAL, PHASE
     13      FORMAT (1P7E12.4)
     14      IF (L) 17, 16, 17
     15      OMEGA = OMEGA + OMGINC
     16      GO TO 18
     17      OMEGA = OMEGA * OMGINC
     18      IF (OMEGA - OMGLST) 5, 5, 9
     19      STOP
             END
```

Р и с. 4.7. Программа для вычисления частотной характеристики сервомеханизма (практический пример 5).
Программа содержит разнообразные ошибки.

Существует очень много возможностей допустить ошибку в программе. Наиболее часто ошибки возникают при пробивке перфокарт. Если программист сам пробивает перфокарты, то он может допустить ошибку потому, что у него нет опыта работы на перфораторе: он может случайно нажать не ту клавишу. Если перфокарты пробивает лаборант, даже имеющий большой опыт работы с перфоратором, то всегда есть опасность ошибиться при чтении программы,

написанной от руки. Так или иначе, можно сберечь много времени, тщательно проверяя всю колоду перфокарт с программой.

Одним из наиболее употребительных способов проверки правильности пробивки перфокарт является *фиктивная пробивка* с помощью контрольника[1]). Этот аппарат имеет внешний вид и клавиатуру такие же, как и у перфоратора, но

```
    READ 2, AMP, T1, T2, OMGFST, OMGLST, OMGINC, L
    FORMAT (6F10.0, I1)
    PRINT 4, AMP, T1, T2
    FORMAT (1P3E14.5)
    OMEGA = OMGFST
    OMGSQ = OMEGA * OMEGA
    DENOM = (1. + T1**2 * OMGSQ)*(1. + T2**2 * OMGSQ)
    TREAL = -AMP$(T1 + T2)/DENOM
    TIMAG = -AMP*(1 - T1*T2*OMGSQ)/DENOM*OMEGA
    ABSVAL = SQRTF(TREAL**2 + TIMAG**2)
    PHASE = ATANF(TIMAG/TREAL)
    PRINT 13 OMEGA, OMGSQ, DENOM, TREAL, TIMAG,
   1      ABSVAL, PHASE
13  FORMAT (1P7E12.4)
UJ  IF (L) 17, 16, 17
15  OMEGA = OMEGA + OMGINC
16  GO TO 18
17  OMEGA = OMEGA * OMGINC
18  IF (OMEGA - OMGLST) 5, 5, 19
19  STOP
    END
```

Рис. 4.8. Контрольная печать отперфорированной программы рис. 4.7 (практический пример 5).

с его помощью можно только проверять уже пробитые перфокарты. Второй лаборант производит те же самые манипуляции, как и при пробивке перфокарт, но этот аппарат только проверяет соответствие уже имеющихся на карте отверстий тем, которые должны появиться при правильной пробивке. Если появляется несоответствие, то на панели загорается красный свет.

Второй способ проверки правильности пробивки — *контрольная печать* колоды перфокарт с помощью табулятора. На рис. 4.8 показана контрольная печать перфокарт, пробитых согласно программе на рис. 4.7. При тщательной про-

[1]) В оригинале «verifier».

верке можно обнаружить три ошибки, допущенные при перфорации. В операторе 3 вместо PRINT пробито PR1NT, что, по-видимому, объясняется ошибкой при чтении рукописной программы. В трансляторе ФОРТРАНа этот оператор был бы отвергнут, как неправильный. В операторе 8 пробит знак доллара там, где должна бы стоять звездочка; такая ошибка легко объяснима: эти два символа находятся на одной и той же клавише, но на разных регистрах. Этой же причиной объясняется и тот факт, что в операторе 14 вместо цифры 14 пробиты буквы UJ.

При тщательном изучении этой контрольной печати внимательный читатель наверняка обнаружит много различных ошибок. Это справедливо и для подавляющего большинства других программ. Контрольная фиктивная пробивка перфокарт или контрольная печать и ее внимательная проверка — это очень целесообразная затрата времени программиста, потому что время, потраченное на контрольные операции, сокращает длительный процесс отладки программы и экономит дорогостоящее машинное время. Но сейчас давайте вести себя так, как будто мы больше не в состоянии обнаружить никаких ошибок, и посмотрим, как эти ошибки будут обнаружены при преобразовании программы в трансляторе ФОРТРАНа.

Попытаемся преобразовать программу с помощью транслятора. Все трансляторы ФОРТРАНа предусматривают «диагностическую» проверку операторов, которые могут оказаться *синтаксически неправильными*, т. е. составленными с нарушением правил построения операторов, независимо от того, что они означают. При помощи транслятора нельзя определить, написали ли вы то, что хотели написать, но иногда можно несомненно установить, что некоторый оператор построен неправильно. Тщательность и подробность такого рода «диагностики» весьма различны для разных трансляторов ФОРТРАНа.

В нашем случае с помощью транслятора были с первой же пробы обнаружены две ошибки, как показано на рис. 4.9. В первой строке напечатан оператор, во второй строке поставлен комментарий — диагноз ошибки. Диагноз переводится на русский язык так: СМЕШАННОЕ ВЫРАЖЕНИЕ. «СМЕШАННОЕ» в данном диагнозе означает, что целые и действительные числа или переменные объедине-

ны в выражении с нарушением правил, изложенных в гл. 1. Совешенно ясна причина постановки такого диагноза: после константы 1 отсутствует десятичная точка.

В третьей строке напечатан второй неправильный оператор, а в четвертой строке напечатан диагноз. Диагноз, означающий в переводе на русский язык: БОЛЕЕ ШЕСТИ ЗНАКОВ В НЕКОТОРОМ СИМВОЛЕ, не поддается на этот раз такому непосредственному толкованию, как предыдущий. Иногда, установив, что оператор построен неправильно, весьма трудно определить точно, где именно заключена

```
    9       TIMAG = -AMP*(1 - T1*T2*OMGSQ)/DENOM*OMEGA
11252       MIXED EXPRESSION.

   12       PRINT 13 OMEGA, OMGSQ, DENOM, TREAL, TIMAG,
04336       MORE THAN SIX CHARACTERS IN SOME SYMBOL.
```

Р и с. 4.9. Диагноз ошибок, полученный с помощью программы-транслятора для программы рис. 4.7 (практический пример 5).

ошибка[1]). В данном случае ошибочно отсутствует запятая после 13, номера второго оператора FORMAT.

Работа транслятора, использованного для этой программы, распадается на несколько этапов. Если ошибка обнаруживается на первом этапе, то программа отвергается и никаких манипуляций с ней больше не производится. Исправив две ошибки, найденные на первом этапе, мы снова попробуем преобразовать программу. На этот раз программа отвергается с диагнозом: НЕКОТОРЫЙ УЧАСТОК ПРОГРАММЫ НЕ МОЖЕТ БЫТЬ ДОСТИГНУТ. ОН РАСПОЛОЖЕН НА ОПЕРАТОРЕ 15 ИЛИ ВБЛИЗИ ОТ НЕГО. Это означает, что какая-то часть программы не может быть выполнена, так как не существует такой передачи управления, которая приводила бы к этому участку программы. Ошибка заключена не в операторе 15, она может быть где угодно в программе. При внимательном изучении соседних

[1]) Транслятор STP-6, применяемый на ЭЦВМ UNIVAC-1107, поставил бы в этом случае следующий диагноз: PRINT1 НАЧИНАЕТ НАИМЕНОВАНИЕ, КОТОРОЕ СЛИШКОМ ДЛИННО. Место ошибки при данном диагнозе указывается точнее. — *Прим. перев.*

операторов мы обнаруживаем ошибку в операторе 14, который должен выглядеть так:

```
14 IF(L) 17, 15, 17
```

После исправления этих ошибок программа наконец преобразуется с помощью транслятора без дальнейших диагностических замечаний. В ней еще могут быть ошибки. (И они там есть!) Возможности проверки правильности исходной программы с помощью программы-транслятора по необходимости ограничены, так как нельзя чрезмерно увеличивать объем транслятора, а кроме того, существуют такие ошибки, которые с помощью транслятора просто невозможно обнаружить.

Рабочая программа была испытана на ЭЦВМ, причем с перфокарты была введена следующая исходная информация: AMP = 100, T1 = 0.25, T2 = 0.0625, OMGFST = 0.01, OMGLST = 100, OMGINC = 1.3 и L = 1. При выполнении рабочей программы была прочитана перфокарта с входной информацией, после чего началась печать, причем числа во всех строчках были одинаковыми, а печать продолжалась до тех пор, пока машину не остановили принудительно. Числа, напечатанные во всех строчках выходного бланка, были следующими:

```
OMEGA     0.01
OMGSQ     0.0001
DENOM     1.0000
TREAL   —31.250
TIMAG    —0.99999
ABSVAL   31.266
PHASE     0.031989
```

Сразу же возникает вопрос, почему одна и та же строка многократно повторяется? Прежде всего замечаем, что в этих строчках напечатано исходное значение OMEGA; здесь и находится решение проблемы. Нет ли в программе такого перехода, при котором переменной OMEGA каждый раз присваивалось бы исходное значение? Именно так и обстоит дело: оператор 18 передает управление оператору 5, если OMEGA еще не достигла значения OMGLST. Таким

образом, при каждом новом прохождении цикла переменной OMEGA присваивается ее исходное значение, равное OMGFST. Ясно, что оператор 18 необходимо написать в следующем виде:

```
18 IF (OMEGA—OMGLST) 6, 6, 19
```

В этом месте у начинающего программиста возникает почти непреодолимое желание исправить ошибку и снова испытать программу. Именно этого и не следует делать; всегда нужно стремиться исправить *все* ошибки, которые можно обнаружить при каждом очередном испытании программы. Изучим тщательнее выходную информацию и попробуем проверить, правильно ли работают остальные участки программы, прежде чем предпринимать новую попытку произвести вычисления на ЭЦВМ.

OMGSQ вычислено правильно; DENOM также не вызывает никаких сомнений, так как OMGSQ мало и знаменатель не может быть существенно больше 1. Поскольку DENOM равен 1, то легко проверить, что TREAL также вычислено правильно. Однако TIMAG выглядит подозрительным. AMP равно 100, при малом OMGSQ выражение в скобках должно быть приблизительно равно 1.0; DENOM равен 1.0, и поэтому при OMEGA = 0.01 величина TIMAG должна быть равна приблизительно 10 000. Вместо этого мы имеем величину, практически равную 1.0. Что случилось?

Поскольку OMGSQ и DENOM вычислены правильно, необходимо проверить выражение для TIMAG. Действительно, в выражении для TIMAG содержится ошибка: вокруг произведения DENOM*OMEGA не были написаны скобки, так что фактически OMEGA стояло в числителе. Этим полностью объясняются все неприятности с TIMAG, так что у программиста снова появляется желание испытать программу в машине.

Но давайте обратим внимание еще на ABSVAL и PHASE. ABSVAL немного больше, чем TREAL; это так и должно быть. Другое дело PHASE. Неужели 0.03° может быть правильным аргументом для комплексного числа с действительной частью —31 и мнимой частью —1? Едва ли так. Во-первых, угол находится не в нужном квадранте. Когда мнимая и действительная части комплексного числа отрицательны угол должен находиться между 180° и 270°.

В этой ошибке не виноваты ни ФОРТРАН, ни математика. Арктангенс является многозначной функцией: одному и тому же значению тангенса соответствует бесчисленное множество углов. В то же время стандартная программа для вычисления арктангенса, имеющаяся в трансляторе ФОРТРАНа, выбирает ту ветвь арктангенса, которая расположена между $-\pi/2$ и $+\pi/2$.

Другими словами, значение 0.03 вычислено в предположении, что угол находится в первом квадранте. Но и при этом предположении легко убедиться, что угол вычислен неправильно. В самом деле, даже грубая прикидка показывает что угол с тангенсом 1/31 никак не может быть равен 0.03°.

На этот раз ошибка заключается в том, что угол был вычислен в радианах, а мы ожидали, что он будет вычислен в градусах.

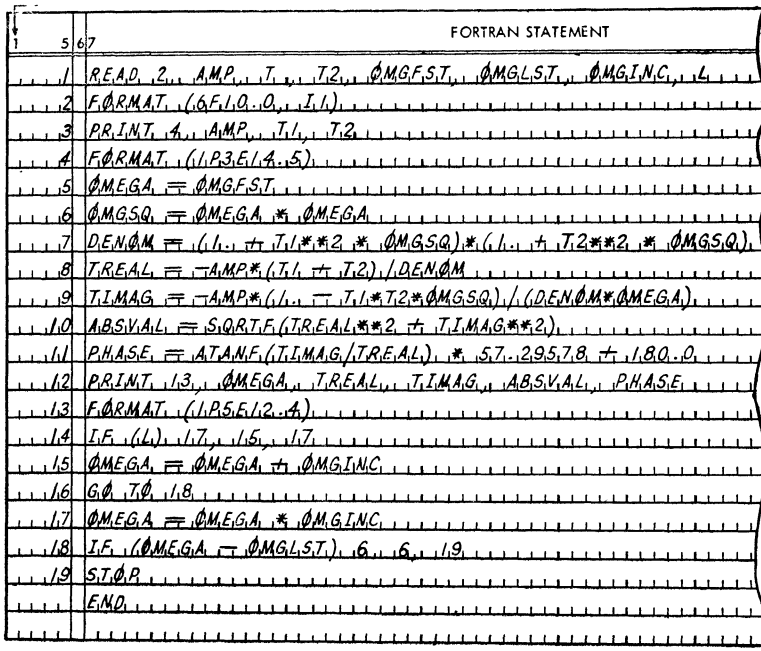
Итак, при вычислении PHASE возникают две проблемы: как получить угол в нужном квадранте и как перевести его из радианов в градусы? Перевод в градусы делается очень просто, для этого достаточно умножить получившийся угол на $180/\pi$, или на 57.29578. Перевод угла в нужный квадрант был бы довольно трудной задачей, если бы угол мог находиться в *любом* из четырех квадрантов. Однако если рассмотреть исходные формулы, то легко убедиться, что действительная часть всегда отрицательна, мнимая же часть может быть и положительной и отрицательной. Таким образом, комплексное число всегда будет находиться во втором или в третьем квадранте. Если оба числа отрицательны, то вычисленный арктангенс будет находиться в первом квадранте и его можно перевести в третий квадрант, прибавив к нему 180°. Если действительная часть отрицательна, а мнимая положительна, то вычисленный арктангенс будет находиться в четвертом квадранте и снова его можно перевести во второй квадрант, прибавив 180°.

Поэтому оператор 11 необходимо записать в следующем виде:

```
11  PHASE = ATANF (TIMAG/TREAL)* 57.29578 + 180.0
```

Вот теперь, по-видимому, мы получили объяснение для всех неполадок в программе. Внеся в программу необходимые исправления, снова производим вычисления на ЭЦВМ.

Окончательный исправленный вариант программы показан на рис. 4.10. Во втором операторе PRINT из списка для печати изъяты переменные OMGSQ и DENOM, так как мы выше убедились, что они вычисляются правильно.



Р и с. 4.10. Окончательно исправленный вариант программы для вычисления частотной характеристики сервомеханизма (практический пример 5).

На этот раз в результате работы программы была напечатана таблица, показанная на рис. 4.11. Внешне эта таблица выглядит правильно и заслуживает доверия. Но все ли уже сделано? Вовсе нет! Теперь известно, что в программе больше нельзя обнаружить ошибок с помощью транслятора ФОРТРАНа; мы знаем также, чем объяснились ошибки первой попытки счета, но не знаем, *правильны ли результаты*. Другими словами, мы, по-видимому, удали-

ли все ошибки из программы, но не *доказали*, что результаты действительно верны.

Следующим шагом должно быть вычисление правильных результатов для нескольких наборов исходных данных вруч-

1.00000E 02	2.50000E-01	6.25000E-02		
10.0000E-03	-3.1250E 01	-9.9999E 03	10.0000E 03	2.6982E 02
1.3000E-02	-3.1250E 01	-7.6922E 03	7.6923E 03	2.6977E 02
1.6900E-02	-3.1249E 01	-5.9170E 03	5.9171E 03	2.6970E 02
2.1970E-02	-3.1249E 01	-4.5515E 03	4.5516E 03	2.6961E 02
2.8561E-02	-3.1248E 01	-3.5010E 03	3.5012E 03	2.6949E 02
3.7129E-02	-3.1247E 01	-2.6930E 03	2.6932E 03	2.6934E 02
4.8268E-02	-3.1245E 01	-2.0714E 03	2.0716E 03	2.6914E 02
6.2749E-02	-3.1242E 01	-1.5931E 03	1.5935E 03	2.6888E 02
8.1573E-02	-3.1236E 01	-1.2252E 03	1.2256E 03	2.6854E 02
1.0604E-01	-3.1227E 01	-9.4213E 02	9.4264E 02	2.6810E 02
1.3786E-01	-3.1211E 01	-7.2425E 02	7.2492E 02	2.6753E 02
1.7922E-01	-3.1183E 01	-5.5652E 02	5.5739E 02	2.6679E 02
2.3298E-01	-3.1138E 01	-4.2732E 02	4.2845E 02	2.6583E 02
3.0288E-01	-3.1061E 01	-3.2770E 02	3.2917E 02	2.6459E 02
3.9374E-01	-3.0931E 01	-2.5078E 02	2.5268E 02	2.6297E 02
5.1186E-01	-3.0715E 01	-1.9124E 02	1.9369E 02	2.6088E 02
6.6542E-01	-3.0356E 01	-1.4497E 02	1.4812E 02	2.5817E 02
8.6504E-01	-2.9767E 01	-1.0883E 02	1.1282E 02	2.5470E 02
1.1246E 00	-2.8819E 01	-8.0385E 01	8.5395E 01	2.5028E 02
1.4619E 00	-2.7339E 01	-5.7845E 01	6.3980E 01	2.4470E 02
1.9005E 00	-2.5140E 01	-3.9941E 01	4.7194E 01	2.3781E 02
2.4706E 00	-2.2093E 01	-2.5886E 01	3.4033E 01	2.2952E 02
3.2118E 00	-1.8264E 01	-1.5264E 01	2.3802E 01	2.1989E 02
4.1754E 00	-1.4001E 01	-7.8075E 00	1.6031E 01	2.0915E 02
5.4280E 00	-9.8628E 00	-3.1377E 00	1.0350E 01	1.9765E 02
7.0564E 00	-6.3621E 00	-6.4046E-01	6.3943E 00	1.8575E 02
9.1733E 00	-3.7574E 00	4.1267E-01	3.7800E 00	1.7373E 02
1.1925E 01	-2.0317E 00	6.6624E-01	2.1381E 00	1.6184E 02
1.5503E 01	-1.0060E 00	5.7217E-01	1.1574E 00	1.5037E 02
2.0154E 01	-4.5787E-01	3.8869E-01	6.0061E-01	1.3967E 02
2.6200E 01	-1.9335E-01	2.2968E-01	3.0023E-01	1.3009E 02
3.4060E 01	-7.6857E-02	1.2367E-01	1.4560E-01	1.2186E 02
4.4278E 01	-2.9217E-02	6.2571E-02	6.9056E-02	1.1503E 02
5.7561E 01	-1.0771E-02	3.0402E-02	3.2254E-02	1.0951E 02
7.4830E 01	-3.8928E-03	1.4398E-02	1.4915E-02	1.0513E 02
9.7279E 01	-1.3894E-03	6.7120E-03	6.8543E-03	1.0169E 02

Р и с. 4.11. Выходная печать для программы рис. 4.10 (практический пример 5).

ную или на настольной счетной машине. При этом необходимо быть осторожным, чтобы не ввести каких-либо специальных условий. В качестве примера таких специальных условий предположим, что вычисления произведены для

OMEGA = 1.0. При этом не удалось бы обнаружить отсутствующие скобки в операторе 9. Или предположим, что две постоянные времени были бы сделаны равными; тогда нельзя было бы обнаружить некоторые типы ошибок при вычислении переменной DENOM.

Ручные вычисления нескольких значений показывают согласие с результатами вычислений на ЭЦВМ и только после этого мы имеем право считать, что программа работает правильно.

На практике большинство программ не преобразуются транслятором с первой попытки, а если и преобразуются, то результаты их работы обычно бывают неверными. Опытные программисты всегда предусматривают довольно значительные затраты времени на отладку программ.

Мы заканчиваем этот раздел некоторыми правилами, которые следует соблюдать при отладке программ.

1. Отладка обычно значительно упрощается, если выводить на печать промежуточные переменные. Наиболее распространенный прием состоит в том, что в программу вводятся дополнительные операторы PRINT, с помощью которых печатаются промежуточные переменные. Когда отладка программы кончается, то из отлаженной программы просто вынимаются перфокарты с дополнительными операторами PRINT (и соответствующими операторами FORMAT).

2. Тщательная проверка программы, написанной от руки или отпечатанной на табуляторе, сократит затраты времени на отладку программы и сэкономит дорогостоящее машинное время. Такую проверку необходимо делать до и после пробивки перфокарт.

3. Никогда не думайте, что программа составлена правильно только на том основании, что с помощью транслятора не удалось обнаружить никаких ошибок.

4. Исправляйте по возможности *все* ошибки, которые удается обнаружить при каждой попытке машинного счета. Сопротивляйтесь почти непреодолимому желанию пробовать считать на машине после нахождения каждой ошибки.

5. Окончательная проверка программы состоит в сравнении результатов работы программы с ручными вычислениями, если такое сравнение возможно. При подборе входных данных для отладки программы избегайте всякого рода специальных условий, если только они не требуются чтобы

проверить отдельные участки программы. Старайтесь задавать такие значения, которые приводили бы в действие все части программы.

4.4. ПРАКТИЧЕСКИЙ ПРИМЕР 6: ИНТЕГРАЛ ВЕРОЯТНОСТЕЙ

В статистике часто встречаются программы, где используются значения интеграла вероятностей

$$P(x) = \frac{1}{\sqrt{2\pi}} \int_{-x}^{x} e^{-t^2/2} dt.$$

Этот интеграл представляет собой площадь, заключенную между $-x$, x и общеизвестной кривой нормальной плотности распределения вероятности. График этой кривой приведен на рис. 4.12. Полная площадь, лежащая под этой кривой, от $-\infty$ до ∞ равна 1. В табл. 4.1 приведены значения интеграла для некоторых значений x:

Таблица 4.1

x	Интеграл
0.0	0.0000
0.1	0.0797
0.2	0.1585
0.3	0.2358
1.0	0.6827
2.0	0.9545
3.0	0.9973
4.0	0.9999

Мы здесь не будем обсуждать, как будет использовано вычисленное значение интеграла, а рассмотрим только, как построить программу для вычисления этого интеграла по известной величине x. Программа, которую мы должны написать, должна производить вычисление интеграла для любого x от 0.0 до 4.0. Значение интеграла должно быть вычислено с точностью до четвертого знака после запятой.

Программа будет использоваться очень широко и должна быть достаточно быстрой. Можно предположить, что сами значения x будут нормально распределены, другими словами, малые значения x более вероятны, чем большие. Глядя на табл. 4.1, можно прийти к выводу, что в двух третях от общего числа случаев x будет меньше 1, так как

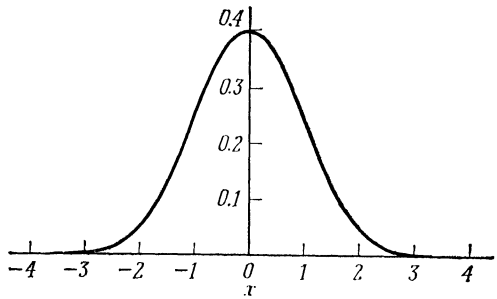


Рис. 4.12. Кривая нормального распределения вероятностей (практический пример 6).

$P(1) = 0.6827$. Поэтому целесообразно обратить внимание на скорость программы при малых x и не обращать большого внимания на скорость программы при больших x.

Величина $P(x)$ будет вычисляться из степенного ряда. (Как мы увидим в гл. 6, возможно и непосредственное численное интегрирование, но по сравнению со степенным рядом оно безнадежно медленно.) Выводом выражения для степенного ряда мы здесь заниматься не будем, а просто приведем этот ряд без доказательства:

$$P(x) = x\left(\frac{2}{\pi}\right)^{1/2}\left(1 - \frac{x^2}{2\cdot 1!\cdot 3} + \frac{x^4}{2^2\cdot 2!\cdot 5} - \frac{x^6}{2^3\cdot 3!\cdot 7} + \ldots\right).$$

Как и для всякого сходящегося знакопеременного ряда, ошибка ограничения не превосходит величины первого отброшенного члена ряда. Подумаем, каким образом определить количество членов ряда, которые необходимы для обеспечения требуемой точности. Четыре знака после запятой — это абсолютная, а не относительная точность; $P(0.1) = 0.0797$ является удовлетворительным результатом, хотя относительная точность и уменьшается для малых x. Для нас это требование имеет тот смысл, что значение

$P(x)$, вычисляемое с помощью программы, должно отличаться от истинного значения не более чем на 0.00005. Другими словами, произведение первого отброшенного члена ряда, умноженного на $x(2/\pi)^{1/2}$, должно быть не больше 0.00005 по абсолютной величине.

Попробуем произвести вычисления для нескольких значений x и определить, сколько при этом потребуется членов ряда. При этом достаточно вычислять члены ряда с относительной точностью порядка 0,1%.

$$P(x) = x\left(\frac{2}{\pi}\right)^{1/2}\left(1 - \frac{x^2}{6} + \frac{x^4}{40} - \frac{x^6}{336} + \frac{x^8}{3456} - \right.$$
$$\left. - \frac{x^{10}}{42\,240} + \frac{x^{12}}{599\,040} - \frac{x^{14}}{9\,676\,800} + \frac{x^{16}}{175\,472\,640} - \ldots\right)$$
$$P(0.1) = 0.1 \cdot 0.80 \cdot (1 - 0.0017 + 0.0000025 - \ldots)$$

Здесь явно можно ограничиться членом с x^2.

$$P(1) = 1.0 \cdot 0.80 \cdot (1 - 0.167 + 0.025 - 0.0030 +$$
$$+ 0.00029 - 0.000024 + \ldots)$$

На этот раз можно ограничиться членом с x^8. Таким образом, для всех x от 0.0 до 1.0 можно принять следующую формулу вычисления интеграла вероятности:

$$P(x) = x\left(\frac{2}{\pi}\right)^{1/2}\left(1 - \frac{x^2}{6} + \frac{x^4}{40} - \frac{x^6}{336} + \frac{x^8}{3456}\right).$$

Эта формула обеспечивает требуемую точность вычислений.

$$P(2) = 2 \cdot 0.80 \cdot (1 - 0.667 + 0.400 - 0.190 +$$
$$+ 0.0741 - 0.0243 + 0.00685 - 0.00169 + 0.00037 -$$
$$- 0.000078 + 0.000013)$$

Это означает, что для обеспечения требуемой точности нужно просуммировать члены ряда до x^{18} включительно.

$$P(3) \approx 3 \cdot 0.80 \cdot (1 - 1.5 + 2.02 - 2.17 + 1.90 - 1.40 +$$
$$+ 0.887 - 0.494 + 0.0246 - 0.0110 + 0.00446 -$$
$$- 0.00166 + 0.000574 - 0.0001835 + 0.0000549 - 0.0000154)$$

На этот раз потребовались члены ряда до x^{28} включительно. Не воспроизводя здесь выкладки, можно сказать,

что для вычисления $P(4)$ потребуются члены ряда вплоть до x^{50}.

Решение, которое нам сейчас необходимо принять, зависит от тех требований, которые предъявляются к скорости и точности вычислений, а также от ожидаемого распределения значений аргумента x. Было бы нецелесообразно использовать для всех x ряд со степенями вплоть до 50-й только потому, что он обеспечивает необходимую точность при $x = 4.0$. Даже если пойти по такому пути, то ошибки округления, по-видимому, снизят точность для больших значений x. Совершенно ясно, что для больших и для малых x придется вычислять интеграл различными способами, поэтому можно предвидеть, что диапазон значений аргумента придется разделить на три или даже на четыре интервала.

Рассмотрим сначала случай малых x. Рассматривая снова формулу для $P(1)$, убеждаемся, что пять членов, до x^8 включительно, обеспечивают требуемую точность. Так как между 0 и 1 заключено 68% ожидаемых значений x, то стоило бы попытаться ускорить работу этой части программы и обращать меньше внимания на скорость работы при больших x. По-видимому, целесообразно попробовать экономизировать этот ряд и уменьшить тем самым количество членов в нем на один или даже на два.

Из приложения 2, часть С, имеем

$$x^8 = \frac{1}{128}(256x^6 - 160x^4 + 32x^2 - 1 + T_8).$$

Пренебрегая всеми членами порядка выше x^8 и заменяя x^8 вышеприведенным выражением, имеем

$$P(x) = x\left(\frac{2}{\pi}\right)^{1/2}\left[\left(1 - \frac{1}{128\cdot 3456}\right) - \left(\frac{1}{6} - \frac{32}{128\cdot 3456}\right)x^2 + \right.$$
$$+ \left(\frac{1}{40} - \frac{160}{128\cdot 3456}\right)x^4 - \left(\frac{1}{336} - \frac{256}{128\cdot 3456}\right)x^6 +$$
$$\left. + \frac{1}{128\cdot 3456}T_8\right].$$

При $x < 1$ T_8 также меньше 1, поэтому ошибка, возникающая в результате экономизации, меньше $2.2\cdot 10^{-6}$ и пренебрежима по сравнению с требуемой точностью вычислений.

Можно ли продолжить процесс экономизации дальше? Если заменить x^6 выражением из приложения 2, то ошибка, возникающая в результате экономизации, составит $(1/32 \cdot 336)T_6$, или, другими словами, ошибка может достигать величины 0.000094. Эта ошибка в большинстве случаев искажала бы последнюю цифру результата, и поэтому в данном случае неприемлема.

Прежде чем рассматривать другие значения x, неплохо было бы написать участок программы для вычисления интеграла по полученной выше экономизированной формуле. Необходимо убедиться, что эта формула правильна; с другой стороны, может оказаться, что она пригодна для вычислений также и при x, несколько превосходящих 1. Это могло случиться потому, что мы оценивали только границы возможных ошибок, а сами ошибки могут оказаться существенно меньше.

Переписывая ряд в виде

$$P(x) = 0.79788455 \cdot x \cdot (0.99999774 - 0.16659433x^2 + \\ + 0.02463831 0x^4 - 0.0023974867x^6)$$

получаем очень простую формулу для программирования. После этого составляем программу, которая перебирает последовательность значений x, вычисляет интеграл и печатает значение x и значение интеграла.

Вычисления были произведены на двоичной ЭЦВМ, в которой действительные числа представляются приблизительно с восемью десятичными значащими цифрами. Значения интеграла вычислялись при x, изменявшемся от 0.0 до 2.0 с шагом 0.01.

Результаты работы программы сравнивались с четырехзначными математическими таблицами. Для $0 \leqslant x < 1.20$ результаты совпали с четырьмя исключениями. Например, для $x = 0.96$ табличное значение равно 0.6629, а по программе было вычислено 0.6630. Это, однако, не значит, что программа работает неправильно. В процессе вычислений по этой программе печатались также значения интеграла с шестью знаками после запятой. Значение, вычисленное для $x = 0.96$, было равно 0.662957, а точное значение, взятое из шестизначной математической таблицы, равно 0.662945. Совершенно ясно, что правильное значение и значение, вычисленное по нашей программе, будут округлены

по-разному, несмотря на то что значения эти различаются всего на 0.000012. Необходимо напомнить, что основным требованием к программе было то, чтобы вычисленное значение интеграла не отличалось от истинного значения больше чем на 0.00005, а это вовсе не то же самое, если бы мы потребовали совпадения всех вычисленных значений с табличными. Для совпадения с таблицей потребовалась бы гораздо более высокая точность вычислений. В действительности даже невозможно сказать заранее, какая точность вычислений потребовалась бы для полного совпадения с таблицей. Предположим, что для некоторого x истинное значение интеграла равно 0.34564999999; в этом случае, если значение интеграла, вычисленное с помощью программы, будет больше истинного значения всего на 10^{-11}, то при округлении до четырех знаков после запятой результаты округления получатся разными.

Все четыре случая, когда результат, полученный при помощи программы, не совпадал с табличным, объясняются подобным же образом; это относится к интервалу $0.0 \leqslant x \leqslant 1.2$. При x, больших 1.2, точность вычислений падает, что вполне естественно.

Таким образом, для малых x все обстоит хорошо; теперь пора заняться случаем больших x. Использование того же самого ряда для больших x не очень целесообразно. Для x, расположенных между 3 и 4, ряд сходится медленно, так что требуется вычислять много членов ряда и становится трудно обеспечить необходимую точность вычислений. При $x = 4$ один из членов ряда становится равен почти 25, и возникает вопрос, можно ли обеспечить при этом вычислении точность в четыре знака после запятой (см. практический пример 3, вычисление синусов больших углов).

При вычислении значений интеграла для больших x возможно использование *асимптотического ряда*, который в данном случае выглядит так:

$$P(x) \approx 1 - \left(\frac{2}{\pi}\right) \frac{e^{-x^2/2}}{x} \left(1 - \frac{1}{x^2} + \frac{1 \cdot 3}{x^4} - \right.$$
$$\left. - \frac{1 \cdot 3 \cdot 5}{x^6} + \frac{1 \cdot 3 \cdot 5 \cdot 7}{x^8} - \ldots\right).$$

Члены асимптотического ряда сначала уменьшаются, потом увеличиваются, так что асимптотический ряд не схо-

дится в том смысле, что ошибку ограничения можно сделать сколь угодно малой, просто взяв достаточное количество членов ряда. В случае асимптотического ряда ошибка ограничения меньше *последнего оставленного* члена ряда (в противоположность тому, что для обычного знакопеременного сходящегося ряда ошибка меньше первого отброшенного члена).

Вопрос, таким образом, заключается в том, будет ли наименьший член ряда достаточно мал для обеспечения требуемой точности вычисления интеграла? Снова произведем проверку для нескольких значений x:

$$P(2) \approx 1 - 0.80 \cdot \frac{e^{-2}}{2} (1 - 0.25 + 0.188 - \\ - 0.235 + 0.411 - \ldots)$$

Это нас явно не устраивает: наименьший член ряда, даже умноженный на общий множитель перед скобками, недостаточно мал для того, чтобы обеспечить требуемую точность.

$$P(3) \approx 1 - 0.80 \cdot \frac{e^{-4.5}}{3} (1 - 0.111 + 0.037 - 0.0205 + \\ + 0.016 - 0.016 + 0.0196 - \ldots)$$

Вот это уже может быть интересно. Общий множитель перед скобками равен приблизительно 0.0030, так что при умножении наименьшего члена ряда на этот множитель получается 0.000048. Такая точность уже приемлема, и, по-видимому, асимптотический ряд можно использовать для $x \geqslant 3$, если ограничить его членом $105/x^8$.

Этот вариант вычисления интеграла также полезно испытать для определения точности вычислений. Снова была написана короткая программа для ЭЦВМ, но на этот раз значения интеграла вычислялись для $2.90 \leqslant x \leqslant 4.00$. Шаг по x снова был равен 0.01.

Результаты оказались правильными для $2.90 \leqslant x \leqslant 4.00$. В шести случаях результат вычисления, округленный до четырех знаков, не совпал с табличным, но расхождение каждый раз составляло одну единицу младшего разряда, а при расчете с шестью знаками после запятой отличие не превосходило 0.00005, как и требовалось. Для $x < 2.90$ расхождение между вычисленным и таблич-

ным значениями стало систематическим, как и ожидалось: асимптотический ряд при малых значениях аргумента теряет точность.

Теперь у нас есть методика вычисления интеграла для малых и для больших x. Остается решить вопрос, что делать с промежуточными значениями. Можно было бы взять исходный степенной ряд, выписать в нем такое количество членов, при котором обеспечивалась бы точность вычисления интеграла для наибольшего требуемого x и попытаться применить экономизацию. Однако при таком подходе пришлось бы выписывать члены вплоть до x^{26}, и экономизация превратилась бы в довольно трудоемкую работу. Иногда такие затраты труда бывают оправданы, но не в данном случае.

Скорее всего, следует найти компромисс между скоростью работы программы и сложностью программирования. Один из способов сделать это — взять исходный степенной ряд и построить программу, где будут вычисляться и суммироваться последовательные члены ряда до тех пор, пока не появится член ряда, обеспечивающий требуемую точность вычислений. При таком подходе придется тщательно исследовать вопросы округления, но в общем такая программа вполне приемлема. Преимущество такой программы заключается еще и в том, что при меньших x эта программа работает быстрее. Это уменьшает общую затрату машинного времени, так как ожидается, что малые значения x будут встречаться чаще, чем большие. Для $x = 1.5$, например, достаточно ограничить ряд членом с x^{10}, что не так уж плохо.

Сейчас мы ведем речь о том, как найти наивыгоднейшее соотношение между затратами машинного времени, сложностью программы и затратами времени на программирование. Для установления таких соотношений не существует никаких строгих правил; приходится ограничиваться приблизительными оценками и даже догадками. Представляется вероятным, что непосредственное вычисление интеграла по ряду Тейлора при учете необходимого количества членов ряда не приведет к чрезмерным затратам машинного времени для $1.2 < x < 2.9$.

Остается проблема округления. Здесь имеются две основные причины потери точности. Во-первых, вычитание двух

больших почти равных чисел, разность между которыми по порядку величины сравнима с каким-нибудь меньшим членом ряда. Во-вторых, прибавление малого числа к большому, при котором теряются младшие значащие цифры меньшего. За сложением всегда следует вычитание (при вычислении знакопеременного ряда), а в полученной разности потерянные значащие цифры меньшего числа иногда могли бы изменить результат.

Было бы довольно трудно исследовать эти проблемы в полном объеме. Будет достаточно, если мы проверим, что получается в наихудшем случае, вблизи $x = 3$, учитывая, что если необходимая точность вычислений обеспечивается для наибольшего значения аргумента, то она будет вполне достаточной и для меньших значений. Рассматривая числа в выражении для $P(3)$, нетрудно убедиться, что ни та, ни другая проблема в данном случае не возникают. Вычитание двух наибольших чисел, 2.02... и 2.17..., не может привести к существенной потере точности, так как при записи исходных чисел с восемью значащими цирами результат вычитания имеет около семи достоверных значащих цифр, что более чем достаточно для обеспечения необходимой точности при вычислениях с четырьмя знаками после запятой. Вторая проблема не возникает из-за порядка арифметических операций: очень малые числа никогда не прибавляются к очень большим; они добавляются к частичным суммам, которые не возрастают существенно выше 1.

Итак, округление, по-видимому, не вызовет никаких затруднений в основном потому, что при вычислениях требуется точность всего в четыре знака, а ЭЦВМ работает с восьмизначными числами. (Если бы требовалась шести- или семизначная точность, то пришлось бы не только использовать большее количество членов, но и применить довольно сложные приемы вычислений, чтобы не допустить потери точности из-за описанных выше причин.)

Остается последний вопрос: какова должна быть величина допуска, необходимого для того, чтобы сравнивать с ним очередной вычисленный член ряда и определять момент окончания вычислений? Немедленно возникает идея взять в качестве такого допуска 0.00005, т. е. сам критерий точности. Но это было бы не совсем правильно, так как члены ряда, расположенные в скобках, умножаются на $x\,(2/\pi)^{1/2}$,

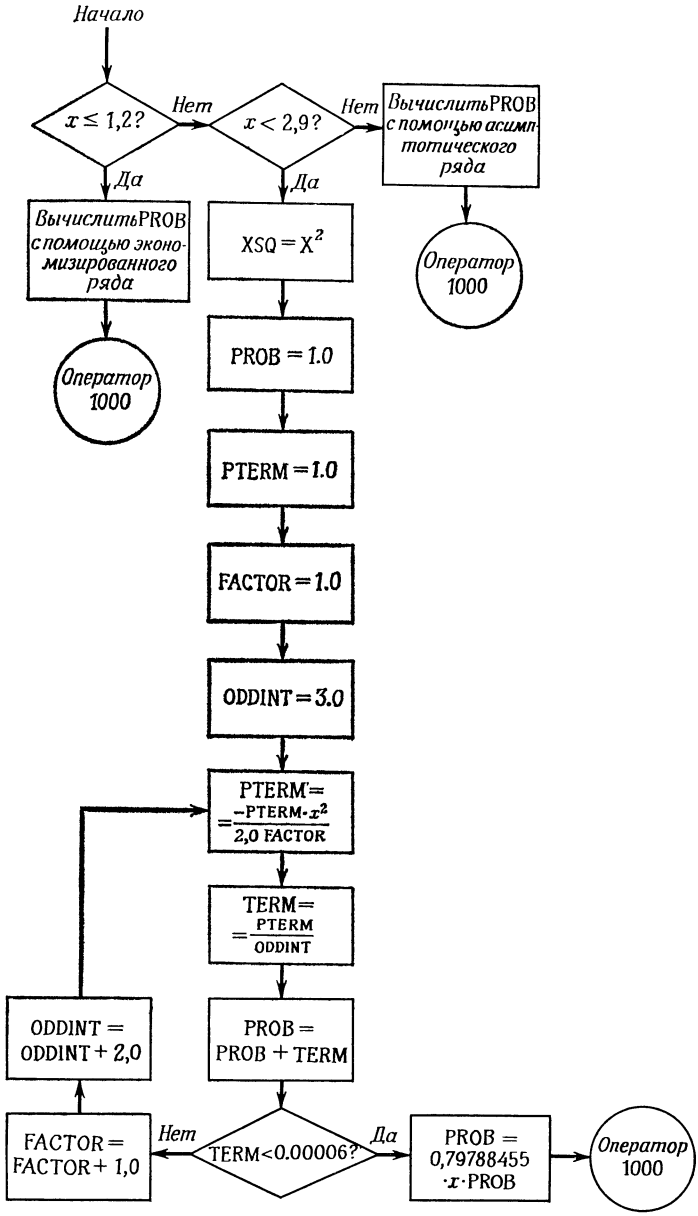
которое для наибольшего x составляет примерно 2.4. Поэтому логично предположить, что допуск должен быть равен примерно 0.00002. Но это также не совсем справедливо, потому что в действительности сравнение будет производиться с тем членом ряда, который уже вычислен и просуммирован, а ошибка ограничения связана с величиной *первого отброшенного* члена ряда. Допуск поэтому может быть несколько больше чем 0.00002, и при этом в программе будет вычислено на один член ряда меньше. (Конечно, не так уж важно, если и будет вычислен лишний член ряда, но лучше избежать этого, если соответствующий анализ достаточно прост.)

Чтобы не вычислять этого лишнего члена, необходимо знать отношение двух соседних членов ряда в наихудшем случае, т. е. когда это отношение минимально. Оказывается, что наихудший случай имеет место при больших x и что для $x = 3$ это отношение составляет приблизительно 3.3. Поэтому если последний вычисленный член ряда не больше $3.3 \cdot 0.00002$, то можно быть уверенным, что следующий член (первый отброшенный член ряда) будет лежать уже за пределами требуемой точности. Для надежности допуск был взят равным 0.00006 вместо точного значения 0.000066.

Вычисление очередных членов ряда может быть произведено таким же образом, как и в практическом примере 3, за исключением того что следующий член нельзя получить непосредственно из предыдущего, так как в знаменателе имеются последовательные нечетные числа. Для этих последовательных нечетных чисел приходится вводить отдельную переменную, а в остальном методика вычислений остается неизменной.

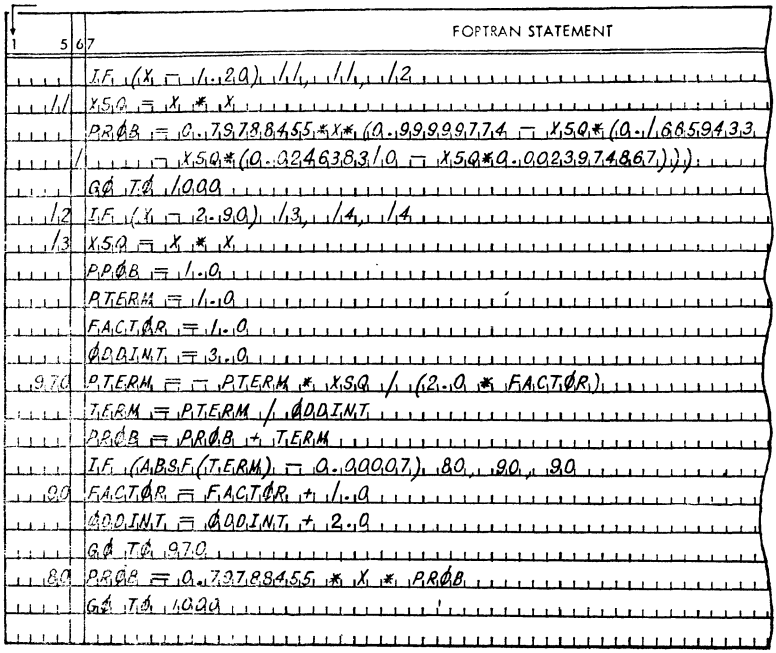
Этот участок программы также был испытан с помощью ЭЦВМ и дал правильные результаты, причем, как и прежде, некоторые значения интеграла, вычисленные с помощью этой программы, отличались от истинных значений на одну единицу младшего разряда. Тот же участок программы был испытан вторично, причем допуск на этот раз был равен не 0.00006, а 0.00007. Все результаты снова оказались правильными; в отдельных случаях количество вычисленных и просуммированных членов ряда было на один меньше[1]).

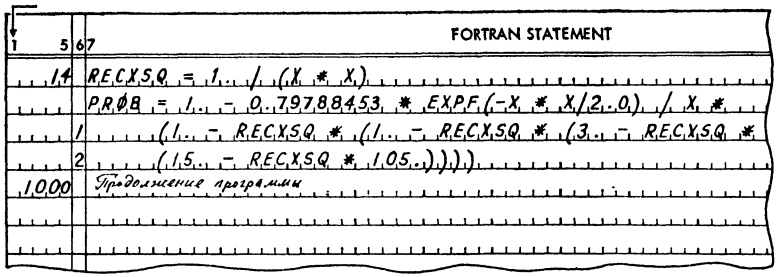
[1]) При испытании программы печаталось также количество вычисленных членов ряда.



Р и с. 4.13. Блок-схема программы для вычисления интеграла вероятностей.

Номер 1000 присвоен тому оператору, которому должно быть передано управление после вычисления интеграла (практический пример 6).





Р и с. 4.14. Программа для вычисления интеграла вероятностей (практический пример 6).

Возможность дальнейшего увеличения допуска не исследовалась; допуск и так уже близок к предельному, и небольшое его увеличение не привело бы к существенной экономии машинного времени.

Блок-схема всех трех частей программы, дополненных операторами условного перехода для выбора способа вычисления, соответствующего величине переменной x, приведена на рис. 4.13. Предполагается, что этот участок программы будет частью некоторой большой программы, так что операции ввода и вывода не предусмотрены; предполагается, что в предшествующей части программы было присвоено некоторое значение переменной X, а в последующей части программы, начиная с оператора 1000, будет использовано вычисленное значение PROB.

Сама программа приведена на рис. 4.14. В общепринятой практике такой участок программы обычно выделяют в подпрограмму, в так называемую FORTRAN FUNCTION, так что, написав в программе P(X), можно получить значение интеграла вероятности так же, как можно получить значение синуса, просто написав SINF(X). Способы написания подпрограмм такого рода и работа с ними описываются в гл. 9.

ГЛАВА

5

ЧИСЛЕННОЕ РЕШЕНИЕ УРАВНЕНИЙ

5.1. ВВЕДЕНИЕ

Нахождение корней уравнения — это одна из древнейших математических проблем, которая не потеряла своей остроты и в наши дни: она часто встречается в самых разнообразных областях науки и техники.

Рассмотрим общеизвестное квадратное уравнение

$$ax^2 + bx + c = 0.$$

Говорят, что значения

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

являются корнями этого уравнения, потому что для этих значений x уравнение удовлетворяется. В более общем случае, если имеется некоторая функция $F(x)$, то бывает необходимо найти такие значения аргумента x, для которых

$$F(x) = 0. \tag{5.1}$$

Функция $F(x)$ может быть алгебраической или трансцедентной; мы обычно будем предполагать, что она дифференцируема.

В общем случае функции, которые мы будем рассматривать, не имеют аналитических формул для своих корней в противоположность, например, квадратному уравнению. Поэтому приходится пользоваться приближенными методами нахождения корней, которые в основном состоят из двух этапов:

1. Отыскание приближенного значения корня.
2. Уточнение приближенного значения до некоторой заданной степени точности.

Мы не будем заниматься первым этапом и лишь кратко расскажем о нем в разд. 5.11. Очень часто приближенное

значение корня бывает известно из физических соображений, в других случаях можно использовать графические методы. Кроме того, существуют специальные методы нахождения приближенного корня для того практически важного случая, когда $F(x)$ является полиномом.

В этой главе будут в основном рассмотрены различные методы, относящиеся ко второму этапу — уточнению первоначального приближения. Численный метод, в котором производится последовательное, шаг за шагом, уточнение первоначального грубого приближения, называется *методом итераций*. Каждый шаг в таком методе называется *итерацией*. Если при последовательных итерациях получаются значения, которые все ближе и ближе приближаются к истинному значению корня, то говорят, что метод итераций *сходится*.

Сейчас мы рассмотрим несколько различных методов итераций для решения уравнений и исследуем, при каких условиях эти методы сходятся.

5.2. МЕТОД ПОСЛЕДОВАТЕЛЬНЫХ ПРИБЛИЖЕНИЙ

Предположим, что (5.1) переписано в виде

$$x = f(x). \tag{5.2}$$

Это преобразование можно сделать различными путями. Например, если

$$F(x) = x^2 - c = 0,$$

где $c \geqslant 0$, то можно прибавить к правой и к левой частям x

$$x = x^2 + x - c \tag{5.3}$$

или можно разделить все выражение на x и получить

$$x = \frac{c}{x}. \tag{5.4}$$

Наконец, можно преобразовать уравнение к следующему виду:

$$x = x - \left(\frac{x^2 - c}{2x}\right) = \frac{1}{2}\left(x + \frac{c}{x}\right). \tag{5.5}$$

Очевидно, что значения x, являющиеся корнями этого уравнения, равны $\pm\sqrt{c}$.

Пусть x_0 будет исходным приближенным значением корня уравнения (5.2). Тогда в качестве следующего приближения примем

$$x_1 = f(x_0).$$

В качестве следующего приближения возьмем

$$x_2 = f(x_1).$$

Продолжая этот процесс дальше, в качестве n-го приближения необходимо положить

$$x_n = f(x_{n-1}). \tag{5.6}$$

Основной вопрос, который необходимо выяснить при пользовании этим методом, — сходится ли x_n к решению уравнения (5.2) при возрастании n?

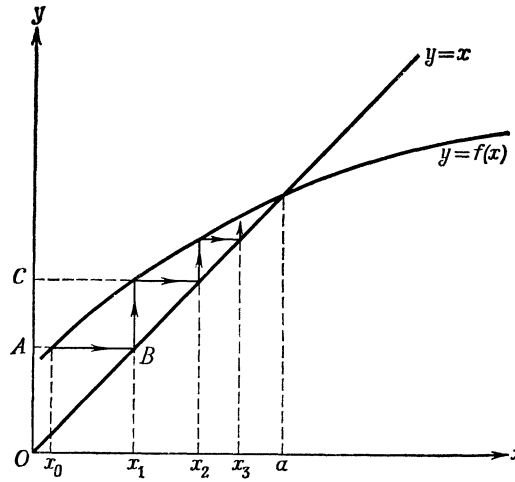
Мы выведем сейчас *достаточные* условия для сходимости метода; иными словами, выведем условия, которые являются гарантией того, что последовательные значения x_n будут приближаться к решению уравнения (5.2). Необходимо отметить, что эти условия *не являются необходимыми*, так как существуют функции, для которых эти условия не выполняются, но для которых тем не менее с помощью (5.6) можно найти их решения.

Рассмотрим сначала геометрическое представление процесса. При решении уравнения (5.2) отыскивается точка пересечения кривой $y = f(x)$ и прямой $y = x$. Рассмотрим рис. 5.1, на котором изображена некоторая кривая $y = f(x)$. Кривая эта может представлять собой какую угодно функцию, но для нас сейчас важно то обстоятельство, что производная этой кривой положительна и меньше 1, т. е. $0 < f'(x) < 1$. Пусть $x = a$ — значение x в точке пересечения; тогда a является корнем этого уравнения. Естественно, приступая к решению задачи, мы не знаем значения корня.

Зададимся некоторым x_0. Значение x_1 равно $f(x_0)$. Так как OA на рис. 5.1 равно $f(x_0)$, то найти x_1 можно следующим образом: проведем через точку A горизонтальную линию до пересечения с прямой $y = x$ в точке B, как показано на рисунке. Значение $x_2 = f(x_1)$ можно найти, проведя через точку B вертикальную линию до пересечения с кривой $y = f(x)$. При этом мы получаем отрезок OC, равный

$f(x_1)$, и проводя через точку C горизонтальную линию до пересечения с прямой $y = x$, получаем x_2. Процесс продолжается в том же порядке и дальше; на рисунке последовательность операций показана стрелками.

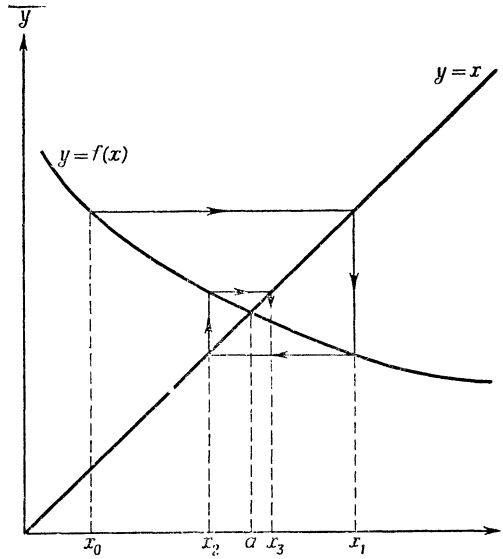
На рисунке видно, как последовательные значения x сходятся к $x = a$. Важно помнить, что для рассмотрения мы взяли кривую, производная которой положительна и меньше 1.



Р и с. 5.1. Геометрическое представление метода последовательных приближений для $0 < f'(x) < 1$.

Рассмотрим теперь другую кривую $y = f(x)$, производная которой отрицательна, но меньше 1 по абсолютной величине. Этот случай изображен на рис. 5.2. Последовательные операции вычисления решения этого уравнения снова изображены стрелками; приближения опять сходятся к решению $x = a$. В противоположность тому, что имело место для функции с положительной производной (см. рис. 5.1), на этот раз каждое последующее приближение находится с противоположной стороны от $x = a$. Для функции с положительной производной все последовательные приближения находились с одной стороны от истинного значения корня.

Наконец, рассмотрим случаи, когда производная функции больше 1 (рис. 5.3) и меньше —1 (рис. 5.4). В обоих случаях метод расходится. Каждое последующее значение x отстоит дальше от истинного значения корня, чем предшествующее. Поэтому кажется обоснованным предположение, что итерации по формуле (5.6) сходятся при условии,



Р и с. 5.2. Геометрическое представление метода последовательных приближений для $0 > f'(x) > -1$.

что производная $f'(x)$ меньше 1 по абсолютной величине.

Действительно, именно так и обстоит дело, и сейчас мы в этом убедимся с помощью элементарных выкладок. Заметим, что

$$a = f(a),$$

$$x_n = f(x_{n-1}),$$

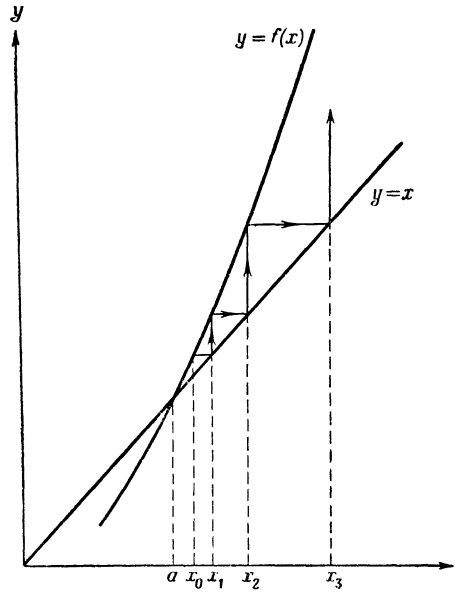
так что

$$x_n - a = f(x_{n-1}) - f(a).$$

Умножая правую часть на $(x_{n-1}-a)/(x_{n-1}-a)$ и используя теорему о среднем значении [1]), получаем

$$x_n - a = f'(\xi)(x_{n-1}-a),$$

где ξ лежит между x_{n-1} и a.



Р и с. 5.3. Геометрическое представление метода последовательных приближений для $f'(x) > 1$.

Теперь пусть m будет максимальным значением $f'(x)$ во всем рассматриваемом интервале, т. е. в интервале, включающем в себя x_0, x_1, x_2, ..., x_n, a. Тогда

$$|x_n - a| \leqslant m\,|x_{n-1} - a|.$$

---

[1]) Теорема о среднем значении утверждает, что если производная функции $y = f(x)$ непрерывна на некотором интервале a, b, то тангенс угла наклона хорды, проведенной между a и b, т. е.

$$\frac{f(b)-f(a)}{b-a},$$

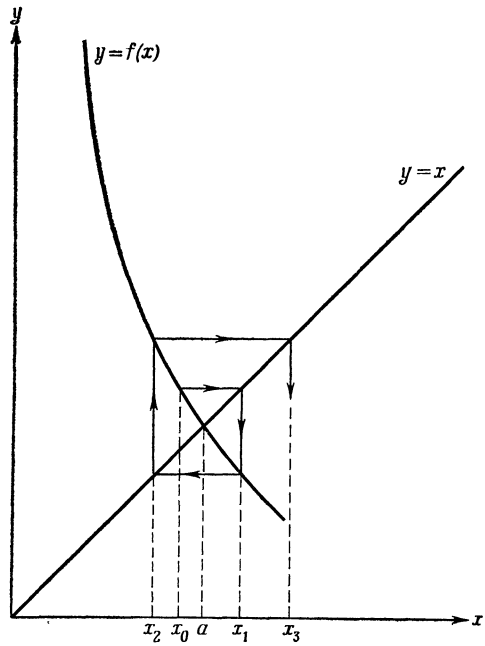
равен производной функции в некоторой промежуточной точке, лежащей между a и b.

Таким же образом получаем

$$|x_{n-1}-a| \leqslant m|x_{n-2}-a|$$

и поэтому

$$|x_n-a| \leqslant m^2|x_{n-2}-a|.$$



Р и с. 5.4. Геометрическое представление метода последовательных приближений для $f'(x) < -1$.

Продолжая те же выкладки, получаем

$$|x_n-a| \leqslant m^n|x_0-a|. \qquad (5.7)$$

Очевидно, что, если во всем интервале $m < 1$, то независимо от выбора начального значения x_0 с возрастанием n правая часть неравенства становится малой и x_n сходится к a.

С другой стороны, для случая $|f'(x)| > 1$ величина $|x_n - a|$ неограниченно возрастает с ростом n. Доказательство этого мы предоставляем читателям.

Таким образом, если $|f'(x)| < 1$, *то процесс сходится, если же* $|f'(x)| > 1$, *то процесс расходится.* Обратите внимание, что неравенства должны выполняться при *всех* значениях x_n, вычисляемых в ходе решения задачи.

Что произойдет в случае, когда производная $f'(x)$ в некоторых точках x_i меньше, а в других точках x_j больше 1 по абсолютной величине? Точного ответа на этот вопрос не существует, процесс иногда сходится, иногда расходится.

Вернемся к примеру, рассмотренному в начале раздела где корнями уравнения являются $\pm\sqrt{c}$. В формуле (5.3)

$$f(x) = x^2 + x - c,$$

так что $|f'(x)| < 1$, если $-1 < x < 0$. В этом случае если число c меньше 1, то процесс сходится к отрицательному значению корня.

С другой стороны, если воспользоваться формулой (5.4), то

$$f'(x) = -\frac{c}{x^2}$$

и при x, близких к $\sqrt{c}$ (как и должно быть при отыскании корня уравнения), $|f'(x)|$ становится почти равным 1; можно проверить, что процесс расходится.

Наконец, применяя формулу (5.5), получаем выражение для $f'(x)$ в виде

$$\frac{1}{2}\left(1 - \frac{c}{x^2}\right).$$

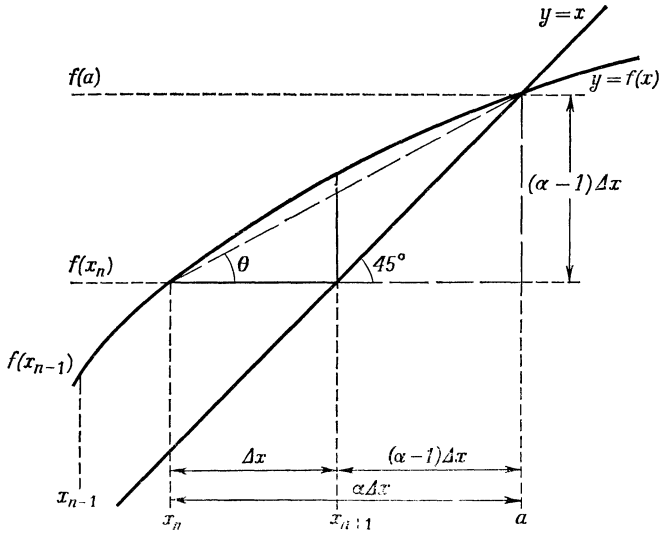
В этом случае когда $x \approx \sqrt{c}$, то $f'(x) \approx 0$ и процесс очень быстро сходится.

Вообще говоря, хотя для всякого уравнения можно найти большое количество соответствующих ему функций $f(x)$ в выражении (5.2), нужно с большой осторожностью подходить к их конкретному выбору, так как от него зависит сходимость метода итераций.

5.3. УСОВЕРШЕНСТВОВАННЫЙ МЕТОД ПОСЛЕДОВАТЕЛЬНЫХ ПРИБЛИЖЕНИЙ

Рассмотрим снова рис. 5.1. Нетрудно заметить, что, хотя каждое последующее значение x_n находится ближе к решению уравнения, чем предшествующее, все они сильно отли-

чаются от a. По-видимому, можно было бы добиться более быстрой сходимости метода, если при каждой очередной



Р и с. 5.5. Геометрическое представление усовершенствованного метода последовательных приближений для $0 < f'(x) < 1$.

итерации делать бо́льшую поправку к очередному значению x_n. Иначе говоря, вместо того, чтобы полагать

$$x_{n+1} + x_n = \Delta x,$$

где

$$\Delta x = f(x_n) - x_n,$$

можно принять следующую формулу для x_{n+1}:

$$x_{n+1} = x_n + \alpha \Delta x_n,$$

где $\alpha > 1$.

Эта идея поясняется на рис. 5.5, где в увеличенном виде изображена небольшая часть рис. 5.1. Наилучшим выбором α следует признать тот, что изображен на рисунке, так как тогда x_{n+1} получается равным a. Попытаемся определить это наилучшее значение α.

Заметим, что расстояние между x_{n+1} и a равно $(\alpha - 1)\Delta x$, и так как $y = x$ есть прямая линия, идущая под углом 45°

к осям координат, расстояние между $f(a)$ и $f(x_n)$ также равно $(\alpha - 1)\Delta x$. Поэтому тангенс угла θ равен

$$\operatorname{tg}\theta = \frac{(\alpha-1)\,\Delta x}{\alpha \Delta x} = \frac{\alpha-1}{\alpha}. \tag{5.8}$$

С другой стороны,

$$\operatorname{tg}\theta = \frac{f(a)-f(x_n)}{a-x_n},$$

и, используя теорему о среднем значении,

$$\operatorname{tg}\theta = f'(\xi), \tag{5.9}$$

где $x_n \leqslant \xi \leqslant a$.

Из (5.8) и (5.9) получаем значение α в виде

$$\alpha = \frac{1}{1-f'(\xi)}. \tag{5.10}$$

Значение ξ, конечно, остается неизвестным, но для значения $f'(\xi)$ можно принять следующее приближение:

$$f'(\xi) \approx \frac{f(x_n)-f(x_{n-1})}{x_n - x_{n-1}} = \frac{f(x_n)-x_n}{x_n - x_{n-1}}. \tag{5.11}$$

Геометрически процесс отыскания следующего приближения, x_{n+1}, сводится к тому, что проводится хорда между точками $(x_n, f(x_n))$ и $(x_{n-1}, f(x_{n-1}))$ и определяется точка ее пересечения с прямой $y = x$.

Формула итерационного метода приобретает при этом следующий вид:

$$x_{n+1} = x_n + \alpha(f(x_n) - x_n), \tag{5.12}$$

где α определяется по формулам (5.10) и (5.11).

Возникает вопрос, как это усовершенствование влияет на сходимость метода. Из формулы (5.10) видно, что при $0 < f'(x) < 1$ должно получиться $1 < \alpha < \infty$. Этот случай изображен на рис. 5.1, где последовательные поправки были слишком малы; так как $\alpha > 1$, усовершенствованный метод увеличит эти поправки и ускорит сходимость вычислений.

При $-1 < f'(x) < 0$ имеем $1/2 < \alpha < 1$. Этот случай изображен на рис. 5.2, где каждая поправка также была слишком велика. В усовершенствованном методе все поправки уменьшаются на коэффициент, расположенный между 1/2 и 1; сходимость метода при этом, естественно, также улучшается.

Более важны случаи, когда простой метод последовательных приближений расходится. Если $f'(x) < 1$,

то $\alpha < 0$. Как показано на рис. 5.3, каждая очередная поправка имеет неправильный знак и соответствующее приближение отстоит от a дальше, чем предыдущее. Так как α для этого случая отрицательно, то в усовершенствованном методе знаки поправок изменяются нужным образом.

Наконец, при $f'(x) < -1$ имеем $0 < \alpha < 1/2$. В этом случае, как показано на рис. 5.4, поправки были слишком велики; при усовершенствованном методе каждая поправка умножается на коэффициент, расположенный между 0 и 1/2. Естественно, что уменьшение поправок должно быть в этом случае более резким, чем на рис. 5.2, где приближения сходятся, в то время как на рис. 5.4 они расходятся.

Описанная модификация метода принадлежит Вегстейну[1]). Процессы экстраполяции и интерполяции характерны для итерационных методов. Мы опять встретимся с ними в гл. 8 в связи с итерационным решением системы линейных алгебраических уравнений.

Небольшая дальнейшая модификация метода последовательных приближений приводит к одному из наиболее известных численных методов — к методу Ньютона — Рафсона для нахождения корней уравнения. Однако в некоторых случаях методы, описанные выше, предпочтительнее метода Ньютона — Рафсона. К этому вопросу мы вернемся в разд. 5.6 после того, как рассмотрим метод Ньютона — Рафсона.

5.4. МЕТОД НЬЮТОНА — РАФСОНА

Вспомним, что в формуле (5.11) мы заменяли производную разностью. Вспомним также, что оптимальное значение ξ лежало в интервале $x_n \leqslant \xi \leqslant a$.

Предположим, что для простоты вычислений мы выбрали $\xi = x_n$. Тогда (5.10) приобретает следующий вид:

$$\alpha = \frac{1}{1 - f'(x_n)}, \tag{5.13}$$

x_{n+1} находим из формулы

$$x_{n+1} = \frac{f(x_n) - x_n f'(x_n)}{1 - f'(x_n)}. \tag{5.14}$$

[1]) См. *Comm. ACM*, I, 9—13 (1958).

Нетрудно видеть, что (5.14) эквивалентно простому методу последовательных приближений

$$x_{n+1} = g(x_n),$$

где

$$g(x) = \frac{f(x) - xf'(x)}{1 - f'(x)}.$$

Вспомним также, что если $|g'(x)| < 1$, то метод сходится. Для $g'(x)$ имеем

$$g'(x) = \frac{f''(x)\,[f(x) - x]}{[1 - f'(x)]^2}.$$

Но поскольку $f(x)$ подчиняется соотношению (5.2), то для x, достаточно близких к a, выражение в скобках становится малым. Поэтому итерационный метод, описываемый формулой (5.14), сходится, если

1. x_0 выбрано достаточно близко к решению $x = f(x)$.
2. Производная $f''(x)$ не становится слишком большой.
3. Производная $f'(x)$ не слишком близка к 1.

Это и есть знаменитый метод Ньютона — Рафсона. Обычно его записывают в более привычной форме

$$x_{n+1} = x_n - \frac{F(x_n)}{F'(x_n)}, \tag{5.15}$$

где

$$F(x) = f(x) - x = 0.$$

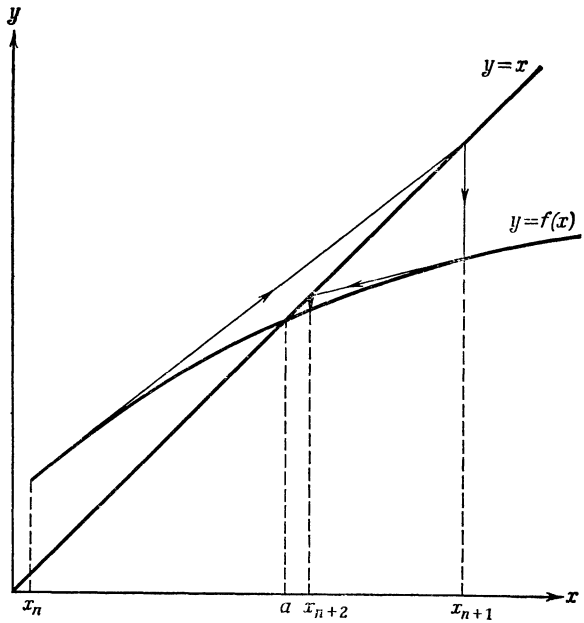
Таким образом, мы возвращаемся к уравнению в форме (5.1), и условия сходимости принимают следующий вид:

1. x_0 выбрано достаточно близко к корню уравнения $F(x) = 0$.
2. Производная $F''(x)$ не становится очень большой.
3. Производная $F'(x)$ не слишком близка к нулю.

Последнее условие означает, что никакие два корня не находятся слишком близко один к другому. В следующем разделе мы вернемся к этому вопросу.

Рассмотрим геометрическое толкование метода Ньютона — Рафсона. В формуле (5.13) мы выбрали точку ξ совпадающей с x_n. На рис. 5.5 это соответствует тому, что угол θ равен углу наклона касательной к $y = f(x)$ в точке $x = x_n$. Нахождение следующего приближения сводится при этом к тому, что проводится касательная к кривой $y = f(x)$ в точке $x = x_n$ и отыскивается точка ее пересечения с пря-

мой $y = x$. Эта точка и будет новым приближением x_{n+1}. Чтобы найти $f(x_{n+1})$, через значение x_{n+1} проводится вертикальная линия. После этого проводится новая касательная, точка пересечения которой с прямой $y = x$ даст значение



Р и с. 5.6. Геометрическое представление метода Ньютона — Рафсона для $f(x) = x$.

x_{n+2}. Последовательность операций показана на рис. 5.6, где изображен случай $0 < f'(x) < 1$.

Заметим, что теперь сходимость гораздо лучше, чем для простого метода последовательных приближений, изображенного на рис. 5.1. Такая быстрая сходимость типична для метода Ньютона — Рафсона, так как величина $g'(x)$ очень мала.

Если уравнение задано в форме (5.1) и используется итерационная формула (5.15), то геометрически можно представить себе ход вычислений по рис. 5.7. В этом случае отыскивается точка пересечения кривой $y = F(x)$ с осью x.

Исходя из некоторого начального приближения x_n, находим соответствующее ему значение $F(x_n)$, проводим касательную к кривой $y = F(x)$ и ищем точку пересечения этой касательной с осью x. Легко видеть, что эта точка и будет значением x_{n+1} из формулы (5.15), так как там и требуется

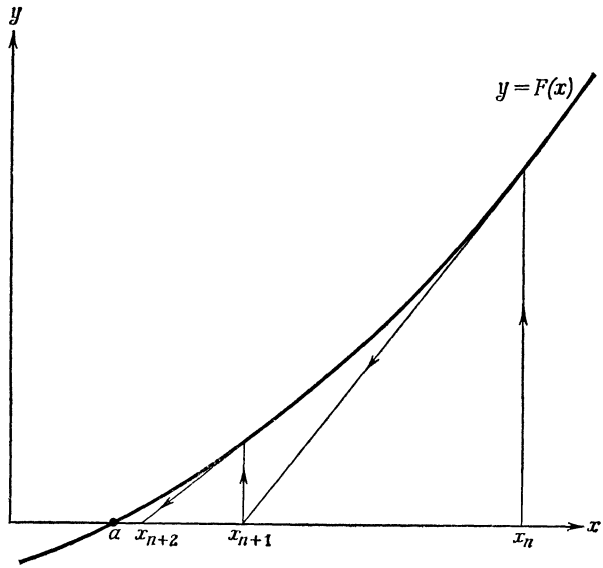


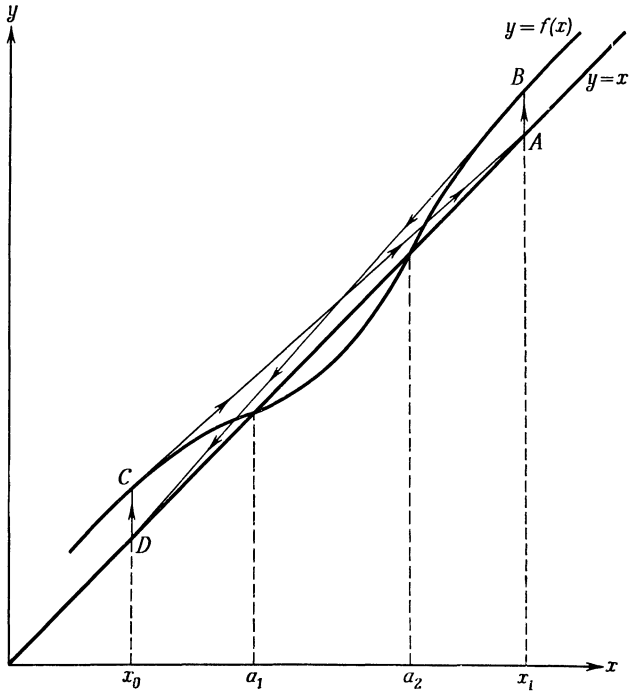
Рис. 5.7. Геометрическое представление метода Ньютона — Рафсона для $F(x) = 0$.

провести через точку с координатами x_n, $F(x_n)$ прямую с угловым коэффициентом $F'(x_n)$ и затем найти ее пересечение с осью x.

5.5. СЛУЧАЙ ПОЧТИ РАВНЫХ КОРНЕЙ

Мы уже указывали, что при решении уравнений методом Ньютона — Рафсона могут возникнуть трудности в том случае, если уравнения (5.1) или (5.2) имеют пары близко расположенных корней. В этом случае условие 3 сходимости метода нарушается вблизи такой пары корней. Соответствующая ситуация проиллюстрирована на рис. 5.8 (мас-

штаб сильно увеличен). Заметим, что производная $f'(x)$ близка к 1 при x, равном обоим значениям корней, a_1 и a_2 Более того, на основании теоремы о среднем значении. можно утверждать, что $f'(x)$ равна 1 где-то между a_1 и a_2.



Р и с. 5.8. Геометрическое представление того случая, когда с помощью метода Ньютона — Рафсона нельзя определить значение корня ($f'(x)$ близка к 1).

Рассмотрим, что случится, если принять x_0 в качестве исходного значения для корня a_1. Касательная, проведенная через точку C, пересечет прямую $y = x$ в точке A, и следующее приближение будет равно x_1. Касательная, проведенная через точку B, пересекает прямую $y = x$ в точке D, и в качестве следующего приближения снова получается x_0. Итерационный процесс, таким образом, осциллирует между x_0 и x_1 до бесконечности, не сходясь ни к одно-

му значению корня. Другими словами, не удается *отделить* эти два корня, потому что они расположены слишком близко один к другому. Конечно, мы вправе сказать в этом случае, что нарушено условие 1 сходимости метода и что начальное значение x_0 было недостаточно близко к a_1.

Это утверждение совершенно верно. Поэтому мы попытаемся разработать метод, с помощью которого можно было бы найти начальное приближение, достаточно близкое к искомому значению корня. Трудности возникают потому, что вычисление знаменателя в формуле (5.14) включает в себя вычитание двух почти равных чисел, а мы уже неоднократно убеждались, что такое вычитание приводит к снижению точности.

Натаниель Мейкон[1]) предложил метод, согласно которому сначала находится значение x, при котором $f'(x) = 1$, т. е. решается уравнение

$$x = x + f'(x) - 1.$$

Пусть решением этого уравнения будет некоторое $x = \bar{x}$. Эта точка расположена между двумя корнями, a_1 и a_2. Чтобы получить начальное приближение для решения уравнения, предположим, что $\bar{x}$ лежит посредине между a_1 и a_2. (Этот случай изображен на рис. 5.9.) Другими словами, мы предполагаем, что $\bar{x}+d$ и $\bar{x}-d$ являются корнями уравнения (5.2). Разлагая $f(x)$ в ряд Тейлора в окрестности точки $\bar{x}$ и замечая, что $f'(\bar{x}) = 1$, получаем

$$f(x) = f(\bar{x}) + (x-\bar{x}) + \frac{1}{2} f''(\bar{x})(x-\bar{x})^2 + \ldots .$$

Как показано, мы ограничиваем ряд тремя членами. Подставляя $\bar{x}+d$ вместо x, имеем

$$f(\bar{x}+d) = f(\bar{x}) + d + \frac{1}{2} f''(\bar{x}) d^2.$$

Но по условию

$$f(\bar{x}+d) = \bar{x} + d,$$

поэтому, решая эти уравнения относительно d, получаем

$$d = \sqrt{\frac{2(\bar{x} - f(\bar{x}))}{f''(\bar{x})}}.$$

[1]) Macon N., Numerical analysis, Wiley, New York, 1963, p. 34—36.

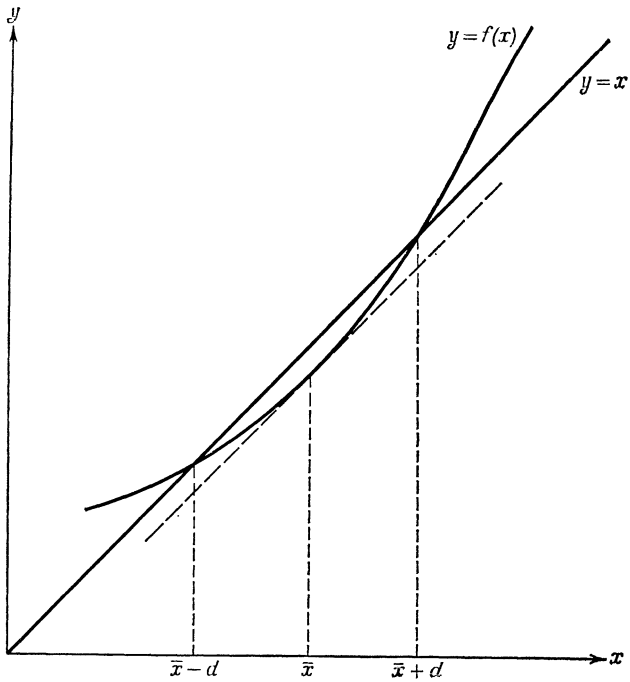
Не воспроизводя выкладки, скажем также, что если уравнение задано в форме (5.1)

$$F(x)=0,$$

то для d получаем

$$d=\sqrt{\frac{-2F(\bar{x})}{F''(\bar{x})}},$$

так как приходится решать уравнение $F'(x)=0$. Рассматривая рис. 5.8, читатель может легко убедиться в том, что величина под корнем положительна.



Р и с. 5.9. Геометрическое представление усовершенствованного метода Ньютона — Рафсона для $f'(x)$, близкой к 1.

Процесс решения уравнения сводится теперь к следующему. Если дано уравнение с двумя почти равными корня-

ми, то, определив приблизительно местонахождение этих корней, необходимо решить уравнение

$$x = x + f'(x) - 1$$

и определить значение $\bar{x}$. Для решения этого уравнения можно использовать любой подходящий метод, например метод Ньютона — Рафсона. Найдя значение $\bar{x}$, можно определить значение d. Наконец,. значения $\bar{x} - d$ и $\bar{x} + d$ используются в качестве начальных приближений для определения соответственно a_1 и a_2.

Конечно, и в этом случае можно не суметь отделить корни, если $f''(x)$ близко к нулю. Это означает, что уравнение $f'(x) = 1$ имеет более, чем один корень вблизи $\bar{x}$. В этом случае сначала необходимо найти решение уравнения $f''(x) = 0$. Интересующийся читатель может найти необходимые сведения в книге Н. Мейкона.

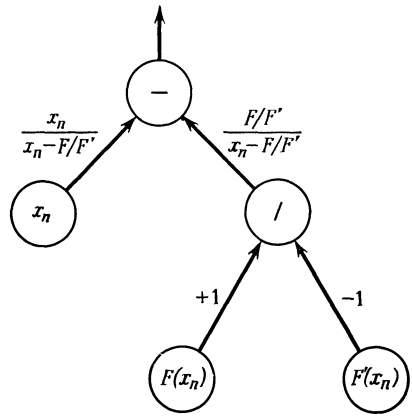
5.6. СРАВНЕНИЕ МЕТОДОВ И ИХ ОШИБОК ОКРУГЛЕНИЯ

Так как метод Ньютона — Рафсона сходится гораздо быстрее, чем метод последовательных приближений, возникает вопрос, почему все же используется и тот и другой. Дело в том, что при использовании метода Ньютона — Рафсона при каждой очередной итерации требуется вычислять не только функцию, но и ее производную. Эти вычисления могут оказаться трудными, длительными или даже вообще невозможными. Например, функция $f(x)$ может быть задана не формулой, а таблицей значений. Производная может даже не существовать в некоторых точках. В таких случаях часто применяется метод последовательных приближений или различные его модификации.

Другими словами, выбор метода зависит от конкретного вида функции $f(x)$ или $F(x)$.

Интересно и важно отметить тот факт, что ошибка округления *не накапливается* при использовании итерационных методов решения уравнений. Общая ошибка округления равна ошибке, возникшей в последней итерации, и не зависит от арифметических операций, выполнявшихся в предыдущих итерациях. Это общее свойство всех итерационных методов; оно является одним из важнейших преимуществ

итерационных методов перед всеми другими. Причина того, что ошибки округления не накапливаются, ясна — каждое новое приближение, включая и предпоследнее, можно рассматривать как исходное приближение. Ошибка округления при вычислении последнего приближения зависит, таким



Р и с. 5.10. Граф вычислительного процесса для метода Ньютона — Рафсона.

образом, только от арифметических операций, с помощью которых это последнее приближение получается из предпоследнего.

Граф вычислительного процесса для метода Ньютона — Рафсона изображен на рис. 5.10. Исходная ошибка в x_n отсутствует: можно считать, что x_n представляется в виде бесконечной десятичной дроби, которая содержит одни нули, начиная с некоторой цифры. При вычислении $F(x_n)$ и $F'(x_n)$ появляются относительные ошибки округления; назовем их соответственно r и r'. Обозначим относительную ошибку округления при делении и при вычитании соответственно через d и s. Тогда абсолютная ошибка округления при вычислении x_{n+1} будет равна

$$e = \frac{F(x_n)}{F'(x_n)}(r - r' + d) + s.$$

В большинстве случаев общая ошибка определяется неточностью r и r' при вычислении $F(x_n)$ и $F'(x_n)$.

5.7. КОРНИ МНОГОЧЛЕНОВ

Рассмотрим практически важный случай, когда $F(x)$ представляет собой многочлен степени m

$$F(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_m x^m. \tag{5.16}$$

Применим метод Ньютона—Рафсона согласно формуле (5.15).

Вычисление $F(x_n)$ по правилу Горнера было изложено в разд. 3.4. Имея рекуррентные формулы

$$\left.\begin{aligned} b_m &= a_m, \\ b_j &= a_j + x_n b_{j+1}, \quad j = m-1, \ldots, 0, \end{aligned}\right\} \tag{5.17}$$

находим

$$F(x_n) = b_0.$$

Но, согласно формуле (3.11) из разд. 3.4,

$$F(x) = (x - x_n) G(x) + b_0,$$

где

$$G(x) = b_1 + b_2 x + \ldots + b_m x^{m-1}.$$

Поэтому

$$F'(x) = (x - x_n) G'(x) + G(x).$$

Следовательно,

$$F'(x_n) = G(x_n).$$

Но ведь $G(x)$ является многочленом степени $m-1$, так что, используя правило Горнера, можно вычислить $G(x_n)$ и таким образом найти $F'(x_n)$. Воспользовавшись теми же рекуррентными формулами, имеем

$$\left.\begin{aligned} c_m &= b_m, \\ c_j &= b_j + x_n c_{j+1}, \quad j = m-1, \ldots, 1, \end{aligned}\right\} \tag{5.18}$$

и соответственно

$$F'(x_n) = G(x_n) = c_1.$$

Подставляя найденные значения $F(x_n)$ и $F'(x_n)$ в формулу (5.15) для метода Ньютона—Рафсона, получаем

$$x_{n+1} = x_n - \left(\frac{b_0}{c_1}\right).$$

b_0 и c_1 вычислены здесь по формулам (5.17) и (5.18). Этот метод нахождения корней полиномов часто называют методом Бирге—Виета.

Пример

Задан многочлен

$$F(x) = x^3 - x - 1.$$

Требуется найти корень, расположенный вблизи $x_0 = 1.3$. Последовательность вычислений приводится в табл. 5.1.

Таблица 5.1

i	a_i	b_i	c_i
3	1	1	1
2	0	1.3	2.6
1	—1	0.69	4.07
0	—1	—0.103	

$$x_1 = x_0 - \frac{b_0}{c_1} = 1.3 - \frac{-0.103}{4.07} = 1.325$$

i	a_i	b_i	c_i
3	1	1	1
2	0	1.325	2.65
1	—1	0.755625	4.267
0	—1	0.001203	

$$x_2 = x_1 - \frac{b_0}{c_1} = 1.325 - \frac{0.001203}{4.267} = 1.3247181$$

i	a_i	b_i	c_i
3	1	1	1
2	0	1.324718	2.649436
1	—1	0.154878	4.264634
0	—1	0.0000004	

$$x_3 = x_2 - \frac{b_0}{c_1} = 1.324718 - \frac{0.0000004}{4.264634} = 1.3247179$$

Из таблицы видно, что x_2 и x_3 совпадают с точностью до седьмого знака; поэтому x_3 имеет по крайней мере семь достоверных значащих цифр и почти наверняка больше.

Дальнейшее рассмотрение методов нахождения корней многочленов продолжается в практическом примере 7.

5.8. ВЛИЯНИЕ НЕТОЧНОСТИ КОЭФФИЦИЕНТОВ МНОГОЧЛЕНА

Очень часто коэффициенты a_i многочлена (5.16) получаются из эксперимента или из предыдущих вычислений. В любом случае значения коэффициентов содержат в себе некоторые ошибки и очень важно знать, как влияет ошибка в значении коэффициента на ошибку в вычисленном значении корня.

Необходимо отметить, что эта ошибка не зависит от использованного при решении задачи метода вычислений. Мы будем предполагать, что ошибка округления при вычислениях отсутствует; точнее, она пренебрежимо мала по сравнению с ошибками, возникающими из-за неточности коэффициентов. На практике это допущение оправдывается довольно часто. Например, нет ничего необычного в том, что с помощью ЭЦВМ, числа в которой представляются с точностью до 10 значащих цифр, отыскиваются корни многочлена, коэффициенты которого известны всего с точностью до нескольких процентов. Естественно, что в таких обстоятельствах ошибкой округления можно пренебречь.

Пусть каждое значение a_i задано с ошибкой δ_i, т. е. *фактический* многочлен имеет следующий вид:

$$F^*(x) = (a_0 + \delta_0) + (a_1 + \delta_1)\, x + \\ + (a_2 + \delta_2)\, x^2 + \ldots + (a_m + \delta_m)\, x^m, \qquad (5.19)$$

причем $|\delta_i|$ малы по сравнению с $|a_i|$. Пусть $\bar{x}$ является корнем многочлена

$$F(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_m x^m. \qquad (5.20)$$

Пусть, наконец,

$$x^* = \bar{x} + \varepsilon \qquad (5.21)$$

представляет собой корень многочлена (5.19). Предположим также, что $|\varepsilon|$ мало по сравнению с $|\bar{x}|$. Наша задача

заключается в том, чтобы оценить ε. Необходимо отметить, однако, что проведенное ниже рассуждение основано на предположении о малости $|\varepsilon|$ сравнительно с $|\bar{x}|$; в противном случае результаты окажутся неверными. Во многих случаях, однако, это предположение оправдано; кроме того, всегда можно проверить его справедливость, если имеется оценка величины ε.

Подставляя (5.21) в (5.19) и учитывая, что

$$F^*(x^*) = 0,$$

имеем

$$(a_0 + \delta_0) + (a_1 + \delta_1)(\bar{x} + \varepsilon) + \ldots + (a_m + \delta_m)(\bar{x} + \varepsilon)^m = 0.$$

Раскрывая скобки в $(\bar{x} + \varepsilon)^j$ и пренебрегая членами, содержащими ε в степенях выше первой, получаем

$$(a_0 + \delta_0) + (a_1 + \delta_1)(\bar{x} + \varepsilon) + \ldots + (a_{m-1} + \delta_{m-1}) \times$$
$$\times (\bar{x}^{m-1} + (m-1)\varepsilon\bar{x}^{m-2}) + (a_m + \delta_m)(\bar{x}^m + m\varepsilon\bar{x}^{m-1}) = 0.$$

Пренебрегая также членами, содержащими произведения $\delta_i\varepsilon$, находим окончательно

$$\delta_0 + \delta_1\bar{x} + \ldots + \delta_m\bar{x}^m + \varepsilon(a_1 + 2a_2\bar{x} + \ldots + ma_m\bar{x}^{m-1}) = 0,$$

так как $F(\bar{x}) = 0$,

или

$$\sum_{i=0}^{m} \delta_i\bar{x}^i + \varepsilon f'(\bar{x}) = 0.$$

Верхняя граница возможной величины ε выразится тогда в виде

$$|\varepsilon| \leqslant \frac{\left|\sum_{i=0}^{m} \delta_i(\bar{x}^i)\right|}{|f'(\bar{x})|}. \qquad (5.22)$$

Рассмотрим теперь два частных случая:

1. Коэффициенты многочлена измерены в результате эксперимента с точностью до некоторого определенного знака после запятой (например, p знаков).

2. Коэффициенты многочлена вычислены заранее с определенным количеством достоверных значащих цифр.

Речь идет о разнице между абсолютной и относительной точностью задания коэффициентов.

Случай 1. Если все a_i заданы с точностью p знаков после запятой, то

$$|\delta_i| \leqslant \frac{1}{2} \cdot 10^{-p}$$

и из (5.22) получаем

$$|\varepsilon| \leqslant \frac{\frac{1}{2} \cdot 10^{-p}}{|f'(\bar{x})|} \left| \sum_{i=0}^{m} x^i \right|.$$

В силу неравенства треугольника

$$\left| \sum_{i=0}^{m} \bar{x}^i \right| \leqslant \sum_{i=0}^{m} |\bar{x}|^i.$$

Правая часть этого последнего неравенства представляет собой геометрическую прогрессию и равна

$$\frac{1-|\bar{x}|^{m+1}}{1-|\bar{x}|}.$$

Поэтому окончательно

$$|\varepsilon| \leqslant \frac{10^{-p}(1-|\bar{x}|^{m+1})}{2|c_1|(1-|\bar{x}|)}. \tag{5.23}$$

Вместо $f'(x)$ в этой формуле подставлено c_1, вычисленное по формулам (5.17) и (5.18).

Пример

$F(x) = x^3 - x - 1$. Предположим, что коэффициенты заданы с точностью до четвертого знака после запятой ($p = 4$). Один из корней этого уравнения, как было показано выше, равен 1.324718. Верхняя граница (5.23) при этом равна

$$0.75 \cdot 10^{-4}$$

Это означает, что

$$x = 1.324718 \pm 0.000075$$

Таким образом, дальнейшие итерации следует признать бессмысленными, так как они не приведут к уточнению значения корня.

Случай 2. Если коэффициенты a_i заданы с точностью в t значащих цифр

$$|\delta_i| \leqslant 5 \cdot 10^{-t} |a_i|,$$

то, исходя из (5.22),

$$|\varepsilon| \leqslant 5 \cdot 10^{-t} \frac{\sum_{i=0}^{m} |a_i \bar{x}^i|}{|c_1|}.$$

Пример

$F(x) = x^3 - x - 1$. Предположим, что коэффициенты заданы с точностью до четырех значащих цифр ($t = 4$). Тогда для $x = 1.324718$

$$|\varepsilon| \leqslant 0.00055,$$

так что

$$x = 1.324718 \pm 0.00055.$$

Таким образом, в качестве значения корня должно быть взято 1.325, причем возможна ошибка на одну единицу младшего разряда.

5.9. СИСТЕМЫ УРАВНЕНИЙ

Весьма часто приходится решать задачи, где для определения нескольких неизвестных величин задается столько же уравнений. Пусть, например, необходимо найти такие x и y, при которых

$$x^2 + y = 3,$$
$$y^2 + x = 5.$$

В этом случае можно выразить y через x из первого уравнения и поставить во второе уравнение. При этом получается

$$x^4 - 6x^2 + x + 4 = 0.$$

Теперь мы получили многочлен, корни которого можно найти с помощью методов, рассмотренных выше. Один из корней равен $x = 1$, и поэтому $y = 2$.

Очень часто, однако, бывает трудно или нецелесообразно сводить задачу к решению одного уравнения с одним неизвестным. Соответствующая ситуация характерна для решения системы линейных уравнений; этот случай специально рассматривается в гл. 8, так как для линейных уравнений разработаны специальные методы решения. Здесь мы приводим итерационные формулы для решения системы двух

нелинейных уравнений с двумя неизвестными, полученные путем обобщения метода Ньютона — Рафсона. Вывод этих формул предоставляется читателю (упражнение 32).

Пусть заданы уравнения

$$F(x, y) = 0,$$
$$G(x, y) = 0$$

и известны приближенные значения корней x_n, y_n.

Положим

$$J(x_n, y_n) = \frac{\partial F(x_n, y_n)}{\partial x}\frac{\partial G(x_n, y_n)}{\partial y} - \frac{\partial F(x_n, y_n)}{\partial y}\frac{\partial G(x_n, y_n)}{\partial x}.$$

Это выражение называется якобианом системы; предполагается, что он отличен от нуля. Это аналогично предположению $F'(x) \neq 0$ в случае одной переменной.

При этих условиях следующее приближение можно вычислить по формулам

$$x_{n+1} = x_n - \frac{F(x_n, y_n)\dfrac{\partial G(x_n, y_n)}{\partial y} - G(x_n, y_n)\dfrac{\partial F(x_n, y_n)}{\partial y}}{J(x_n, y_n)},$$

$$y_{n+1} = y_n + \frac{F(x_n, y_n)\dfrac{\partial G(x_n, y_n)}{\partial x} - G(x_n, y_n)\dfrac{\partial F(x_n, y_n)}{\partial x}}{J(x_n, y_n)}.$$

Итерационный процесс продолжается до тех пор, пока два последовательных приближения не будут достаточно близки друг к другу. Численные примеры можно найти в упражнениях 33 и 34.

Обобщение простого метода последовательных приближений для случая системы двух уравнений дается в упражнении 35.

5.10. КОМПЛЕКСНЫЕ КОРНИ

Все методы, описанные выше, применялись нами для нахождения вещественных корней уравнений или систем из двух уравнений. Сейчас мы очень кратко рассмотрим решение уравнений, корни которых являются комплексными числами.

Должно быть ясно, что если все значения функции вещественны и в качестве начального приближения x_0 также

взято вещественное число, то и все последующие значения x_i также будут вещественными. Однако если взять в качестве x_0 комплексное число, то последующие x_i также могут оказаться комплексными. Все методы, описанные выше, годятся для работы с комплексными числами так же хорошо, как и с действительными. Во многих вариантах ФОРТРАНа предусмотрена возможность работы с комплексными числами, так что в этом случае для нахождения комплексных корней достаточно небольшого изменения программы.

Кроме того, известно, что если $a + bi$ есть корень многочлена с действительными коэффициентами, то $a - bi$ также является его корнем. Поэтому многочлен $p_n(x)$ степени n, имеющий такую пару корней, можно записать в следующем виде:

$$p_n(x) = (x^2 - 2ax + (a^2 + b^2))\, p_{n-2}(x),$$

где выражение в скобках называется квадратичным множителем; $p_{n-2}(x)$ является многочленом степени $n - 2$. Тогда можно провести анализ, подобный тому, что проведен в разд. 5.7, и найти a и b, не выходя за пределы арифметики вещественных чисел. Интересующимся читателям рекомендуем гл. 10 книги Гильдебранда[1]).

Использование метода последовательных приближений для нахождения комплексных корней известно под названием метода Лина, а использование для той же цели метода Ньютона — Рафсона известно под названием метода Берстоу.

5.11. НАХОЖДЕНИЕ ИСХОДНОГО ПРИБЛИЖЕНИЯ

Приближенное значение корня уравнения

$$F(x) = 0$$

иногда известно из физических соображений. Если же это значение неизвестно, то его часто можно найти с помощью грубого анализа функции.

В основном этот анализ сводится к тому, что отыскиваются такие два значения x, для которых $F(x)$ имеет противо-

[1]) Hildebrand F. B., Introduction to numerical analysis, McGraw-Hill, New York, 1956.

положные знаки, т. е. определяются такие x^* и x_*, для которых

$$F(x^*) > 0$$

и

$$F(x_*) < 0.$$

Тогда между x^* и x_* есть по крайней мере одна точка, где $F(x) = 0$. В качестве исходного приближения для нахождения корня $F(x)$ можно взять

$$x_0 = \frac{1}{2}(x^* + x_*).$$

Одним из путей для нахождения x^* и x_* является подбор более простого уравнения, корни которого расположены вблизи от корней исходного. Например, если

$$F(x) = \frac{\sin x}{10} + x^3 - 1,$$

то при $0 \leqslant x \leqslant \pi/2$ первый член меняется между 0 и 1/10. Так как по сравнению с двумя другими членами эта величина мала, то рассмотрим упрощенное уравнение для двух крайних значений $(\sin x)/10$:

$$0 + x^3 - 1 = 0$$

и

$$\frac{1}{10} + x^3 - 1 = 0.$$

Корни каждого из этих двух простых выражений равны соответственно

$$x^* = +1$$

$$x_* = \sqrt[3]{0.9} \approx 0.965489$$

Проверяем для исходного уравнения

$$F(x^*) = 0.084$$

и

$$F(x_*) = -0.18$$

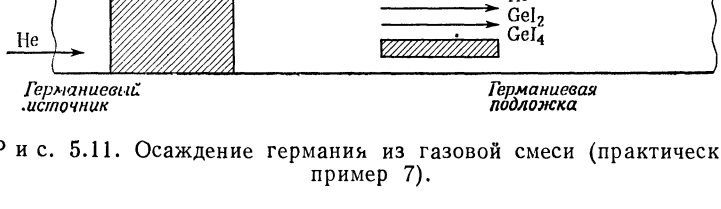
Таким образом, в качестве первого приближения можно принять

$$x_0 = \frac{1}{2}(1 + 0.965489) = 0.982749$$

Для приближенного нахождения корней многочленов существует много специальных приемов. Заинтересованному читателю рекомендуем гл. 2 книги К. Кунца [1]).

5.12. ПРАКТИЧЕСКИЙ ПРИМЕР 7: ПРОЦЕСС РОСТА МОНОКРИСТАЛЛА ИЗ ПАРА

Следующий практический пример иллюстрирует круг вопросов, относящихся к приложениям численных методов. Этот пример показывает, как при решении научной задачи



Р и с. 5.11. Осаждение германия из газовой смеси (практический пример 7).

возникает полиномиальное уравнение, как следует его задавать, а также как иногда оно может упрощаться применительно к конкретной ситуации.

Когда пары иода и гелия (I_2 и He) проходят над поверхностью германия (Ge), то некоторое количество германия вступает в реакцию с иодом и образует GeI_2 и GeI_4. Выходящие пары состоят из I, I_2, GeI_2, GeI_4 и He (рис. 5.11).

Если теперь пропускать этот пар над кристаллической подложкой из германия, то некоторое количество GeI_2 диссоциирует на Ge и GeI_4; германий при этом осаждается на подложке. Добавляя определенные примеси в подложку и в осаждаемый кристалл, можно изготовлять различные полупроводниковые приборы [2]).

Вкратце анализ сводится к следующему. Предположим, что полное давление постоянно и равно 760 *мм рт. ст.*

---

[1]) Kunz K. W., Numerical analysis, McGraw-Hill, 1957.

[2]) Полный термодинамический анализ этого процесса и других связанных с ним явлений проводится в статье Riesman A., Alyanakyan S. A., *Journal of the Electrochemical Society*, 111 (1964).

Предположим также, что газы подчиняются законам идеального газа. Рассмотрим реакции, которые протекают в ходе процессов сублимации и осаждения:

$$I_{2(г)} \rightleftarrows 2I_{(г)}; \tag{5.24}$$

$$Ge_{(т)} + I_{2(г)} \rightleftarrows GeI_{2(г)}; \tag{5.25}$$

$$Ge_{(т)} + GeI_{4(г)} \rightleftarrows 2GeI_{2(г)}. \tag{5.26}$$

Индексы ($т$) и ($г$) поставлены для того, чтобы обозначить, находится ли вещество в твердом или в газообразном состоянии.

Для некоторой фиксированной температуры T отношения парциальных давлений различных газов, возведенных в соответствующую степень, называются константами равновесия. Например, в реакции (5.24)

$$\frac{(p_I)^2}{p_{I_2}} = K_{24},$$

где p_I — парциальное давление атомарного иода (I),

p_{I_2} — парциальное давление молекулярного иода (I_2),

$$\log_{10} K_{24} = 8.362 - 7991/T.$$

Примем температуру равной

$$T = 273.2° K = 0° C.$$

Тогда

$$K_{24} = 1.30 \cdot 10^{-21} = \frac{(p_I)^2}{p_{I_2}}. \tag{5.27}$$

Аналогичным образом можно найти константы равновесия для реакций (5.25) и (5.26)

$$K_{25} = 1.09 \cdot 10^{4} = \frac{p_{GeI_2}}{p_{I_2}}, \tag{5.28}$$

$$K_{26} = 3.25 \cdot 10^{-17} = \frac{(p_{GeI_2})^2}{p_{GeI_4}}. \tag{5.29}$$

Парциальное давление твердого германия не входит в эти соотношения, так как в первом приближении оно не зависит от внешнего давления.

Так как полное давление в смеси равно 760 *мм рт. ст.*, то имеем

$$p_{\text{полн}} = 760 = p_{\text{I}} + p_{\text{I}_2} + p_{\text{GeI}_2} + p_{\text{GeI}_4} + p_{\text{He}}. \qquad (5.30)$$

Далее, из закона сохранения вещества следует

$$X = \frac{P_{\text{He}}}{P_{\text{I}_2}} = \frac{p_{\text{He}}}{\frac{1}{2} p_{\text{I}} + p_{\text{I}_2} + p_{\text{GeI}_2} + 2p_{\text{GeI}_4}}, \qquad (5.31)$$

где X есть отношение давления гелия к давлению иода. Это отношение устанавливается в начале опыта и впоследствии должно оставаться постоянным. Величина X определяется экспериментально; в том эксперименте, который описывается здесь, это отношение составляло 49.36.

Уравнения от (5.27) до (5.31) являются пятью уравнениями относительно пяти неизвестных парциальных давлений. Если обозначить $Z^4 = P_{\text{GeI}_4}$, то можно свести эти пять уравнений к одному уравнению для Z, а именно

$$(2X+1)Z^4 + \left[(1+X)\sqrt{K_{26}}\left(1+\frac{1}{K_{25}}\right)\right]Z^2 + \\ + \left(\frac{K_{24}\sqrt{K_{26}}}{K_{25}}\right)^{1/2}\left(1+\frac{X}{2}\right)Z - 760 = 0. \qquad (5.32)$$

Остальные парциальные давления можно найти из соотношений

$$p_{\text{I}_2} = \frac{\sqrt{K_{26}}}{K_{25}} Z^2,$$

$$p_{\text{I}} = \left(\frac{K_{24}\sqrt{K_{26}}}{K_{25}}\right)^{1/2} Z,$$

$$p_{\text{GeI}_2} = \sqrt{K_{26}}\, Z^2,$$

$$p_{\text{He}} = X\left[2Z^4 + \sqrt{K_{26}}\left(1+\frac{1}{K_{25}}\right)Z^2 + \frac{1}{2}\left(\frac{K_{24}\sqrt{K_{26}}}{K_{25}}\right)^{1/2} Z\right].$$

При заданных $T = 273.2°$ и $X = 49.36$ уравнение (5.32) принимает вид

$$9.972 \cdot 10^1 Z^4 + 2.870565 \cdot 10^{-7} Z^2 + \\ + 6.674356 \cdot 10^{-16} Z - 7.600 \cdot 10^2 = 0 \qquad (5.33)$$

Уравнение (5.33) представляет собой многочлен четвертой степени относительно Z. Его можно легко решить методом Ньютона — Рафсона. Программа для решения этого уравнения приведена на рис. 5.12.

Из физических соображений известно, что парциальное давление GeI_4 лежит между 0 и 10 *мм рт. ст.*; в качестве исходного приближения Z_0 мы взяли 5.0.

Метод Ньютона — Рафсона был изложен в виде последовательных итераций, причем каждой очередной итерации присваивался новый индекс (1, 2, 3, ...). Нет никакой необходимости сохранять это обозначение в программе; ведь все, что требуется для вычислений — это результаты предшествующей и текущей итераций. В программе на на рис. 5.12 переменная Z представляет собой результат предшествующей итерации (первоначально равный 5.0), а переменная ZNEW — новое значение, которое вычисляется по формуле Ньютона — Рафсона.

Мы ожидаем, что итерационный процесс будет сходиться, но опытный программист всегда примет меры к тому, чтобы избежать такой ситуации, когда ошибки в исходной информации или в программе могли бы вызвать неопределенно долгую работу ЭЦВМ без выхода из цикла. Поэтому мы введем специальную переменную, так называемый счетчик. В программе этой переменной присвоено наименование ITN. Если процесс не сойдется после 15 попыток, то он должен быть прекращен.

Значения функции и производной вычисляются по рекуррентным формулам, приведенным в разд. 5.7. Конечно, можно было бы написать

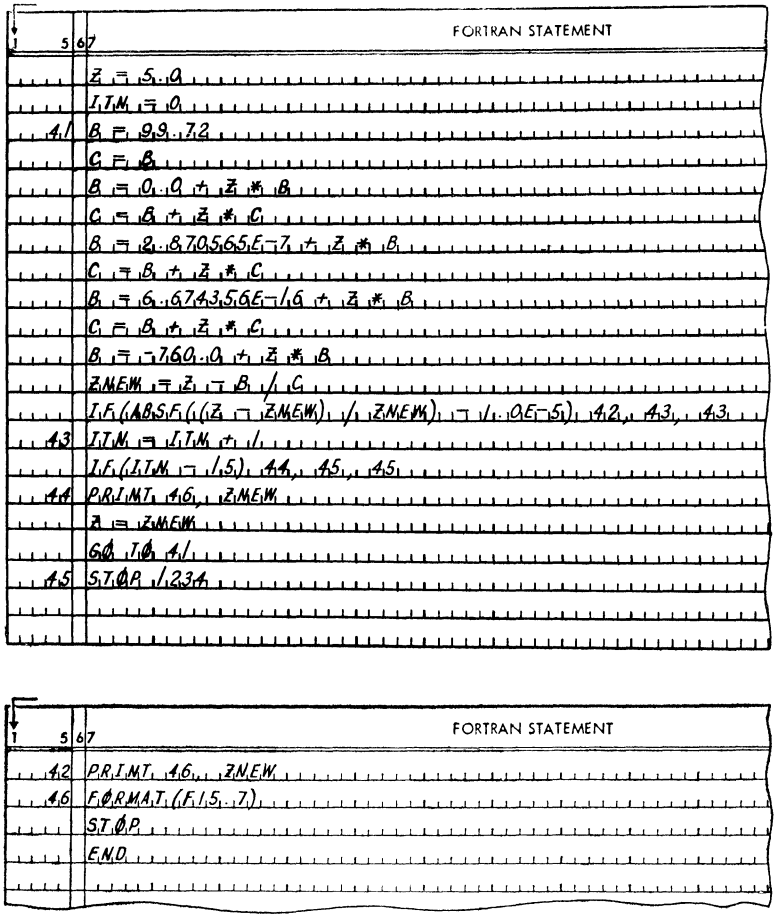
$$\frac{F(Z)}{F'(Z)} = \frac{A_0 + Z(A_1 + Z(A_2 + Z(A_3 + ZA_4)))}{A_1 + Z(2A_2 + Z(3A_3 + 4ZA_4))},$$

и тогда вся итерационная формула выразилась бы с помощью одного длинного арифметического оператора. Мы предпочли сделать вычисления так, как это показано в программе, отчасти потому, что это немного быстрее[1]), отчасти потому, что такой способ вычисления лучше согласуется с использованием переменных с индексами, чему будет посвящена гл. 7.

После вычисления нового значения ZNEW мы должны проверить, не стали ли два очередных приближения достаточно близки друг к другу; это укажет, что достигнута

[1]) Читатель может сам убедиться, что в приведенной программе на вычисление $F(Z)$ и $F'(Z)$ затрачивается на три умножения меньше, чем потребовалось бы в случае явной формулы.

необходимая точность. Согласно общему правилу, не следует производить это испытание по абсолютной точности



Р и с. 5.12. Программа для нахождения корней многочлена четвертого порядка с помощью метода Ньютона — Рафсона (практический пример 7).

совпадения двух очередных приближений. Например, если бы корень оказался порядка 10^{-4}, то было бы бессмысленно

проверять, не отличаются ли два очередных приближения на 10^{-3}; процесс остановился бы почти немедленно, а значение «корня» имело бы очень небольшую относительную точность. С другой стороны (хотя здесь этот случай и невозможен), если бы корень был порядка 10^5, а мы производили бы проверку на совпадение с точностью 10^{-3}, то ошибки округления привели бы к тому, что процесс вообще не удалось бы закончить. Очевидное решение проблемы состоит в том, чтобы проверять относительную точность совпадения двух очередных приближений. Поэтому в программе проверка состоит в сравнении абсолютной величины разности между двумя последовательными приближениями, деленной на последнее из них, с допуском, скажем, 10^{-5}. По-видимому, такая проверка в данном случае достаточна, если учесть ограниченную точность исходной информации.

```
3.7652427
2.8596258
2.2261980
1.8423436
1.6864486
1.6620756
1.6615289
1.6615286
```

Рис. 5.13. Выходная печать для программы рис. 5.12 практический пример 7).

Если процесс сошелся, то мы печатаем окончательное значение ZNEW с помощью оператора PRINT и прекращаем вычисления. Если процесс не закончился, то мы проверяем с помощью переменной ITN, сделано ли уже 15 итераций, и при достижении этого предела останавливаем программу. В противном случае мы присваиваем переменной Z (результату предшествующей итерации) значение ZNEW, возвращаемся к началу программы и вычисляем очередное приближение.

Чтобы можно было наблюдать сходимость, в итерационный цикл введен оператор PRINT. Обычно этого не делают.

Результаты вычислений показаны на рис. 5.13; быстрая сходимость метода Ньютона — Рафсона очевидна.

Парциальные давления получаются равными

$$p_{GeI_4} = 7.621 \cdot 10^{0}$$

$$p_{I_2} = 1.428 \cdot 10^{-12}$$

$$p_{I} = 4.3 \cdot 10^{-17}$$

$$p_{GeI_2} = 1.573 \cdot 10^{-8}$$

$$p_{He} = 7.524 \cdot 10^{2}$$

Читатели, вероятно, уже заметили, что в уравнении (5.33) коэффициенты при Z^2 и Z гораздо меньше, чем при Z^4 и Z^0. Если пренебречь членами, содержащими Z^2 и Z, и решать уравнение

$$9.972 \cdot 10^1 Z^4 - 760 = 0,$$

то получится тот же результат, а именно $Z = 1.6615286$. При этом метод Ньютона — Рафсона даст те же последовательные приближения, как на рис. 5.13.

Упражнения

«Вычислительные» упражнения, приведенные ниже (а также в последующих главах), составлены так, что в случае необходимости они могут выполняться вручную или на клавишной вычислительной машине. Некоторые из них, однако, вполне могут быть решены с помощью ЭЦВМ, причем рекомендуется использовать приемы из упражнений 22—27. Решение этих упражнений с помощью ЭЦВМ, по-видимому, не приведет к существенной экономии времени, но может дать ценные практические навыки программирования итерационных методов.

*1. Вычислите отрицательный квадратный корень из 0.5, написав уравнение $F(x) = x^2 - 0.5$ и решая уравнение $x = x^2 + x - 0.5$ методом последовательных приближений. В качестве x_0 возьмите —0.6. Можно ли тем же методом найти положительный корень?

2. Найдите отрицательный квадратный корень из 0.25 тем же способом, что и в предыдущем упражнении. В качестве x_0 снова возьмите —0.6. Почему на этот раз метод сходится быстрее?

3. Используйте метод Ньютона — Рафсона для того, чтобы найти квадратный корень из 4 с точностью до четвертого знака. В качестве x_0 возьмите 1.5. Повторите вычисления, взяв в качестве x_0: 2.5, —1.5, 10.0.

*4. Выведите на основе метода Ньютона — Рафсона итерационную формулу для вычисления кубического корня из положительного числа c.

5. Выведите на основе метода Ньютона — Рафсона итерационную формулу для вычисления $\arcsin A$, если задано A.

6. Используя результат упражнения 5, найдите $\arcsin 0.5$ с точностью до третьего знака.

*7. Вычислите с точностью до третьего знака корень уравнения $0.1\, x^2 - x \ln x = 0$, лежащий между 1 и 2.

8. Найдите с точностью до третьего знака корень уравнения

$$\operatorname{ch} x + \cos x - 3 = 0.$$

*9. Используйте метод Ньютона—Рафсона для того, чтобы найти с точностью до третьего знака все корни уравнения

$$x^3 - 1.473x^2 - 5.738x + 6.763 = 0.$$

10. Используйте метод Ньютона — Рафсона для того, чтобы найти с точностью до третьего знака корни уравнения $x^2 - x - 6 = 0$. В качестве x_0 возьмите 0, затем повторите для $x_0 = 4$. Итерационную формулу можно существенно упростить алгебраическими преобразованиями.

11. Уравнение $4x^3 - 12.3x^2 - x + 16.2 = 0$ имеет два корня в интервале между 1 и 2. Найдите их с точностью до четвертого знака.

***12.** Найдите с точностью до третьего знака корень уравнения $x^3 - 0.39\,x^2 - 10.5\,x + 11.0 = 0$, лежащий между 2 и 3. Если коэффициенты уравнения известны с точностью 2%, какова верхняя граница возможной ошибки при вычислении корня?

13. То же, что и в упражнении 12, но коэффициенты содержат погрешность 4%.

14. Покажите, что если каждый коэффициент полиномиального уравнения имеет погрешность $P\%$, то погрешность при вычислении корня будет линейной функцией от P, если оправдываются допущения о малости этой погрешности (см. разд. 5.8).

15. Уравнение $2.0x^2 - 5.0x + 2.0 = 0$ имеет корни $x_1 = 0.5$ и $x_2 = 2.0$. Если коэффициенты известны с погрешностью 20%, то верхняя граница возможной ошибки при вычислении x_2 равна 1.33. Однако больший корень уравнения $1.6\,x^2 - 6.0x + 1.6 = 0$, в котором коэффициенты изменились на 20% по сравнению с исходным, равен 3.47, т. е. ошибка составляет 1.47. Почему фактическая ошибка оказалась выше верхнего предела? С другой стороны, если взять уравнение $1.6x^2 - 4.0x + 1.6 = 0$, то его больший корень равен 1.79, т. е. ошибка составляет всего 0.21; почему теперь ошибка гораздо меньше, чем в первом случае и гораздо меньше верхнего предела?

16. Рассмотрим уравнение

$$x^4 - 26x^3 + 131x^2 - 226x + 120 = 0.$$

Корни этого уравнения равны 1, 2, 3 и 20. Предположим, что в постоянном члене имеется малая погрешность, а все остальные коэффициенты заданы точно. Покажите, что эта погрешность оказывает вдвое большее влияние на верхнюю границу ошибки для корня, близкого к 3, нежели для корня, близкого к 1, и почти не влияет на корень, близкий к 20. Теперь предположите, что имеется малая погрешность в коэффициенте при x^3, а все остальные коэффициенты заданы точно. Покажите, что верхняя граница ошибки для корня, близкого к 1, гораздо меньше этой погрешности и превосходит ее для корня, близкого к 3.

17. Примените метод Ньютона — Рафсона для решения уравнения $x^3 - 2x^2 - 3x + 10 = 0$, причем в качестве x_0 возьмите 1.9. Можете ли вы объяснить странное поведение последовательных значений корня?

18. Используйте метод Ньютона — Рафсона для решения уравнения $x^3 - 2x^2 - 3x + 10 = 0$, производя все вычисления с комплексными числами. В качестве x_0 возьмите $3 + i$.

19. Уравнение

$$x^5 + 8x^4 + 17x^3 - 8x^2 - 14x + 20 = 0$$

имеет корни —1 и +2. Однако если использовать метод Ньютона — Рафсона, взяв в качестве x_0: —0.3, то процесс сойдется к корню, равному +5. Объясните, почему так получается.

20. Попытайтесь использовать метод Ньютона — Рафсона для решения уравнения

$$x^4 - 7x^3 + 12x^2 + 4x - 16 = 0.$$

Что при этом получается?

***21.** Напишите на ФОРТРАНе программу для вычисления квадратного корня из переменной A, которой предварительно присвоено некоторое значение. Назовите результат вычислений SQRTA.

22. Предположите, что задано уравнение вида

$$F(x) = a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + \ldots + a_0 = 0.$$

Так как любой из коэффициентов может оказаться равным нулю, то это уравнение может быть шестого порядка или ниже. Предположите, кроме того, что имеется перфокарта с числами, пробитыми в следующем формате:

Колонки:	
1—10	x_0
11—20	a_0
21—30	a_1
31—40	a_2
41—50	a_3
51—60	a_4
61—70	a_5
71—80	a_6

Все числа пробиты в формате F10.0. Напишите программу для считывания данных с этой карты, вычисления $F(x_0)$ и печати коэффициентов, x_0 и $F(x_0)$, используя для этого спецификацию F15.6.

23. В дополнение к перфокарте, описанной в упражнении 22, имеется вторая карта, которая содержит в колонках 1—10 значение переменной EPS в формате F10.0. Напишите программу для вычисления корня многочлена из упражнения 22, применяя метод Ньютона — Рафсона, полагая x_0 в качестве исходного приближения и продолжая итерационный процесс до тех пор, пока два приближения не будут отличаться по абсолютной величине меньше, чем на EPS. Когда корень найден, напечатайте его и всю исходную информацию. Рекомендуется вначале ввести оператор PRINT в итерационный цикл, чтобы можно было следить за сходимостью.

24. В дополнение к информации, содержащейся в упражнениях 22 и 23, вторая перфокарта содержит также значение переменной P, которая представляет собой максимальную ошибку коэффициентов в процентах. После нахождения корня вычислите верхнюю границу для его возможной ошибки и напечатайте это число наряду с прочей информацией.

25. Предположим, что имеются две перфокарты с входной информацией. Первая карта та же самая, что в упражнении 22,

но значение x_0 не используется. (При том расположении чисел, которое описано в упражнении 22, необходимо будет прочесть значение x_0, но не использовать его.) Вторая перфокарта содержит следующую информацию:

Колонки: 1—10 XF
11—20 XL
21—30 DELTA

Напишите программу, где читаются эти перфокарты, и затем производятся следующие вычисления:

1. Присвоить переменной X значение XF.
2. Вычислить F(X) и присвоить это значение переменной BEFORE.
3. Вычислить F(X + DELTA) и присвоить это значение переменной THIS.
4. Если BEFORE и THIS имеют разный знак, то напечатайте X и X+DELTA. (Простейший способ определить, одинаковы или различны знаки двух чисел, заключается в анализе знака их произведения.)
5. Увеличьте значение переменной X на величину DELTA.
6. Если X стало больше XL, прекратите вычисления; в противном случае переходите к следующей операции.
7. Присвоить переменной BEFORE значение THIS.
8. Перейти к операции 3.

26. Используя методику предыдущих упражнений, составьте программу, с помощью которой можно было бы прочесть коэффициенты XF, XL, DELTA и EPS (выберите сами подходящие форматы для входных перфокарт), найти места перемены знака функции и вычислить корни при каждой перемене знака. (Это будет очень ценное упражнение и очень полезная программа, но необходимо помнить о тех неожиданностях, с которыми пришлось столкнуться в упражнениях 17, 19 и 20. Полная программа для нахождения корней многочлена 6-го порядка будет значительно сложнее.)

27. Усовершенствуйте программу упражнения 26, введя в нее счетчик; он должен прекращать вычисления, если процесс не сходится после N итераций. N является целым числом, которое вводится с второй перфокарты. (Для N целесообразно принять значение порядка 15).

28. Пусть задано уравнение $F(x) = 0$ и два значения X, равные XL и XH, причем XL < XH и F(XL) и F(XH) имеют разные знаки; можно утверждать, что уравнение $F(x) = 0$ имеет по крайней мере один корень между XL и XH. Предположим, что в этом интервале имеется точно один корень и рассмотрим следующую вычислительную процедуру:

1. Вычислить $F\left(\frac{XL+XH}{2}\right)$.
2. Если F(XL) и $F\left(\frac{XL+XH}{2}\right)$ имеют разные знаки, присвоить переменной XH значение $\left(\frac{XL+XH}{2}\right)$, оставить XL неизмен-

ным и перейти к операции 3; если же $F(XL)$ и $F\left(\frac{XL+XH}{2}\right)$ имеют одинаковые знаки, присвоить переменной XL значение $\left(\frac{XL+XH}{2}\right)$, оставить XH неизменным и перейти к операции 3.

3. Если (XH — XL) меньше, чем некоторый заданный допуск EPS, то прекратить вычисления, так как значение XL или XH равно корню уравнения с точностью EPS. Если (XH — — XL) больше EPS, перейти к операции 1.

Описанная процедура называется *методом деления пополам*. Он не столь элегантен, как те способы, которые рассматривались выше, и вообще не так быстро сходится, но его иногда применяют из-за простоты. Истолкуйте геометрически процесс вычислений, начертите блок-схему и напишите программу для нахождения корня уравнения

$$x^4 - 0.486x^3 - 5.792x^2 + 0.486x + 4.792 = 0,$$

лежащего между 2 и 3. Используйте допуск 10^{-3}.

29. Пусть в методе последовательных приближений m^* является минимальным значением $|f'(x)|$ для всех x. Покажите, что если $m^* > 1$, то процесс расходится.

30. Предположим, что имеются два значения x_0 и x_1, такие, что $F(x_0) < 0$ и $F(x_1) > 0$. Пусть x_2 — точка пересечения оси x с прямой, соединяющей точки $(x_0, F(x_0))$ и $(x_1, F(x_1))$. Продемонстрируйте геометрически, что x_2 является лучшим приближением к корню уравнения $F(x) = 0$, чем x_0 и x_1. Разработайте вычислительный алгоритм для нахождения корня уравнения $F(x) = 0$, подобный алгоритму из упражнения 28.

31. Предположите, что функция $F(x)$ разложена в ряд Тейлора в окрестности точки x_n и ограничьте разложение двумя членами ряда

$$F(x) = F(x_n) + (x - x_n) F'(x_n).$$

Покажите, что при этих условиях получается приближенная формула

$$x = x_n - \frac{F(x_n)}{F'(x_n)}.$$

Что означает эта формула?

32. Ограничиваясь членами не выше первого порядка в разложении функций двух переменных в ряд Тейлора

$$F(x, y) = F(x_n, y_n) + (x - x_n)\frac{\partial F(x_n, y_n)}{\partial x} + (y - y_n)\frac{\partial F(x_n, y_n)}{\partial y} + \ldots$$

$$G(x, y) = G(x_n, y_n) + (x - x_n)\frac{\partial G(x_n, y_n)}{\partial x} + (y - y_n)\frac{\partial G(x_n, y_n)}{\partial y},$$

покажите, что последующее приближение к корню уравнений

$$F(x, y) = 0,$$
$$(x, y) = 0$$

дается формулами

$$x = x_n - \left[F(x_n, y_n) \frac{\partial G(x_n, y_n)}{\partial y} - G(x_n, y_n) \frac{\partial F(x_n, y_n)}{\partial y} \right] \Big/ J,$$

$$y = y_n + \left[F(x_n, y_n) \frac{\partial G(x_n, y_n)}{\partial x} - G(x_n, y_n) \frac{\partial F(x_n, y_n)}{\partial x} \right] \Big/ J,$$

где

$$J = \frac{\partial F(x_n, y_n)}{\partial x} \frac{\partial G(x_n, y_n)}{\partial y} - \frac{\partial F(x_n, y_n)}{\partial y} \frac{\partial G(x_n, y_n)}{\partial x}.$$

***33.** Пусть

$$F(x, y) = x^2 + y^2 - 4,$$
$$G(x, y) = xy - 1.$$

Примените метод, описанный в разд. 5.9, взяв $x_0 = 2$, $y_0 = 0$. Имеет ли система другие решения?

34. Пусть

$$F(x, y) = x^3 - x - 2x^2 - x + 2 - y,$$
$$G(x, y) = x - y.$$

Примените метод, описанный в разд. 5.9, принимая $x_0 = y_0 = 0$. Объясните сходство между двумя итерационными формулами.

35. Обобщение метода последовательных приближений на случай системы двух уравнений

$$x = f(x, y),$$
$$y = g(x, y)$$

дается формулами

$$x_{n+1} = f(x_n, y_n),$$
$$y_{n+1} = g(x_n, y_n).$$

а. Покажите, что достаточные условия сходимости этого метода записываются в виде

$$\left| \frac{\partial f}{\partial x} \right| + \left| \frac{\partial g}{\partial x} \right| < 1$$

и

$$\left| \frac{\partial f}{\partial y} \right| + \left| \frac{\partial g}{\partial y} \right| < 1.$$

(Указание: исследуйте верхнюю границу величины $|x - x_n| + |y - y_n|$.)

б. Являются ли условия, изложенные в пункте а, *необходимыми* для сходимости?

в. Рассмотрим линейные уравнения

$$a_{11}x + a_{12}y = b_1,$$
$$a_{21}x + a_{22}y = b_2.$$

Если $a_{11} \neq 0$ и $a_{22} \neq 0$, то эту систему можно переписать в виде

$$x = \frac{1}{a_{11}}(b_1 - a_{12}y),$$

$$y = \frac{1}{a_{22}}(b_2 - a_{21}x).$$

Как запишутся для данной системы достаточные условия сходимости из пункта а? Какие ограничения накладывают эти условия на наклоны прямых, представленных исходными уравнениями?

36. Рассмотрим кубическое уравнение

$$p_0(x) = x^3 + a_2x^2 + a_1x + a_0 = 0.$$

а. Если x_1, x_2 и x_3 — корни этого уравнения, то покажите, что

$$a_2 = -(x_1 + x_2 + x_3),$$

$$a_1 = x_1x_2 + x_1x_3 + x_2x_3,$$

$$a_0 = -x_1x_2x_3.$$

б. Покажите, что если x_1, x_2 и x_3 вещественны и, кроме того, $|x_1| \gg |x_2| \gg |x_3|$, то

$$x_1 \approx -a_2,$$

$$x_2 \approx -\frac{a_1}{a_2},$$

$$x_3 \approx -\frac{a_0}{a_1}.$$

в. Докажите, что

$$p_1(y) = -p_0(-x)\,p_0(x) = y^3 + (-a_2^2 + 2a_1)\,y^2 + (a_1^2 - 2a_0a_2)\,y - a_0^2,$$

где

$$y = x^2.$$

г. Покажите, что корни уравнения $p_1(y) = 0$ суть следующие

$$y_1 = x_1^2,$$

$$y_2 = x_2^2,$$

$$y_3 = x_3^2.$$

д. Покажите, что корни

$$p_2(z) = -p_1(-y)\,p_1(y)$$

имеют вид

$$z_1 = x_1^4,$$

$$z_2 = x_2^4,$$

$$z_3 = x_3^4,$$

где

$$z = y^2 = x^4.$$

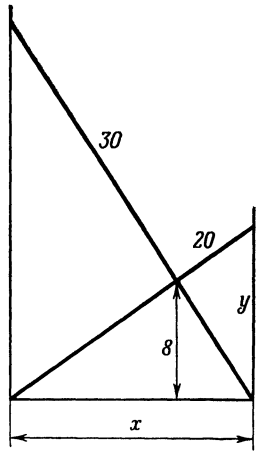
е. Что произойдет с корнями уравнения после многократного повторения процесса, описанного в пунктах в и д?

ж. Опишите, как бы вы нашли приближенные значения x_1, x_2 и x_3 после того, как процесс, описанный в пунктах в и д, был проделан достаточное количество раз и корни раздвинулись.

Этот процесс известен под названием алгоритма Греффе. Он обобщается на случай многочленов более высокого порядка и комплексных корней. Интересующимся читателям рекомендуем книгу Гильдебранда, § 10.11 [1]).

37. Следующая общеизвестная задача приводит к уравнению четвертого порядка.

Две лестницы, одна в 20 футов, другая в 30 футов длиной, поставлены поперек улицы, как показано на схеме, и опираются своими концами на противостоящие дома. Определить ширину улицы, если точка пересечения лестниц находится на высоте 8 футов над землей.



Покажите, что эта задача сводится к решению уравнения

$$y^4 - 16y^3 + 500y^2 - 8000y + 32\,000 = 0$$

и тогда

$$x = \sqrt{400 - y^2}.$$

---

[1]) Описанный алгоритм известен в отечественной литературе под названием метода Лобачевского. Подробное описание этого метода можно найти в книге А. Н. Крылова, Лекции по приближенным вычислениям, ГТТИ, М.— Л., 1950.— *Прим. ред.*

ГЛАВА

6 ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ

6.1. ВВЕДЕНИЕ

Задачи, в которых требуется вычисление интегралов, возникают почти во всех областях прикладной математики. Иногда удается найти аналитическую формулу, т. е. выразить неопределенный интеграл в виде комбинации алгебраических и трансцедентных функций, после чего остается вычислить значение определенного интеграла, подставляя в формулу пределы интегрирования.

Во многих случаях, однако, не удается найти никакой аналитической формулы или же она получается настолько сложной, что вычислять интеграл с ее помощью труднее, чем другими способами. В таких ситуациях приходится применять различные методы *численного интегрирования*, которые основаны на том, что интеграл представляется в виде предела суммы площадей, и позволяют вычислить эту сумму с достаточной точностью.

Обращаясь к содержанию этой главы, поставим задачу и сделаем необходимые предположения. Требуется вычислить определенный интеграл

$$I = \int_a^b f(x)\,dx \tag{6.1}$$

при условии, что a и b конечны и $f(x)$ является непрерывной функцией x во всем интервале $a \leqslant x \leqslant b$.

Представляют интерес также те случаи, когда один или оба предела интегрирования бесконечны, либо когда подинтегральная функция имеет особенности внутри интервала интегрирования или на его концах. Иногда удается привести соответствующие интегралы к виду (6.1), после чего их можно вычислить с помощью методов, изложенных

в этой главе. Мы будем рассматривать подобные задачи только в упражнениях в конце главы[1]).

Общий подход к решению задачи будет следующим. Определенный интеграл I представляет собой площадь, ограниченную кривой $f(x)$, осью x и прямыми $x=a$ и $x=b$. Мы будем пытаться вычислить I, разбивая интервал от a до b на множество меньших интервалов, находя приблизительно площадь каждой полоски, получающейся при таком разбиении, и суммируя площади этих полосок.

При этом придется рассмотреть два способа разбиения исходного интервала на меньшие.

1. Разбиение на интервалы производится заранее; обычно интервалы выбираются равными. Кроме того, если вычисление интеграла предполагается производить «вручную», то интервалы выбираются так, чтобы значения x, соответствующие концу каждого интервала, было возможно легче вычислять. Из этой категории методов будут рассмотрены правило трапеций и метод парабол (Симпсона).

2. Местоположение и длина интервалов определяются путем анализа; сначала ставится требование достичь наивысшей точности с заданным числом интервалов, а затем в соответствии с этим определяются их границы. Примером такого подхода является метод Гаусса.

Существует еще много других методов обоих категорий. Те из них, которые будут рассмотрены в этой главе, позволяют ясно представить общий подход к задаче численного интегрирования и оценке ошибок. В подавляющем большинстве случаев эти методы вполне применимы для практических вычислений.

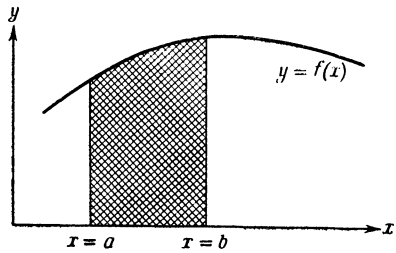
6.2. ПРАВИЛО ТРАПЕЦИЙ

Рассмотрим интеграл (6.1), который представляет собой заштрихованную площадь на рис. 6.1. Разобьем интервал интегрирования на n равных частей, каждая длиной

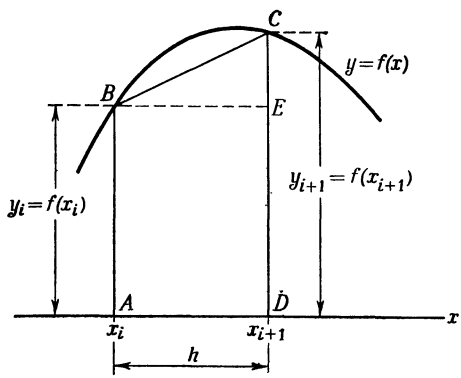
---

[1]) Интересующимся читателям рекомендуются книги A l t F., Electronic digital computers, Academic Press, New York, 1958, p. 203—205; K o p a l Z., Numerical Analysis, Wiley, New York, 1961, p. 370—386.

$h = (b - a)/n$. Рассмотрим теперь один из этих интервалов, как изображено на рис. 6.2, где масштаб по оси x сильно



Р и с. 6.1. Геометрическое представление задачи о численном интегрировании.



Р и с. 6.2. Один интервал из заштрихованной области, изображенной на рис. 6.1.
Масштаб по оси x увеличен.

увеличен. Площадь, лежащая под кривой $y = f(x)$ между x_i и x_{i+1}, равна

$$I_i = \int_{x_i}^{x_{i+1}} f(x)\,dx.$$

Но если h достаточно мало, то эту площадь без большой ошибки можно приравнять к площади трапеции $ABCD$. Если написать $y_i = f(x_i)$, то площадь прямоугольника

$ABED$ будет равна $y_i h$, а площадь треугольника BEC будет равна $\frac{1}{2}(y_{i+1} - y_i)\, h$, так что

$$I_i \approx \frac{1}{2} h (y_i + y_{i+1}). \tag{6.2}$$

Но поскольку

$$\int_a^b f(x)\,dx = \int_a^c f(x)\,dx + \int_c^b f(x)\,dx,$$

получаем

$$I = \sum_{i=0}^{n-1} I_i, \tag{6.3}$$

где $x_0 = a$ и $x_n = b$. Теперь, подставляя (6.2) в (6.3), окончательно получаем

$$I \approx \frac{h}{2}(y_0 + 2y_1 + \ldots + 2y_{n-1} + y_n). \tag{6.4}$$

Эта формула описывает хорошо известное правило трапеций для численного интегрирования; согласно этому правилу, приближенное значение интеграла (6.1) получается в виде суммы площадей n трапеций. Правило трапеций — один из простейших методов численного интегрирования. Ошибки ограничения, которые мы подробно исследуем в следующем параграфе, для этого метода больше, нежели для большинства других, но его привлекательность иногда увеличивается из-за простоты. Во всяком случае, этот метод полезен тем, что он ярко демонстрирует основной принцип построения всех приближенных формул численного интегрирования (имеется в виду тот случай, когда разбиение на интервалы произведено заранее). Сущность его состоит в том, что интервал интегрирования разбивается на множество меньших отрезков, внутри которых подинтегральная кривая $y = f(x)$ заменяется с некоторой степенью точности более простыми функциями; интегралы от них можно вычислить, используя только ординаты на концах отрезков.

6.3. ОШИБКА ОГРАНИЧЕНИЯ ДЛЯ МЕТОДА ТРАПЕЦИЙ

При использовании формулы (6.4) для вычисления определенного интеграла возникает ошибка, равная сумме площадей между кривой $y = f(x)$ и хордами, соединяющи-

ми y_i и y_{i+1} (BC на рис. 6.2). Мы оценим эту ошибку, разлагая функцию $y = f(x)$ в ряд Тейлора в точках x_i и x_{i+1}. Это разложение позволит получить уравнение исходной кривой в виде, удобном для сравнения точного значения интеграла с приближенным, вычисленным по формуле (6.4).

Рассмотрим разложение функции $y = f(x)$ в ряд Тейлора в окрестности точки $x = x_i$. Предположим, что $f(x)$ имеет столько непрерывных производных, сколько может потребоваться.

$$y = y_i + (x - x_i)\, y_i' + \frac{(x-x_i)^2}{2} y_i'' + \ldots . \qquad (6.5)$$

Аналогично, разлагая функцию в ряд в окрестности точки x_{i+1}, имеем

$$y = y_{i+1} + (x - x_i - h)\, y_{i+1}' + \frac{(x-x_i-h)^2}{2} y_{i+1}'' + \ldots . \qquad (6.6)$$

Естественно, формулы (6.5) и (6.6) обе справедливы, но ни одна из них в отдельности не позволяет получить интересующий нас результат. Поэтому возьмем среднее из обеих формул:

$$y = \frac{y_i + y_{i+1}}{2} + \frac{x - x_i}{2}(y_{i+1}' + y_i') - \frac{h}{2} y_{i+1}' + \\ + \frac{(x-x_i)^2}{4}(y_{i+1}'' + y_i'') - \frac{(x-x_i)\,h}{2} y_{i+1}'' + \frac{h^2}{4} y_{i+1}'' + \ldots .$$

Интегрируя $y\,dx$ от x_i до x_{i+1}, получаем

$$\int_{x_i}^{x_{i+1}} y\,dx = \frac{h}{2}(y_{i+1} + y_i) - \frac{h^2}{4}(y_{i+1}' - y_i') + \\ + \frac{h^3}{12}(y_{i+1}'' + y_i'') + \ldots . \qquad (6.7)$$

Это выражение представляет собой оценку истинного значения интеграла. Оценка может быть сделана как угодно точной, потому что можно взять сколь угодно большое количество членов в разложении функции в ряд Тейлора. Правило трапеций получается, если в формуле (6.7) отбросить все члены, содержащие h в степенях выше первой.

Поэтому ошибка ограничения при использовании метода трапеций равна

$$E_{T_i} = -\frac{h^2}{4}(y'_{i+1} - y'_i) + \frac{h^3}{12}(y''_{i+1} + y''_i) + \ldots . \quad (6.8)$$

Для малых h первый член гораздо больше всех остальных, поэтому можно было бы предположить, что ошибка как раз им и определяется. Заметим, однако, что если разложить $y' = df/dx$ в ряд в окрестности точки x_i и умножить на h^2, то получается

$$y'_{i+1}h^2 = y'_i h^2 + y''_i h^3 + \ldots,$$

так что члены, содержащие $y''_i h^3$, вносят вклад в величину первого члена в формуле (6.8). В действительности все члены высших порядков вносят некоторый вклад в величину первого члена.

Поэтому *предположим*, что ошибка при использовании метода трапеций выражается формулой

$$E_{T_i} \approx Kh^2(y'_{i+1} - y'_i), \quad (6.9)$$

где константу K необходимо определить. Конечно, это будет только некоторое приближение, основанное на допущении о постоянстве K; такое приближение справедливо только до тех пор, пока y'' и производные высших порядков не изменяются сильно между x_i и x_{i+1}.

Попытаемся определить K. Прежде всего заметим, что формула (6.7) справедлива для *любой* функции. Поэтому мы можем взять любую функцию, для которой ошибка ограничения при интегрировании по методу трапеций не равна нулю. Результат будет справедлив для всех функций.

Самой простой функцией было бы $y = x$, но очень легко убедиться, что для $y = x$ ошибка ограничения равна нулю; это означает, что метод трапеций дает точный результат при интегрировании линейных функций. Рассмотрим $y = x^2$. В этом случае

$$I_i = \int_{x_i}^{x_{i+1}} x^2\,dx = x_i^2 h + x_i h^2 + \frac{h^3}{3}. \quad (6.10)$$

Но из формул (6.7) и (6.8) и подстановки $y_i = x_i^2$ и $y_{i+1} = (x_i + h)^2$ следует

$$I_i = x_i^2 h + x_i h^2 + \frac{h^3}{2} + E_{T_i}. \quad (6.11)$$

Из уравнений (6.10) и (6.11) находим

$$E_{T_i} = -\frac{h^3}{6}.$$

Но так как $y' = 2x$, то из (6.9) имеем

$$E_{T_i} = 2Kh^2(x_i + h - x_i) = 2Kh^3.$$

Поэтому для K получаем

$$K = -\frac{1}{12}.$$

И окончательно

$$E_{T_i} \approx -\frac{h^2}{12}(y'_{i+1} - y'_i). \tag{6.12}$$

Полную ошибку ограничения можно оценить из соотношения

$$e_T = \sum_{i=0}^{n-1} E_{T_i} = -\frac{h^2}{12}(y'_b - y'_a), \tag{6.13}$$

где y'_b равно значению df/dx при $x = b$ и y'_a равно значению df/dx при $x = a$.

Мы предоставляем читателям произвести те же выкладки для $y = x^3$ и убедиться, что результат не зависит от конкретного выбора функции. Конечно, и в этом случае следует определять K из члена, содержащего h^3, так как формула (6.9) справедлива только в предположении, что y'' является постоянным.

Чаще ошибку ограничения для правила трапеций выражают с помощью несколько преобразованной формулы. На основании теоремы о среднем значении можно написать

$$y'_b - y'_a = (b-a)\,y''(\xi),$$

где $a < \xi < b$, так что

$$e_T \approx -\frac{h^2}{12}(b-a)\,y''(\xi). \tag{6.14}$$

Более того, можно показать[1]), что если в интервале $a \leqslant \xi \leqslant b$

$$M = \max y''(\xi),$$

[1]) См., например, Мейкон, гл. 9.

то

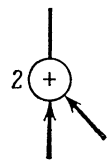
$$|e_T| \leqslant \frac{h^2}{12}(b-a)M.$$

Необходимо напомнить еще раз, что формула (6.13) дает только оценку ошибки, но не ее верхнюю границу.

6.4. ОШИБКИ ОКРУГЛЕНИЯ ПРИ ИСПОЛЬЗОВАНИИ МЕТОДА ТРАПЕЦИЙ

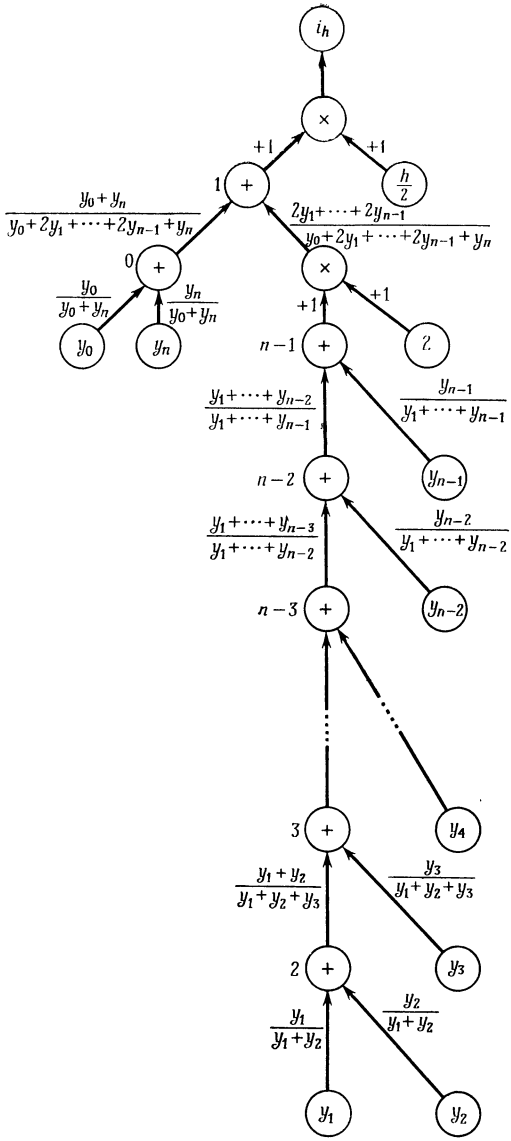
Граф вычислительного процесса для метода трапеций изображен на рис. 6.3. Предполагается, что все члены суммы, которые необходимо умножить на 2, сначала складываются, а их сумма умножается на 2. При этом не только экономится машинное время, но и уменьшается общая ошибка округления.

Пусть δ_i $(i = 0, 1, \ldots, n)$ — относительная ошибка каждой величины y_i. Пусть α_i $(i = 0, 1, \ldots, n)$ — относительная ошибка округления при выполнении каждого из $n + 1$ сложений. На рис. 6.3 сложениям присвоены порядковые номера, поставленные около кружков со знаком «плюс». Например, α_2 есть относительная ошибка округления при выполнении сложения.



Наконец, пусть μ_1 — относительная ошибка округления при выполнении умножения $(y_1 + y_2 + \ldots + y_{n-1})$ на 2, и пусть μ_2 — относительная ошибка округления при выполнении умножения $(y_0 + 2y_1 + \ldots + 2y_{n-1} + y_n)$ на $h/2$. Тогда относительная ошибка округления I_h принимает вид

$$\begin{aligned}\frac{e_R}{I_h} = \mu_2 + \alpha_1 + \alpha_0 \frac{y_0+y_n}{y_0+2y_1+\ldots+2y_{n-1}+y_n} + \\ + \delta_0 \frac{y_0}{y_0+2y_1+\ldots+2y_{n-1}+y_n} + \\ + \delta_n \frac{y_n}{y_0+2y_1+\ldots+2y_{n-1}+y_n} + \\ + \mu_1 \frac{2y_1+\ldots+2y_{n-1}}{y_0+2y_1+\ldots+2y_{n-1}+y_n} + \\ + \alpha_{n-1} \frac{2y_1+\ldots+2y_{n-1}}{y_0+2y_1+\ldots+2y_{n-1}+y_n} +\end{aligned}$$



Р и с. 6.3. Граф вычислительного процесса для интегрирования по правилу трапеций.

$$
\begin{aligned}
&+\alpha_{n-2}\frac{2(y_1+\ldots+y_{n-2})}{y_0+2y_1+\ldots+2y_{n-1}+y_n}+\\
&+\delta_{n-1}\frac{2y_{n-1}}{y_0+2y_1+\ldots+2y_{n-1}+y_n}+\\
&+\alpha_{n-3}\frac{2(y_1+\ldots+y_{n-3})}{y_0+2y_1+\ldots+2y_{n-1}+y_n}+\\
&+\delta_{n-2}\frac{2y_{n-2}}{y_0+2y_1+\ldots+2y_{n-2}+y_n}+\\
&+\ldots\\
&+\alpha_2\frac{2(y_1+y_2)}{y_0+2y_1+\ldots+2y_{n-1}+y_n}+\\
&+\delta_3\frac{2y_3}{y_0+2y_1+\ldots+2y_{n-1}+y_n}+\\
&+\delta_2\frac{2y_2}{y_0+2y_1+\ldots+2y_{n-1}+y_n}+\\
&+\delta_1\frac{2y_1}{y_0+2y_1+\ldots+2y_{n-2}+y_n}.
\end{aligned}
$$

Абсолютная ошибка равна

$$
\begin{aligned}
e_R = h\left(\delta_0\frac{y_0}{2}+\delta_1 y_1+\ldots+\delta_{n-1}y_{n-1}+\delta_n\frac{y_n}{2}\right)+\\
+h[\alpha_2(y_1+y_2)+\alpha_3(y_1+y_2+y_3)+\ldots+\\
+\alpha_{n-2}(y_1+\ldots+y_{n-2})+\alpha_{n-1}(y_1+\ldots+y_{n-1})]+\\
+\frac{h}{2}[\alpha_0(y_0+y_n)+\alpha_1(y_0+2y_1+\ldots+2y_{n-1}+y_n)+\\
+\mu_1(2y_1+\ldots+2y_{n-1})+\mu_2(y_0+2y_1+\ldots+2y_{n-1}+y_n)].
\end{aligned}
$$

Теперь предположим, что $y_i = \bar{y} + \theta_i$, где $\bar{y}$ равно среднему из всех значений y_i и $|\theta_i| \ll |\bar{y}|$, так что можно пренебречь членами, содержащими произведения $\theta_i\alpha_i$, $\theta_i\mu_i$ и $\theta_i\delta_i$. Кроме того, предположим, что

$$
\begin{aligned}
|\alpha_i| &\leqslant \varepsilon,\\
|\mu_i| &\leqslant \varepsilon,\\
|\delta_i| &\leqslant \varphi\varepsilon,
\end{aligned}
$$

где ε — относительная ошибка округления при выполнении арифметической операции (мы уже записывали ее в виде $5\cdot 10^{-t}$), а φ — некоторая постоянная, определяющая соот-

ношение между ошибками исходной информации и ошибками округления. Тогда

$$|e_R| \leqslant h|\bar{y}|\varepsilon\varphi n + h|\bar{y}|\varepsilon\sum_{j=2}^{n-1} j + h|\bar{y}|\varepsilon(3n).$$

Но так как

$$\sum_{j=1}^{m} j = \frac{m(m+1)}{2},$$

то формула для ошибки приобретает следующий вид:

$$|e_R| \leqslant \frac{h|\bar{y}|\varepsilon}{2}[n^2 + (5+2\varphi)n - 2].$$

Но ведь

$$n = \frac{b-a}{h},$$

поэтому

$$|e_R| \leqslant \frac{|\bar{y}|\varepsilon}{2}\left[\frac{(b-a)^2}{h} + (5+2\varphi)(b-a) - 2h\right].$$

Очевидно, при малом h первый член из заключенных в скобках значительно превосходит по своей величине оба остальных члена, поэтому в качестве верхней границы возможной ошибки e_R можно принять выражение

$$|e_R| \leqslant \frac{|\bar{y}|\varepsilon(b-a)^2}{2h}. \qquad (6.15)$$

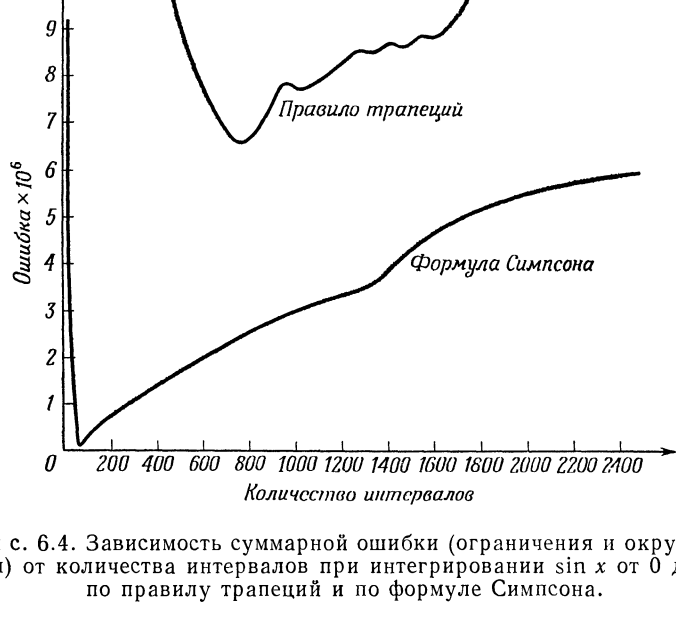
Итак, мы получили интересный результат: верхняя граница для возможной ошибки округления возрастает как $1/h$ и при некотором h превосходит ошибку ограничения, пропорциональную h^2. В действительности сама ошибка округления возрастает медленнее, а именно пропорционально h^{-p}, где $0 < p < 1$, но все же при достаточно малом h становится больше ошибки ограничения.

Это очередной поучительный пример расхождения между теорией и практикой вычислений. Теоретически можно сделать I_h сколь угодно близким к I, взяв h достаточно малым. Однако на практике ошибки округления ставят предел достижимой точности результата интегрирования.

В качестве примера рассмотрим интеграл

$$I = \int_0^{\pi} \sin x \, dx = 1.$$

На рис. 6.4 изображена зависимость общей ошибки E_I от числа интервалов n. Зависимость эта определена для



Р и с. 6.4. Зависимость суммарной ошибки (ограничения и округления) от количества интервалов при интегрировании $\sin x$ от 0 до π по правилу трапеций и по формуле Симпсона.

правила трапеций и правила Симпсона, которое будет рассмотрено в разд. 6.6. Заметьте, что с возрастанием n общая ошибка, равная сумме ошибок ограничения и округления, уменьшается до $n = 775$. После этого ошибка округления доминирует: увеличение n *приводит к росту* общей ошибки.

Наконец, заметим, что δ_i, μ_i, α_0 и α_1 не дают вклада в величину члена, зависящего от $1/h$ в формуле для e_I.

Вся ошибка, выражаемая этим членом, возникает во время суммирования

$$y_1 + y_2 + \ldots + y_{n-1},$$

так что нет смысла повышать точность вычисления самих величин y_i.

Эту ошибку можно существенно уменьшить, производя сложение с удвоенной точностью (см. разд. 3.8), в то время как остальные операции будут выполняться с обычной точностью. Такой метод называется вычислениями с *частичной удвоенной точностью.*

6.5. ЭКСТРАПОЛЯЦИОННЫЙ ПЕРЕХОД К ПРЕДЕЛУ

Чтобы найти более точное значение интеграла, можно воспользоваться сравнительно простым усовершенствованием метода трапеций.

Вспомним (см. формулу (6.14)), что для шага h ошибка ограничения составляет

$$e_T = Ch^2,$$

где

$$C = -\frac{b-a}{12} y''(\xi);\ a < \xi < b.$$

Если вторую производную от y считать постоянной, то C также является константой.

Предположим теперь, что выбрана некоторая другая величина шага разбиения $k = (b-a)/m$, причем $m \neq n$. Тогда

$$e_T = Ck^2.$$

Теперь пусть I_h — значение интеграла, вычисленное по правилу трапеций с шагом h, а I_k — значение, вычисленное с шагом k. При этом

$$I = I_h + Ch^2 \tag{6.16}$$

и

$$I = I_k + Ck^2.$$

Если вычесть эти два уравнения друг из друга, то можно определить C

$$C = \frac{I_h - I_k}{k^2 - h^2}. \tag{6.17}$$

Подставляя это значение C в (6.16), получаем

$$I = I_h + \frac{I_h - I_k}{\frac{k^2}{h^2} - 1}. \tag{6.18}$$

Вычисленное таким образом значение интеграла I является лучшим приближением, чем I_h или I_k. Если же вторая производная $y''(x)$ действительно постоянна при $a \leqslant x \leqslant b$, то ошибка ограничения в формуле (6.18) равна нулю.

Этот метод называется экстраполяционным переходом к пределу; предложен он Ричардсоном[1]).

6.6. ПРАВИЛО СИМПСОНА

В этом параграфе мы рассмотрим один из наиболее широко известных и применяемых методов численного интегрирования, а именно правило Симпсона. Этот метод аналогичен правилу трапеций в той части, что интегрирование производится путем разбиения общего интервала интегрирования на множество более мелких отрезков; однако теперь для вычисления площади над каждым из них через три последовательных ординаты разбиения проводится квадратичная парабола. Можно было бы ожидать, что аналогично тому, как правило трапеций дает точный результат при интегрировании линейных функций, правило Симпсона даст точный результат при интегрировании многочленов второго порядка; в действительности же получается несколько парадоксальный результат: формула Симпсона дает точные значения интеграла при интегрировании многочленов до третьего порядка включительно. Поэтому при всей своей простоте этот метод весьма точен, хотя формула для численного интегрирования получается ненамного сложней, чем для правила трапеций. Простота и точность правила Симпсона сильно способствуют его широкому применению при вычислениях на ЭЦВМ.

Вспомним, что количество отрезков n в случае правила трапеций определялось формулой

$$n = \frac{b-a}{h}.$$

[1]) Richardson L. F., Gaunt J. A., The deferred approach to the limit, *Trans. Roy. Soc. London*, **226 A**, 300 (1927),

Предположим теперь, что число n является четным и что

$$k = 2h. \tag{6.19}$$

Тогда

$$I_h = \frac{h}{2}(y_0 + 2y_1 + 2y_2 + \ldots + 2y_{n-1} + y_n), \tag{6.20}$$

$$I_k = h(y_0 \quad + 2y_2 + \ldots + 2y_{n-2} + y_n). \tag{6.21}$$

Уравнения (6.19), (6.20) и (6.21) можно подставить в (6.18). При этом получим

$$I = I_h + \frac{I_h - I_k}{\frac{k^2}{h^2} - 1} =$$

$$= h\left(\frac{1}{2}y_0 + y_1 + y_2 + \ldots + y_{n-2} + y_{n-1} + \frac{1}{2}y_n\right) +$$

$$+ h\left(\frac{1}{6}y_0 + \frac{1}{3}y_1 + \frac{1}{3}y_2 + \ldots + \frac{1}{3}y_{n-2} + \frac{1}{3}y_{n-1} + \frac{1}{6}y_n\right) -$$

$$- h\left(\frac{1}{3}y_0 \quad + \frac{2}{3}y_2 + \ldots + \frac{2}{3}y_{n-2} \quad + \frac{1}{3}y_n\right) =$$

$$= h\left(\frac{1}{3}y_0 + \frac{4}{3}y_1 + \frac{2}{3}y_2 + \ldots + \frac{2}{3}y_{n-2} + \frac{4}{3}y_{n-1} + \frac{1}{6}y_n\right).$$

И окончательно

$$I = \frac{h}{3}(y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + \ldots$$

$$\ldots + 2y_{n-2} + 4y_{n-1} + y_n). \tag{6.22}$$

Формула (6.22) называется формулой Симпсона. Ее можно было вывести другим путем, а именно проводя параболу через три ординаты на концах двух соседних интервалов и потом складывая получившиеся при этом площади. Читатели могут проделать эти выкладки самостоятельно и геометрически истолковать разницу между формулами интегрирования.

Ошибка ограничения при интегрировании с помощью формулы Симпсона может быть вычислена таким же способом, как и для правила трапеций в разд. 6.3. Не воспроизводя здесь выкладки, приведем окончательный результат:

$$e_T \approx -\frac{h^4}{180}(b-a) f^{\mathrm{IV}}(\xi), \quad a < \xi < b,$$

Заметим, что ошибка пропорциональна h^4, в то время как для метода трапеций ошибка была пропорциональна h^2. Это означает, что метод Симпсона соответствует ряду Тейлора в формуле (6.7) с точностью до членов *третьего* порядка включительно, а метод трапеций соответствует этому ряду только с точностью до членов первого порядка. Поэтому при интегрировании многочленов степени не выше третьей метод Симпсона дает точные значения интеграла (так как $f^{IV}(x) = 0$).

Если предположить, что четвертая производная практически постоянна, то снова можно применить экстраполяционный переход к пределу и улучшить результаты интегрирования по методу Симпсона. Вообще говоря, подобно тому, как была выведена формула (6.22), можно повышать точность, проводя через последовательные ординаты многочлены более высоких степеней. В результате получаются формулы Ньютона — Котеса[1]).

Наконец, опять без вывода, заметим, что верхняя граница возможной ошибки округления при интегрировании по правилу Симпсона пропорциональна $1/h$, как и для формулы трапеций.

На рис. 6.4 изображен также график зависимости общей ошибки от числа интервалов при интегрировании $\sin x$ от 0 до π по формуле Симпсона. На графике ясно видно, что при использовании формулы Симпсона ошибка уменьшается гораздо быстрее (h^4 по сравнению с h^2). Так как ошибки округления для обоих случаев примерно одинаковы, то при использовании формулы Симпсона ошибка округления начинает преобладать в общей ошибке гораздо раньше. Из рисунка видно, что общая ошибка возрастает при $n > 50$.

6.7. МЕТОД ГАУССА

В предыдущих параграфах рассматривались методы численного интегрирования с произвольным разбиением интервала. Фактически разбиение производилось на равные отрезки.

---

[1]) См., например, гл. 3 книги Гильдебранда: Hildebrand F. B., Introduction to numerical analysis, McGraw-Hill, 1956.

Зададимся теперь вопросом: нельзя ли уменьшить ошибку ограничения при заданном количестве интервалов, если располагать концы интервалов там, где это требуется из условий достижения наивысшей точности интегрирования? С философской точки зрения, следует ожидать улучшения: пожертвовав свободой при выборе разбиения интервала, мы вправе потребовать взамен увеличения точности. Оказывается, точность в самом деле увеличивается.

Вспомним, что рассмотренное ранее правило трапеций позволяло при двух ординатах найти точное значение интеграла от линейной функции (многочлена первого порядка). Сейчас будет показано, что при соответствующем выборе местоположения двух ординат можно получить точный результат для многочлена третьего порядка. Интуитивно очевидно, что погрешность метода тем меньше, чем выше порядок многочлена, при численном интегрировании которого получается точный результат. Мы не будем здесь доказывать этого во всей полноте.

Чтобы упростить выкладки, изменим пределы интегрирования так, чтобы они стали равными $(+1, -1)$. Для этого введем новую переменную

$$\mu = \frac{2x-(b+a)}{b-a},$$

так что

$$x = \frac{1}{2}(b-a)\mu + \frac{1}{2}(b+a).$$

Интеграл (6.1) после такой подстановки запишется в виде

$$I = \int_{-1}^{+1} \varphi(\mu)\, d\mu, \tag{6.23}$$

где

$$\varphi(\mu) = \frac{1}{2}(b-a)\cdot f\left[\frac{1}{2}(b-a)\mu + \frac{1}{2}(b+a)\right].$$

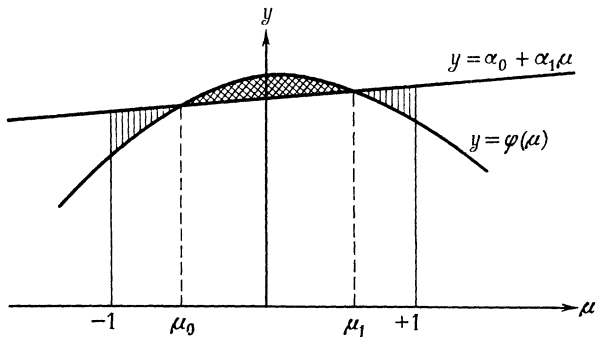
Это означает, что соответствующей заменой переменных можно свести все интегралы к виду (6.23). (Под «всеми» подразумеваются интегралы с конечными пределами интегрирования и с непрерывной подинтегральной функцией.)

Попытаемся определить, чего можно добиться при наличии всего двух ординат, т. е. в том случае, когда кривая, которой мы заменяем подинтегральную функцию, является прямой линией. Другими словами, попытаемся найти такую линейную функцию

$$y = \alpha_0 + \alpha_1 \mu,$$

для которой

$$\int_{-1}^{1} (\alpha_0 + \alpha_1 \mu)\, d\mu = \int_{-1}^{1} \varphi(\mu)\, d\mu. \tag{6.24}$$



Р и с. 6.5. Геометрическое представление метода Гаусса с двумя ординатами.

Интеграл, стоящий в левой части этого уравнения, представляет собой площадь трапеции на рис. 6.5. Пусть сумма площадей вертикально заштрихованных участков (между -1 и μ_0 и от μ_1 до $+1$) равна площади зачерненного участка (между μ_0 и μ_1). Тогда площадь трапеции в точности равна площади под кривой $y = \varphi(\mu)$. Задача заключается в нахождении такой прямой линии, для которой достигается это равенство.

Положим

$$I_G = A_0 \varphi(\mu_0) + A_1 \varphi(\mu_1), \tag{6.25}$$

где необходимо определить A_0, A_1, μ_0 и μ_1. Так как в формуле имеются четыре параметра, то естественно предположить,

что формула даст точный результат при интегрировании кубической параболы

$$\varphi(\mu) = a_0 + a_1\mu + a_2\mu^2 + a_3\mu^3.$$

Перепишем эту функцию в виде

$$\varphi(\mu) = \alpha_0 + \alpha_1\mu + (\mu - \mu_0)(\mu - \mu_1)(\beta_0 + \beta_1\mu).$$

Если α_0 и α_1 должны удовлетворять уравнению (6.24), то μ_0 и μ_1 определяется из условия

$$\int_{-1}^{1} (\mu - \mu_0)(\mu - \mu_1)(\beta_0 + \beta_1\mu)\, d\mu = 0.$$

Так как это равенство должно быть справедливо для *любых* β_0 и β_1, то необходимо потребовать выполнения двух следующих равенств:

$$\int_{-1}^{1} (\mu - \mu_0)(\mu - \mu_1)\, d\mu = 0,$$

$$\int_{-1}^{1} (\mu - \mu_0)(\mu - \mu_1)\,\mu\, d\mu = 0.$$

Выполнив интегрирование, можно записать два последних равенства в виде

$$\frac{2}{3} + 2\mu_0\mu_1 = 0,$$

$$\mu_0 + \mu_1 = 0,$$

откуда следует, что

$$\mu_1 = -\mu_0 = \frac{1}{\sqrt{3}}. \tag{6.26}$$

Теперь остается только найти A_0 и A_1 в формуле (6.25). Заметим, что

$$\int_{-1}^{1} \varphi(\mu)\, d\mu = \int_{-1}^{1} (\alpha_0 + \alpha_1\mu)\, d\mu = 2\alpha_0 \tag{6.27}$$

и из формул (6.25) и (6.26) следует, что

$$I_G = A_0(\alpha_0 + \alpha_1\mu_0) + A_1(\alpha_0 + \alpha_1\mu_1) =$$

$$= \alpha_0(A_0 + A_1) - \frac{\alpha_1}{\sqrt{3}}(A_0 - A_1).$$

Так как значение выражения в правой части последней формулы должно быть равно значению интеграла в формуле (6.27) при всех α_0 и α_1, то для A_0 и A_1 получаем систему уравнений

$$A_0 + A_1 = 2,$$
$$A_0 - A_1 = 0,$$

из которой находим

$$A_0 = A_1 = 1. \tag{6.28}$$

Уравнение (6.25) окончательно запишется в виде

$$I_G = \varphi\left(-\frac{1}{\sqrt{3}}\right) + \varphi\left(\frac{1}{\sqrt{3}}\right).$$

Это и есть формула численного интегрирования Гаусса для случая двух ординат. Ошибка ограничения равна нулю при интегрировании многочленов до третьего порядка включительно. Естественно ожидать, что при интегрировании многочленов высших степеней и прочих функций ошибка ограничения будет определяться формулой

$$e_T = K\varphi^{\mathrm{IV}}(\xi), \quad -1 < \xi < 1,$$

где

$$\int_{-1}^{1} \varphi(\mu)\, d\mu = I_G + e_T = \varphi\left(-\frac{1}{\sqrt{3}}\right) + \varphi\left(\frac{1}{\sqrt{3}}\right) + e_T. \tag{6.29}$$

Чтобы найти K, предположим, что

$$\varphi(\mu) = \mu^4.$$

При этом

$$\varphi^{\mathrm{IV}}(\mu) = 24,$$

так что

$$e_T = 24K. \tag{6.30}$$

Точное значение интеграла легко вычислить:

$$\int_{-1}^{1} \varphi(\mu)\, d\mu = \int_{-1}^{1} \mu^4\, d\mu = \frac{2}{5}.$$

С другой стороны, из формул (6.29) и (6.30)

$$\int_{-1}^{1} \varphi(\mu)\, d\mu = \left(-\frac{1}{\sqrt{3}}\right)^4 + \left(\frac{1}{\sqrt{3}}\right)^4 + 24K.$$

Поэтому

$$K = \frac{1}{135},$$

и окончательная формула для ошибки ограничения запишется в следующем виде:

$$e_T = \frac{\varphi^{IV}(\xi)}{135}, \quad -1 < \xi < 1.$$

Можно вывести гауссовы формулы численного интегрирования более высоких порядков, предусматривая большее количество ординат и вводя различные *весовые коэффициенты* A_i:

$$\int_{-1}^{1} \varphi(\mu)\, d\mu = \sum_{i=0}^{n} A_i \varphi(\mu_i). \tag{6.31}$$

В общем случае, при $(n+1)$ ординате, получается точная формула для нахождения интеграла от многочлена степени $2n+1$.

Оказывается, что значения μ_i в уравнении (6.31) являются корнями полиномов Лежандра степени n. По этой причине вышеописанный метод численного интегрирования часто называют методом Лежандра — Гаусса. Полиномы Лежандра $P_n(\mu)$ можно найти с помощью рекуррентных формул

$$\left.\begin{aligned} &P_0(\mu) = 1, \\ &P_1(\mu) = \mu, \\ &P_m(\mu) = \frac{1}{m}\left[(2m-1)\,\mu P_{m-1}(\mu) - (m-1)\,P_{m-2}(\mu)\right]. \end{aligned}\right\} \tag{6.32}$$

Например, для $m = 2$

$$P_2(\mu) = \frac{1}{2}(3\mu \cdot \mu - 1 \cdot 1) = \frac{3\mu^2}{2} - \frac{1}{2}.$$

Заметим, что корни $P_2(\mu)$ равны $\pm 1/\sqrt{3}$, как мы уже определили ранее.

Весовые коэффициенты в формуле (6.31) можно найти из следующего соотношения:

$$A_i = \frac{2}{(1-\mu_i^2)\,[P'_n(\mu_i)]^2}.$$

В качестве примера возьмем $n=2$, так что $\mu_0 = -1/\sqrt{3}$, $\mu_1 = 1/\sqrt{3}$ и $P'_2 = 3\mu$. При этом

$$A_0 = \frac{2}{\left(1-\frac{1}{3}\right)\cdot 3} = 1,$$

и совершенно аналогично $A_1 = 1$.

В общем случае ошибка ограничения определяется формулой

$$e_T = \frac{\varphi^{2n}(\xi)}{2n!}\left(\frac{2}{2n+1} - \sum_{i=0}^{n} A_i \mu_i^{2n}\right).$$

Таблица μ_i и A_i для $n=2, \ldots, 6$ приведена в приложении. Заметим, что μ_i расположены симметрично относительно начала координат и что коэффициенты A_k одинаковы для μ_k и для $-\mu_k$. Более подробная таблица вплоть до $n = 48$ приведена в приложении к книге В. И. Крылова «Приближенное вычисление интегралов».

Итак, метод Гаусса позволяет достичь большей точности, нежели формула Симпсона при том же количестве ординат, но зато точки, где следует определять эти ординаты, полностью определены и совершенно не зависят от произвола программиста. Как и во многих других случаях, при численном интегрировании приходится делать выбор между простотой формулы Симпсона и возможной экономией машинного времени при использовании метода Гаусса. На практике чаще используется метод Симпсона.

6.8. ЧИСЛЕННЫЕ ПРИМЕРЫ И СРАВНЕНИЕ МЕТОДОВ

Чтобы проиллюстрировать возможности трех разобранных нами методов численного интегрирования, а также сравнить их точность, рассмотрим следующий интеграл:

$$I = \int_{-2}^{2} e^{-\frac{x^2}{2}}\,dx.$$

Это уже знакомый нам интеграл вероятностей из практического примера 6 (разд. 4.4), только на этот раз перед ним нет коэффициента $1/\sqrt{2\pi}$. Точное значение интеграла до пятой значащей цифры равно 2.3925.

Для начала воспользуемся методом трапеций с довольно широким интервалом $h = 1.0$. При этом

$$I_T = \frac{1.0}{2}(e^{-2.0} + 2e^{-0.5} + 2e^{0.0} + 2e^{-0.5} + e^{-2.0}) =$$

$$= 0.5\,(0.13534 + 2 \cdot 0.60653 + 2 \cdot 1.00000 + 2 \cdot 0.60653 + 0.13534) =$$

$$= 2.3484$$

$$\varepsilon = 0.0441$$

Метод Симпсона при $h = 1.0$ дает

$$I_S = \frac{1.0}{3}(e^{-2.0} + 4e^{-0.5} + 2e^{0.0} + 4e^{-0.5} + e^{-2.0}) =$$

$$= 0.33333\,(0.13534 + 4 \cdot 0.60653 + 2 \cdot 1.00000 + 4 \cdot 0.60653 + 0.13534) =$$

$$= 2.3743$$

$$\varepsilon = 0.0182$$

При использовании метода Гаусса с двумя ординатами приходится вычислять следующее выражение:

$$I_G = A_0\varphi(\mu_0) + A_1\varphi(\mu_1),$$

где

$$\varphi(\mu) = \frac{b-a}{2} f\left(\frac{b-a}{2}\mu + \frac{b+a}{2}\right) = 2 \cdot e^{-(2\mu)^2/2}.$$

Подставляя численные значения, получаем

$$I_G = 2 \cdot 2 \cdot e^{\left[\frac{-(2 \cdot 0.57735)^2}{2}\right]} + e^{\left[\frac{-(2 \cdot 0.57735)^2}{2}\right]} =$$

$$= 2.0536$$

$$\varepsilon = 0.3389$$

Повторяя эти же вычисления при $h = 0.5$ и при $h = 0.25$, а также при трех, четырех, пяти и шести точ-

ках в методе Гаусса, получаем результаты, приведенные в табл. 6.1.

Таблица 6.1

Правило трапеций	h	I_T	ε
	1,0	2,3484	0,0441
	0,5	2,3813	0,0112
	0,25	2,3898	0,0027
Правило Симпсона	h	I_S	ε
	1,0	2,3743	0,0182
	0,5	2,3923	0,0002
	0,25	2,3926	0,0001
Метод Гаусса	Число точек	I_G	ε
	2	2,0536	0,3389
	3	2,4471	—0,0546
	4	2,3859	0,0066
	5	2,3931	—0,0006
	6	2,3925	0,0000

Конечно, нельзя выводить сколько-нибудь общие заключения из рассмотрения одного простого примера, но оказывается, что нижеследующие выводы имеют довольно общий характер.

1. Формула Симпсона при n ординатах дает примерно ту же степень точности, что и формула трапеций при $2n$ ординатах.

2. Метод Гаусса при n ординатах дает примерно ту же степень точности, что и формула Симпсона при $2n$ ординатах.

Хотя ни одно из этих утверждений не является *совершенно* точным, все же можно показать, что они *приблизительно* справедливы.

3. Для достижения той же степени точности при использовании формулы Симпсона приходится производить примерно вдвое меньше вычислений, чем по формуле трапеций, которая требует вдвое большего количества ординат.

4. Для достижения той же степени точности при использовании метода Гаусса приходится производить примерно вдвое меньше вычислений, чем по формуле Симпсона благодаря вдвое меньшему количеству ординат.

При «ручных» вычислениях метод Гаусса несколько неудобен тем, что абсциссы заданы заранее и выражаются, как правило, такими числами, с которыми очень неудобно работать. Например, гораздо легче искать по таблицам $e^{-0.5}$, нежели $e^{-0.16667}$. Однако при использовании ЭЦВМ эти соображения не играют никакой роли.

Экономия времени при использовании метода Гаусса имеет свою оборотную сторону. Дело в том, что если приходится заново вычислять тот же интеграл с бóльшим количеством точек, то нельзя использовать ранее вычисленные ординаты, так как они находятся в неподходящих местах. В случае формулы Симпсона ранее вычисленные ординаты можно использовать снова (см. упражнение 14).

Для интегрирования при экспериментальных данных метод Гаусса можно использовать только тогда, когда абсциссы заранее правильно расположены; это случается очень редко. Формулой Симпсона можно пользоваться при условии, что интервалы разбиения одинаковы. Если же интервалы разбиения случайны, то для интегрирования можно применить лишь формулу трапеций. Формула Симпсона обеспечивает достаточную точность при умеренном количестве ординат, проста и удобна, так как позволяет легко измельчать разбиение интервала и заново вычислять интеграл. Поэтому неудивительно, что она получила широкое распространение в практических вычислениях.

6.9. ПРАКТИЧЕСКИЙ ПРИМЕР 8: СВЕТИМОСТЬ ЭЛЕКТРИЧЕСКОЙ ЛАМПОЧКИ

В этом практическом примере будут использованы описанные методы численного интегрирования, а также проиллюстрированы некоторые полезные приемы программирования.

Абсолютно черное тело (идеальный излучатель) излучает энергию пропорционально четвертой степени абсолютной температуры. Этот процесс описывается уравнением Стефана — Больцмана

$$E = 36.9 \cdot 10^{-12} T^4,$$

где E — мощность излучения $вт/см^2$,
T — температура в градусах Кельвина.

Нас интересует та часть общей энергии, которая заключена в видимом спектре частот. Здесь мы предположим, что видимый спектр соответствует длинам волн от $4 \cdot 10^{-5}$ до $7 \cdot 10^{-5}$ *см*. Эту часть энергии можно найти, интегрируя между вышеуказанными пределами уравнение Планка

$$E_{\text{видимая}} = \int_{4 \cdot 10^{-5}}^{7 \cdot 10^{-5}} \frac{2.39 \cdot 10^{-11}\, dx}{x^5 \left(e^{\frac{1.432}{Tx}} - 1\right)},$$

где x — длина волны в *см*.

E и T означают то же, что и в предыдущей формуле.

Светимостью мы назовем отношение энергии, заключенной в видимом спектре, к общей энергии излучения. Если умножить это отношение на 100, чтобы получить окончательный результат в процентах и перемножить коэффициенты, то задача сведется к вычислению интеграла

$$\text{EFF} = \frac{64.77 \int_{4 \cdot 10^{-5}}^{7 \cdot 10^{-5}} \frac{dx}{x^5 \left(e^{\frac{1.432}{Tx}} - 1\right)}}{T^4}.$$

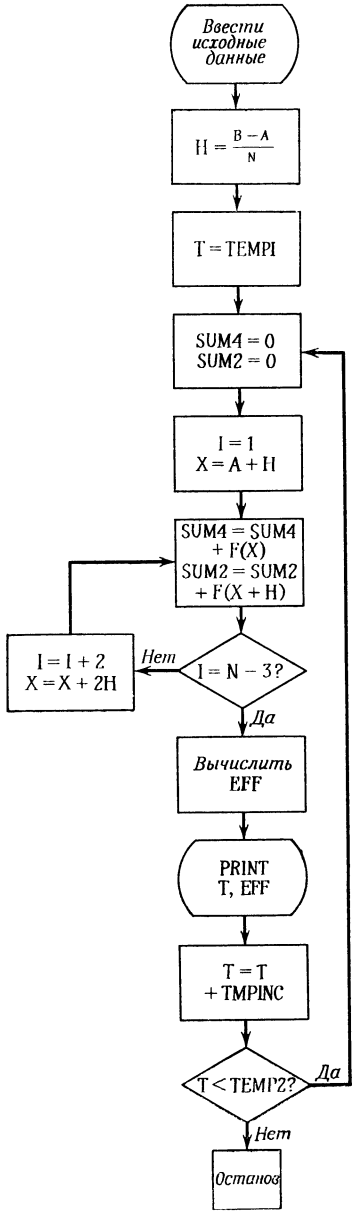
Необходимо написать программу, с помощью которой можно было бы вычислить EFF для некоторой последовательности температур, начиная с начальной температуры TEMP1 до конечной TEMP2 с шагом TMPINC. С перфокарт необходимо также считывать границы видимого спектра A и B, чтобы эти величины тоже можно было при желании варьировать. Число отрезков разбиения при интегрировании задается с помощью целой переменной N, значение которой также следует прочитать с перфокарты. Ниже мы вернемся к вопросу о том, как велико должно быть N для достижения достаточной точности.

Блок-схема программы изображена на рис. 6.6. Работа программы начинается с ввода шести исходных значений, после чего производятся два действия, которые в дальнейшем не повторяются: вычисляется H — интервал разбиения, а переменной T присваивается исходное значение ТЕМП1. После этого начинается собственно интегрирование. В программе предусмотрено вычисление двух сумм при интегрировании. Первая из них, обозначаемая SUM4, представляет собой сумму тех ординат, перед которыми в формуле Симпсона стоит коэффициент 4; вторая, обозначаемая SUM2, является суммой ординат с коэффициентом 2. Поскольку по ходу программы потребуется вычислять значение подинтегральной функции в первой внутренней точке X + H (это значение будет прибавлено к SUM4) и во второй внутренней точке (это значение будет прибавлено к SUM2), то переменной X присваивается соответствующее начальное значение.

Теперь мы сталкиваемся с некоторой трудностью. Для вычисления сумм необходимо пройти все внутренние значения X и затем остановиться. Если определять окончание цикла, сравнивая X и B — H, т. е. проверять, находится ли X внутри интервала интегрирования, то возникают два неприятных обстоятельства. Во-первых, в формуле Симпсона количество ординат с коэффициентом 4 на единицу больше, нежели с коэффициентом 2. Поэтому нам придется остановиться, не доходя до конца интервала интегрирования, и прибавить к сумме значение ординаты в последней внутренней точке уже после окончания цикла. Во-вторых, если присвоить X исходное значение A + H и многократно прибавлять по 2H, то ошибки округления будут накапливаться и в общем случае нам не удастся получить *точное* равенство X и B — 3H.

Одно из возможных решений заключается в подсчете количества пройденных внутренних точек с помощью целой переменной I. Перед началом цикла интегрирования переменной I присваивается значение 1. (Легко определить значение I, при достижении которого следует выйти из цикла.)

Таким образом, мы вычисляем подинтегральную функцию для двух очередных внутренних точек, прибавляем эти значения к соответствующим суммам и проверяем, не пора ли окончить цикл. Если нет, то мы увеличиваем



Р и с. 6.6. Блок-схема программы для вычисления светимости
Интегрирование производится по формуле Симпсона (практический пример 8)

I и X и повторяем вычисления для следующей пары внутренних точек. Если цикл окончился, то необходимо перейти к вычислению EFF. В момент окончания цикла значение переменной SUM 4 равно сумме всех значений ординат, которые необходимо умножить на 4, за исключением ординаты в точке B — H, которая должна быть вычислена дополнительно. Значение переменной SUM 2 равно сумме всех значений ординат, которые необходимо умножить на 2. Кроме того, остается добавить значения ординат в точках A и B. Таким образом, вычисление величины EFF производится с помощью длинного арифметического оператора, где вычисляются ординаты для A, B и B — H, производятся умножения на 4 и на 2, а также на величину H/3 и на $64.77/T^4$.

После печатания результатов T увеличивается на величину TEMPINC и сравнивается с TEMP 2. Если новое значение не превосходит TEMP 2, т. е. если вычисления произведены еще не для всех температур, то мы снова переходим к вычислению интеграла. Если же светимость уже вычислена для наивысшей из заданных температур, то вычисления нужно прекратить.

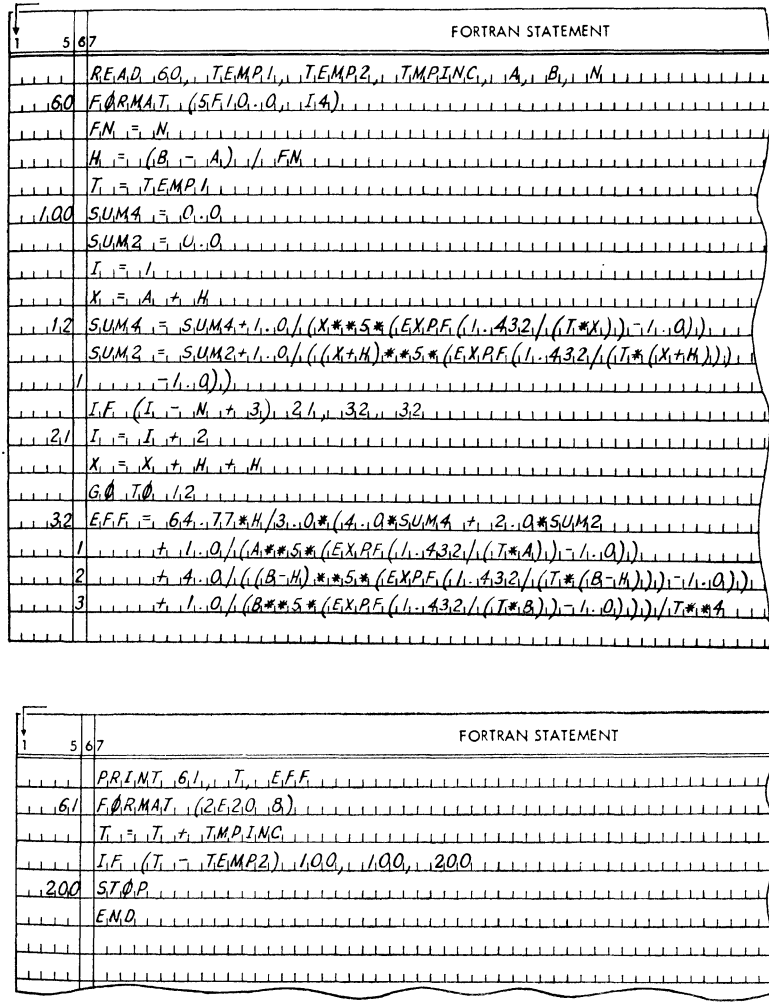
Сама программа приведена на рис. 6.7. Предоставляем читателям самим внимательно прочесть ее, чтобы убедиться, что она в самом деле производит вычисления, графически изображенные на блок-схеме.

Ошибку ограничения можно было бы оценить по ее верхней границе (см. разд. 6.6), но практически это вычисление требует гораздо больше труда, чем стоит затрачивать в данном случае. Во-первых, нужно было бы найти четвертую производную от

$$\frac{1}{x^5\left(e^{\frac{1.432}{Tx}}-1\right)},$$

что само по себе является довольно трудоемкой задачей, хотя и вполне осуществимо. Затем нужно было бы каким-нибудь образом определить, где эта четвертая производная имеет максимум.

Гораздо удобнее воспользоваться помощью самой ЭЦВМ. Аналогично (6.18) в случае правила трапеций, следующая формула описывает экстраполяционный переход к пределу



Р и с. 6.7. Программа для вычисления светимости.
Интегрирование производится по формуле Симпсона (практический пример 8).

для формулы Симпсона

$$I = I_h + \frac{I_h - I_k}{1 - \left(\frac{k}{h}\right)^4}.$$

Это означает, что если, например, вычислить интеграл сначала при N = 10, а затем при N = 20, то можно оценить истинное значение интеграла. Кроме того, из величины разности между двумя значениями интеграла можно решить, каково должно быть N для достижения приемлемой точности результата.

Такие вычисления были проделаны по этой программе для T = 3500° K. При 10 отрезках вычисленное значение

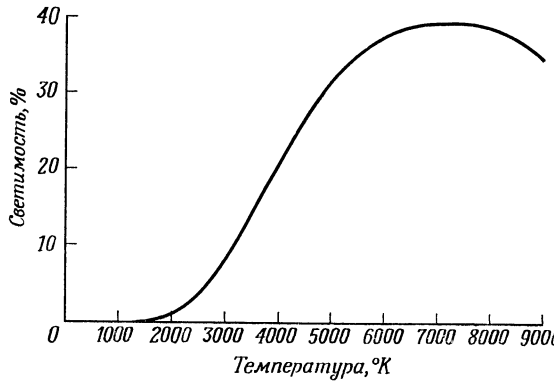


Рис. 6.8. Зависимость светимости от температуры (°K) для черного тела (практический пример 8).

светимости было равно 14.512723%, а при 20 отрезках — 14.512664%. Разница двух значений настолько мала, что мы немедленно прекращаем дальнейшую погоню за точностью.

Наконец, хотя это уже и не имеет прямого отношения к численному анализу, интересно построить график зависимости светимости от температуры, как показано на рис. 6.8. Из графика следует, что видимая часть общей энергии пренебрежимо мала при температурах, меньших 2000° C, что при температуре плавления вольфрама (3600° K) только около 15% общей энергии излучения находится в види-

мой области спектра и что кривая имеет широкий максимум около 7000° K. Соображения этого рода определяют верхний предел коэффициента полезного действия осветительных ламп накаливания. Интересно было бы также сделать некоторые предположения, относящиеся, например, к стоимости электроэнергии, времени жизни нити накаливания в зависимости от температуры и стоимости лампочки, и вычислить температуру, при которой «полная стоимость» света, включая электроэнергию и лампочку, была бы минимальной.

Упражнения

1. Вычислите точное значение интеграла $I = \int_0^2 f(x)\,dx$, а также приближенные значения по формуле трапеций при $h = 1.0$ и по формуле Симпсона с $h = 1.0$. В качестве $f(x)$ возьмите следующие функции:

а. $f(x) = 1 + x$,

б. $f(x) = 1 + x^2$,

в. $f(x) = 1 + x^3$,

г. $f(x) = 1 + x^4$.

2.. Дайте доказательство и геометрическое истолкование следующего факта: если в интервале $a \leqslant x \leqslant b$ $f''(x) > 0$, то приближенное значение интеграла $\int_a^b f(x)\,dx$, вычисленное по формуле трапеций, будет всегда больше, нежели его точное значение. (Говорят, что подобные функции $f(x)$ направлены выпуклостью книзу.)

3. Рассмотрим интеграл $\int_{-1}^{1} (1 + x + x^2 + x^3)\,dx$. Покажите, что формула Симпсона с $h = 1$ и метод Гаусса с двумя ординатами позволяют вычислить точное значение интеграла, хотя при использовании метода Гаусса требуется одной ординатой меньше по сравнению с формулой Симпсона.

4. Рассмотрим интеграл $\int_0^1 \sin x\,dx = 0.45970$. Покажите, что приближенное значение, вычисленное по формуле Симпсона при $h = 0.5$, поразительно близко к истинному значению (с точностью примерно 1/3000). Объясните «качественно», почему получается такое близкое совпадение.

5. В противоположность упражнению 4, рассмотрим часто встречающийся интеграл $\int_{0.1}^{1} dx/x = 2.30259$. Ниже приведены приближенные значения интеграла, вычисленные по формуле Симпсона при трех различных разбиениях, а также ошибки для каждого случая

n	h	I_S	Ошибка
2	0.45	3.3500	—1.0474
4	0.225	2.4079	—0.1053
8	0.1125	2.3206	—0.0180

Объясните разницу между результатами этого и предыдущего упражнения.

6. Примените экстраполяционный переход к пределу, чтобы найти лучшее приближение к точному значению интеграла $\int_{0.1}^{1} dx/x$. Используйте две последние строки предыдущей таблицы. По виду уравнения (6.18) можно было бы предположить, что окончательный результат будет точным. Почему это не так?

7. Полный эллиптический интеграл первого рода определяется формулой

$$K(\theta) = \int_0^{\frac{\pi}{2}} \frac{d\varphi}{\sqrt{1-\sin^2\theta\,\sin^2\varphi}}.$$

Вычислите $K(30°)$, воспользовавшись формулой Симпсона с четырьмя отрезками. Точный результат до четвертого знака после запятой равен 1.6858. Вычислите также $K(85°)$, снова воспользовавшись формулой Симпсона с четырьмя отрезками. На этот раз точный результат до третьего знака после запятой равен 3.832. Почему $K(85°)$ получилось с такой большой ошибкой, в то время как $K(30°)$ было вычислено довольно точно?

8. Рассмотрим следующий интеграл [1])

$$\int_{-1}^{1} \frac{x^7\sqrt{1-x^2}\,dx}{(2-x)^{13/2}}.$$

[1]) Scarborough J. B., Numerical mathematical analysis, The Johns Hopkins Press, 1950.

Напишите программу для вычисления этого интеграла по формуле Симпсона. Программу составьте так, чтобы значение шага разбиения h можно было вводить с перфокарты. Вычислите значение интеграла при $h = 0.25$, затем 0.1, 0.05, 0.02 и 0.01. Объясните несколько неожиданное поведение окончательного результата при уменьшении h. (Заметим, что объяснить поведение окончательного результата, возможно, будет легче, если предусмотреть в программе печать ординат функции.)

* **9.** Рассмотрим интеграл

$$I = \int_0^{10} e^{-x}\,dx = 0.999955$$

Вычислите этот интеграл с помощью:

а. Метода Гаусса с 6 точками;

б. Правила трапеций с 10 отрезками;

в. Правила Симпсона с 10 отрезками;

г. Правила прямоугольников из упражнения 18 с 10 отрезками.

Сравните результаты вычислений. В какую сторону лучше двигаться при интегрировании: от 0 к 10 или от 10 к 0? Почему?

***10.** Напишите программу вычислений из упражнения 9.

11. Обобщите программу из упражнения 10 для формулы Симпсона так, чтобы в нее можно было ввести с перфокарты значения a, b и n и вычислить требуемое значение h.

12. К программе из упражнения 11 добавьте проверку, является ли n четным. При нечетном n вычисление интеграла производить не следует. (Указание: при делении целых чисел дробная часть результата отбрасывается; если N является нечетным, то (N/2) *2 не равно N.)

13. Усовершенствуйте программу, приведенную на рис. 6.7, так, чтобы при ее работе вычислялись и печатались значения светимости, полученные по формуле трапеций и по формуле Симпсона. Ординаты должны быть вычислены только один раз.

14. Предположите, что вычисления по программе, приведенной на рис. 6.7, уже закончены и теперь необходимо вычислить тот же интеграл, но для вдвое более мелкого разбиения. Все ординаты, рассчитанные ранее и просуммированные в ячейках SUM4 и SUM2, понадобятся при новом вычислении интеграла, так как это как раз те ординаты, сумму которых необходимо умножить на 2. Кроме того, придется рассчитать ординаты в центрах каждого из отрезков предыдущего разбиения. Видоизмените программу таким образом, чтобы после вычисления интеграла с разбиением, определяемым величинами A, B и N, те же расчеты повторились с удвоенным количеством точек; при этом ни одна ордината не должна вычисляться дважды. Имея два приближенных значения интеграла, используйте экстраполяционный переход к пределу, чтобы получить еще более точное значение.

15. Напишите программу, которая предусматривала бы ввод значений a_0, a_1, a_2, a_3, a_4, a_5 и a_6 с одной перфокарты и значений

a, b и n—с др угой, вычисление и печать значения интеграла

$$I = \int_a^b (a_0 + a_1 x + a_2 x^2 + \ldots + a_6 x^6)\, dx.$$

Для вычислений используйте метод Симпсона с n отрезками.

16. Видоизмените программу, приведенную на рис. 6.7, предусмотрев в ней определение максимума светимости, если он, максимум, попадает в заданную для вычислений область температур. Если существуют такие три последовательных значения температуры, что EFF $(T_1) <$ EFF $(T_2) >$ EFF (T_3), то необходимо их напечатать вместе с соответствующими светимостями. Если три таких значения найти не удается, то вместо всех этих шести величин необходимо напечатать нули.

17. Предположите, что заданы три значения $x = -h$, 0 и h и три соответствующих значения $y = y_0$, y_1 и y_2. Подставьте эти три пары значений в общее уравнение параболы $y = a + bx + cx^2$ и решите три получившихся уравнения относительно a, b и c. Используя вычисленные таким образом a, b и c, проинтегрируйте уравнение параболы в пределах от $-h$ до h и покажите, что результат будет равен

$$\int_{-h}^{h} (a + bx + cx^2)\, dx = \frac{h}{3}(y_0 + 4y_1 + y_2).$$

Таким способом обычно выводится формула Симпсона для численного интегрирования в учебниках по математическому анализу.

18. Выведите формулу численного интегрирования для

$$I = \int_a^b f(x)\, dx,$$

разделив интервал интегрирования на n равных отрезков длиной

$$h = \frac{b-a}{n}.$$

В качестве приближенного значения площади для каждого интервала примите площадь прямоугольника, высота которого равна значению $f(x)$ на *левом* краю интервала (см. схему).

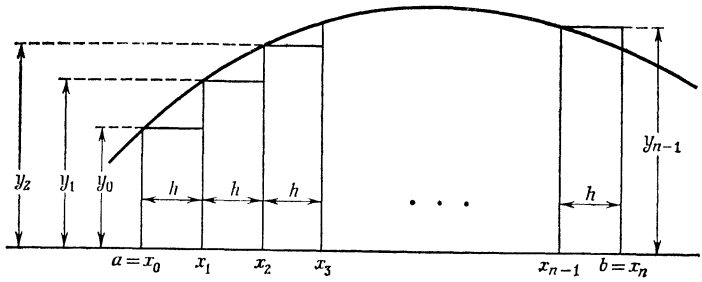
19. Покажите, что ошибка ограничения для формулы, выведенной в упражнении 18, равна

$$E_T = \left[\frac{b-a}{2} y'(\xi)\right] h,$$

где $a < \xi < b$.

20. Примените экстраполяционный переход к пределу, чтобы с помощью формулы численного интегрирования, выведенной

в упражнении 18, и формулы ошибки ограничения, выведенной в упражнении 19, получить новую формулу численного интегри-



рования

$$\int_a^b f(x)\,dx \approx h\,(y_{1/2}+y_{3/2}+y_{5/2}+\ldots+y_{n-1/2}),$$

где

$$y_{1/2}=f\left(a+\frac{h}{2}\right),$$

$$y_{3/2}=f\left(a+\frac{3h}{2}\right),$$

$$y_{i+1/2}=f\left(a+\left(i+\frac{1}{2}\right)h\right);\quad i=0,\,1,\,2,\,\ldots\,n-1.$$

Объясните формулу геометрически.

21. Ошибка ограничения при численном интегрировании по формуле из упражнения 20 равна

$$E_T=\left[\frac{(b-a)\,y''(\xi)}{24}\right]h^2,\quad a<\xi<b.$$

Сравните ее с ошибкой ограничения при интегрировании по формуле трапеций. Объясните, почему эти ошибки имеют одинаковый порядок.

22. Покажите, что формула численного интегрирования по методу Гаусса, которая дает точный результат только для линейной функции, имеет следующий вид:

$$\int_0^1 \varphi(u)\,du \approx 2\varphi(0).$$

23. Предположим, что при интегрировании с помощью метода трапеций $n=3m$, т. е. количество отрезков разбиения кратно 3. Напишите формулы численного интегрирования по формуле трапеций для случая, когда длина отрезка разбиения равна h, и для случая, когда она равна $k=3h$. Используйте экстраполяционный переход к пределу и выведите новую формулу численного интегрирования

$$I=\frac{3}{8}h\,(y_0+3y_1+3y_2+2y_3+3y_4+3y_5+2y_6+3y_7+\ldots$$
$$\ldots+3y_{n-4}+2y_{n-3}+3y_{n-2}+3y_{n-1}+y_n).$$

Эта формула иногда называется *правилом трех восьмых*. Ее можно было бы вывести, приближая интеграл от исходной функции на трех соседних отрезках интегралом от кубической параболы, проведенной через четыре последовательные ординаты.

Ошибка ограничения для этой формулы имеет вид

$$E_T=-\frac{(b-a)\,y^{\mathrm{IV}}(\xi)}{80}h^4;\quad a<\xi<b.$$

Сравните ее с ошибкой ограничения при интегрировании по правилу Симпсона. Обладает ли формула трех восьмых какими-нибудь преимуществами перед формулой Симпсона?

24. Предположим, что при интегрировании по формуле Симпсона $n=4m$. Напишите формулы численного интегрирования по правилу Симпсона для случаев, когда величина отрезка разбиения равна h и $k=2h$. Используйте экстраполяционный переход к пределу для вывода формулы

$$I=\frac{2h}{45}(7y_0+32y_1+12y_2+32y_3+14y_4+32y_5+\ldots$$
$$\ldots+14y_{n-4}+32y_{n-3}+12y_{n-2}+32y_{n-1}+7y_n).$$

Это *четвертая формула Ньютона — Котеса* (первыми тремя являются формула трапеций, формула Симпсона и формула трех восьмых). Соответствующая ошибка ограничения записывается в виде

$$E_T\approx Kh^6.$$

25. Рассмотрим несобственный интеграл

$$I=\int_0^\infty f(x)\,dx,$$

где $f(x)$ представляется в виде

$$f(x)=e^{-x}\varphi(x).$$

Покажите, что если $\varphi(x)$ является многочленом степени не выше 3, то I можно точно вычислить по формуле

$$I=A_0\varphi(x_0)+A_1\varphi(x_1),$$

где

$$x_0 = 2 - \sqrt{2},$$
$$x_1 = 2 + \sqrt{2},$$
$$A_0 = (2+\sqrt{2})/4,$$
$$A_1 = (2-\sqrt{2})/4.$$

Это *формула Лагерра — Гаусса* второго порядка. Названием своим она обязана тому факту, что x_0 и x_1 являются корнями полинома Лагерра второго порядка.

Читателям, интересующимся общим выводом и таблицами абсцисс и весовых коэффициентов для формул Лагерра — Гаусса более высоких порядков, рекомендуем книги Гильдебранда (разд. 8.6) и Крылова (разд. 7.5 и приложение С).

26. Покажите, что если $f(x)$ является многочленом степени не выше 3, то

$$\int_{-2}^{\infty} e^{-x} f(x)\, dx = \frac{e^2}{4}\left[(2+\sqrt{2})\, f(-\sqrt{2}) + (2-\sqrt{2})\, f(\sqrt{2})\right].$$

(Указание: заменой переменных сведите задачу к виду, рассмотренному в упражнении 25.)

27. Рассмотрим несобственный интеграл

$$I = \int_{-\infty}^{\infty} f(x)\, dx,$$

где $f(x)$ представляется в виде

$$f(x) = e^{-x^2}\varphi(x).$$

Покажите, что если $\varphi(x)$ является многочленом степени не выше 3, то точное значение интеграла можно вычислить по формуле

$$I = \frac{\sqrt{\pi}}{2}\left[\varphi\left(-\frac{\sqrt{2}}{2}\right) + \varphi\left(\frac{\sqrt{2}}{2}\right)\right].$$

Это *формула Эрмита — Гаусса* второго порядка. Своим названием она обязана тому факту, что абсциссы, где вычисляется φ, являются корнями полинома Эрмита второго порядка.

Более подробные сведения можно найти в книгах Гильдебранда и Крылова.

Указание: вспомните, что

$$\int_{-\infty}^{\infty} e^{-x^2}\, dx = \sqrt{\pi}.$$

28. Используйте дважды метод трапеций и выведите формулу

$$\int_{x_0}^{x_n}\int_{y_0}^{y_m} f(x, y)\,dx\,dy =$$

$$= \frac{hk}{4}[f_{0,0}+f_{n,0}+f_{0,m}+f_{n,m}+$$

$$+2(f_{1,0}+f_{2,0}+\ldots+f_{n-1,0}+f_{0,1}+f_{0,2}+\ldots+f_{0,m-1}+$$
$$+f_{n,1}+f_{n,2}+\ldots+f_{n,m-1}+f_{1,m}+f_{2,m}+\ldots+f_{n-1,m})+$$
$$+4(f_{1,1}+f_{2,1}+\ldots+f_{n-1,1}+$$
$$+f_{1,2}+f_{2,2}+\ldots+f_{n-1,2}+$$
$$\ldots\ldots\ldots\ldots\ldots$$
$$+f_{1,m-1}+f_{2,m-1}+\ldots+f_{n-1,m-1})],$$

где

$$f_{i,j}=f(x_i, y_j),$$
$$h=x_i-x_{i-1},$$
$$k=y_j-y_{j-1}.$$

С помощью этой формулы можно производить вычисление двойных интегралов в прямоугольной области. Можно получить и другие формулы для вычисления двойных интегралов, применяя дважды соответствующие правила численного интегрирования (например, правило Симпсона).

***29.** Инженер желает получить «наилучший» средний отсчет измерительного прибора на протяжении часа. Через сколько минут после начала измерений необходимо снимать показания прибора, если

на протяжении часа можно снять два отсчета;
на протяжении часа можно снять три отсчета;
на протяжении часа можно снять четыре отсчета.

Ответ следует округлить до ближайшего целого.

30. Предположим, что имеются экспериментальные результаты, согласно которым y является некоторой (неизвестной) функцией от x.

x	y	x	y
1.0	1.00	2.2	5.12
1.2	1.82	2.4	6.38
1.4	2.08	2.6	6.98
1.6	3.18	2.8	8.22
1.8	3.52	3.0	9.00
2.0	4.70		

Вычислите приближенное значение интеграла $\int_1^3 y\,dx$, воспользовавшись следующими методами:

а. Правилом трапеций,

б. Формулой Симпсона,

в. Правилом трапеций для интервалов от 1.0 до 1.2 и от 2.8 до 3.0 и формулой Симпсона для интервала от 1.2 до 2.8.

Если о характере зависимости y от x больше ничего неизвестно, можно ли сказать, какой результат является «наиболее точным»?

31. Имеются некоторые экспериментальные данные, согласно которым y является некоторой (неизвестной) функцией от x.

x	y	x	y
20	0.21	100	0.43
25	0.30	120	0.37
30	0.37	140	0.33
40	0.45	160	0.29
50	0.49	180	0.25
60	0.50	200	0.19
70	0.49	250	0.13
80	0.47	300	0.08
90	0.45	400	0.04

Вычислите приближенное значение интеграла $\int_{20}^{400} y\,dx$, воспользовавшись формулой трапеций.

ГЛАВА

7

ПЕРЕМЕННЫЕ С ИНДЕКСАМИ И ОПЕРАТОР DO

Приемы программирования, разобранные в предыдущих главах, позволяют производить разнообразные вычисления на ЭЦВМ, но страдают отсутствием достаточно мощных средств при решении определенного круга распространенных задач. Дело в том, что очень часто при вычислениях приходится обрабатывать обширные массивы информации, например коэффициенты системы алгебраических уравнений и т. п. В этой главе мы рассмотрим употребление переменных с индексами, с помощью которых можно обозначать целый массив чисел одним общим наименованием, а также изучим оператор DO, сильно упрощающий обработку таких массивов.

7.1. ОПРЕДЕЛЕНИЯ

Переменные с индексами позволяют представить большое количество величин одним общим наименованием. Каждая отдельная величина определяется индексом (или индексами), стоящим в скобках после наименования переменной. Полная система таких величин называется *массивом*, а каждая отдельная величина называется *элементом массива*. Переменная с индексами может иметь один, два или три индекса и тогда она представляет соответственно одно-, дву- или трехмерный массив. Следует обратить внимание, что термин «одномерный» и т. д. относится к количеству *индексов*, но отнюдь не к количеству элементов: одномерный массив может содержать большое их количество, в то время как трехмерный массив в принципе может состоять из одного единственного элемента.

Первый элемент одномерного массива — это элемент номер 1, второй элемент — это элемент номер 2, и т. д.

до конца, т. е. пока не будут перенумерованы все элементы. В обычных математических обозначениях можно, например, написать $X_1, X_2, X_3, \ldots, X_{19}, X_{20}$. При записи той же последовательности чисел для программирования на ФОРТРАНе следует написать X (1), X (2), X (3), ..., X (19), X (20).

Двумерный массив можно представить себе в виде матрицы из горизонтальных строк и вертикальных столбцов. При этом первый из двух индексов определяет номер строки (он изменяется от 1 до числа строк), второй — номер столбца (изменяющийся от 1 до количества столбцов). Например, массив, состоящий из двух строк и трех столбцов, в обычной математической записи выглядит в виде

$$A_{1,1} \quad A_{1,2} \quad A_{1,3}$$
$$A_{2,1} \quad A_{2,2} \quad A_{2,3}$$

В обозначениях ФОРТРАНа он будет записан следующим образом: A (1,1), A (1,2), A (1,3), A (2,1), A (2,2), A (2,3). Заметим, что индексы отделяются друг от друга запятыми, и то же самое имеет место при записи трехмерного массива.

При желании можно представить себе трехмерный массив в виде набора плоскостей, каждая из которых состоит из строк и столбцов. Такое толкование не является единственно возможным, в приложениях может оказаться более удобной другая геометрическая интерпретация.

Наименование переменной с индексами не должно оканчиваться на букву F, остальные же требования к наименованиям такие, как и для обычных целых и действительных переменных. Массив может состоять или из целых, или из действительных элементов, и правило, согласно которому первая буква наименования определяет, является ли переменная целой или действительной, действует и в данном случае. Однако *все элементы* массива должны быть *или целыми, или действительными*, смешивать разнородные элементы в одном массиве нельзя.

7.2. ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ ПЕРЕМЕННЫХ С ИНДЕКСАМИ

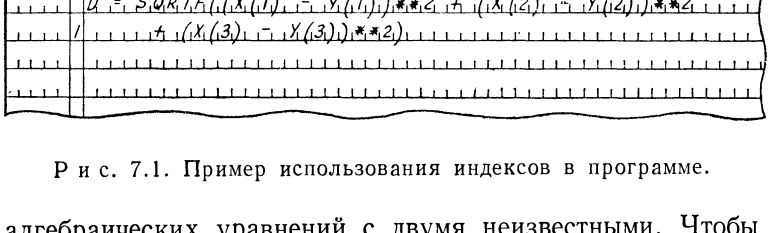
Предположим, что имеются две точки в пространстве с координатами X_1, X_2, X_3 и Y_1, Y_2, Y_3. Необходимо вычислить расстояние между этими точками, которое опре-

деляется формулой

$$D = \sqrt{(X_1 - Y_1)^2 + (X_2 - Y_2)^2 + (X_3 - Y_3)^2}\,.$$

Теперь предположим, что имеется некоторый массив X, три элемента которого являются действительными числами (координатами точки X), и аналогичный ему массив Y. Тогда вычисление расстояния между двумя точками в трехмерном пространстве можно произвести с помощью арифметического оператора, приведенного на рис. 7.1.

В качестве другого примера использования переменных с индексами рассмотрим задачу о решении двух линейных



Р и с. 7.1. Пример использования индексов в программе.

алгебраических уравнений с двумя неизвестными. Чтобы подчеркнуть аналогию между использованием индексов в ФОРТРАНе и в обычной математической записи, запишем систему уравнений в следующем виде:

$$C_{1,1}X_1 + C_{1,2}X_2 = B_1,$$
$$C_{2,1}X_1 + C_{2,2}X_2 = B_2.$$

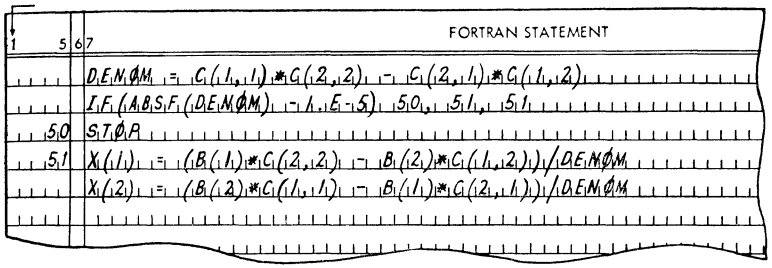
При такой постановке задачи свободные члены B_1 и B_2 составляют одномерный массив из двух элементов, а неизвестные X_1 и X_2 образуют другой одномерный массив из двух элементов; коэффициенты же при неизвестных являются элементами двумерного массива из двух строк и двух столбцов.

Решение такой простой системы удобно найти по правилу Крамера, согласно которому

$$X_1 = \frac{B_1C_{2,2} - B_2C_{1,2}}{C_{1,1}C_{2,2} - C_{2,1}C_{1,2}},$$

$$X_2 = \frac{B_2C_{1,1} - B_1C_{2,1}}{C_{1,1}C_{2,2} - C_{2,1}C_{1,2}}.$$

Программа для вычисления обоих неизвестных приведена на рис. 7.2. В этой программе есть две детали, которые нуждаются в пояснении. Во-первых, так как знаменатель обоих выражений одинаков, то он вычисляется отдельно и затем используется при расчете X_1 и X_2. Во-вторых, не исключена возможность того, что знаменатель окажется нулевым, т. е. система будет иметь или бесчисленное множество решений или вовсе не будет иметь решения в зависимости от свободных членов. Очевидно, что в таком случае



Р и с. 7.2. Участок программы для решения системы двух линейных уравнений, в котором используются индексы.

правило Крамера использовать нельзя из-за деления на нуль. При попытке разделить на нуль ЭЦВМ обычно посылает указание, что произошла ошибка (вид этого указания зависит от устройства машины). Очевидно, следует предусмотреть проверку знаменателя, чтобы избежать деления на нуль.

В действительности знаменатель сравнивается по модулю с некоторым малым числом, например с 10^{-5}. Дело в том, что ошибки округления при малой абсолютной величине знаменателя могут привести к неточным результатам, даже если знаменатель и не равен нулю.

7.3. ДЛЯ ЧЕГО НУЖНЫ ПЕРЕМЕННЫЕ С ИНДЕКСАМИ?

Примеры, приведенные в предыдущем разделе, показывают, как можно использовать в программах переменные с индексами, но не дают полного представления о возмож-

ностях метода. В самом деле, в этих программах нет ничего такого, чего нельзя было бы сделать столь же легко, присвоив каждой переменной отдельное наименование. Почему же в таком случае переменные с индексами так важны в ФОРТРАНе?

Ответ на вопрос заключается в том факте, что *сами индексы могут быть переменными или некоторыми выражениями*. Это означает, что, написав программу для некоторой последовательности расчетов, можно затем производить

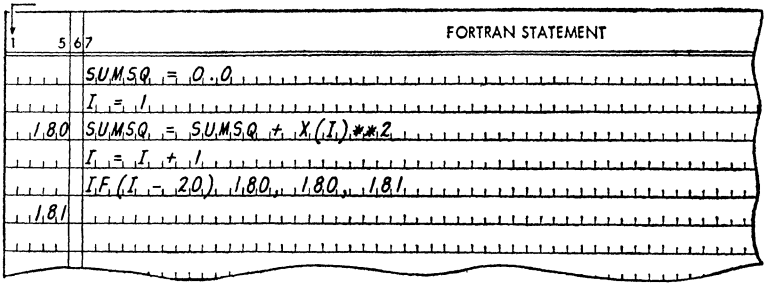


Рис. 7.3. Участок программы для вычисления суммы квадратов 20 чисел.

Здесь использованы индексы и оператор IF.

их для множества различных чисел, просто изменяя значения индексов.

Предположим, например, что необходимо вычислить сумму квадратов двадцати величин от X_1 до X_{20}, которые записаны в памяти ЭЦВМ. Конечно, можно было бы присвоить им 20 различных наименований и вычислить сумму их квадратов с помощью длинного арифметического оператора, однако проще подойти к этой задаче по-другому. Будем рассматривать 20 величин как элементы одномерного массива X. Тогда любую величину из этих 20 можно выбрать согласно ее наименованию X (I), и программу следует организовать так, чтобы I принимало последовательно все значения от 1 до 20.

В обычной математической записи

$$\text{SUMSQ} = \sum_{i=1}^{20} X_i^2.$$

Эти вычисления можно произвести с помощью программы, приведенной на рис. 7.3. Сначала мы присваиваем переменной SUMSQ значение 0, так что для вычисления всей суммы необходимо только последовательно добавлять очередное слагаемое. Затем переменной I присваивается значение 1, так что, когда очередь доходит до оператора 180, из массива X выбирается первый элемент. Затем I увеличивается на 1 и производится проверка, все ли значения массива X использованы в вычислениях. Заметим, что, когда оператор IF показывает I = 20, необходимо еще раз вернуться к оператору 180, так как I было увеличено *перед проверкой*.

Три последних оператора в этой программе выполняются точно 20 раз; в результате получается сумма квадратов всех 20 элементов массива. В разд. 7.6 мы увидим, что эта программа может быть сделана еще проще, если использовать оператор DO.

7.4. ОПЕРАТОР DIMENSION

Если в программе используются переменные с индексами, то в ней должна содержаться и некоторая дополнительная информация:

1. Какие переменные снабжены индексами?
2. Сколько индексов имеет каждая из переменных?
3. Сколько элементов содержится в каждом из массивов, т. е. какова наибольшая величина каждого индекса?

Ответ на эти вопросы содержится в операторе DIMENSION. Каждая переменная с индексом должна быть упомянута в операторе DIMENSION, а сам он должен располагаться до первого появления этой переменной в программе. Обычно все операторы DIMENSION, которые содержат сведения о переменных с индексами, ставятся в самом начале программы. Оператор DIMENSION может содержать сведения о любом количестве переменных; кроме того, в программе можно употреблять любое количество операторов DIMENSION.

Оператор DIMENSION записывается в виде

DIMENSION V, U, W, ...,

где V, U, W обозначают наименования переменных, за которыми должны следовать скобки, содержащие одну, две или три целых константы без знака. Эти константы определяют максимальную величину каждого индекса, а количество констант в скобках определяет размерность массива. Когда транслятор ФОРТРАНа распределяет память, необходимую для того или иного массива, то используются как раз сведения, содержащиеся в операторе DIMENSION. Например, если в программе содержится оператор

```
DIMENSION X(20), A(3, 10), K(2, 2, 5)
```

то при распределении памяти будут предусмотрены 20 ячеек для одномерного массива X, 30 ячеек для двумерного (3×10) массива A и 20 ячеек для трехмерного ($2 \times 2 \times 5$) массива K.

При составлении программы программист должен предусматривать такие размеры массивов информации, которые не превосходили бы максимальных, указанных в операторе DIMENSION. Кроме того, индексы не должны быть меньше 1, нулевые и отрицательные индексы не допускаются. Если нарушить эти ограничения, то исходная программа будет переведена в рабочую, но эта последняя почти наверняка даст ошибочные результаты.

Оператор DIMENSION принадлежит к числу *неисполняемых* операторов, т. е. он только содержит информацию для транслятора, но сам по себе не определяет никаких вычислений (ему не соответствуют команды в рабочей программе). Он может стоять в любом месте программы, даже между двумя арифметическими операторами. Однако, как уже было отмечено ранее, оператор DIMENSION, в котором содержится информация о какой-либо переменной, должен быть расположен до первого ее появления в программе. Кроме того, оператор DIMENSION не может быть первым в области действия оператора DO (см. разд. 7.6).

За небольшим числом исключений, о которых мы скажем позже, переменные с индексами могут появляться в программе в любом месте, где могла бы быть использована обычная переменная. Например, рассмотрим оператор READ, который мог бы потребоваться для ввода информации в задаче с системой линейных уравнений из разд. 7.2.

Операторы DIMENSION, READ и FORMAT в данном случае могли бы, например, выглядеть следующим образом:

```
DIMENSION B(2), C(2, 2), X(2)
READ 16, C(1, 1), C(2, 1), C(1, 2), C(2, 2), B(1), B(2)
16 FORMAT (6F 10. 0)
```

Когда в операторе READ перечислены все элементы массива, их можно вводить в произвольной последовательности. Например, программисту может показаться более удобным следующий порядок ввода:

```
READ 16, C(1, 1), C(1, 2), B(1), C(2, 1) C(2, 2), B(2)
```

Естественно, что на перфокарте числа должны быть пробиты в том же порядке, в каком они перечислены в операторе READ.

Иногда бывает желательно ввести или вывести все элементы массива, не перечисляя их по одному. В таких случаях в операторе ввода или вывода следует писать название массива без индексов; при этом будет введен или выведен весь массив. Поэтому, если бы после оператора DIMENSION в нашем случае стоял оператор

```
READ 16, C, B
```

то были бы целиком введены оба массива. Конечно, необходимо при этом иметь правило, строго определяющее последовательность элементов массива. Это правило заключается в следующем. Одномерный массив вводится по порядку, начиная с элемента, имеющего индекс 1, и кончая максимальным индексом, указанным в операторе DIMENSION. Для случая двумерного массива ввод осуществляется так, что первый индекс изменяется быстрее. Поэтому операторы

```
DIMENSION R(2, 3)
READ 16, R
```

определяют следующий порядок ввода: R (1,1), R (2,1), R (1,2), R (2,2), R (1,3), R (2,3); именно так и следует располагать исходные числа на перфокарте. Таким образом, элементы двумерного массива перечисляются *по столбцам*.

Для случая трехмерного массива также действует правило, согласно которому первый индекс изменяется быстрее всего, а последний — медленнее всего.

7.5. ДОПУСТИМЫЕ ФОРМЫ ИНДЕКСОВ

До сих пор мы использовали в качестве индексов целые константы и целые переменные. Допустим еще три формы индексов. Если I — целая переменная, а L и K — целые константы, то допустимы следующие формы индексов[1]):

I

L

I ± L

L * I

L * I ± K

Значение вычисленного любым способом индекса не должно быть меньше 1 или превосходить максимальную величину, указанную в операторе DIMENSION. Переменные, входящие в состав индекса, *сами не должны иметь индексов.*

Существует множество различных задач, в которых очень удобно использовать последние три формы индексов. Рассмотрим одну из таких задач и решим ее тремя различными, хотя и эквивалентными способами.

Предположим, что в некоторой программе требуется произвести такие вычисления:

$$y = \begin{cases} a + bX + cX^2, & \text{если } K = 1, \\ d + eX + fX^2, & \text{если } K = 2, \\ g + hX + iX^2, & \text{если } K = 3. \end{cases}$$

Значения X и K вычислены ранее по ходу программы. Мы знаем, как написать три арифметических оператора с различными коэффициентами и затем использовать оператор IF, чтобы произвести вычисления по нужной формуле. Однако все это можно сделать гораздо проще, если воспользоваться индексами.

---

[1]) В некоторых последних версиях языка ФОРТРАН переменные могут иметь до 7 индексов, причем допускается более общая форма каждого из них, а именно

$$I_1^* I_2^* \ldots {}^* I_h \pm J_1^* J_2^* \ldots {}^* J_m \pm \ldots \pm L_1^* L_2^* \ldots {}^* L_n$$

где I_1, I_2, I_3, . . ., L_{n-1}, L_n — наименования целых переменных или целые константы.— *Прим. ред.*

Предположим, что девять коэффициентов образуют одномерный массив, который мы назовем C:

a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*
1	2	3	4	5	6	7	8	9

Если K = 1, то для вычислений нам необходимы элементы со значениями индексов

$$1 = 3K - 2,$$
$$2 = 3K - 1,$$
$$3 = 3K.$$

Если K = 2, то необходимы элементы со значениями индексов

$$4 = 3K - 2,$$
$$5 = 3K - 1,$$
$$6 = 3K.$$

Если K = 3, то необходимы элементы со значениями индексов

$$7 = 3K - 2,$$
$$8 = 3K - 1,$$
$$9 = 3K.$$

Таким образом, при любом значении K будут отобраны необходимые коэффициенты, если арифметический оператор записан в виде

```
Y = C(3*K-2)+C(3*K-1)*X+C(3*K)*X**2
```

Если же организовать массив несколько иначе, а именно

a	*d*	*g*	*b*	*e*	*h*	*c*	*f*	*i*
1	2	3	4	5	6	7	8	9

то арифметический оператор нужно будет записать следующим образом:

```
Y = C(K)+C(K+3)*X+C(K+6)*X**2
```

Другой подход, дающий те же самые результаты, заключается в задании двумерного массива из девяти

коэффициентов

$$\begin{matrix} a & b & c \\ d & e & f \\ g & h & i \end{matrix}$$

В этом случае с помощью переменной K можно выбрать строку (первый индекс), а константы 1, 2 и 3 определяет столбец (второй индекс). Арифметический оператор при этом запишется так:

```
Y=C(K, 1)+C(K, 2)*X+C(K, 3)*X**2
```

Итак, мы видим, что переменные с индексами облегчают выборку чисел из большого массива. Мы видим также, что элементы массива вовсе не обязаны каким-либо образом зависеть друг от друга.

Все же чаще всего переменные с индексами используются тогда, когда нужно произвести одни и те же вычисления для большого набора взаимосвязанных величин. Некоторые примеры такого использования индексов мы рассмотрим ниже в связи с оператором DO.

7.6. ОПЕРАТОР DO

Оператор DO является одним из наиболее мощных средств языка ФОРТРАН. Этот оператор позволяет производить вычисления в некотором участке программы многократно, причем при каждом новом прохождении цикла вычислений изменяется значение некоторой целой переменной. Индексация и оператор DO облегчают программирование вычислений, которые были бы слишком сложными при непосредственном использовании машинных команд. Материал, излагаемый в этом параграфе, весьма важен; поэтому мы проиллюстрируем работу оператора DO на нескольких примерах.

Оператор DO может быть записан в одной из двух форм:

$$\text{DO}\, ni = m_1,\ m_2$$

$$\text{DO}\, ni = m_1,\ m_2,\ m_3$$

Здесь n — номер некоторого оператора, i — наименование целой переменной без индекса и без знака, а m_1, m_2,

m_3 — либо целые константы без знака, либо целые переменные без индекса и без знака. Если m_3 отсутствует, то подразумевается, что оно равно 1. Операторы, следующие за оператором DO вплоть до оператора номер n включительно, будут выполнены многократно. Сначала они будут выполнены при $i = m_1$, а перед каждым следующим прохождением цикла i увеличивается на величину m_3. Повторные прохождения цикла будут продолжаться до тех пор, пока вычисления не будут произведены при наибольшем значении i, не превосходящем m_2[1]).

Чтобы практически проиллюстрировать применение оператора DO, рассмотрим снова задачу о нахождении суммы квадратов 20 чисел (см. разд. 7.3). Эту сумму можно вычислить с помощью короткой программы, приведенной на рис. 7.4.

Оператор DIMENSION устанавливает, что переменная X имеет один индекс, максимальное значение которого равно 20. Сначала мы присваиваем переменной SUMSQ исходное значение, равное нулю, а затем ставим оператор DO, где указан номер оператора 180. Это означает, что все операторы после DO до оператора 180 включительно будут выполнены многократно. В нашем случае после DO стоит всего один оператор, оператор 180; он-то и будет повторяться. Сначала оператор 180 выполняется при I = 1, так что

[1]) Здесь подразумевается, что значения i, m_1, m_2 и m_3 положительны; так как при каждом прохождении цикла m_3 добавляется к i, а сумма сравнивается с m_2, то $m_1 < m_2$. В силу этого условия цикл DO действует в *порядке возрастания* индекса i. В последних версиях языка ФОРТРАН такое ограничение отсутствует: m_3 может быть отрицательной целой константой либо целой переменной, принимающей отрицательные значения (в этих случаях, разумеется, $m_1 > m_2$). Если значение m_3 отрицательно, то выход из цикла происходит тогда, когда i становится в первый раз меньше, чем m_2. Например, при работе оператора

DO 523 I = 10, 2, —2

индекс I принимает последовательные значения 10,8 . . ., 2 (цикл DO действует в *порядке убывания* индекса I). Кроме того, операторы типа

DO nI = 10, 1

автоматически интерпретируются как

DO nI = 10, 1, —1

—Прим. ред.

на этот раз к переменной SUMSQ прибавляется X_1^2. Затем значение I увеличивается на 1 и оператор 180 выполняется теперь уже при I = 2, так что к предыдущему значению

```
1    5 6 7  FORTRAN STATEMENT
       DIMENSION X(20)
       SUMSQ = 0.0
       DO 180 I = 1, 20
180    SUMSQ = SUMSQ + X(I)**2
```

Рис. 7.4. Программа для вычисления суммы квадратов 20 чисел. Здесь использован оператор DO.

переменной SUMSQ прибавляется X_2^2. Этот процесс продолжается до тех пор, пока I не достигнет 20, после чего значение переменной SUMSQ будет равно сумме квадратов всех

```
1    5 6 7  FORTRAN STATEMENT
       DIMENSION DATA(21)
       SUM = 0.0
       DO 500 J = 1, 21, 2
500    SUM = SUM + DATA(J)
```

Р и с. 7.5. Программа для вычисления суммы элементов массива, имеющих нечетные порядковые номера.

20 чисел. После этого управление передается оператору, следующему за 180.

В качестве другого примера использования оператора DO рассмотрим одномерный массив DATA, содержащий 21 элемент. Необходимо вычислить сумму всех элементов массива с нечетными индексами и присвоить этой сумме наименование SUM. Эти вычисления реализованы в программе на рис. 7.5.

В данном случае оператор DO написан таким образом, что оператор с номером 500 выполнится при значениях переменной J, равных 1, 3, 5, ..., 17, 19, 21. При первом прохождении цикла первый элемент массива DATA будет прибавлен к переменной SUM; при втором прохождении цикла значение переменной J будет увеличено на 2, а к SUM добавится третий элемент массива DATA и т. д. Наконец, при J=21 к сумме прибавляется последний элемент массива, а управление передается оператору, следующему за 500.

7.7. ДАЛЬНЕЙШИЕ ОПРЕДЕЛЕНИЯ

Перед рассмотрением новых примеров введем некоторые определения, относящиеся к оператору DO.

Областью действия оператора DO называется последовательность идущих друг за другом операторов, первый из которых непосредственно следует за оператором DO, а последний имеет номер, упомянутый в записи этого DO. Иными словами, это та последовательность операторов, которая исполняется многократно.

Целая переменная *i* в общей форме оператора DO называется его *индексом*. Во время повторных прохождений цикла переменную *i* можно использовать в вычислениях как любую другую целую переменную. Мы уже видели, как ее можно использовать в качестве индекса при переменной; позже мы покажем, как эту переменную можно использовать и для некоторых других целей.

Существуют два основных способа выхода из цикла, определяемого оператором DO. *Нормальный выход* из цикла происходит тогда, когда определяющие оператор DO индексные условия *удовлетворены*, т. е. когда вычисления проделаны столько раз, сколько необходимо, исходя из значений m_1, m_2 и m_3. Как мы уже видели в этом случае, управление передается оператору, непосредственно стоящему после цикла. Второй способ выхода из цикла — это применение внутри него операторов IF и (или) GO TO. Такой способ может понадобиться, например, тогда, когда желательно с помощью чисел m_1, m_2 и m_3 определить *максимально возможное* количество прохождений цикла, а внутри него организовать проверку, определяющую *действительно необходимое* количество повторных вычислений.

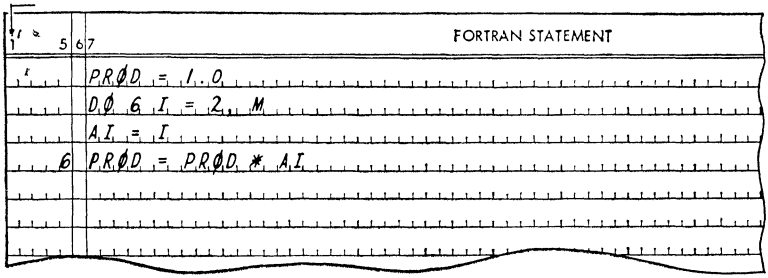
Если выход из цикла произошел по оператору GO TO или IF до удовлетворения индексных условий, то значение индекса i в момент выхода может использоваться в дальнейших вычислениях в качестве целой переменной. Такая особенность оператора DO может оказаться полезной. После нормального выхода из цикла в большинстве вариантов ФОРТРАНа переменной i пользоваться больше нельзя, но на практике это ограничение не приводит к серьезным неудобствам. Дело в том, что в подавляющем большинстве случаев постоянная m_3 опускается, т. е. принимается равной 1. После нормального выхода из такого цикла значение его индекса равно m_2.

Рассмотрим теперь другой способ использования переменной i внутри цикла. В теории соединений требуется вычислять произведения всех целых чисел от 1 до M, где M могло, например, быть вычислено ранее по ходу программы. Называя переменную M, мы определяем, что это должна быть целая переменная, однако значение произведения желательно иметь в виде действительного числа. Чтобы избежать вычислений со слишком большими целыми числами, мы должны перед умножением переводить каждое число из целой в действительную форму. Программа на рис. 7.6 реализует все эти вычисления и присваивает произведению название PROD. Сначала мы присваиваем переменной PROD исходное значение 1.0 и затем организуем цикл с помощью оператора DO, где индекс должен пройти все значения от 2 до M. Чтобы использовать I в качестве действительного числа, сначала выполняется оператор AI=I, который, как мы об этом уже говорили ранее, переводит число I из целой формы в действительную. При первом прохождении цикла в операторе 6 будет выполнено умножение 1.0 на 2.0, так как переменной PROD перед началом цикла было присвоено значение 1.0. При следующем прохождении цикла это произведение будет умножено уже на 3.0 и новое произведение будет иметь то же наименование PROD. Процесс будет повторяться до тех пор, пока два оператора в области действия DO не будут выполнены в последний раз при I = M.

При составлении этой программы подразумевается, что M по величине не меньше 2. Если M может принимать значение 1, то нельзя предсказать заранее, что произойдет,

если в операторе DO поставить в качестве нижнего предела 2, а в качестве верхнего предела 1 [1]). Если же M будет равно 2, то все будет в порядке — два оператора в области действия оператора DO будут выполнены один-единственный раз и управление будет передано на часть программы, следующую за оператором 6.

Как показывает этот пример, очень важно убедиться в выполнении точно необходимого количества прохождений



Р и с. 7.6. Программа для вычисления произведения целых чисел от 1 до M.

цикла. Из опыта известно, что здесь очень легко ошибиться на 1 в ту или в другую сторону. Весьма хорошо проверять себя в таких случаях вопросом: «Каковы должны быть параметры цикла, если его необходимо выполнить только один раз?» Исходя из соответствующего подсчета, обычно нетрудно бывает установить, правильно ли выбраны параметры цикла в программе.

Оператор DO часто очень удобно использовать для перебора последовательности значений некоторой независимой переменной. Например, в гл. 3 мы подсчитывали синус для углов 30, 390, 750 и т. д. до 2910 градусов. На рис. 3.10 было указано, что эти числа вводятся с перфокарт. Те же самые значения аргумента можно было бы получить гораздо проще с помощью оператора DO, как показано на рис. 7.7. Индекс оператора DO должен быть целой переменной; на рис. 7.7 он обозначается буквой *I*. В процессе вычислений, однако, угол должен быть действительной

[1]) См. предыдущее примечание.

величиной; в программе на рис. 3.10 эта переменная называлась DEGREE. Поэтому первый же оператор в области действия DO присваивает переменной DEGREE значение I, и при этом значение I переводится в действительную форму. Остальные вычисления остались без изменений. В конце

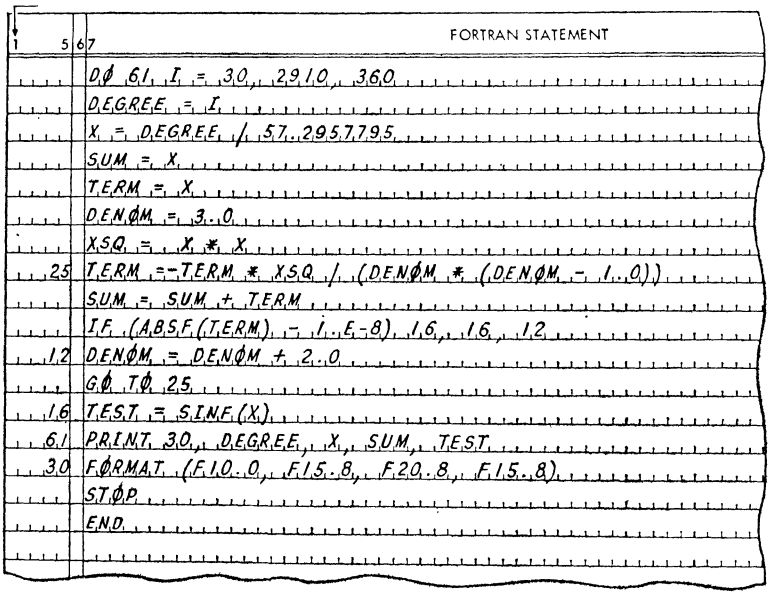


Рис. 7.7. Усовершенствованный вариант программы рис. 3.10. Последовательные значения аргумента получаются в этой программе не чтением с перфокарт, а с помощью оператора DO.

области действия оператора DO стоит оператор PRINT, с помощью которого печатаются результаты вычислений.

В качестве другого примера подобного же использования оператора DO рассмотрим предварительные проверочные программы из практического примера 6 (вычисление интеграла вероятностей). В одной из таких программ требовалось перебрать значения от 2.00 до 4.00 с шагом 0.01. На этот раз невозможно непосредственно использовать значение индекса оператора DO, так как этот индекс является целой величиной, а нам необходимо увеличивать значения аргу-

мента каждый раз на 0.01. Эту трудность можно легко преодолеть, написав следующие операторы перед началом основных вычислений:

```
DO 73 I = 200, 400
X = I
X = X/100.0
```

Индекс оператора DO пройдет все значения от 200 до 400 с шагом 1. После преобразования в действительную форму и деления на 100 переменная X пройдет все значения от 2.00 до 4.00 с шагом 0.01.

7.8. ПРАВИЛА ИСПОЛЬЗОВАНИЯ ОПЕРАТОРА DO

Оператор DO позволяет строить очень изящные и экономные программы, но при его использовании необходимо строго соблюдать определенные правила. Мы сначала приведем список этих правил, а затем на нескольких примерах поясним соответствующие ситуации.

Правило 1. Первый оператор в области действия оператора DO должен быть исполняемым. Таким образом, операторы DIMENSION и FORMAT исключаются из этого числа, потому что, не вызывая сами никаких вычислений, они только содержат в себе некоторую информацию для программы-транслятора. Как мы увидим ниже, еще несколько типов операторов являются неисполняемыми.

Правило 2. В области действия одного оператора DO (который мы здесь назовем «внешним» DO) может содержаться другой оператор DO (который мы здесь назовем «внутренним» DO). В этом случае обязательно, чтобы все операторы из области действия внутреннего DO принадлежали также к области действия внешнего DO. При этом допустимо, чтобы области действия двух или более операторов DO кончались одним и тем же оператором; однако область действия внутреннего DO не должна выходить за пределы области действия внешнего DO.

Правило 3. Последний оператор в области действия оператора DO не должен вызывать передачи управления.

Тем самым из этого числа исключаются операторы GO TO, IF и DO. Эти операторы можно использовать без всяких ограничений внутри области действия оператора DO. Ниже будет описан пустой оператор CONTINUE, который введен специально для того, чтобы не нарушать этого правила.

Правило 4. Никакой оператор в области действия оператора DO не должен каким-либо образом изменять его индексные параметры; это значит, что внутри области действия оператора DO недопустимо присваивать новые значения переменным i, m_1, m_2 и m_3. Как уже было отмечено выше, эти переменные можно использовать при любых вычислениях, если при этом их значения внутри области действия оператора DO не изменяются.

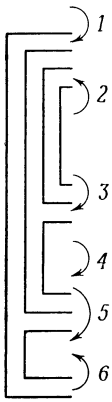


Рис. 7.8. Примеры допустимых циклов DO, вложенных друг в друга, и примеры допустимых и недопустимых передач управления.
Передачи 2, 3, 4 допустимы, передачи 1, 5, 6 недопустимы.

Правило 5. За одним исключением, недопустимо передавать управление внутрь области действия DO из какого-либо оператора, лежащего вне этой области. Поэтому категорически запрещается передавать управление внутрь области действия оператора DO с помощью GO TO или IF, не выполнив вначале его самого. Это правило запрещает также передачу управления из области действия внешнего DO в область действия внутреннего DO, но оно отнюдь не запрещает передачу управления из области действия внутреннего DO в область действия внешнего DO. Дело в том, что с «точки зрения» внешнего DO последний переход вполне допустим, так как управление передается от оператора, лежащего внутри области действия внешнего DO, другому оператору, также лежащему внутри нее. Схематическая иллюстрация этого правила приведена на рис. 7.8. Квадратные скобки на этом рисунке показывают области действия операторов DO, а стрелками изображаются передачи управления. Передачи 2, 3 и 4 допустимы, так как

2 и 3 производятся из области действия внутреннего DO в область действия внешнего DO, а передача 4 не выводит из области действия одного и того же оператора DO. Передачи управления 1, 5 и 6 недопустимы, так как они направлены внутрь области действия оператора DO извне.

Единственное исключение из правила, запрещающего передачу управления извне внутрь области действия оператора DO, заключается в следующем: в ходе выполнения оператора DO можно передать управление в какое-либо другое место программы, чтобы проделать вычисления, не изменяющие индексных параметров оператора DO, и затем вернуться в область действия того же оператора DO. Ограничение, налагаемое на точки выхода и возврата[1]), можно сформулировать следующим образом: между точками выхода и возврата не может находиться оператор DO или оператор, являющийся последним в его области действия.

Оператор CONTINUE является пустым: в рабочей программе отсутствуют соответствующие ему действия. Он вводится тогда, когда необходимо удовлетворить правилу, согласно которому последний оператор в области действия оператора DO не должен вызывать передачи управления. Этот оператор также приходится использовать, если внутри области действия DO имеется оператор IF, с помощью которого определяется момент окончания вычислений внутри цикла. В таких случаях нельзя передавать управление самому оператору DO, поскольку при такой передаче управления весь цикл начнется сначала с исходного значения индексных параметров. Поэтому если вычисления внутри области действия оператора DO закончены и момент окончания вычислений определен с помощью оператора IF, то для очередного прохождения цикла необходимо, чтобы оператор IF передавал управление оператору CONTINUE, стоящему в конце области действия оператора DO. Пример использования оператора CONTINUE можно найти в программе на рис. 7.9.

---

[1]) Некоторые программисты привыкли называть возвратом команду, находящуюся в подпрограмме и передающую управление на основную программу. Во избежание недоразумений отметим, что здесь под точкой возврата подразумевается оператор, на который передается управление после окончания вычислений в каком-либо другом месте программы.— *Прим. ред*

7.9. ДАЛЬНЕЙШИЕ ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ ОПЕРАТОРА DO

Оператор DO является весьма мощным средством и очень широко применяется при составлении программ на ФОРТРАНе; мы приведем некоторые дополнительные примеры его использования.

Предположим, что информация, введенная в машину, состоит из некоторого набора экспериментально измеренных значений. Каждая пара экспериментальных данных содержит значение X и соответствующее ему значение Y (абсциссу и ординату точки на графике). Вся эта информация была введена в машину хаотическим образом. Нам известно лишь то, что каждое значение Y соответствует значению X, имеющему тот же порядковый номер, т. е. Y(1) соответствует X(1), Y(2) соответствует X(2) и т. д., но первое значение X не обязано быть наименьшим из всего набора значений. Для дальнейших вычислений нам желательно упорядочить информацию так, чтобы первое значение X было минимальным, следом за ним шло второе по величине и т. д. Иными словами, необходимо расположить значения X в порядке возрастания.

Предположим, что исходная информация образует два массива: массив X, содержащий 25 элементов, и соответствующий ему массив Y, также содержащий 25 элементов. Значения X расположены хаотически.

Программа для упорядочения этой информации, составленная на ФОРТРАНе, включает в себя два цикла DO, находящихся один внутри другого. Мы, однако, опишем процесс построения такой программы, рассмотрев упрощенный вариант внутреннего цикла, а уже потом составим программу полностью. Этот упрощенный цикл поместит наименьшее значение X на первое место в массиве.

Такую операцию можно организовать следующим образом. Вначале сравним первое и второе значения X в исходном массиве. Если первое значение X меньше второго или равно ему, то ничего делать не нужно; если же первое значение X больше второго, то необходимо поменять их местами внутри массива. Исследовав первый и второй элементы массива и в случае необходимости поменяв их местами, перейдем к сравнению первого и третьего элементов

и также поменяем их местами, если третий элемент окажется меньше первого. «Первым» теперь может оказаться элемент, который в исходном массиве был вторым, но это не играет никакой роли. Точно так же сравним первый и четвертый, затем первый и пятый элементы и т. д., осуществляя всякий раз в случае необходимости перестановку, пока первый элемент массива не сравнится со всеми остальными. После окончания этого процесса наименьшее значение X будет первым элементом массива. Вспоминая, что каждому значению X соответствует некоторое значение Y, мы должны предусмотреть такие же перестановки в массиве Y, только на этот раз без сравнения элементов.

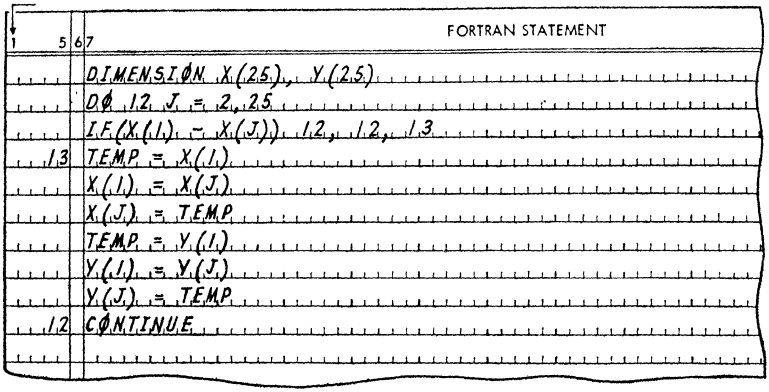
Для перестановки элементов массива используется трехступенчатый процесс: (1) переставить первый элемент массива в некоторую временную ячейку[1]) (в программе этой ячейке присвоено наименование TEMP); (2) переставить второй элемент туда, где ранее был первый элемент; (3) переставить первый элемент, который теперь записан в ячейке TEMP, в ту ячейку, где ранее был второй элемент.

Программа, с помощью которой можно произвести все эти операции, приведена на рис. 7.9. Мы предполагаем, что ввод информации осуществлен заранее, и также не приводим операторы, полностью завершающие упорядочение или производящие дальнейшие вычисления.

Эта программа достаточно ярко иллюстрирует положения, о которых говорилось в этой главе. В ней имеется оператор DO, индекс которого начинается не с 1. Там же имеется пример использования оператора CONTINUE. Этот последний оператор необходим по следующей причине: если с помощью оператора IF установлено, что первое значение X уже меньше очередного, то необходимо опустить 6 последующих операторов, осуществляющих перестановку. Единственное, что нужно сделать в этом случае,— это начать весь процесс сравнения сначала, увеличив предвари-

[1]) Здесь и дальше используется понятие ячейки памяти ЭЦВМ. Дело в том, что при трансляции программы на ФОРТРАНе каждой переменной отводится ячейка памяти, где хранится численное значение; переменной же с индексом соответствует целый массив последовательных ячеек. Строго говоря, при описании языка не следовало бы использовать понятие ячейки памяти: однако в конкретных случаях это оказывается удобным.— *Прим. ред.*

тельно на 1 значение индекса. Но, как мы уже отмечали ранее, передавать управление снова оператору DO в таком случае нельзя, так как при этом цикл повторился бы снова с начального значения индекса J = 2. Поэтому управление



Р и с. 7.9. Участок программы, позволяющий поместить наименьшее значение X на первое место в массиве X и соответствующее ему значение Y — на первое место в массиве Y.

передается пустому оператору CONTINUE, который был упомянут в операторе DO в качестве конечного в области его действия.

При прочтении этой программы необходимо твердо помнить определение арифметического оператора: переменная в левой части получает то значение, которое имеет выражение в правой части. Поэтому, например, оператор вида

$$X(1) = X(J)$$

означает, что число с наименованием X(J) должно быть записано в ячейку, обозначенную через X(1). Прежнее число, записанное в ячейке X(1), при этом теряется; число же из ячейки X(J) остается неизменным.

Может оказаться, что несколько значений X одинаковы. При таком написании программы, как на рис. 7.9, наличие нескольких одинаковых значений X ничуть не усложняет задачу. Если какие-либо значения X оказались одинаковыми, то нет никакого смысла их переставлять, и мы просто

передаем управление оператору CONTINUE и заново повторяем цикл.

По окончании этого цикла (оператор DO будет «удовлетворен») информация уже реорганизована таким образом, что наименьшее значение X будет первым элементом массива X, а соответствующее ему значение Y будет первым элементом в массиве Y. Теперь нужно сделать следующий шаг и поставить второе по величине значение X на второе место в массиве X. Это можно сделать, сравнивая второе значение с третьим и со всеми последующими и при необходимости переставляя эти числа точно так же, как это было сделано для первого элемента массива. После этого нужно поставить третий по величине элемент на третье место, четвертый — на четвертое и т. д. Короче говоря, необходимо расположить все значения X в порядке возрастания величины.

По-видимому, для составления такой программы необходимо сделать переменным индекс того элемента, который мы собираемся сравнивать со всеми остальными. Тогда с помощью этого переменного индекса можно будет выбрать тот элемент, который необходимо сравнивать со всеми последующими. Этот переменный индекс, который мы назовем I, должен будет пройти значения от 1 до 24. Тот индекс, который в предыдущей программе обозначался через J, будет и теперь обозначаться через J, но он должен будет проходить значения от I + 1 до 25. Таким образом, в программе приходится использовать два оператора DO, один из которых имеет индекс I, а второй — индекс J. Оператор DO с индексом I является внешним, а с индексом J — внутренним. Внешний оператор управляет индексом внутреннего. При составлении программы возникает одно небольшое усложнение. Дело в том, что индекс внутреннего DO должен начинаться с числа, превосходящего текущее значение I на 1. Однако в определении оператора DO говорится, что индексные параметры этого оператора должны быть либо константами, либо целыми переменными. Нельзя, например, писать оператор в таком виде:

```
DO 12 J=I+1,25
```

Чтобы обойти это ограничение, приходится вводить дополнительный арифметический оператор, который вычис-

ляет значение I + 1 и присваивает его переменной IP 1. Полная программа для упорядочения информации приведена на рис. 7.10[1]).

Не следует думать, что такой способ упорядочения исходной информации является наиболее эффективным; он приведен здесь лишь для иллюстрации программ с оператором DO.

Приведем теперь еще один пример использования оператора DO. Предположим, что введенные в машину и затем

```
1    5 6 7  FORTRAN STATEMENT
      DIMENSION X(25), Y(25)
      DO 12 I = 1, 24
      IP1 = I + 1
      DO 12 J = IP1, 25
      IF (X(I) - X(J)) 12, 12, 13
   13 TEMP = X(I)
      X(I) = X(J)
      X(J) = TEMP
      TEMP = Y(I)
      Y(I) = Y(J)
      Y(J) = TEMP
   12 CONTINUE
```

Рис. 7.10. Программа, позволяющая переупорядочить массивы X и Y так, чтобы значения X шли в порядке возрастания.

упорядоченные числа являются координатами точек на графике некоторой функции Y(X). Попытаемся найти площадь под кривой, т. е. вычислить определенный интеграл от этой функции. Если окажется, что значения X расположены на равном расстоянии друг от друга с шагом H, то приближенное значение интеграла можно найти с помощью метода трапеций, знакомого читателям из гл. 6:

$$\text{AREA} = \frac{H}{2}(y_1 + 2y_2 + 2y_3 + \ldots + 2y_{24} + y_{25}).$$

[1]) В технической литературе на английском языке этот алгоритм обычно называется «bubble sorting».

Чтобы подсчитать сумму всех Y с индексами от 2 до 24, можно использовать оператор DO. После этого сумму можно умножить на 2, добавить первое и последнее значения Y и умножить всю сумму на H/2. Такая программа, приведенная на рис. 7.11, может размещаться непосредственно вслед за оператором CONTINUE в программе из рис. 7.10. Величина шага H вычисляется как расстояние

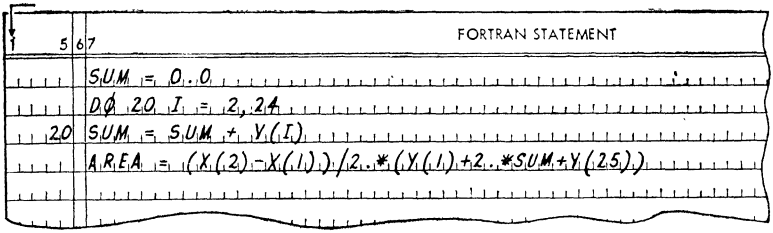


Рис. 7.11. Участок программы для интегрирования по правилу трапеций.

между двумя соседними значениями X. Если же расстояния между соседними значениями X неодинаковы, то программа даст неверный результат. При работе с таким массивом, где расстояния между соседними X случайны, метод численного интегрирования нужно изменить.

Возвращаясь к случаю равномерно расположенных X, составим еще программу для вычисления интеграла с помощью метода Симпсона:

$$\text{AREA} = \frac{H}{3}(y_1 + 4y_2 + 2y_3 + \ldots + 2y_{23} + 4y_{24} + y_{25}).$$

Эта программа будет немного сложнее предыдущей из-за того, что приходится попеременно умножать ординаты то на 2, то на 4. Совершенно очевидно, что в данном случае проще всего использовать два оператора DO, отдельно накапливая суммы ординат, соответствующие этим коэффициентам. Участок программы для интегрирования с помощью метода Симпсона приведен на рис. 7.12.

Те же вычисления можно произвести и с помощью всего лишь одного оператора DO. (В некотором смысле этот вариант предпочтительнее, потому что составленная таким

образом программа требует несколько меньших затрат машинного времени.) Предположим, что некоторый индекс проходит значения от 1 до 11. Тогда значение этого индекса,

```
1    5 6 7  FORTRAN STATEMENT
       ODD = 0.0
       EVEN = 0.0
       DO 47 I = 2,24,2
   47  EVEN = EVEN + Y(I)
       DO 48 I = 3,23,2
   48  ODD = ODD + Y(I)
       AREA = (X(2) - X(1))/3.*(Y(1) + 4.*EVEN + 2.*ODD + Y(25))
```

Рис. 7.12. Один из вариантов программы для интегрирования по формуле Симпсона.

умноженное на 2, будет номером ординаты, которую необходимо умножить на 4, а если к этому удвоенному значению индекса прибавить 1, то получится номер ординаты

```
1    5 6 7  FORTRAN STATEMENT
       ODD = 0.0
       EVEN = 0.0
       DO 51 I = 1,11
       EVEN = EVEN + Y(2*I)
   51  ODD = ODD + Y(2*I + 1)
       AREA = (X(2) - X(1))/3.*(Y(1) + 4.*(EVEN + Y(24))
      1    + 2.*ODD + Y(25))
```

Рис. 7.13. Еще один вариант программы для интегрирования по формуле Симпсона.

с коэффициентом 2. Участок программы, построенный по такому принципу, приведен на рис. 7.13.

Здесь мы использовали то обстоятельство, что индексы при переменных могут быть выражениями. Без этого было

бы довольно затруднительно вычислить сумму ординат при помощи только одного оператора DO.

Можно заметить, что все формулы численного интегрирования были написаны здесь таким образом, что их индексы начинались с единицы, в то время как обычно принято начинать индексы с нуля. Это было сделано для того, чтобы легче было переходить от формулы к программе, так как в ФОРТРАНе индексы должны быть ненулевыми и положительными. Формулы с нулевыми и отрицательными индексами также можно запрограммировать, потратив дополнительные усилия; мы не будем рассматривать этого вопроса. Некоторые варианты ФОРТРАНа или сходные с ним языки допускают использование нулевых и отрицательных индексов.

7.10. ПРАКТИЧЕСКИЙ ПРИМЕР 9: ЛИНЕЙНАЯ ИНТЕРПОЛЯЦИЯ

Чтобы проиллюстрировать практическое использование переменных с индексами и оператора DO, рассмотрим программу линейной интерполяции в некотором массиве. Обычно такая программа является частью большей программы; здесь мы предусмотрим ввод табличных значений X и Y, а затем — тех X, для которых необходимо найти Y с помощью линейной интерполяции.

Мы предполагаем, что табличные значения X и Y отперфорированы на картах, каждая из которых содержит одно X и одно Y. Предполагается, что колода перфокарт расположена в порядке возрастания величин X, а их количество не превышает 200. При этом вовсе не предполагается, что последовательные значения X находятся на равном расстоянии друг от друга. Значения X и Y пробиты каждое в 10 колонках перфокарты и могут вводится в машину с помощью спецификации поля F10.0. Последняя карта из этой колоды будет содержать какую-либо ненулевую цифру в колонке 21. Программа приведена на рис. 7.14.

Рассмотрим теперь ту часть программы, которая определяет ввод информации. Оператор DIMENSION указывает, что 200 ячеек памяти будут зарезервированы для массива X и еще 200 — для массива Y. Оператор DO используется в этой части программы просто для того, чтобы присвоить индексы каждой паре значений X и Y. Всего может быть

не более 200 пар значений, поэтому пределы работы оператора DO определены между 1 и 201; предполагается, что окончание цикла будет определяться не индексными параметрами оператора DO, а наличным количеством перфокарт с вводимой информацией. При удовлетворении индексных

```
1    5 6 7  FORTRAN STATEMENT
      DIMENSION X(200), Y(200)
      DO 63 I = 1, 201
      READ 57, X(I), Y(I), K
   57 FORMAT (2F10.0, I1)
      IF (K) 43, 63, 43
   63 CONTINUE
      STOP 1234
   43 READ 70, XE, L
   70 FORMAT (F10.0, I1)
      DO 110 J = 1, I
      IF(X(J) - XE) 110, 111, 112
  110 CONTINUE
      STOP 2345
  111 YE = Y(J)
      GO TO 200
  112 YE = Y(J-1)+(Y(J)-Y(J-1))/(X(J)-X(J-1))*(XE-X(J-1))
  200 PRINT 201, XE, YE
  201 FORMAT (2E15.8)
      IF(L) 202, 43, 202
  202 STOP
      END
```

Р и с. 7.14. Программа для линейной интерполяции (практический пример 9).

условий количество перфокарт с вводимой информацией превышает 200 и дальнейшие вычисления нужно прекратить. Поэтому сразу после цикла расположен оператор STOP[1]).

[1]) Как уже упоминалось в разд. 1.11, большие вычислительные машины почти всегда снабжены некоторой управляющей программой (программой-монитором). При наличии программы-монитора использование оператора STOP в большинстве случаев нецелесообразно,

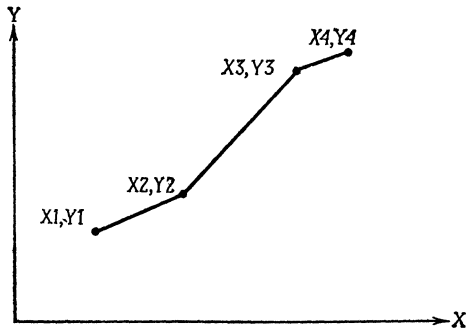
При выполнении оператора STOP, записанного в такой форме, следующее за ним число печатается. Это необходимо, потому что вычисления могут остановиться и по другим причинам: например, в конце программы также должен стоять оператор STOP. Всегда следует знать, по какой причине прекращены вычисления: то ли из-за ошибки, то ли потому, что вся программа уже выполнена.

Предполагается, что колонка 21 на всех перфокартах, кроме последней, является свободной. Если в колонке ничего не пробито, то при вводе она интерпретируется так, как будто в ней пробит нуль. Поэтому IF после оператора 57 будет все время передавать управление оператору CONTINUE, пока в последней карте в колонке 21 не окажется какой-либо ненулевой символ. Другими словами, мы остаемся в области действия оператора DO до тех пор, пока не будет обнаружена последняя карта с вводимой информацией, причем при каждом очередном прохождении цикла значение индекса I увеличивается на 1. При обнаружении последней карты (с ненулевым символом в колонке 21) оператор IF передаст управление оператору 43, откуда начинается обработка информации.

Теперь в памяти ЭЦВМ записаны пары значений X и Y; схематически их можно изобразить на графике в виде набора точек, соединенных отрезками прямых линий, как это показано на рис. 7.15. Последующие перфокарты, которые вводятся в машину, содержат только значения X в колонках 1 — 10. Нам необходимо ввести число с перфокарты, произвести интерполяцию табличных значений, найти соответствующее значение Y, напечатать пару значений X и Y, вернуться к началу этого участка программы, ввести новое значение X и т. д., пока не будет обработана последняя перфокарта, содержащая ненулевой символ в колонке 11.

а иногда и просто недопустимо. Взамен в программе-мониторе предусматриваются команды, с помощью которых программист может получить соответствующую информацию (т. е. где и по какой причине остановилась его программа), причем ЭЦВМ продолжает вычисления уже по совершенно другой программе. Необходимые подробности можно выяснить при работе на конкретной ЭЦВМ, но здесь эти детали не приводятся, поскольку различные программы-мониторы очень существенно отличаются друг от друга.

После ввода очередного значения абсциссы мы перебираем элементы массива X, чтобы обнаружить или такое же значение X, или два табличных, между которыми заключено введенное. Этот процесс можно осуществить, сравнивая введенное значение XE по очереди с каждым элементом массива X. Сделать это тем более легко, что значения X



Р и с. 7.15. Схематическое представление пар значений X и Y в задаче о линейной интерполяции (практический пример 9).

вводились в порядке возрастания величины. Для такого сравнения используется оператор DO, индекс J которого проходит значения от 1 до I. (I равно числу пар табличных значений X и Y.) Если XE окажется меньше второго табличного значения X, то оператор IF немедленно прекратит работу оператора DO, причем J будет при этом равно 2. Если же XE больше второго значения X, то индекс J увеличится на 1 и значение XE сравнится с третьим элементом массива X. Этот процесс продолжается до тех пор, пока не отыщется значение X, больше введенного значения XE. При этом индекс определяет, какой именно элемент массива X оказался больше XE. Графически эта ситуация изображена на рис. 7.16. Если в результате какой-либо ошибки XE окажется больше наибольшего значения X, индексные условия оператора DO будут удовлетворены и управление будет снова передано оператору STOP.

После обнаружения табличного значения X, большего или равного XE, можно переходить к вычислению соответ-

ствующего значения Y. Если XE просто равно некоторому X, то никакой интерполяции производить не следует, взяв вместо этого соответствующее значение Y из таблицы. Если же, что более вероятно, значение XE попадет между двумя табличными значениями X, то YE необходимо вычислить, интерполируя между значениями Y(J) и Y(J — 1). Геометрически этот процесс изображен на рис 7.16; нетрудно убедиться, что арифметический оператор 112 вычисляет

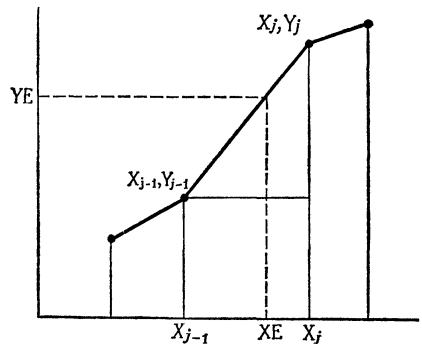


Рис. 7.16. Геометрическое представление процесса линейной интерполяции, когда найдено, что XE меньше некоторого табличного значения X(J).

именно это значение Y. В конце концов, каким бы способом ни было вычислено YE, необходимо напечатать XE и YE. С помощью оператора IF мы определяем, содержался ли на только что введенной перфокарте в колонке 11 ненулевой символ. Если нет, то мы возвращаемся к оператору 43 для ввода нового значения XE, а в противном случае передаем управление оператору STOP, означающему на этот раз нормальное окончание по программе.

Существуют интерполяционные схемы более высоких порядков, нежели только что описанный линейный алгоритм. В таких схемах через три, четыре или более последовательных точек функции проводятся многочлены второго, третьего или более высокого порядка, по которым и вычисляются промежуточные значения YE. Описание интерполяционных алгоритмов можно найти практически в любой книге по численному анализу. Два подобных метода вкратце описаны в упражнениях 9 и 10.

Упражнения

В каждую из программ, которые придется составлять при решении этих упражнений, необходимо включить оператор DIMENSION, который определит количество элементов в данных массивах.

*1. Предположите, что координаты какой-либо точки в пространстве заданы тремя элементами одномерного массива X. (Обратите внимание на различный характер понятия количества измерений: элементы одномерного массива используются для задания координат точки в трехмерном пространстве!) Напишите арифметический оператор, с помощью которого можно было бы вычислить расстояние точки от начала координат. Это расстояние дается корнем квадратным из суммы квадратов координат точки.

2. Если координаты точки в пространстве равны X_1, X_2 и X_3, то направляющие косинусы можно вычислить по следующим формулам:

$$CA = \frac{X_1}{\sqrt{X_1^2+X_2^2+X_3^2}},$$

$$CB = \frac{X_2}{\sqrt{X_1^2+X_2^2+X_3^2}},$$

$$CC = \frac{X_3}{\sqrt{X_1^2+X_2^2+X_3^2}}.$$

Напишите арифметические операторы для вычисления направляющих косинусов. Предположите, что координаты являются элементами одномерного массива X.

*3. Заданы два двумерных массива A и B. Напишите программу, с помощью которой можно вычислить элемент двумерного массива C согласно следующим формулам (максимальная величина индексов равна 2):

$$C_{11} = a_{11}b_{11} + a_{12}b_{21},$$
$$C_{12} = a_{11}b_{12} + a_{12}b_{22},$$
$$C_{21} = a_{21}b_{11} + a_{22}b_{21},$$
$$C_{22} = a_{21}b_{12} + a_{22}b_{22}.$$

Читатели, знакомые с матричной алгеброй, легко узнают формулу умножения матриц.

4. Задан двумерный массив R, причем величина каждого индекса не превосходит 3. Элементы этого массива можно рассматривать как элементы матрицы, имеющей три строчки и три столбца. Напишите программу для вычисления определителя этой матрицы.

*5. Два одномерных массива A и B содержат каждый по 30 элементов. Вычислите

$$D = \left[\sum_{i=1}^{30} (A_i - B_i)^2\right]^{1/2}$$

(расстояние в 30-мерном пространстве). Напишите участок программы для этих вычислений с использованием оператора DO и без него.

*6. Если имеются некоторые табличные значения, заданные, например, в виде одномерного массива, то *первые разности* этой таблицы можно получить, вычитая каждый элемент, кроме последнего, из следующего за ним элемента. Предположим, что имеется одномерный массив X, состоящий из 50 элементов. Вычислите 49 элементов другого массива, который можно обозначить D(X), используя для вычисления элементов этого массива соотношения

$$DX(I) = X(I+1) - X(I); \quad I = 1, 2, 3, \ldots, 49.$$

Напишите участок программы для этих вычислений с использованием оператора DO и без него.

7. Предположим, что задан некоторый одномерный массив Y, состоящий из 32 элементов. Эти элементы можно рассматривать как 32 ординаты экспериментальной кривой, соответствующие некоторым 32 равноотстоящим абсциссам. Предполагая, что расстояние между соседними абсциссами известно и равно H, вычислите приближенное значение интеграла от этой кривой, воспользовавшись формулой трапеций:

$$TRAP = \frac{H}{2}(Y_1 + 2Y_2 + 2Y_3 + \ldots + 2Y_{31} + Y_{32}).$$

Напишите участок программы для этих вычислений с использованием оператора DO и без него.

8. Некоторый двумерный массив AMATR содержит 10 строк и 10 столбцов. Одномерный массив DIAG содержит 10 элементов. Вычислите элементы массива DIAG по формуле

$$DIAG(I) = AMATR(I, I), \quad I = 1, 2, \ldots, 10.$$

Напишите участок программы для этих вычислений с использованием оператора DO и без него.

*9. Задан одномерный массив Y, содержащий 50 элементов и числа U и I. Напишите участок программы для вычисления величины S согласно следующей формуле:

$$S = y_i + u\frac{y_{i+1} - y_{i-1}}{2} + u^2\frac{y_{i+1} - 2y_i + y_{i-1}}{2}.$$

Эта формула называется *интерполяционной формулой Стирлинга* (интерполяция с помощью вторых разностей). Ее можно интерпретировать геометрически следующим образом: имеются три точки кривой, через эти три точки проводится квадратичная парабола. В этой формуле предполагается, что $x_{i+1} - x_i = x_i - x_{i-1} = h$. Предполагается, что величина u равна $(x - x_i)/h$. Формула Стирлинга дает значение y по уравнению квадратичной параболы, проходящей через три соседние ординаты кривой.

10. Аналогично упражнению 9 напишите участок программы для вычисления значения T по следующей формуле:

$$T = y_i + u(y_{i+1} - y_i) + \frac{u(u-1)(y_{i+2} - y_{i+1} - y_i - y_{i-1})}{4} + \frac{\left(u - \frac{1}{2}\right) u(u-1)(y_{i+2} - 3y_{i+1} + 3y_i - y_{i-1})}{6}.$$

Эта формула называется *интерполяционной формулой Бесселя* (интерполяция с помощью третьих разностей). Согласно формуле Бесселя, интерполированные значения находятся с помощью уравнения кубической параболы, проведенной через четыре соседние точки на кривой.

***11.** Заданы два одномерных массива, каждый из которых содержит по 7 элементов. Предположите, что все семь элементов массива A пробиты на одной перфокарте, а все семь элементов массива B — на другой. Каждый элемент занимает на перфокарте 10 колонок и может вводиться с помощью спецификации поля типа F10.0. Напишите программу, в которой предусматривалось бы чтение перфокарт, вычисление и печать значения ANORM. Для вычисления ANORM используйте формулу

$$\text{ANORM} = \sqrt{\sum_{i=1}^{7} A_i B_i}.$$

Для печати значения ANORM используйте спецификацию поля E20.8.

12. Дополните программу из упражнения 11 следующей операцией: сравните A_i с B_i; в том случае, когда все значения $A_i > B_i$ для $i = 1, 2, \ldots, 7$, присвойте некоторой переменной значение 1 и напечатайте его; если же это условие не выполнено, то присвойте этой переменной значение 0 и также напечатайте его.

Во всех нижеследующих программах используйте оператор DO.

***13.** Целочисленный одномерный массив M содержит 20 элементов. Напишите участок программы, где каждому элементу массива присваивается новое значение, равное старому, умноженному на номер этого элемента. Другими словами, заменить каждое m_i на $m_i \cdot i$ для $i = 1, 2, 3, \ldots, 20$.

***14.** Два одномерных массива R и S могут содержать каждый не более 40 элементов. Фактическое количество элементов задается предварительно вычисляемой переменной M. Вычислите первые M элементов массива T, который также может содержать не более 40 элементов, согласно формуле

$$T(I) = R(I) + S(I); \quad I = 1, 2, 3, \ldots, M.$$

15. Два одномерных массива A и B могут содержать каждый не более 18 элементов. Имеется некоторое целое число N, величина которого не более 18. Вычислите

$$C = \sum_{k=1}^{N} A_k B_k.$$

***16.** Одномерный массив F содержит не более 50 элементов. Каждый из первых M элементов массива, исключая первый и M-й элементы, должен быть заменен следующим значением:

$$F_i = \frac{F_{i-1} + F_i + F_{i+1}}{3}.$$

Это пример *сглаживания* экспериментальных данных, которое уменьшает влияние случайных ошибок.

***17.** Одномерный массив B содержит 50 элементов. Поместите в ячейку BIGB значение наибольшего из них, а в ячейку NBIGB его номер.

18. Два одномерных массива X и Y содержат каждый по 50 элементов. Известно, что значение переменной XS равно одному из элементов массива X. Найдите этот элемент и запомните соответствующее ему значение Y в ячейке YS. Иными словами, если $XS = X_i$, то переменной YS необходимо присвоить значение Y_i.

19. То же, что и в упражнении 18, только теперь заранее не известно, равно ли значение XS какому-либо элементу из массива X. Если XS равно какому-либо X_i, то запомните Y_i в ячейке YS, если же XS не равно никакому X_i, то запомните в ячейке YS нуль.

***20.** Двумерный массив A содержит 15 строчек и 15 столбцов. Одномерный массив X содержит 15 элементов. Вычислите 15 элементов одномерного массива B по формуле

$$B_i = \sum_{j=1}^{15} A_{ij} X_j, \quad i = 1, 2, \ldots, 15.$$

Эта формула описывает умножение матрицы на вектор.

21. Три двумерных массива A, B и C содержат каждый по 15 строчек и по 15 столбцов. Если заданы массивы A и B, напишите программу для вычисления элементов массива C по формуле

$$C_{ij} = \sum_{k=1}^{15} A_{ik} B_{kj}, \quad i, j = 1, 2, \ldots, 15.$$

Эта формула описывает умножение матриц.

***22.** Двумерный массив RST содержит 20 строчек и 20 столбцов. Вычислите произведение всех элементов массива RST, лежащих на главной диагонали, и присвойте это значение переменной DPROD. Элемент, лежащий на главной диагонали,— это тот элемент, оба индекса у которого одинаковы, так что

$$\text{DPROD} = \prod_{\text{I}=1}^{15} \text{RST (I, I)}.$$

***23.** Необходимо вычислить значения Y по формуле

$$Y = 41.926 \sqrt{1 + X^2} + X^{1/3} e^X$$

для следующих значений X:

$$X = 1.00,\ 1.01,\ 1.02, \ldots, 3.00$$

Каждая пара значений X и Y должна быть напечатана с помощью спецификации поля типа E 20.8.

При составлении программы используйте оператор DO.

24. Вычислите значения Z по формуле

$$Z=\frac{e^{A\cdot X}-e^{-A\cdot X}}{2}\sin(X+B)+A\ln\left(\frac{B+X}{2}\right)$$

для всех комбинаций параметров:

$$X=1.0\,(0.1)\,2.0$$
$$A=0.10\,(0.05)\,0.80$$
$$B=1.0\,(1.0)\,10.0$$

где X = 1.0(0.1)2.0 означает X = 1.0, 1.1, 1.2, . . . , 2.0 и т. д. Для каждой комбинации параметров (всего имеется 1650 комбинаций) необходимо напечатать строку значений X, A, B и Z. Для печати используйте спецификацию поля E 20.8. Напишите программу для этих вычислений, содержащую три оператора DO.

25. Необходимо найти решение следующей системы линейных уравнений:

$$\begin{aligned}
A_{11}X_1 &= B_1\\
A_{21}X_1+A_{22}X_2 &= B_2\\
&\cdots\cdots\cdots\cdots\\
A_{n1}X_1+A_{n2}X_2+\ldots+A_{nn}X_n &= B_n
\end{aligned}$$

Очевидно, что такую систему просто решить, найдя значение X_1 из первого уравнения, подставив это значение во второе уравнение и т. д. Трудность решения такой системы с помощью ЭЦВМ заключается в том, что если организовать из коэффициента этих уравнений двумерный массив, то почти половина ячеек памяти будет занята нулями; это не экономно, так как ограничивает размеры системы, которую можно решить на данной машине. Разработайте метод записи коэффициентов в виде одномерного массива и напишите программу для нахождения неизвестных. Предположите, что максимальный размер системы не превосходит 30 уравнений с 30 неизвестными, и включите в программу соответствующий оператор DIMENSION. Фактическое количество уравнений и неизвестных дается целой переменной N.

26. При тех же условиях, что и в упражнении 25, система уравнений выглядит следующим образом:

$$\begin{aligned}
A_{11}X_1+A_{12}X_2+\ldots+A_{1n}X_n &= B_1\\
A_{22}X_2+\ldots+A_{2n}X_n &= B_2\\
&\cdots\cdots\cdots\\
A_{nn}X_n &= B_n
\end{aligned}$$

Заметим, что нельзя писать оператор DO в виде

```
DO 12 I=N, 1, -1
```

что могло бы оказаться здесь очень полезным [1]). Для того чтобы работать с индексами, которые не увеличиваются, а уменьшаются, необходимо составить эквивалентный цикл с использованием оператора IF. Кроме того, при записи

```
DO 12 J=1, N
I=N-J+1
```

можно использовать переменную I в качестве индекса, который изменяется от N до 1.

***27.** Использование целой переменной в качестве индекса и коэффициента позволяет легко избежать такой ситуации, когда при многократном прибавлении к переменной x малой величины h ошибки округления накапливаются и процесс интегрирования прекращается не там, где нужно. Напишите участок программы для вычисления интеграла

$$\int_{1.0}^{5.0} \sqrt{1+x^2}\, e^{-x}\, dx$$

по формуле трапеций с $h = 0.2$.

28. То же, что и в упражнении 27, но при интегрировании используйте формулу Симпсона.

29. Напишите программу, осуществляющую вычисления из упражнения 31 гл. 6. Предусмотрите в программе, что значения x и y вводятся с перфокарт, причем каждая из них содержит одно значение x и одно значение y. В конце колоды должна находиться пробная карта, по обнаружении которой нужно прекратить вычисления. На этой карте должно быть пробито $x = 999999$. Необходимо применять метод трапеций, вычисляя h для каждого очередного интервала. (Можно было бы, конечно, вычислять h сразу для целой группы интервалов, но при этом программа излишне усложняется.)

30. Задан двумерный массив C, содержащий 10 строк и 11 столбцов. Сравните C(1,1) со всеми элементами первого столбца и определите элемент, который имеет наибольшую абсолютную величину. Присвойте некоторой переменной L значение, равное номеру строки с максимальным элементом в первом столбце. Если в результате этих операций значение L получится равным 1, то больше ничего не делайте, если же L не равно 1, то переставьте все элементы строки L в строку 1, а все элементы строки 1 в строку L, иначе говоря, поменяйте местами строки 1 и L.

***31.** Переделайте программу, приведенную на рис. 5.12, введя в нее переменные с индексами и оператор DO. Предположите, что коэффициенты многочлена заданы в виде одномерного массива A:

A(1)	A_4
A(2)	A_3

[1]) См. примечание на стр. 255.

A (3) A_2

A (4) A_1

A (5) A_0

32. То же, что и в упражнении 31, но на этот раз элементы массива организованы следующим образом:

A (1) A_0

A (2) A_1

A (3) A_2

A (4) A_3

A (5) A_4

(Напоминаем еще раз, что нулевые индексы в ФОРТРАНе не допускаются.) Для осуществления цикла можно воспользоваться тем же приемом, который предложен в конце упражнения 26.

33. Напишите программу для нахождения корня многочлена с помощью метода Ньютона — Рафсона. Входная информация задается следующим образом: первая перфокарта содержит в колонках 1 и 2 целое число, задающее степень многочлена N. В колонках 3—12 первой перфокарты содержится начальное приближение x_0 для вычисления корня по методу Ньютона — Рафсона. Все следующие карты содержат в колонках 1—10 по одному коэффициенту полинома: вторая карта содержит A_0, третья карта содержит A_1 и т. д. Всего необходимо ввести N + 1 коэффициент, причем необходимо подсчитывать их количество в процессе ввода, так как никакой пробной перфокарты в программе не предусмотрено. Коэффициенты многочлена нужно представить в виде одномерного массива; при этом можно использовать любую из двух систем индексации, предложенных в предыдущих упражнениях.

***34.** Во многих вариантах ФОРТРАНа предусмотрена возможность использования буквенных переменных, при этом можно составлять программы для печати графиков в координатах x, y.

Предположите, что ось y направлена поперек, а ось x направлена вдоль бумажной ленты, на которой печатаются выходные данные. Предположите также, что в направлении оси y имеется 41 позиция, где можно печатать точки графика. (В направлении оси x размер графика ограничивается только количеством бумаги, заряженной в печатающий аппарат.)

Образуем массив из 41 элемента под наименованием GRAPH. Поскольку элементы этого массива являются буквенными переменными, то они могут принимать любые буквенные значения, в том числе и пробел. Например, если перед началом работы программы всем элементам массива было присвоено значение пробел, а затем в программе встретился оператор

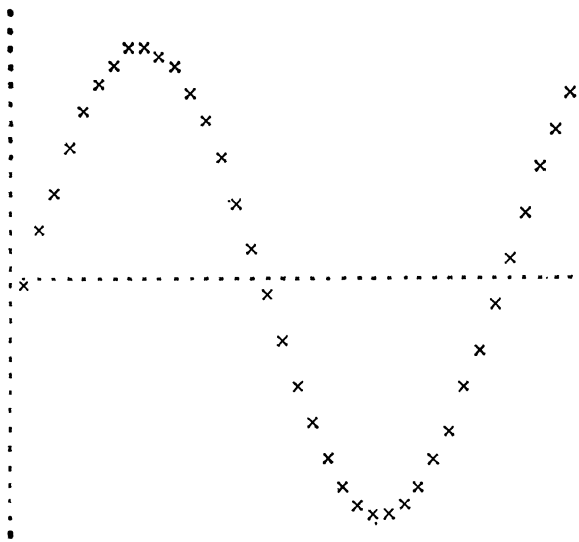
GRAPH (2) = 1HX,

то второй элемент массива примет значение X, причем при печати этого элемента на бумаге будет напечатана именно буква X. Если

теперь напечатать все элементы массива в одной строчке с помощью спецификации типа 41A1, то буква будет напечатана на второй позиции, считая слева, а на остальных 40 позициях не будет напечатано ничего, так как остальным элементам массива было присвоено значение пробел.

В качестве конкретного примера составьте программу для печати графика функции синуса. Предусмотрите в программе, что аргумент *x* должен изменяться от нуля до 7.0 с шагом 0.2. После того как будет вычислено значение синуса, необходимо вычислить номер элемента в массиве GRAPH со значением X, принимая при этом во внимание, что диапазон возможных значений синуса от —1 до +1 необходимо преобразовать в диапазон от 1 до 41. После печати очередной строчки графика необходимо очистить занятый элемент массива GRAPH, заменив его пробелом.

35. Нижеприведенная иллюстрация показывает график того типа, который был описан в упражнении 34 (с некоторым отличием в масштабе). Оси *x* и *y* изображены в виде пунктирных линий. Напишите программу для построения такого графика вместе с пунктирными линиями. Предусмотрите также некоторый способ разметки шкалы по оси *x*.



ГЛАВА

8

СИСТЕМЫ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ

8.1. ВВЕДЕНИЕ

Системы уравнений появляются почти в каждой области прикладной математики. В некоторых случаях эти системы уравнений непосредственно составляют ту задачу, которую необходимо решать, в других случаях задача сводится к такой системе. В практическом примере 10 мы увидим, например, что для проведения кривой, наилучшим образом соответствующей экспериментальным данным, приходится решать систему линейных уравнений, а в гл. 11 будут описаны методы решения уравнений в частных производных, где также требуется решать системы алгебраических уравнений. Существует множество других задач, сводящихся к решению систем алгебраических уравнений.

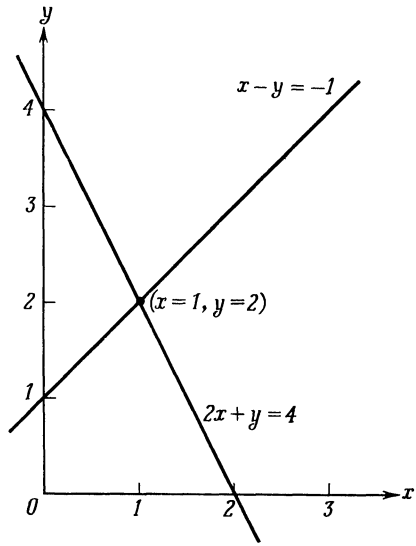
В этой главе мы будем рассматривать системы из n уравнений с n неизвестными. Каждый член такого уравнения содержит только одно неизвестное, и каждое неизвестное входит только в первой степени. Такая система уравнений называется *линейной*. В случае двух неизвестных каждое уравнение графически изображается прямой линией, в случае трех неизвестных ему соответствует плоскость в трехмерном пространстве, а для четырех и более неизвестных — гиперплоскость. Искомое решение системы уравнений представляет собой набор значений неизвестных, удовлетворяющих одновременно всем уравнениям.

Если задана некоторая произвольная система уравнений, то без предварительного исследования нельзя сказать, имеет ли она какое-либо решение и, в случае если решение существует, является ли оно единственным. На этот вопрос существуют три и только три ответа.

1. Решение системы уравнений существует и является единственным. Например,

$$\begin{cases} 2x+y=4, \\ x-y=-1. \end{cases} \tag{8.1}$$

Решение этой системы $x = 1$ и $y = 2$, никакие другие значения x и y не способны одновременно удовлетворить



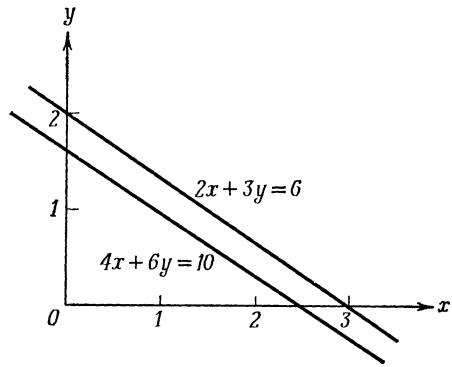
Р и с. 8.1. Геометрическое представление системы двух линейных уравнений, имеющей единственное решение.
Прямые линии, соответствующие уравнениям системы, образуют между собой довольно большой угол.

этим двум уравнениям. В этой главе мы будем в основном рассматривать именно такие системы уравнений, т. е. имеющие единственное решение. Геометрически эта система уравнений представлена на рис. 8.1, где видно, что две прямые линии, соответствующие двум уравнениям, пересекаются в одной и только в одной точке. Координаты этой точки как раз и представляют собой искомое решение.

2. Система уравнений вообще не имеет решения. Например,

$$\begin{cases} 4x+6y=10, \\ 2x+3y=6. \end{cases} \tag{8.2}$$

На рис. 8.2 показаны прямые линии, соответствующие этим двум уравнениям. Две прямые параллельны, они нигде не пересекаются, и система уравнений не имеет решения.



Р и с. 8.2. Геометрическое представление системы двух линейных уравнений, не имеющей решения.
Прямые линии параллельны.

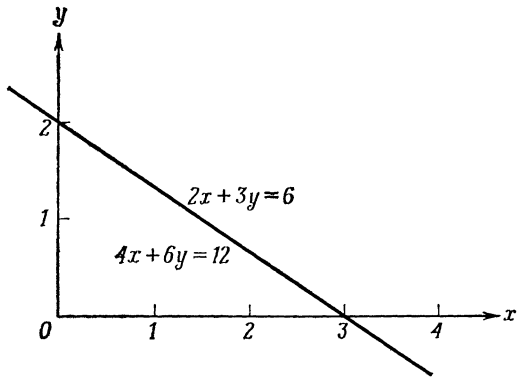
3. Система уравнений имеет бесконечное множество решений. Например,

$$\begin{cases} 4x+6y=12, \\ 2x+3y=6. \end{cases}$$

Как видно из рис. 8.3, эти два уравнения описывают одну и ту же прямую линию. Любая точка, лежащая на этой линии, является решением такой системы уравнений, например: $x=0.$, $y=2.$; $x=1.$, $y=4/3$ и т. д.

Системы уравнений типа 2 и 3 называются вырожденными. Иногда непосредственно из поставленной задачи бывает ясно, что система уравнений не может быть вырожденной. Если же, как это бывает в большинстве случаев, соответствующая информация отсутствует, то приходится или проверять

вырожденность системы уравнений в процессе решения, или исследовать такую возможность непосредственно. В разд. 8.2 мы увидим, что при решении системы уравнений методом Гаусса проверка вырожденности системы производится автоматически в процессе решения. Конечно,



Р и с. 8.3. Геометрическое представление системы двух линейных уравнений, имеющей бесконечное множество решений.
Оба уравнения изображаются одной и той же прямой линией.

непосредственная проверка состояла бы в вычислении определителя системы, тогда равенство определителя нулю прямо указывало бы на вырожденность. К сожалению, вычислить определитель ничуть не легче, чем просто решить систему уравнений.

С точки зрения обычной математики система линейных уравнений всегда является или вырожденной, или невырожденной. С точки же зрения практических вычислений могут существовать *почти* вырожденные системы, при решении которых получаются недостоверные значения неизвестных. Рассмотрим систему уравнений:

$$\begin{cases} 5x + 7y = 12, \\ 7x + 10y = 17. \end{cases} \tag{8.3}$$

Эта система имеет единственное решение $x = 1$, $y = 1$. Теперь рассмотрим пару значений неизвестных $x = 2.415$, $y = 0$. При подстановке этих значений в исходные уравне-

ния получаем

$$\begin{aligned} 5x+\ 7y &= 12.075 \\ 7x+10y &= 16.905 \end{aligned} \qquad (8.4)$$

После округления до двух значащих цифр правые части равенств (8.4) совпадают с правыми частями исходных уравнений. А так как исходные уравнения были заданы

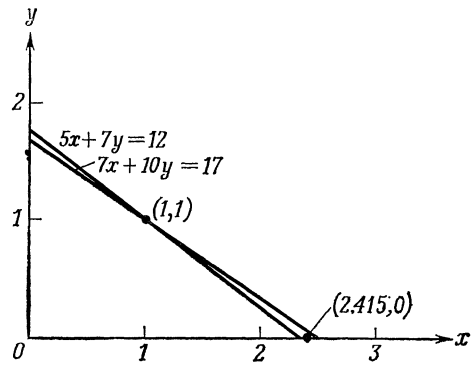


Рис. 8.4. Геометрическое представление системы двух линейных уравнений, которая является почти вырожденной. Прямые, соответствующие двум уравнениям, почти совпадают.

только с точностью до двух значащих цифр, то решение (8.4) так же хорошо отвечает условиям поставленной задачи, как и решение $x = 1$, $y = 1$.

Дело в том, что две прямые линии, описываемые двумя уравнениями этой системы, *почти* параллельны, как это показано на рис. 8.4. Точка $x = 2.415$, $y = 0$ хотя и не лежит ни на одной из этих прямых линий, но очень близка к ним.

Системы типа (8.3) называются *плохо обусловленными*. В любом случае, когда две линии (либо плоскости и гиперплоскости) почти параллельны, система уравнений становится плохо обусловленной. В этом случае найти численное решение системы трудно, а точность его весьма сомнительна. Более того, система из трех и более уравнений может оказаться плохо обусловленной, даже если никакие плоскости не являются параллельными или почти параллельны-

ми[1]). Вне зависимости от геометрической интерпретации обусловленность системы можно исследовать при ее решении методом Гаусса или с помощью специальной проверки.

Перед тем как переходить к подробному рассмотрению методов решения системы уравнений, остановимся вкратце на двух основных направлениях в этой области.

Методы численного решения системы линейных уравнений подразделяются на два типа, *прямые* (*конечные*) и *итерационные* (*бесконечные*). Первая группа методов рассматривается в разд. 8.2, вторая — в разд. 8.6. Естественно, никакой практический метод решения системы уравнений не может быть бесконечным. Мы имеем в виду только то, что прямые методы могут в принципе (с точностью до ошибок округления) дать точное решение, если оно существует, с помощью конечного числа арифметических операций. С другой стороны, итерационный метод требует бесконечного числа арифметических операций, приводящих к точному решению. Иными словами, при использовании итерационного метода появляется ошибка ограничения, отсутствующая в прямых методах.

Из сказанного выше вовсе не следует, что решение, полученное с помощью прямого метода, обязательно будет точнее, так как ошибка округления играет большую роль. При решении большой плохо обусловленной системы прямым методом ошибки округления могут привести к бессмысленным результатам. Несмотря на неизбежную ошибку ограничения, итерационный метод может оказаться наиболее удобным, так как при его использовании ошибки округления не накапливаются.

После разбора обоих методов решения систем линейных уравнений мы кратко сравним их и проанализируем ошибки. Мы увидим, что оба метода полезны и удобны для практических вычислений, а каждый из них имеет свои преимущества и недостатки.

---

[1]) Представьте себе три параллельные линии, проходящие через три вершины треугольника и перпендикулярные к его плоскости. Теперь проведите плоскость через каждую пару параллельных линий. Эти три плоскости не параллельны, но они нигде не пересекаются в одной точке. Если же одна из плоскостей слегка наклонена, то три плоскости пересекутся в одной точке, но система уравнений будет плохо обусловленной.

8.2. МЕТОД ИСКЛЮЧЕНИЯ (МЕТОД ГАУССА)

В этом разделе мы рассмотрим один из наиболее известных и широко применяемых прямых методов решения систем линейных уравнений. Обычно этот метод называют *методом исключения* или *методом Гаусса.*

Чтобы проиллюстрировать этот метод, рассмотрим сначала систему из трех уравнений с тремя неизвестными:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1, \tag{8.5}$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2, \tag{8.6}$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3. \tag{8.7}$$

В такой системе по крайней мере один из коэффициентов a_{11}, a_{21}, a_{31} должен быть отличен от нуля, иначе мы имели бы дело в этих трех уравнениях только с двумя неизвестными. Если $a_{11} = 0$, то можно переставить уравнения так, чтобы коэффициент при x_1 в первом уравнении был отличен от нуля. Очевидно, что перестановка уравнений оставляет систему неизменной: ее решение остается прежним.

Теперь введем множитель

$$m_2 = \frac{a_{21}}{a_{11}}.$$

Умножим первое уравнение системы (8.5) на m_2 и вычтем его из уравнения (8.6). («Первое» и «второе» уравнения мы берем уже после перестановки, если она была необходима.) Результат вычитания равен

$$(a_{21} - m_2a_{11})\,x_1 + (a_{22} - m_2a_{12})\,x_2 + \\ + (a_{23} - m_2a_{13})\,x_3 = b_2 - m_2b_1. \tag{8.8}$$

Но ведь

$$a_{21} - m_2a_{11} = a_{21} - \frac{a_{21}}{a_{11}}\,a_{11} = 0,$$

так что x_1 исключено из второго уравнения (конечно, именно для достижения такого результата и было выбрано значение m_2). Определим теперь новые коэффициенты

$$a'_{22} = a_{22} - m_2a_{12},$$

$$a'_{23} = a_{23} - m_2a_{13},$$

$$b'_2 = b_2 - m_2b_1.$$

Тогда уравнение (8.6) приобретает вид

$$a'_{22}x_2 + a'_{23}x_3 = b'_2. \tag{8.9}$$

Заменим второе из первоначальных уравнений (8.6) уравнением (8.9) и введем множитель для третьего уравнения

$$m_3 = \frac{a_{31}}{a_{11}}.$$

Умножим первое уравнение на этот множитель и вычтем его из третьего. Коэффициент при x_1 снова становится нулевым, и третье уравнение приобретает вид

$$a'_{32}x_2 + a'_{33}x_3 = b'_3, \tag{8.10}$$

где

$$a'_{32} = a_{32} - m_3 a_{12},$$
$$a'_{33} = a_{33} - m_3 a_{13},$$
$$b'_3 = b_3 - m_3 b_1.$$

Если теперь в исходной системе уравнений заменить (8.7) на (8.10), то новая система выглядит так:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1, \tag{8.5}$$
$$a'_{22}x_2 + a'_{23}x_3 = b'_2, \tag{8.9}$$
$$a'_{32}x_2 + a'_{33}x_3 = b'_3. \tag{8.10}$$

Эти новые уравнения полностью эквивалентны исходным уравнениям с тем преимуществом, что x_1 входит только в первое уравнение и не входит ни во второе, ни в третье. Таким образом, два последних уравнения представляют собой систему из двух уравнений с двумя неизвестными; если теперь найти решение этой системы, т. е. определить x_2 и x_3, то результат можно подставить в первое уравнение и найти x_1. Иначе говоря, задача сведена к решению системы из двух уравнений с двумя неизвестными.

Попытаемся теперь исключить x_2 из двух последних уравнений. Если $a'_{22} = 0$, то снова мы переставим уравнения так, чтобы a'_{22} было отлично от нуля. (Если же $a'_{22} = 0$

и $a'_{32} = 0$, то система вырождена и либо вовсе не имеет решения, либо имеет бесчисленное множество решений.)

Введем новый множитель

$$m'_3 = \frac{a'_{32}}{a'_{22}}.$$

Умножим уравнение (8.9) на m'_3 и вычтем его из (8.10). Результат вычитания равен

$$(a'_{32} - m'_3 a'_{22})\, x_2 + (a'_{33} - m'_3 a'_{23})\, x_3 = b'_3 - b'_2 m'_3.$$

В силу выбора m'_3

$$a'_{32} - a'_3 a'_{22} = 0.$$

Полагая

$$a''_{33} = a'_{33} - m'_3 a'_{23},$$
$$b''_3 = b'_3 - m'_3 b'_2,$$

окончательно получаем

$$a''_{33} x_3 = b''_3. \qquad (8.11)$$

Уравнение (8.10) можно заменить уравнением (8.11), после чего система уравнений приобретает следующий вид:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1, \qquad (8.5)$$
$$a'_{22}x_2 + a'_{23}x_3 = b'_2, \qquad (8.9)$$
$$a''_{33}x_3 = b''_3. \qquad (8.11)$$

Такая система уравнений иногда называется треугольной из-за своего внешнего вида.

Теперь совершенно очевидно, что надо делать для решения системы. Необходимо определить x_3 из (8.11), подставить этот результат в (8.9), определить x_2 из (8.9), подставить x_3 и x_2 в (8.5) и определить x_1. Этот процесс, который обычно называется *обратной подстановкой*, определяется в нашем случае формулами

$$x_3 = \frac{b''_3}{a''_{33}},$$

$$x_2 = \frac{b'_2 - a'_{23}x_3}{a'_{22}},$$

$$x_1 = \frac{b_1 - a_{12}x_2 - a_{13}x_3}{a_{11}}.$$

Вспомним, что мы всегда переставляли уравнения таким образом, чтобы a_{11} и a'_{22} не были равны нулю. Если же окажется, что $a''_{33} = 0$, то система уравнений вырождена.

Рассмотрим пример

$$\begin{cases} x+y+z=4, \\ 2x+3y+z=9, \\ x-y-z=-2. \end{cases} \tag{8.12}$$

Легко убедиться, что множители для второго и третьего уравнений равны соответственно 2 и 1. После исключения x из второго и из третьего уравнений новый множитель, исключающий y из третьего уравнения, равен —2. Треугольная система уравнений имеет вид

$$\begin{aligned} x+y+z&=4, \\ y-z&=1, \\ -4z&=-4. \end{aligned}$$

Из последнего уравнения $z = 1$, из второго $y = 2$, из первого $x = 1$. Можно подставить эти значения в исходные уравнения и убедиться, что они точно удовлетворяются. Таким образом мы нашли точное решение системы с помощью конечного числа арифметических операций. В данном случае ошибки округления отсутствовали.

В качестве полезного упражнения предлагаем читателям решить методом исключения систему (8.1) и попытаться решить вырожденную систему (8.2).

Теперь можно обобщить этот метод на случай системы из n уравнений с n неизвестными. После «алгебраического» описания алгоритма мы рассмотрим блок-схему программы, которая увеличивает наглядность алгоритма и позволяет непосредственно составить программу для вычислений на ЭЦВМ.

Обозначим неизвестные через $x_1, x_2, \ldots, x_n$ и запишем систему уравнений в следующем виде:

$$\begin{cases} a_{11}x_1+a_{12}x_2+\ldots+a_{1n}x_n=b_1, \\ a_{21}x_1+a_{22}x_2+\ldots+a_{2n}x_n=b_2, \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1+a_{n2}x_2+\ldots+a_{nn}x_n=b_n. \end{cases} \tag{8.13}$$

Предполагается, что в силу расположения уравнений a_{11} не равно нулю. Введем $n-1$ множителей

$$m_i = \frac{a_{i1}}{a_{11}}, \quad i = 2, 3, \ldots, n,$$

и вычтем из каждого i-го уравнения первое, домноженное на m_i. Обозначая

$$a'_{ij} = a_{ij} - m_i a_{1j}, \qquad i = 2, 3, \ldots, n,$$
$$b'_i = b_i - m_i b_1, \qquad j = 1, 2, \ldots, n,$$

легко убедиться в том, что для всех уравнений, начиная со второго,

$$a'_{i1} = 0, \quad i = 2, 3, \ldots, n.$$

Преобразованная система уравнений запишется в следующем виде:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1, \\ 0 + a'_{22}x_2 + \ldots + a'_{2n}x_n &= b'_2, \\ \ldots\ldots\ldots\ldots\ldots\ldots & \\ 0 + a'_{n2}x_2 + \ldots + a'_{nn}x_n &= b'_n. \end{aligned}$$

Продолжая таким же образом, мы можем исключить x_2 из последних $n-2$ уравнений, затем x_3 из последних $n-3$ уравнений и т. д. На некотором k-м этапе мы исключаем x_k с помощью множителей

$$m_i^{(k-1)} = \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}, \quad i = k+1, \ldots, n, \tag{8.14}$$

причем предполагается, что $a_{kk}^{(k-1)}$ не равно нулю. Тогда

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - m_i^{(k-1)} a_{kj}^{(k-1)}, \tag{8.15}$$

$$b_i^{(k)} = b_i^{(k-1)} - m_i^{(k-1)} b_k^{(k-1)} \tag{8.16}$$

для $i = k+1, \ldots, n$ и для $j = k, \ldots, n$. Индекс k принимает последовательные целые значения от 1 до $n-1$ включительно. При $k = n-1$ происходит исключение x_{n-1} из последнего уравнения.

Окончательная треугольная система уравнений записывается следующим образом:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1, \\ \qquad a'_{22}x_2 + \ldots + a'_{2n}x_n = b'_2, \\ \qquad\qquad \ldots\ldots\ldots\ldots\ldots\ldots \\ \qquad\qquad\qquad a^{n-1}_{nn}x_n = b^{(n-1)}_n. \end{cases} \tag{8.17}$$

Блок-схема программы для процесса исключения неизвестных приведена на рис. 8.5. Эта блок-схема точно соответствует разобранному выше алгебраическому процессу с двумя принципиальными различиями. Прямоугольник, где написано «переставить уравнения так, чтобы $a_{kk} \neq 0$», означает некоторый алгоритм, описанный в разд. 8.3 после разбора ошибок округления. Мы увидим, что ошибки округления могут быть существенно уменьшены, если следовать определенным правилам перестановки уравнений.

Второе различие сводится к тому, что в блок-схеме все множители обозначаются одним и тем же символом m. В самом деле, если вычисления в программе правильно организованы, то на каждом определенном этапе вычислений не требуется больше одного множителя.

При анализе блок-схемы полезно четко представлять себе смысл индексов i, j и k:

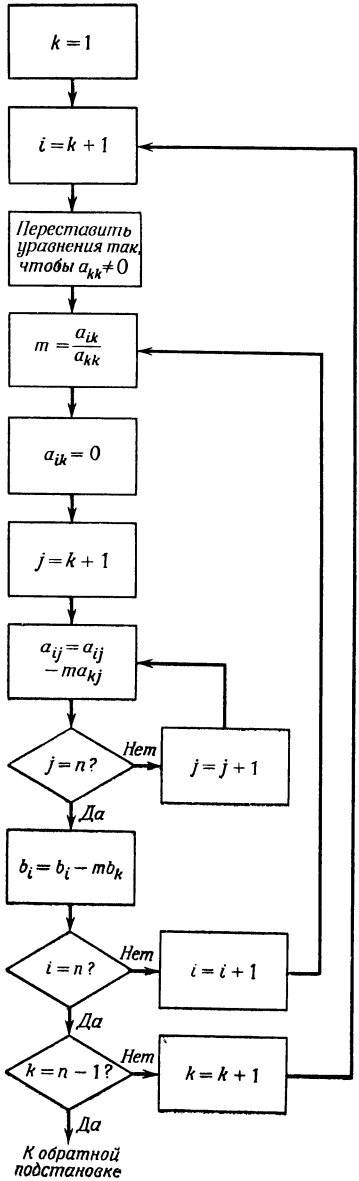
k означает номер того уравнения, которое вычитается из остальных, а также номер того неизвестного, которое исключается из оставшихся $n - k$ уравнений.

i означает номер уравнения, из которого в данный момент исключается неизвестное.

j означает номер столбца.

Мы рекомендуем тщательно проанализировать блок-схему, приведенную на рис. 8.5. Она не только позволяет ясно представить себе метод исключения, но также иллюстрирует некоторые полезные приемы, применяемые при написании блок-схемы, в частности использование индексов.

Полезно заметить, что процесс исключения неизвестных не изменяет абсолютной величины определителя системы, хотя знак определителя и изменяется при каждой перестановке уравнений. После окончания процесса значение определителя равно произведению диагональных элементов, причем знак этого произведения надо изменить на обратный.



Ф и г. 8.5. Блок-схема программы для решения системы линейных уравнений методом исключения.

если количество перестановок было нечетным. Таким образом, значение определителя системы удобно вычислять именно с помощью метода исключения.

Обратная подстановка для нахождения значений неизвестных задается формулами

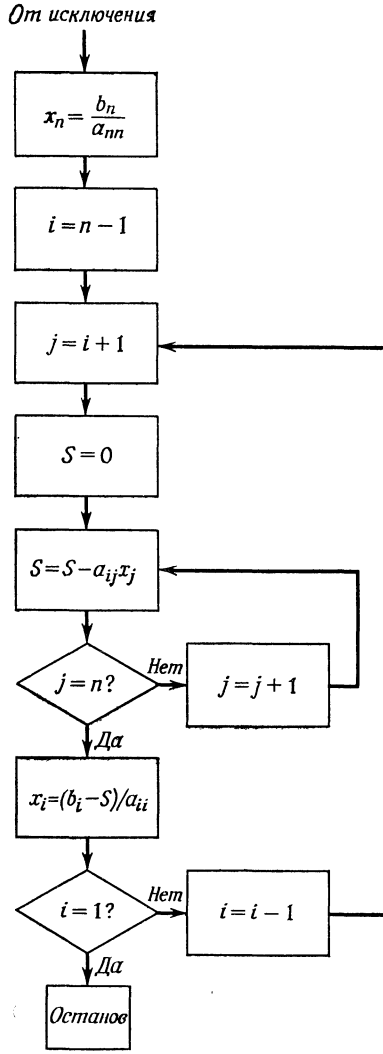
$$x_n = b_n^{(n-1)}/a_{nn}^{(n-1)},$$
$$x_{n-1} = \frac{b_{n-1}^{(n-2)} - a_{n-1,\,n}^{(n-2)} x_n}{a_{n-1,\,n-1}^{(n-2)}}, \qquad (8.18)$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$x_j = (b_j^{(j-1)} - a_{jn}^{(j-1)} x_n - \ldots - a_{j,\,j+1}^{(j-1)} x_{j+1})/a_{jj}^{(j-1)}$$
$$\text{для } j = n-2,\ n-3,\ \ldots,\ 1.$$

Блок-схема программы для обратной подстановки приведена на рис. 8.6. Она существенно проще блок-схемы метода исключения. Оказывается также, что эта блок-схема еще упрощается, если вычислять значение x_n в начале программы, не включая это вычисление в цикл. Конечно, можно составить блок-схему и без этого отдельного шага, но при этом ненужным образом усложнилось бы вычисление суммы членов, стоящих после диагонального: первая «сумма» оказалась бы равной нулю и проверку индекса j пришлось бы производить перед вычислением суммы. Заметим, что, если в блок-схеме на рис. 8.5 все индексы в процессе вычисления увеличивались, в блок-схеме на рис. 8.6 один из индексов, а именно i, уменьшается.

Программа для решения системы линейных уравнений приведена на рис. 8.15 в связи с разбором практического примера 10. Если читатель захочет рассмотреть эту программу, то ему нетрудно будет убедиться, что она довольно точно соответствует блок-схеме.

8.3. ОШИБКИ ОКРУГЛЕНИЯ

В этом разделе мы рассмотрим ошибки округления, которые возникают при вычислениях с плавающей запятой. Здесь мы зададимся целью не столько найти верхнюю границу возможной ошибки округления (хотя эту верхнюю границу и можно определить), сколько указать правило организации вычислений, которое свело бы ее к минимуму. Определение величины самой ошибки отложим до разд. 8.4,



Р и с. 8.6. Блок-схема программы для обратной подстановки при решении системы уравнений методом исключения.

где будет рассмотрен метод, позволяющий использовать ЭЦВМ не только для определения величины ошибки, но и для коррекции ошибочных результатов.

Вспомним, что перед началом процесса исключения очередного неизвестного может понадобиться переставить уравнения в системе, чтобы избежать деления на нуль. Обычно приходится делать несколько таких перестановок. В этом разделе мы покажем, как с помощью соответствующего выбора пары уравнений для перестановки можно существенно уменьшить ошибки округления. Оказывается, что обычно перестановка уравнений желательна, даже если диагональный член и не равен нулю.

Предположим, что мы собираемся исключить x_k. Система уравнений имеет следующий вид:

$$\begin{aligned}
&a_{11}x_1 + a_{12}x_2 + \ldots + a_{1k}x_k + \ldots + a_{1j}x_j + \ldots + a_{1n}x_n = b_1,\\
&a'_{22}x_2 + \ldots + a'_{2k}x_k + \ldots + a'_{2j}x_j + \\
&\qquad + \ldots + a'_{2n}x_n = b'_2,\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&a_{kk}^{(k-1)}x_k + \ldots + a_{kj}^{(k-1)}x_j + \ldots + a_{kn}^{(k-1)}x_n = b_k^{(k-1)},\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&a_{ik}^{(k-1)}x_k + \ldots + a_{ij}^{(k-1)}x_j + \ldots + a_{in}^{(k-1)}x_n = b_i^{(k-1)},\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&a_{nk}^{(k-1)}x_k + \ldots + a_{nj}^{(k-1)}x_j + \ldots + a_{nn}^{(k-1)}x_n = b_n^{(k-1)}.
\end{aligned} \tag{8.19}$$

Теперь коэффициенты $a_{ij}^{(k)}$ определяются формулой (8.15), в которой множители $m_i^{(k-1)}$ вычисляются по формуле (8.14). Граф этого процесса приведен на рис. 8.7.

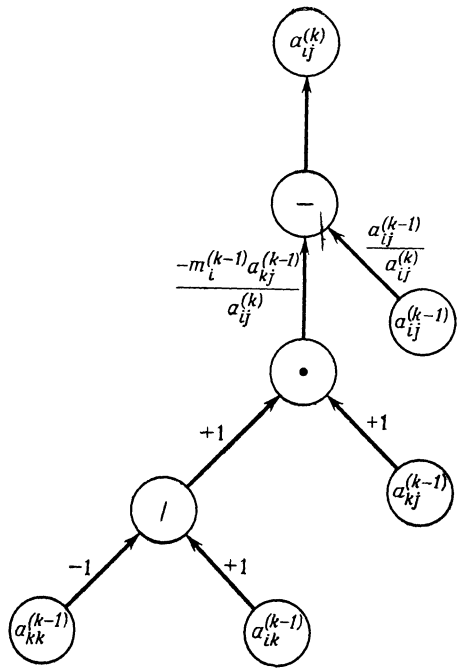
Определим теперь относительную ошибку округления при вычислении коэффициента $a_{ij}^{(k)}$. Пусть α_{ij} — относительная ошибка, содержащаяся в коэффициенте $a_{ij}^{(k-1)}$, и пусть δ, μ и σ — относительные ошибки округления соответственно при делении, умножении и вычитании. Тогда, обозначая через e_{ij} абсолютную ошибку коэффициента $a_{ij}^{(k)}$, имеем

$$\frac{e_{ij}}{a_{ij}^{(k)}} = -\frac{m_i^{(k-1)}a_{kj}^{(k-1)}}{a_{ij}^{(k)}}(\alpha_{ik} - \alpha_{kk} + \alpha_{kj} + \delta + \mu) + \frac{a_{ij}^{(k-1)}}{a_{ij}^{(k)}}\cdot\alpha_{ij} + \sigma.$$

Если δ, μ и σ не превосходят по абсолютной величине $5\cdot10^{-t}$ и α_{ij} не превосходит $K\cdot10^{-t}$, где $K\geqslant 5$, то предыдущую формулу можно переписать в следующем виде:

$$|e_{ij}|\leqslant\{3\cdot(K+5)\cdot|a_{kj}^{(k-1)}|\cdot|m_i^{(k-1)}|+10|a_{ij}^{(k-1)}|\}\cdot10^{-t}.$$

Рассмотрим теперь некоторый столбец j. Можно с уверенностью сказать, что ошибка при вычислении нового коэффициента определяется в основном первым членом, стоящим в фигурных скобках, а величина его уменьшается



Р и с. 8.7. Граф вычислительного процесса при решении системы уравнений методом исключения.

при уменьшении $|m_i^{(k-1)}|$. Поэтому желательно, чтобы $|m_i^{(k-1)}|$ было возможно меньшим. Но ведь для того, чтобы сделать $|m_i^{k-1}|$ возможно меньшим, необходимо, чтобы $|a_{kk}^{(k-1)}|$ было возможно большим. Другими словами, при

перестановке уравнений необходимо добиваться того, чтобы

$$|a_{kk}^{(k-1)}| \geqslant |a_{ik}^{(k-1)}|,$$

потому что тогда

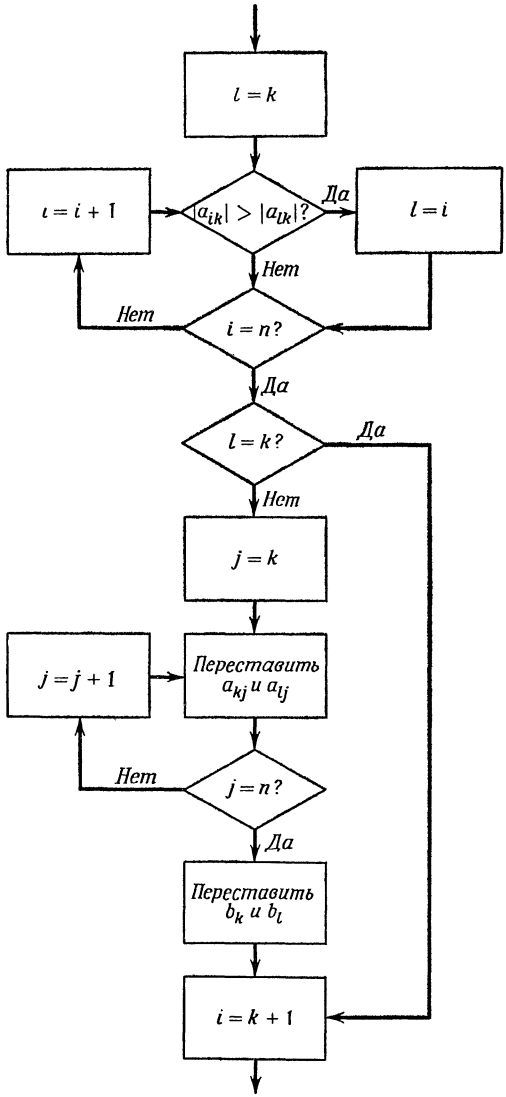
$$|m_i^{(k-1)}| \leqslant 1.$$

Если не соблюдать этого правила при перестановке уравнений, то по крайней мере один из множителей $m_i^{(k-1)}$ окажется по абсолютной величине больше 1.

Поэтому мы будем переставлять $n - k + 1$ оставшихся уравнений так, чтобы наибольший по абсолютной величине коэффициент при x_k попал на главную диагональ; в результате ни один из множителей m_i^{k-1} не превзойдет 1 по абсолютной величине. Описанный способ решения линейных уравнений часто называется методом главного элемента.

Блок-схема программы для определения наибольшего коэффициента и соответствующей перестановки уравнений приведена на рис. 8.8. Она должна входить в блок-схему из рис. 8.5 вместо прямоугольника, обозначенного звездочкой. При рассмотрении рис. 8.8 необходимо помнить, что при переходе к этому этапу вычислений индекс имеет некоторое определенное значение, а $i = k + 1$. Вначале в программу вводится вспомогательный индекс l, которому присваивается значение k. Первое сравнение производится между a_{lk}, элементом, лежащим на главой диагонали, и следующим за ним a_{ik}. Если a_{ik} окажется по абсолютной величине больше a_{lk}, то индексу l присваивается значение i и дальнейшее сравнение производится уже со вторым элементом. Поэтому индекс l является номером элемента, который оказался при сравнении больше других по абсолютной величине. Индекс i пробегает по ходу программы значения от $k + 1$ до n включительно, и в конце этого цикла индекс l определяет номер наибольшего по абсолютной величине элемента в k-м столбце. При перестановке уравнений этот элемент должен попасть на главную диагональ, т. е. должен стать элементом a_{kk}.

Может случиться так, что значение a_{kk} уже является наибольшим по абсолютной величине в k-м столбце. Поэтому такая возможность сразу проверяется и в этом случае перестановка не производится. Перед возвращением к основному процессу вычислений необходимо восстановить перво-



Р и с. 8.8. Блок-схема процесса определения наибольшего коэффициента при x_k и перестановки уравнений при необходимости.

начальное значение индекса i, который был использован при сравнении. Это делается очень просто: перед началом процесса сравнения индекс i имел значение $k + 1$. Остается только снова присвоить ему это значение.

Фактически процесс состоит в перестановке коэффициентов, один из которых содержится в уравнении k, а другой — в уравнении l, каково бы ни было значение l. Как уже указывалось в разд. 7.9, перестановка значений производится с помощью трехступенчатого процесса. Эту перестановку необходимо произвести для всех пар коэффициентов двух уравнений, что делается с помощью цикла, использующего в качестве индекса переменную j. Наконец, необходимо переставить свободные члены уравнений, и на этом процесс перестановки заканчивается.

Можно еще уменьшить ошибку округления, если переставлять не только строки, но и столбцы. Другими словами, необходимо искать наибольший по абсолютной величине коэффициент среди всех оставшихся, а не только среди коэффициентов k-го столбца. Ошибка округления в самом деле уменьшается, но за счет существенного усложнения программы: перестановка столбцов равносильна переименованию неизвестных, которое необходимо запоминать. Конечно, это можно сделать, но обычно соответствующее уменьшение ошибки округления не оправдывает усложнения программы; кроме того, такая программа работает гораздо медленнее. В упражнении 26 рассмотрен практический прием, с помощью которого можно произвести переименование неизвестных.

Чтобы проиллюстрировать практическое значение перестановки уравнений, рассмотрим следующую простую систему:

$$\begin{cases} 3.241 \cdot 10^0 x_1 + 1.600 \cdot 10^2 x_2 = 1.632 \cdot 10^2 \\ 1.020 \cdot 10^4 x_1 + 1.540 \cdot 10^3 x_2 = 1.174 \cdot 10^4 \end{cases} \tag{8.20}$$

Точное решение этой системы следующее:

$$x_1 = 1.000 \cdot 10^0$$
$$x_2 = 1.000 \cdot 10^0$$

Попытаемся решить эти уравнения методом исключения в том порядке, в котором они написаны, используя при

вычислениях числа с четырьмя значащими цифрами в мантиссе.

Первый и единственный множитель будет равен

$$m = \frac{1.020 \cdot 10^4}{3.241 \cdot 10^0} = 3.147 \cdot 10^3$$

Преобразованное второе уравнение запишется так:

$$5.730 \cdot 10^{-1} x_1 - 5.020 \cdot 10^5 x_2 = -5.019 \cdot 10^5$$

Естественно, коэффициент при x_1 должен был бы быть равен нулю, но ошибка округления не позволяет получить точный результат. Так как этот коэффициент нигде не входит в дальнейшие вычисления, то мы принимаем его равным нулю и производим обратную подстановку

$$x_2 = \frac{-5.019 \cdot 10^5}{-5.020 \cdot 10^5} = 9.998 \cdot 10^{-1} \qquad (8.21)$$

Из первого уравнения получаем

$$x_1 = 9 \cdot 873 \cdot 10^{-1}. \qquad (8.22)$$

Теперь переставим уравнения, как это потребовалось бы согласно описанному правилу. Теперь множитель равен

$$m = \frac{3.241 \cdot 10^0}{1.020 \cdot 10^4} = 3.177 \cdot 10^{-4}$$

Преобразованное второе уравнение запишется так:

$$0.000 x_1 + 1.595 \cdot 10^2 x_2 = 1.595 \cdot 10^2$$

откуда

$$x_2 = 1.000 \cdot 10^0$$

и

$$x_1 = 1.000 \cdot 10^0$$

В этом примере коэффициенты сильно отличались по величине, но чаще всего различия эти не так велики. Однако при решении большой системы уравнений ошибки округления могут накапливаться (система из 10 уравнений представляет собой обычное явление, а система из 100 уравнений тоже встречается довольно часто). Первая же перестановка в очень большой степени влияет на точность последующих вычислений. Нетрудно привести

примеры сравнительно малых систем, при решении которых без перестановки уравнений ответы получаются совершенно неверными, не имеющими ни одной правильной значащей цифры. Любая практическая программа для решения системы линейных уравнений на ЭЦВМ методом исключения должна предусматривать перестановку уравнений, кроме, может быть, некоторых весьма специальных случаев, когда заранее можно предсказать поведение коэффициентов в процессе вычислений.

8.4. УТОЧНЕНИЕ РЕШЕНИЯ

Независимо от того, производится при решении системы перестановка уравнений или нет, ошибка округления обычно влияет на результат. В этом разделе мы рассмотрим метод уточнения уже найденного решения системы. Это позволит в большинстве случаев уменьшить влияние ошибок округления на окончательный результат; может оказаться даже, что в некоторых случаях получится правдоподобное решение некоторых плохо обусловленных систем.

Обозначим через $x_1^{(0)}$, $x_2^{(0)}$, $x_3^{(0)}$, $x_4^{(0)}$, ..., $x_n^{(0)}$ приближенное решение уравнения (8.13), найденное с помощью метода исключения или некоторым другим способом. Подставляя это приближенное решение в левые части системы (8.13), получим

$$\begin{cases} a_{11}x_1^{(0)} + a_{12}x_2^{(0)} + \ldots + a_{1n}x_n^{(0)} = b_1^{(0)}, \\ a_{21}x_1^{(0)} + a_{22}x_2^{(0)} + \ldots + a_{2n}x_n^{(0)} = b_2^{(0)}, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1^{(0)} + a_{n2}x_2^{(0)} + \ldots + a_{nn}x_n^{(0)} = b_n^{(0)}. \end{cases} \tag{8.23}$$

Если $b_i^{(0)}$ сильно отличаются от b_i, то $x_i^{(0)}$ не являются достаточно хорошим приближением к решению уравнения. С другой стороны, даже если все b_i^0 достаточно близки к b_i, то x_i^0 все равно могут явиться плохим приближением к решению системы (см. пример (8.3) в разд. 8.1).

Если вычесть каждое уравнение системы (8.23) из соответствующего уравнения системы (8.13) и обозначить

$$\varepsilon_i^{(0)} = x_i - x_i^{(0)}, \quad i = 1, \ldots, n, \tag{8.24}$$

$$\beta_i^{(0)} = b_i - b_i^{(0)}, \quad i = 1, \ldots, n, \tag{8.25}$$

то можно написать

$$\begin{cases} a_{11}\varepsilon_1^{(0)} + a_{12}\varepsilon_2^{(0)} + \ldots + a_{1n}\varepsilon_n^{(0)} = \beta_1^{(0)}, \\ a_{21}\varepsilon_1^{(0)} + a_{22}\varepsilon_2^{(0)} + \ldots + a_{2n}\varepsilon_n^{(0)} = \beta_2^{(0)}, \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}\varepsilon_1^{(0)} + a_{n2}\varepsilon_2^{(0)} + \ldots + a_{nn}\varepsilon_n^{(0)} = \beta_n^{(0)} \end{cases} \tag{8.26}$$

Легко вычислить величины β_i^0 и определить ε_i^0 из системы (8.26) методом исключения. Новое приближение к решению системы (8.13) запишется так:

$$x_i^{(1)} = x_i^{(0)} + \varepsilon_i^{(0)}, \quad i = 1, 2, \ldots, n.$$

Теперь снова можно подставить $x_i^{(1)}$ в левые части уравнений (8.13) и обозначить через $b_i^{(1)}$ значения свободных членов, которые будут вычислены в результате этой подстановки.

Новую поправку к решению системы (8.13) можно получить из системы уравнений

$$\begin{cases} a_{11}\varepsilon_1^{(1)} + a_{12}\varepsilon_2^{(1)} + \ldots + a_{1n}\varepsilon_n^{(1)} = \beta_1^{(1)}, \\ a_{21}\varepsilon_1^{(1)} + a_{22}\varepsilon_2^{(1)} + \ldots + a_{2n}\varepsilon_n^{(1)} = \beta_2^{(1)}, \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}\varepsilon_1^{(1)} + a_{n2}\varepsilon_2^{(1)} + \ldots + a_{nn}\varepsilon_n^{(1)} = \beta_n^{(1)}, \end{cases} \tag{8.27}$$

где

$$\beta_i^{(1)} = b_i - b_i^{(1)}, \quad i = 1, 2, \ldots, n.$$

Новое приближение запишется после этого в следующем виде:

$$x_i^{(2)} = x_i^{(1)} + \varepsilon_i^{(1)}, \quad i = 1, 2, \ldots, n.$$

Естественно, такой процесс можно продолжать до тех пор, пока все ε_i не станут достаточно малыми. Важно отметить, что нельзя останавливать вычисления только потому, что все β_i стали малыми; тот же пример (8.3) ясно показывает, что при малых β_i решение системы вовсе не обязано быть точным.

В качестве примера рассмотрим систему (8.20), решенную без перестановки уравнений. В качестве первого приближения были получены следующие величины:

$$x_1^{(0)} = 9.873 \cdot 10^{-1}; \quad x_2^{(0)} = 9.998 \cdot 10^{-1}$$

Подставляя эти значения в (8.20) и вычисляя все величины с точностью до четырех значащих цифр, получим

$$b_1^{(0)} = 1.632 \cdot 10^2$$
$$b_2^{(0)} = 1.161 \cdot 10^4$$

так что

$$\beta_1^{(0)} = b_1 - b_1^{(0)} = 0$$
$$\beta_2^{(0)} = b_2 - b_2^{(0)} = 1.300 \cdot 10^2$$

Теперь решим систему (8.20), подставив в качестве правых частей $\beta_1^{(0)}$ и $\beta_2^{(0)}$, т. е.

$$3.241 \cdot 10^0 \varepsilon_1^{(0)} + 1.600 \cdot 10^2 \varepsilon_2^{(0)} = 0$$
$$1.020 \cdot 10^4 \varepsilon_1^{(0)} + 1.540 \cdot 10^3 \varepsilon_2^{(0)} = 1.300 \cdot 10^2$$

Снова, не прибегая к перестановке уравнений, получим

$$\varepsilon_1^{(0)} = 1.284 \cdot 10^{-2}$$
$$\varepsilon_2^{(0)} = -2.600 \cdot 10^{-4}$$

так что новое приближение будет равно

$$x_1^{(1)} = x_1^{(0)} + \varepsilon_1^{(0)} = 1.000 \cdot 10^0$$
$$x_2^{(1)} = x_2^{(0)} + \varepsilon_2^{(0)} = 9.995 \cdot 10^{-1}$$

Подставляя эти значения в (8.20), найдем

$$\beta_1^{(1)} = b_1 - b_1^{(1)} = 1.000 \cdot 10^{-1}$$
$$\beta_2^{(1)} = b_2 - b_2^{(1)} = 0$$

так что

$$\varepsilon_1^{(1)} = -9.256 \cdot 10^{-5}$$
$$\varepsilon_2^{(1)} = 6.268 \cdot 10^{-4}$$

Следующее приближение будет равно

$$x_1^{(2)} = x_1^{(1)} + \varepsilon_1^{(1)} = 1.000 \cdot 10^0$$
$$x_2^{(2)} = x_2^{(1)} + \varepsilon_2^{(1)} = 1.000 \cdot 10^0$$

Таким образом, удалось получить точное решение с помощью трех итераций. (Укажем, что при перестановке уравнений достаточно одной итерации.)

Важно отметить, что удалось существенно уточнить решение, даже несмотря на значительную величину ошибок округления. Естественно, при практических вычислениях производится перестановка уравнений. Основной смысл описанного выше метода состоит в уточнении решения, полученного с использованием перестановки уравнений, но тем не менее содержащего ошибки округления.

8.5. ВЛИЯНИЕ ПОГРЕШНОСТЕЙ КОЭФФИЦИЕНТОВ. ДОСТИЖИМАЯ ТОЧНОСТЬ РЕШЕНИЯ

Зададимся вопросом: если a_{ij} и b_i являются экспериментальными данными или вычисленными величинами, то каково будет влияние погрешностей этих величин на решение системы уравнений?

Чтобы получить ответ на этот важный вопрос, мы воспользуемся тем же методом, что и в гл. 5. Пусть истинные значения коэффициентов равны $a_{ij}+\alpha_{ij}$ и $b_i+\beta_i$, причем α_{ij} и β_i малы по сравнению с a_{ij} и b_i. Иначе говоря, мы хотим в действительности получить решение системы

$$\begin{cases} (a_{11}+\alpha_{11})\,x_1+\ldots+(a_{1n}+\alpha_{1n})\,x_n=b_1+\beta_1, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ (a_{i1}+\alpha_{i1})\,x_1+\ldots+(a_{in}+\alpha_{in})\,x_n=b_i+\beta_i, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ (a_{n1}+\alpha_{n1})\,x_1+\ldots+(a_{nn}+\alpha_{nn})\,x_n=b_n+\beta_n. \end{cases} \tag{8.28}$$

Конечно, если бы мы точно знали значения α_{ij} и β_i, то мы просто решили бы уравнения (8.28) вместо исходных. Однако обычно мы знаем только верхнюю границу возможной величины α_{ij} и β_i. Например, если a_{ij} и b_i являются экспериментальными данными, известными с точностью до d знаков после запятой, то

$$|\alpha_{ij}|\leqslant\frac{1}{2}\cdot 10^{-d};\quad |\beta_i|\leqslant\frac{1}{2}\cdot 10^{-d}.$$

Предположим, что $x_1, x_2, x_3, \ldots, x_n$ — решение исходной системы (8.13), и обозначим через

$$x_i+\delta_i$$

решение системы (8.28). Предположим также, что

$$|\delta_i| \ll |x_i|, \quad i = 1, 2, \ldots, n.$$

Подставляя $x_i + \delta_i$ в (8.28) и используя равенства (8.13), находим, что

$$\begin{aligned}
&(a_{11}\delta_1 + \ldots + a_{1n}\delta_n) + (\alpha_{11}x_1 + \ldots + \alpha_{1n}x_n) = \beta_1,\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&(a_{i1}\delta_1 + \ldots + a_{in}\delta_n) + (\alpha_{i1}x_1 + \ldots + \alpha_{in}x_n) = \beta_i,\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&(a_{n1}\delta_1 + \ldots + a_{nn}\delta_n) + (\alpha_{n1}x_1 + \ldots + \alpha_{nn}x_n) = \beta_n,
\end{aligned}$$

где мы пренебрегаем произведениями α_{ij} на δ_i из-за их малости. Теперь δ_i можно получить из системы уравнений

$$\left\{\begin{aligned}
&(a_{11}\delta_1 + \ldots + a_{1n}\delta_n) = \beta_1 - (\alpha_{11}x_1 + \ldots + \alpha_{1n}x_n),\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&a_{i1}\delta_1 + \ldots + a_{in}\delta_n = \beta_i - (\alpha_{i1}x_1 + \ldots + \alpha_{in}x_n),\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&a_{n1}\delta_1 + \ldots + a_{nn}\delta_n = \beta_n - (\alpha_{n1}x_1 + \ldots + \alpha_{nn}x_n).
\end{aligned}\right. \tag{8.29}$$

Рассмотрим четыре возможных случая:

1. Все a_{ij} известны точно, а все b_i заданы с точностью d знаков после запятой, т. е.

$$\alpha_{ij} = 0,$$

$$|\beta_i| \leqslant \frac{1}{2}\cdot 10^{-d}.$$

Тогда из (8.29) следует

$$\left\{\begin{aligned}
&a_{11}\delta_1 + \ldots + a_{1n}\delta_n \leqslant \frac{1}{2}\cdot 10^{-d},\\
&\ldots\ldots\ldots\ldots\ldots\\
&a_{i1}\delta_1 + \ldots + a_{in}\delta_n \leqslant \frac{1}{2}\cdot 10^{-d},\\
&\ldots\ldots\ldots\ldots\ldots\\
&a_{n1}\delta_1 + \ldots + a_{nn}\delta_n \leqslant \frac{1}{2}\cdot 10^{-d}.
\end{aligned}\right. \tag{8.30}$$

Эти неравенства можно «решить» (ниже мы покажем, как это делается). «Решение» неравенств производится одновременно с решением исходных уравнений (8.13), и

в результате получается прямая оценка влияния погрешности правых частей уравнений на достижимую точность результата.

Заметим сначала, что если набор значений Δ_i ($i = 1, 2, \ldots, n$) удовлетворяет неравенствам (8.30), где все правые части равны $+1$, то можно написать

$$\delta_i = \left(\frac{1}{2}\cdot 10^{-d}\right)\Delta_i,\ i = 1, \ldots, n.$$

Таким образом, для оценки влияния погрешности коэффициентов необходимо одновременно с основной решать еще одну систему, где все правые части равны $+1$. Процесс ее решения тот же, что и для системы, с одной разницей: в правой части производится не вычитание, а сложение абсолютных значений. Причина этого видоизменения состоит в том, что формулы (8.30) представляют собой неравенства и, преобразовывая их, нельзя допускать уменьшения правых частей.

Рассмотрим пример. Возьмем уравнения (8.12) и предположим, что их правые части известны с точностью до двух знаков после запятой, т. е.

$$|\beta_i| \leqslant \frac{1}{2}\cdot 10^{-2}.$$

При исключении x из второго уравнения получим

$$(2-2\cdot 1)x+(3-2\cdot 1)y+(1-2\cdot 1)z = 9-2\cdot 4\,|\,1+2\cdot 1$$

или

$$y-z = 1\,|\,3.$$

Число после вертикальной черты является соответствующим элементом дополнительного столбца. Заметим, что $2\cdot 1$ было *прибавлено*, а не вычтено из правой части второго уравнения, так как при вычислении ошибки всегда необходимо складывать абсолютные значения. Аналогично преобразовываем третье уравнение

$$(1-1\cdot 1)x+(-1-1\cdot 1)y+(-1-1\cdot 1)z = -2-1\cdot 4\,|\,1+1\cdot 1$$

или

$$-2y-2z = -6\,|\,2.$$

Окончательная треугольная система уравнений запишется так:

$$\begin{aligned} x+y+z &= 4 \mid 1, \\ y-z &= 1 \mid 3, \\ -4z &= -4 \mid 8. \end{aligned}$$

При решении этой треугольной системы с помощью обратной подстановки также необходимо складывать абсолютные значения. В результате получаем

$$\Delta_z = \frac{8}{4} = 2,$$

$$\Delta_y = 3 + \Delta_z = 5,$$

$$\Delta_x = 1 + \Delta_y + \Delta_z = 8,$$

так что

$$\begin{cases} \delta_x \leqslant 4 \cdot 10^{-2} \\ \delta_y \leqslant 2.5 \cdot 10^{-2} \\ \delta_z \leqslant 1 \cdot 10^{-2} \end{cases} \tag{8.31}$$

где

$$x = 1,$$

$$y = 2,$$

$$z = 1.$$

2. Все b_i заданы точно, но a_{ij} заданы с точностью до d знаков после запятой, т. е.

$$|\alpha_{ij}| \leqslant \frac{1}{2} \cdot 10^{-d},$$

$$\beta_i = 0.$$

Тогда из (8.29) и из неравенства треугольника следует

$$a_{11}\delta_1 + \ldots + a_{1n}\delta_n \leqslant \frac{1}{2} \cdot 10^{-d} (|x_1| + \ldots + |x_n|),$$

. . . . . .

$$a_{i1}\delta_1 + \ldots + a_{in}\delta_n \leqslant \frac{1}{2} \cdot 10^{-d} (|x_1| + \ldots + |x_n|),$$

. . . . . .

$$a_{n1}\delta_1 + \ldots + a_{nn}\delta_n \leqslant \frac{1}{2} \cdot 10^{-d} (|x_1| + \ldots + |x_n|).$$

И снова если Δ_i — решения системы при правых частях, равных $+1$, то можно написать

$$\delta_i = \frac{1}{2} \cdot 10^{-d} (|x_1| + \ldots + |x_n|) \Delta_i.$$

В качестве примера рассмотрим снова систему (8.12), но на этот раз предположим, что коэффициенты в левой части известны с точностью до двух знаков после запятой. Тогда, учитывая, что

$$|x| + |y| + |z| = 4,$$

можно написать на основании формулы (8.31)

$$\delta_x \leqslant 1.6 \cdot 10^{-1}$$
$$\delta_y \leqslant 1 \cdot 10^{-1}$$
$$\delta_z \leqslant 4 \cdot 10^{-2}$$

3. Предположим, что все a_{ij} и все b_i известны с точностью до d знаков после запятой. Вычисления, аналогичные случаям 1 и 2, дают результаты

$$\delta_i = \frac{1}{2} \cdot 10^{-d} (1 + |x_1| + \ldots + |x_n|) \Delta_i.$$

Если предположить, что все числа в системе (8.12) известны с точностью до двух знаков после запятой, то

$$\delta_x \leqslant 2 \cdot 10^{-1}$$
$$\delta_y \leqslant 1.25 \cdot 10^{-1}$$
$$\delta_z \leqslant 5 \cdot 10^{-2}$$

4. Наконец, рассмотрим случай, когда a_{ij} и b_i известны с точностью до t значащих цифр. Тогда

$$\left| \frac{\alpha_{ij}}{a_{ij}} \right| \leqslant 5 \cdot 10^{-t} \text{ и } \left| \frac{\beta_i}{b_i} \right| \leqslant 5 \cdot 10^{-t}.$$

Тогда вычисление максимально возможной ошибки приводит к следующему результату:

$$\delta_i = k \cdot \Delta_i \cdot 5 \cdot 10^{-t},$$

где

$$k = \max (|b_i| + |a_{i1}x_1| + \ldots + |a_{in}x_n|)$$

и отыскивается максимальное значение для всех i от 1 до n.

Снова обратимся к примеру (8.12) и предположим, что все числа известны с точностью до четырех значащих цифр, т. е. $t = 4$. Тогда

$$|b_1| + |a_{11}x| + |a_{12}y| + |a_{13}z| = 4 + 1 + 2 + 1 = 8,$$
$$|b_2| + |a_{21}x| + |a_{22}y| + |a_{23}z| = 9 + 2 + 6 + 1 = 18,$$
$$|b_3| + |a_{31}x| + |a_{32}y| + |a_{33}z| = 2 + 1 + 2 + 1 = 6.$$

Максимальное значение равно 18, так что

$$\delta_x \leqslant 7.2 \cdot 10^{-2}$$
$$\delta_y \leqslant 4.5 \cdot 10^{-2}$$
$$\delta_z \leqslant 1.8 \cdot 10^{-2}$$

Все вычисленные здесь верхние пределы для ошибок являются обычно слишком грубыми оценками; как правило, фактические ошибки много меньше этих верхних пределов. Можно найти более точные оценки верхних пределов — они окажутся меньше, чем найденные в этом параграфе, — но нахождение таких верхних пределов требует основательного знакомства с матричной алгеброй, в частности с правилами обращения матриц. Интересующимся читателям можно рекомендовать книгу Гильдебранда, стр. 436—437.

8.6. ИТЕРАЦИОННЫЕ МЕТОДЫ РЕШЕНИЯ СИСТЕМ ЛИНЕЙНЫХ УРАВНЕНИЙ

В разд. 8.2 мы рассмотрели метод исключения (конечный метод), с помощью которого можно решить систему линейных алгебраических уравнений. Затем в разд. 8.4 мы увидели, как можно уточнить решение, повторно используя метод исключения. В сущности, этот метод уточнения решения является итерационным. Таким образом, и здесь проявляется обычное противоречие между теорией и практикой вычислений на ЭЦВМ — благодаря ошибкам округления решение, получаемое с помощью конечного и точного метода, может сильно отличаться от истинного,

с помощью же итерационного метода это решение можно существенно улучшить.

В этом разделе мы рассмотрим другой, более обычный итерационный метод, отличающийся простотой и легкостью программирования. Подобно итерационным методам из гл. 5, он отличается малой ошибкой округления, но в то же время сходится только при выполнении некоторых условий, которые мы также рассмотрим.

Оказывается, что условия сходимости этого итерационного метода очень часто выполняются тогда, когда с помощью определенных приемов решаются уравнения в частных производных. К этому вопросу мы еще вернемся в гл. 11.

Рассмотрим систему из трех уравнений с тремя неизвестными (8.5), (8.6) и (8.7). Предположим, что $a_{11} \neq 0$, $a_{22} \neq 0$, $a_{33} \neq 0$, и перепишем систему в следующем виде:

$$x_1 = \frac{1}{a_{11}}(b_1 - a_{12}x_2 - a_{13}x_3), \tag{8.32}$$

$$x_2 = \frac{1}{a_{22}}(b_2 - a_{21}x_1 - a_{23}x_3), \tag{8.33}$$

$$x_3 = \frac{1}{a_{33}}(b_3 - a_{31}x_1 - a_{32}x_2). \tag{8.34}$$

Теперь возьмем некоторое первое приближение к решению этой системы, обозначив его через x_1^0, x_2^0, x_3^0. Подставим это решение в (8.32) и вычислим новое значение x_1:

$$x_1^{(1)} = \frac{1}{a_{11}}(b_1 - a_{12}x_2^0 - a_{13}x_3^0).$$

Используя только что вычисленное значение $x_1^{(1)}$ и начальное значение x_3^0, вычислим из уравнения (8.33) новое значение x_2:

$$x_2^{(1)} = \frac{1}{a_{22}}(b_2 - a_{21}x_1^1 - a_{23}x_3^0).$$

Наконец, используя только что вычисленные значения $x_1^{(1)}$ и $x_2^{(1)}$, найдем из (8.34) новое значение x_3:

$$x_3^{(1)} = \frac{1}{a_{33}}(b_3 - a_{31}x_1^{(1)} - a_{32}x_2^{(1)}).$$

Этим заканчивается первая итерация. Теперь можно заменить исходные значения $x_1^{(0)}$, $x_2^{(0)}$ и $x_3^{(0)}$ на $x_1^{(1)}$, $x_2^{(1)}$, $x_3^{(1)}$ и вычислить следующее приближение. В общем слу-

чае k-е приближение определяется формулами

$$\begin{cases} x_1^{(\ )} = \dfrac{1}{a_{11}}(b_1 - a_{12}x_2^{(k-1)} - a_{13}x_3^{(k-1)}), \\ x_2^{(k)} = \dfrac{1}{a_{22}}(b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k-1)}), \\ x_3^{(k)} = \dfrac{1}{a_{33}}(b_3 - a_{31}x_1^{(k)} - a_{32}x_2^{(k)}). \end{cases} \tag{8.35}$$

Заметим, что текущие значения неизвестных сразу же используются для последующих вычислений и что, например, нельзя вычислять $x_2^{(k)}$, пока не получено $x_1^{(k)}$. Аналогично этому для вычисления $x_3^{(k)}$ необходимо сначала определить $x_1^{(k)}$ и $x_2^{(k)}$.

Только что описанный здесь метод известен как *итерационный метод Гаусса — Зейделя*. Этот метод исключительно удобен для использования на ЭЦВМ. Перед обобщением на случай n уравнений с n неизвестными и выводом условия сходимости метода рассмотрим простой численный пример

$$\begin{aligned} 4x_1 - x_2 + x_3 &= 4, \\ x_1 + 6x_2 + 2x_3 &= 9, \\ -x_1 - 2x_2 + 5x_3 &= 2. \end{aligned}$$

Нетрудно убедиться, что точное решение этой системы равно $x_1 = 1$, $x_2 = 1$, $x_3 = 1$. Положим $x_1^0 = x_2^0 = x_3^0 = 0$, как это обычно делается для начального приближения. Тогда, вычисляя согласно формулам

$$\begin{aligned} x_1 &= \frac{1}{4}(4 + x_2 - x_3), \\ x_2 &= \frac{1}{6}(9 - x_1 - 2x_3), \\ x_3 &= \frac{1}{5}(2 + x_1 + 2x_2), \end{aligned}$$

получаем следующее приближение:

$$\begin{aligned} x_1^{(1)} &= 1, \\ x_2^{(1)} &= \frac{4}{3}, \\ x_3^{(1)} &= \frac{17}{15}. \end{aligned}$$

Последовательные приближения, вычисленные каждый раз с точностью до четырех значащих цифр, приведены в табл. 8.1.

Таблица 8.1

Итерация	x_1	x_2	x_3
0	0	0	0
1	$0.1000 \cdot 10^1$	$0.1333 \cdot 10^1$	$0.1133 \cdot 10^1$
2	$0.1050 \cdot 10^1$	$0.9473 \cdot 10^0$	$0.9889 \cdot 10^0$
3	$0.9896 \cdot 10^0$	$0.1005 \cdot 10^1$	$0.9999 \cdot 10^0$
4	$0.1001 \cdot 10^1$	$0.9999 \cdot 10^0$	$0.1000 \cdot 10^1$
5	$0.1000 \cdot 10^1$	$0.1000 \cdot 10^1$	$0.1000 \cdot 10^1$

Интересно отметить, что вычисления с восемью значащими цифрами приводят к такой же точности снова за пять итераций. Однако следующие четыре итерации дают уже решение с точностью до восьми значащих цифр.

Рассмотрим теперь систему (8.13) из n уравнений с n неизвестными. Мы по-прежнему предполагаем, что диагональные коэффициенты a_{ij} отличны от нуля для всех i. Тогда k-е приближение к решению будет задаваться формулой

$$x_i^{(k)} = \frac{1}{a_{ii}} (b_i - a_{i1}x_1^{(k)} - \ldots - a_{i,\,i-1}x_{i-1}^{(k)} - a_{i,\,i+1}x_{i+1}^{(k-1)} - \ldots \\ \ldots - a_{in}x_n^{(k-1)}), \quad i = 1, 2, \ldots, n.$$

Итерационный процесс продолжается до тех пор, пока все $x_i^{(k)}$ не станут достаточно близки к $x_i^{(k-1)}$. Критерий близости можно, например, задать в следующем виде:

$$M^{(k)} = \max | x_i^{(k)} - x_i^{(k-1)} | < \varepsilon,$$

где определяется максимальное значение разности для всех i, а ε — некоторое положительное число. При выполнении критерия итерационный процесс следует остановить. Можно заменить этот критерий близости, сравнивая с ε не абсолютные, а относительные разности

$$\max \left| \frac{x_i^{(k)} - x_i^{(k-1)}}{x_i^{(k)}} \right| < \varepsilon.$$

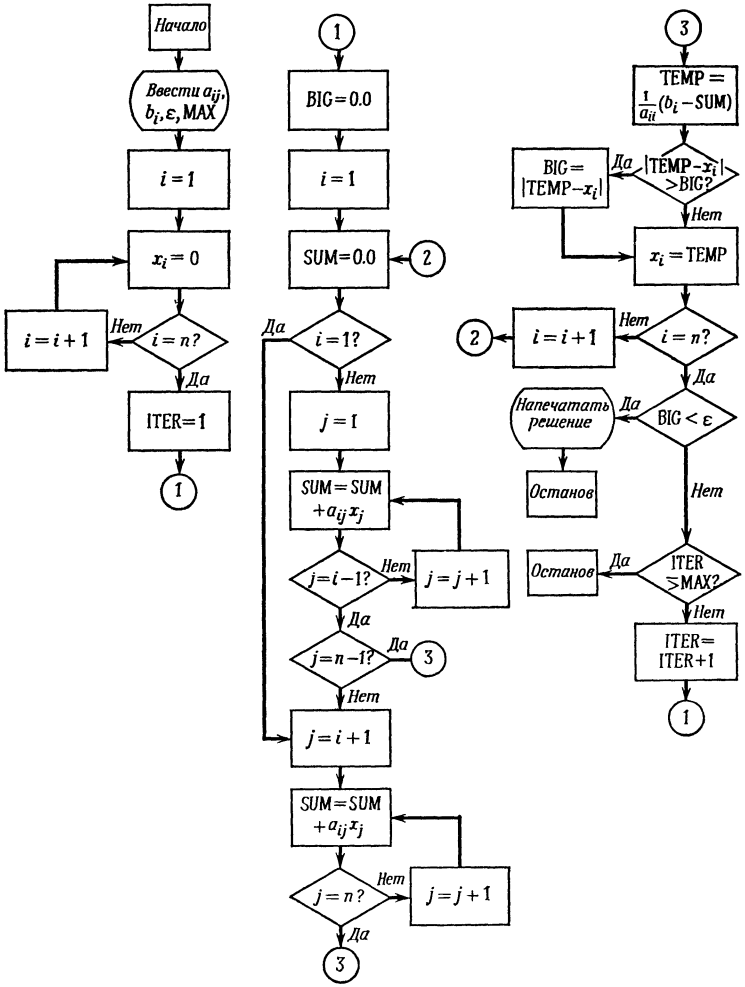
Блок-схема программы для вычислений по методу Гаусса — Зейделя приведена на рис. 8.9. Она вовсе не так сложна, как это может показаться с первого взгляда. Вспомним хотя бы, что блок-схема для метода исключения заняла три рисунка — 8.5, 8.6 и 8.8.

В левой части рисунка изображены действия, связанные со вводом исходных данных и подготовкой к вычислениям. В этой части рисунка указано, что начальное приближение принимается равным нулю, а счетчику количества итераций присваивается значение 1. Как обычно, для подсчета количества итераций используется целая переменная ITER. При дальнейших вычислениях эти начальные действия не повторяются. В конце этой части блок-схемы стоит кружок с цифрой внутри, так называемая связка. Эта связка означает, что блок-схема продолжается, начиная с кружка, содержащего ту же цифру, в нашем случае со средней части рисунка, где вверху поставлен кружок с цифрой 1. Связки часто используются в блок-схемах, чтобы не проводить длинных пересекающихся линий.

Переменная BIG используется для того, чтобы определять наибольшее значение разности между $x_i^{(k)}$ и $x_i^{(k-1)}$. Сначала этой переменной присваивается значение нуль, а затем с ней сравниваются абсолютные значения разностей $|x_i^{(k)} - x_i^{(k-1)}|$. Если какая-либо разность оказывается по абсолютной величине больше BIG, то прежнее значение BIG заменяется этой разностью. После вычисления всех $x_i^{(k)}$ наибольшая разность будет равна значению переменной BIG.

Затем в блок-схеме следует группа действий, с помощью которых вычисляется сумма всех членов уравнения, кроме диагонального. По ходу программы удобнее сначала вычислять и суммировать члены, стоящие перед диагональным (так как в них используются новые значения $x_i^{(k)}$), а уже потом — члены, идущие после него (так как в них используются старые значения $x_i^{(k-1)}$). Необходимо также несколько усложнить эту логику в связи с тем, что в первом и последнем уравнениях отсутствуют члены, стоящие соответственно до и после диагонального.

В правой части блок-схемы указана группа действий, начинающаяся с вычисления нового значения $x_i^{(k)}$; опреде-



Р и с. 8.9. Блок-схема программы для решения системы линейных уравнений итерационным методом Гаусса — Зейделя.

ляется максимальная разность и запоминается новое значение $x_i^{(k)}$. Если это было не последнее уравнение, то мы увеличиваем индекс i и начинаем вычислять очередное

приближение для следующего неизвестного. В случае последнего уравнения мы сравниваем максимальную разность с ε и печатаем результаты, если процесс сошелся. Если же итерационный процесс не сошелся, то мы производим операции, относящиеся к самому программированию, но не к численному методу. Дело в том, что процесс, который должен сходиться, в действительности не всегда сходится. Причин этому может быть очень много: от ошибок в программе до неверных исходных данных. Поэтому в начале программы с перфокарты было прочитано целое число MAX, которое определяет максимально допустимое число итераций. Обычно это число выбирается несколько бо́льшим необходимого количества итераций. (Ориентировочно для MAX можно принять значение 50 при решении системы из 50 уравнений.) Тогда, если по какой-либо причине итерационный процесс не сошелся, ЭЦВМ не будет работать бесконечно долго. Практически для любого итерационного метода необходимо предусматривать в программе такой счетчик в той или иной форме.

В блок-схеме указано, что исходная информация вводится в машину извне и результаты печатаются. Заметим, что на практике исходные данные для системы уравнений часто вычисляются самой же ЭЦВМ в некоторой предшествующей программе, а результаты столь же часто используются для последующих вычислений. Пример использования результатов вычислений в последующей программе мы рассмотрим в разд. 8.8 (практический пример 10).

Теперь обратимся к вопросу сходимости метода. Перед обобщением на случай n уравнений подробно рассмотрим более простой пример — систему из двух уравнений. Запишем уравнения в виде

$$a_{11}x + a_{12}y = b_1, \tag{8.36}$$

$$a_{21}x + a_{22}y = b_2, \tag{8.37}$$

так что

$$x^{(k)} = \frac{1}{a_{11}}(b_1 - a_{12}y^{(k-1)}), \tag{8.38}$$

$$y^{(k)} = \frac{1}{a_{22}}(b_2 - a_{21}x^{(k)}). \tag{8.39}$$

Если обозначить

$$\Delta x^{(k)} = x - x^{(k)},$$

$$\Delta y^{(k)} = y - y^{(k)},$$

то из (8.36) и (8.38) следует

$$\Delta x^{(k)} = -\frac{a_{12}}{a_{11}} \Delta y^{(k-1)},$$

а из (8.37) и (8.39)

$$\Delta y^{(k)} = -\frac{a_{21}}{a_{22}} \Delta x^{(k)}.$$

Сравнивая последние два уравнения, нетрудно получить, что

$$\Delta y^{(k)} = \frac{a_{12}a_{21}}{a_{11}a_{22}} \Delta y^{(k-1)}.$$

Аналогично

$$\Delta y^{(k-1)} = \frac{a_{12}a_{21}}{a_{11}a_{22}} \Delta y^{(k-2)},$$

так что

$$\Delta y^{(k)} = \left(\frac{a_{12}a_{21}}{a_{11}a_{22}}\right)^2 \Delta y^{(k-2)}.$$

Продолжая дальше таким же образом, получим

$$\Delta y^{(k)} = \left(\frac{a_{12}a_{21}}{a_{22}a_{11}}\right)^{(k)} \Delta y^{(0)}.$$

Аналогично

$$\Delta x^{(k)} = \left(\frac{a_{12}a_{21}}{a_{11}a_{22}}\right)^{(k)} \Delta x^{(0)}.$$

Поэтому если

$$\left|\frac{a_{12}a_{21}}{a_{11}a_{22}}\right| < 1, \tag{8.40}$$

то итерационный метод Гаусса — Зейделя сходится к решению уравнений (8.36) и (8.37).

Нетрудно увидеть аналогию между выкладками, произведенными в этом разделе, и теми, которые были исполь-

зованы при решении вопроса о сходимости метода последовательных приближений в разд. 5.2.

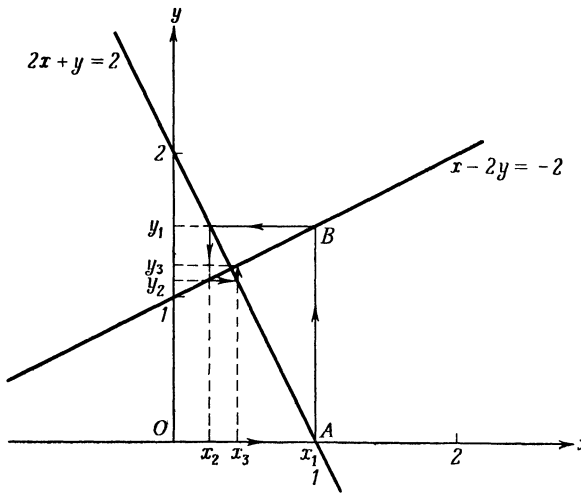
Соотношение (8.40) можно удовлетворить, если

$$\begin{cases} |a_{11}| > |a_{12}|, \\ |a_{22}| \geqslant |a_{21}|, \end{cases} \tag{8.41}$$

или если

$$\begin{cases} |a_{11}| \geqslant |a_{12}|, \\ |a_{22}| > |a_{21}|. \end{cases} \tag{8.42}$$

Иными словами, диагональные члены должны преобладать в уравнении, т. е. они должны быть по абсолютной



Р и с. 8.10. Геометрическое представление итерационного метода Гаусса — Зейделя для случая сходимости.

величине не меньше, а по крайней мере в одном случае больше недиагональных членов.

Рассмотрим простой пример

$$2x + y = 2,$$
$$x - 2y = -2.$$

Точное решение равно $x = 2/5$, $y = 6/5$, как показано на рис. 8.10. Результаты последовательных итераций составляют

Итерация	x	y
0	0	0
1	1	3/2
2	1/4	9/8
3	7/16	39/32

Полезно представить себе процесс решения геометрически. Мы начинаем искать решение от начала координат (0, 0). Так как при вычислении x мы сохраняем неизменным значение y, то геометрически это соответствует движению по горизонтали до пересечения ее с прямой, соответствующей первому уравнению ($2x + y = 2$). Затем, сохраняя неизменным только что найденное значение x, мы начинаем двигаться по вертикали, пока не пересечем прямую, соответствующую второму уравнению ($x - 2y = -2$). На рис. 8.10 такому процессу соответствует путь OAB. На этом заканчивается одна итерация.

Дальнейшие итерации производятся точно таким же образом; на рис. 8.10 их последовательность представлена горизонтальными и вертикальными линиями со стрелками. Заметим, что процесс сходится к решению системы уравнений. Отметим также сходство между рис. 8.10 и рис. 5.2.

Посмотрим теперь, что случится, если мы переставим уравнения:

$$x^{(k)} = -2 + 2y^{(k-1)},$$
$$y^{(k)} = 2 - 2x^{(k)}.$$

Результаты последовательных итераций составляют

Итерация	x	y
0	0	0
1	−2	6
2	10	−18
3	−33	78

Геометрически этот случай проиллюстрирован на рис. 8.11. Снова можно отметить сходство с рис. 5.4. Трудность на этот раз состоит в том, что угол наклона прямой,

соответствующей первому уравнению, меньше 1 и поэтому Δx получается чересчур большим. Аналогично, большой угол наклона прямой, соответствующей второму уравнению, приводит к слишком большой величине Δy. Короче говоря, процесс расходится.

Очевидно, что для сходимости процесса первая прямая

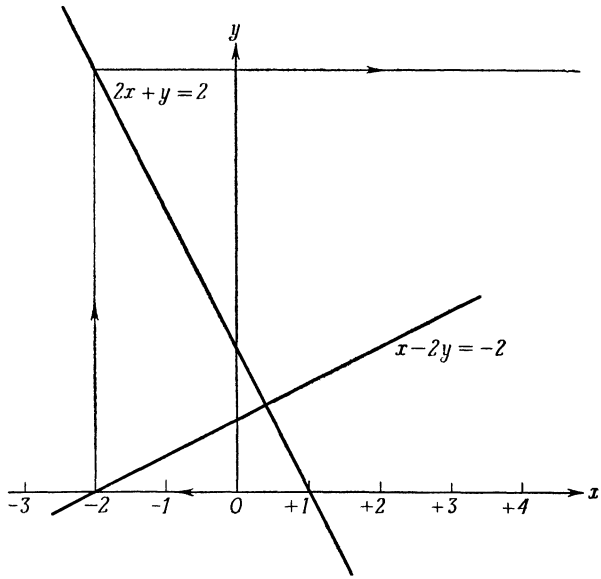


Рис. 8.11. Геометрическое представление итерационного метода Гаусса — Зейделя для случая расходимости.

должна иметь наклон больше 1, а вторая — меньше 1. Эти условия полностью эквивалентны условиям (8.41) и (8.42), записанным в аналитическом виде.

Вспомним, однако, что фактически условие сходимости, которому нужно удовлетворить, выражается формулой (8.40) и что это условие гораздо слабее условий (8.41) и (8.42). Зададимся вопросом: возможен ли случай, когда первая прямая имеет наклон меньше 1, но вторая имеет такой малый угол наклона, что процесс тем не менее сходится? Или, иными словами, если даже Δx велико, то может ли Δy быть достаточно малым, чтобы обеспечить сходи-

мость метода? На этот вопрос можно ответить положительно.

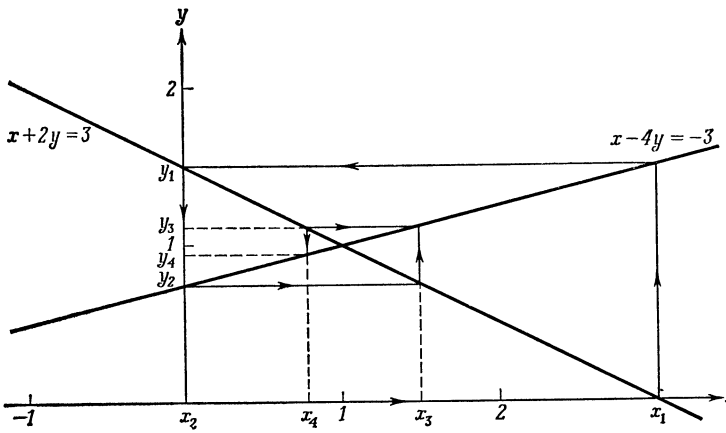
Рассмотрим следующий пример:

$$x+2y=3,$$
$$x-4y=-3.$$

Наклон прямых, соответствующий обоим уравнениям, меньше 1. Поэтому

$$|a_{11}|<|a_{12}|,$$

т. е. нарушено и условие (8.41), и (8.42). Заметим, однако, что условие (8.40) для этой системы удовлетворяется.



Р и с. 8.12. Геометрическое представление итерационного метода Гаусса — Зейделя для случая, когда достаточные условия сходимости не выполнены, но метод тем не менее сходится.

Геометрическое толкование процесса последовательных приближений показано на рис. 8.12. Результаты последовательных итераций составляют

Итерация	x	y
0	0	0
1	3	3/2
2	0	3/4
3	3/2	9/8
4	3/4	15/16

Решением этой системы являются $x = 1$, $y = 1$.

Таким образом, условия (8.41) и (8.42) являются *достаточными* условиями для сходимости, но они не являются *необходимыми*. Иными словами, выполнение какого-нибудь из них гарантирует сходимость метода, но в то же время существуют системы, для которых эти условия не выполняются, но метод итераций все равно сходится.

Мы рассмотрели пример, где наклон первой прямой был меньше -1, а второй — положителен, но меньше $+1$, т. е.

$$k_1 < -1,$$

$$0 < k_2 < +1.$$

Как показывает рис. 8.10, сходимость носила в этом случае колебательный характер.

В другом примере, где наклоны удовлетворяли неравенствам

$$0 < k_1 < 1,$$

$$k_2 < -1,$$

расходимость также являлась колебательной.

Можно легко проверить, что при $|k_1| \geqslant 1$, $|k_2| \leqslant 1$, где по крайней мере одно из неравенств является строгим, справедливы следующие утверждения:

1. Если k_1 и k_2 имеют одинаковые знаки, то процесс сходится к решению с одной стороны.

2. Если k_1 и k_2 имеют разные знаки, то сходимость носит колебательный характер.

Теперь мы приведем без доказательства достаточные условия сходимости итерационного метода Гаусса — Зейделя для системы из n уравнений с n неизвестными.

Если система уравнений неприводима[1]), если

$$|a_{ii}| \geqslant |a_{i1}| + \ldots + |a_{i,\,i-1}| + |a_{i,\,i+1}| + \ldots + |a_{in}|$$

для всех i и если по крайней мере для одного i

$$|a_{ii}| > |a_{i1}| + \ldots + |a_{i,\,i-1}| + |a_{i,\,i+1}| + \ldots + |a_{in}|,$$

[1]) Система уравнений называется неприводимой, если нельзя вычислить какие-либо неизвестные, решая меньше чем n уравнений.

то итерационный метод Гаусса — Зейделя сходится к решению системы уравнений (8.13).

Выполнение этих условий обеспечивает сходимость метода. Мы еще раз подчеркиваем, что эти условия ни в коем случае не являются необходимыми. Они выведены путем соответствующего обобщения условий (8.41) и (8.42).

У читателя может возникнуть вопрос, оставляют ли эти достаточно жесткие условия какую-либо практическую возможность для применения метода Гаусса — Зейделя. Оказывается, что да. Во многих областях прикладной математики возникает необходимость решать системы линейных уравнений, причем коэффициенты этих уравнений получаются такими, что условия сходимости метода Гаусса — Зейделя автоматически выполняются. В частности, именно такими получаются системы, связанные с решением на ЭЦВМ уравнений в частных производных, как мы это увидим в гл. 11.

Наконец, отметим следующее: по аналогии с результатами гл. 5 следует ожидать, что увеличение очередных поправок (экстраполяция) или их уменьшение (интерполяция) может ускорить сходимость метода. В действительности дело именно так и обстоит. Окончательное обсуждение этого вопроса мы откладываем до гл. 11.

8.7. СРАВНЕНИЕ МЕТОДОВ

Мы рассмотрели два основных метода решения систем линейных алгебраических уравнений — метод исключения и итерационный метод Гаусса — Зейделя. Естественно, возникает вопрос, какой из этих методов предпочтительнее.

Метод исключения имеет то преимущество, что он конечен и теоретически с его помощью можно решить любую невырожденную систему уравнений. Итерационный метод Гаусса — Зейделя сходится только для специальных систем уравнений. Для некоторых систем метод исключения является единственно возможным[1]).

[1]) В действительности для любой невырожденной системы уравнений существуют итерационные методы решения, но обычно они неудобны для практических вычислений. См., например, книгу Milne W. E., Numerical solution of differential equations, Wiley, 1953.

Однако, когда итерационные методы сходятся, они обычно предпочтительнее.

1. Время вычислений пропорционально n^2 на итерацию, в то время как для метода исключения время вычислений пропорционально n^3; если для решения системы требуется менее n итераций, то общие затраты машинного времени будут меньше.

2. Как правило, ошибки округления при итерационном методе меньше; иногда это соображение может оказаться достаточно важным и оправдывает дополнительные затраты машинного времени.

Многие системы уравнений, возникающие на практике, имеют среди коэффициентов большой процент нулей. В этих случаях итерационные методы — если они сходятся — в высшей степени предпочтительны, так как при использовании метода исключения получается треугольная система уравнений, которая обычно уже не имеет нулей в качестве коэффициентов[1]). При решении системы уравнений на ЭЦВМ система с большим количеством нулей предпочтительна потому, что можно проверять коэффициенты и не производить умножения, если они равны нулю. Уравнения, получаемые при решении уравнений в частных производных, относятся именно к этому классу.

Наконец, некоторые системы уравнений настолько велики, что их не только нельзя точно решить методом исключения, но даже нельзя поместить целиком в оперативную память ЭЦВМ. Если коэффициенты этих уравнений вычисляются самой ЭЦВМ с помощью некоторой программы, то такое затруднение можно обойти при использовании итерационного метода, вычисляя коэффициенты каждого уравнения тогда, когда в них возникает необходимость. Системы, возникающие при решении уравнений в частных производных, опять-таки относятся к этому классу.

8.8. ПРАКТИЧЕСКИЙ ПРИМЕР 10: ПРОВЕДЕНИЕ КРИВОЙ МЕТОДОМ НАИМЕНЬШИХ КВАДРАТОВ

Очень часто возникает необходимость выразить в виде функциональной зависимости связь между величинами, которые заданы в виде набора точек с координатами (x, y).

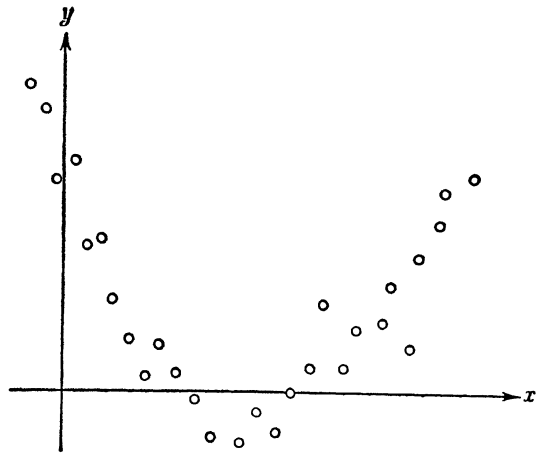
[1]) В некоторых случаях в треугольной системе после исключения все же остаются нулевые коэффициенты (см. упражнение 10).

Предположим, например, что в результате некоторого эксперимента получены точки, отложенные на графике рис. 8.13. Если необходимо использовать эти данные для вычислений на ЭЦВМ, то сразу появляются следующие проблемы:

1. В значениях y наверняка имеются погрешности эксперимента. Было бы желательно каким-либо образом «сгладить» те отклонения, которые обусловлены ошибками эксперимента.

2. Может оказаться желательным знать значения y, соответствующие промежуточным значениям x.

3. Может оказаться, что необходимо экстраполировать функциональную зависимость, т. е. найти значение y,



Р и с. 8.13. Пример экспериментальных результатов, для которых ищется приближенная функциональная зависимость (практический пример 10).

соответствующее значению x, лежащему вне области эксперимента (иногда это является главной целью эксперимента и вычислений). В особенности это относится к экономической информации.

Все эти соображения приводят нас к выводу, что желательно было бы установить некоторую функциональную зависимость между x и y в виде по возможности простой формулы. Вопрос состоит в том, как найти кривую, которая *приближенно* соответствует исходной информации с до-

статочной точностью. Таким образом, нужно выработать критерий, согласно которому та или иная кривая является достаточно «хорошим» приближением к исходной информации.

Рассуждение станет проще, если мы введем новое понятие. Назовем *отклонением* экспериментальной точки разность между экспериментальной ординатой y и той, которая вычислена из функциональной зависимости. Вопрос о том, является ли кривая достаточно «хорошим» приближением к экспериментальным данным, можно поставить в следующем виде: какое условие необходимо наложить на отклонение точек от кривой, чтобы эта кривая представляла экспериментальные данные с достаточной точностью?

Казалось бы, что наиболее простое и логичное условие состоит в том, чтобы сумма отклонений точек от кривой была наименьшей. Если обозначить через y' значение y, вычисленное из функциональной зависимости, то это условие можно записать так: требуется, чтобы

$$\sum_{i=1}^{N} (y_i - y_i')$$

было минимальным, причем в этом выражении N означает количество точек исходной информации. Но привлекательность этого простого критерия сразу становится сомнительной, стоит только рассмотреть простую задачу о проведении линии через две точки, как это показано на рис. 8.14. Мы видим, что пунктирная линия удовлетворяет нашему критерию, но прямую линию, изображенную на этом рисунке, никак нельзя признать удовлетворительным приближением к экспериментальным данным. Можно попытаться обойти это затруднение, используя в критерии сумму абсолютных значений отклонений, т. е. требуя, чтобы

$$\sum_{i=1}^{N} |y_i - y_i'|$$

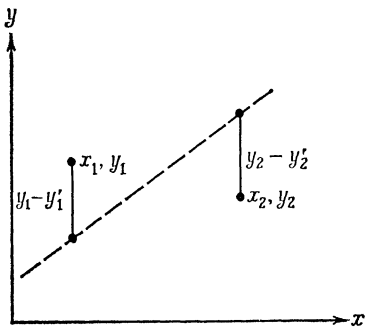
стало минимальным. Но в этом случае для нахождения минимума нельзя воспользоваться производной, так как абсолютное значение не имеет производной в точке минимума. Можно было бы наложить условие, согласно ко-

торому максимальное отклонение должно стать наименьшим (приближение Чебышева), но для определения функциональной зависимости на основе этого критерия приходится использовать длинную и сложную итерационную процедуру.

Поэтому в данном случае мы воспользуемся критерием наименьших квадратов, т. е. будем искать такую функциональную зависимость, при которой

$$\sum_{i=1}^{N} (y_i - y_i')^2$$

обращается в минимум. Это выражение, как мы увидим ниже, можно продифференцировать для нахождения мини-



Р и с. 8.14. Пример, показывающий, что сумма отклонений не может служить критерием для подбора функциональной зависимости (практический пример 10).

мума. Такой критерий во многих практических случаях приводит к линейным уравнениям, которые легко решить, по крайней мере в принципе.

Наконец, можно статистически обосновать, что критерий наименьших квадратов дает достаточно хорошее приближение функциональной зависимости к экспериментальным данным, даже если отвлечься от вопроса о практике вычислений.

Рассмотрим теперь вопрос о том, как при использовании критерия наименьших квадратов получается система уравнений для определения функциональной зависимости y от x. Сначала мы произведем вычисления для

очень простой зависимости, а именно квадратичной, а затем рассмотрим, что получается, если выбирать другие типы функций [1]).

Напишем

$$y'_i = c_1 + c_2 x_i + c_3 x_i^2.$$

Наша задача состоит в том, чтобы определить значения c_1, c_2 и c_3, при которых сумма [2])

$$S = \sum_{i=1}^{N} (y_i - y'_i)^2 = \sum_{i=1}^{N} (y_i - c_1 - c_2 x_i - c_3 x_i^2)^2$$

становится минимальной.

Известно, что в минимуме функции ее производная равна нулю. Рассматривая S как функцию c_1, мы приравниваем

[1]) Предполагается, естественно, что вид функциональной зависимости задан заранее. С помощью метода наименьших квадратов невозможно установить, например, что экспонента была бы хорошим приближением, если не попытаться фактически ее испробовать. Короче говоря, при использовании этого метода можно находить наилучшее приближение *заданного типа*; вообще никакой метод не способен определить тип функциональной зависимости, который лучше всего подошел бы к данному набору экспериментальных точек.

[2]) Для читателей, непривычных к обращению с суммами, мы приводим несколько простых тождеств, в которых через a обозначена некоторая ненулевая константа, а через x_i и y_i — некоторые переменные:

1. $\sum_{i=1}^{N} (x_i + y_i) = \sum_{i=1}^{N} x_i + \sum_{i=1}^{N} y_i,$
2. $\sum_{i=1}^{N} (a x_i) = a \sum_{i=1}^{N} x_i,$
3. $\frac{d}{dx} \sum_{i=1}^{N} f(x_i) = \sum_{i=1}^{N} \frac{df(x_i)}{dx},$
4. $\sum_{i=1}^{N} a = Na.$

В дальнейшем мы будем подразумевать, что суммы вычисляются по всему массиву, т. е. символ $\sum$ будет означать $\sum_{i=1}^{N}$.

нулю частную производную от S по c_1. При этом получается

$$\frac{\partial S}{\partial c_1} = (-2) \sum_{i=1}^{N} (y_i - c_1 - c_2 x_i - c_3 x_i^2).$$

Приравнивая нулю и преобразовывая, получаем

$$Nc_1 + (\sum x_i)\, c_2 + (\sum x_i^2)\, c_3 = \sum y_i.$$

Аналогичным образом, дифференцируя S по c_2 и по c_3 и приравнивая нулю соответствующие выражения, получаем еще два уравнения для определения неизвестных c_1, c_2 и c_3. Эти уравнения называются нормальными уравнениями для выбора наилучшего приближения к экспериментальным точкам в виде квадратичной параболы (подчеркиваем еще раз, что вид функциональной зависимости задан заранее)

$$Nc_1 + (\sum x_i)\, c_2 + (\sum x_i^2)\, c_3 = \sum y_i,$$
$$(\sum x_i)\, c_1 + (\sum x_i^2)\, c_2 + (\sum x_i^3)\, c_3 = \sum (x_i y_i),$$
$$(\sum x_i^2)\, c_1 + (\sum x_i^3)\, c_2 + (\sum x_i^4)\, c_3 = \sum (x_i^2 y_i).$$

Теперь для определения коэффициентов c_1, c_2 и c_3 достаточно вычислить суммы и решить систему из трех уравнений.

Тот же метод можно использовать при выводе нормальных уравнений для многочлена любой степени, к которым желательно приблизить экспериментальную зависимость. (В дальнейшем, однако, мы увидим, что использование многочленов очень больших степеней невозможно из-за ошибок округления.)

Нормальные уравнения для нахождения коэффициентов многочлена степени M можно написать в следующем виде:

$$Nc_1 + \sum x_i c_2 + \ldots + \sum x_i^M c_{M+1} = \sum y_i,$$
$$\sum x_i c_1 + \sum x_i^2 c_2 + \ldots + \sum x_i^{M+1} c_{M+1} = \sum (x_i y_i),$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$\sum x_i^M c_1 + \sum x_i^{M+1} c_2 + \ldots + \sum x_i^{2M} c_{M+1} = \sum (x_i^M y_i).$$

Для многочлена степени M получается система из $M+1$ уравнений относительно $M+1$ коэффициентов. Количество сумм, которое необходимо вычислить, равно $3M+1$.

Напомним, что индексы в ФОРТРАНе должны быть положительными. В приведенной ниже программе коэффициенты c_i образуют одномерный массив и поэтому нумеруются во всех формулах с 1.

Перед тем, как переходить к рассмотрению программы по вычислению требуемых сумм и решению нормальных уравнений, отметим, что метод наименьших квадратов можно применить и к другим функциональным зависимостям, а не только к многочленам. Предположим, например, что мы ищем зависимость в виде показательной функции

$$y = ax^b.$$

Непосредственное использование метода наименьших квадратов приведет к нелинейным нормальным уравнениям, но это затруднение можно обойти, используя вместо экспериментальных значений их логарифмы

$$\log y = \log a + b \log x.$$

Теперь мы стараемся сделать минимальной сумму квадратов разностей между логарифмами экспериментальных значений y_i и логарифмами ординат функции $y = ax^b$. При этом получаем

$$S = \sum (\log y_i - \log a - b \log x_i)^2.$$

Дифференцируя по a и по b, придем к системе нормальных уравнений

$$N \log a + \sum (\log x_i)\, b = \sum \log y_i,$$

$$\sum (\log x_i) \log a + \sum (\log x_i)^2 b = \sum (\log x_i \log y_i).$$

Неизвестными в этой системе являются $\log a$ и b. После их определения a находится по величине логарифма.

В принципе метод наименьших квадратов можно применить для любой функциональной зависимости, хотя нормальные уравнения часто получаются нелинейными. Некоторые типы функциональных зависимостей рассмотрены в упражнениях.

Вернемся теперь к обобщенной программе для нахождения полиномиальных приближений к экспериментальным данным методом наименьших квадратов. С помощью этой программы можно будет работать с многочленами любой степени до 10-й включительно, причем необходимую степень

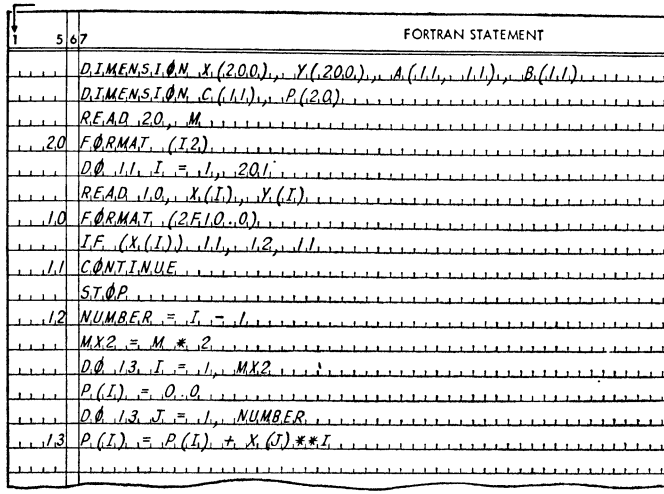
можно будет прочитать с перфокарты. В программе будет предусмотрен ввод экспериментальных данных, вычисление необходимых сумм, решение нормальных уравнений и печать результатов.

Программа, с помощью которой можно произвести эти вычисления, приведена на фиг. 8.15. Чтобы легче было проследить работу программы, мы приведем список основных переменных и их значений в программе:

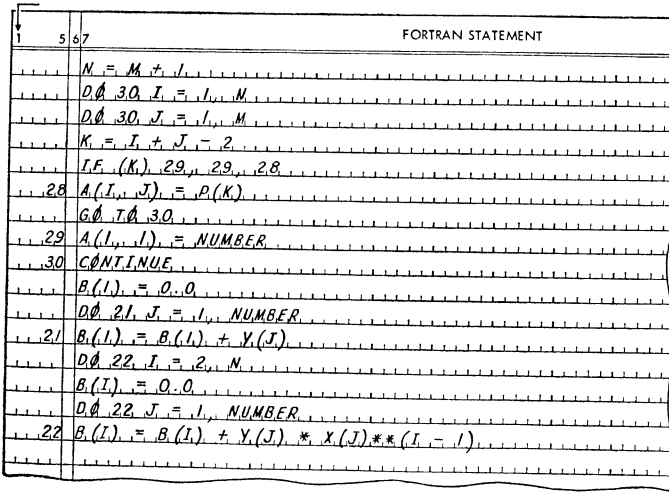
NUMBER — количество экспериментальных точек, не более 200,
M — степень полинома, не выше 10,
N — количество уравнений ($= M + 1$),
X,Y — массивы экспериментальных координат,
A — массив сумм, которые впоследствии становятся коэффициентами нормальных уравнений,
B — массив постоянных членов нормальных уравнений,
C — массив неизвестных, которые впоследствии становятся коэффициентами многочлена,
P — массив степеней x_i от 1 до $2M$.

Два оператора DIMENSION использованы в программе только из-за недостаточной ширины страниц в этой книге. Чтение исходной информации производится с помощью оператора DO, так что по окончании процесса ввода индекс DO определяет количество пар введенных значений. Окончание процесса ввода определяется по пробной перфокарте, на которой пробито значение X = 0. (Если X = = 0 может встретиться в экспериментальных данных, то необходимо использовать какую-либо иную пробную перфокарту, и это изменение нетрудно ввести в программу.) Нормальный выход из цикла DO может произойти только тогда, когда количество точек исходной информации больше 200; если же оно меньше или равно 200, то управление с помощью оператора IF передается оператору 12. После выхода из цикла значение индекса I будет на 1 больше, нежели количество экспериментальных точек, так как пробная карта тоже была сосчитана.

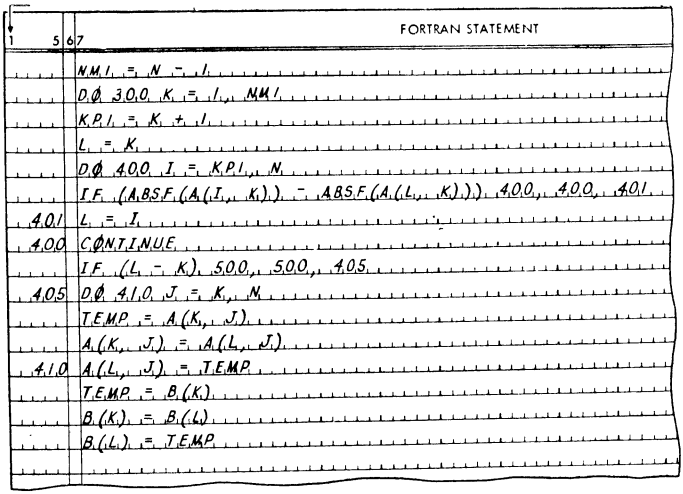
Несколько операторов, стоящих в конце первой страницы программы, позволяют вычислить степени x_i и по-



Р и с. 8.15. Программа для проведения кривой по способу наименьших квадратов. Стр. 1. Ввод исходных данных и вычисление степеней x (практический пример 10).

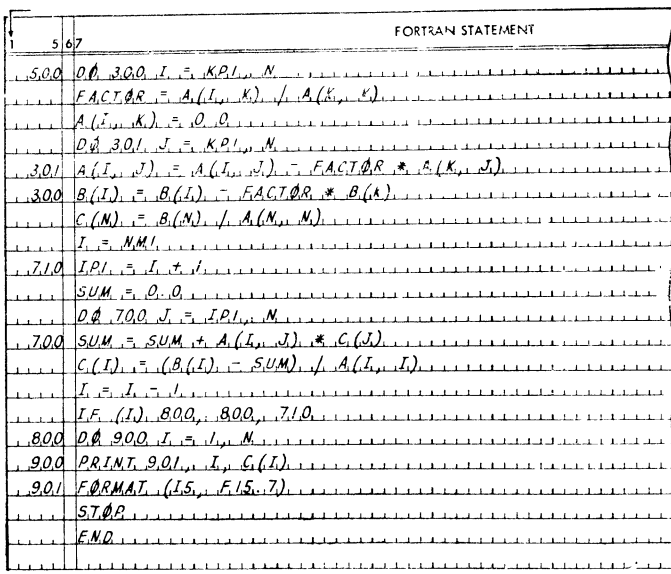


Р и с. 8.15. Стр. 2. Вычисление коэффициентов и постоянных членов в нормальных уравнениях (практический пример 10).



```
1    5 6 7                FORTRAN STATEMENT
     NM1 = N - 1
     DO 300 K = 1, NM1
     KP1 = K + 1
     L = K
     DO 400 I = KP1, N
     IF (ABSF(A(I, K)) - ABSF(A(L, K))) 400, 400, 401
401  L = I
400  CONTINUE
     IF (L - K) 500, 500, 405
405  DO 410 J = K, N
     TEMP = A(K, J)
     A(K, J) = A(L, J)
410  A(L, J) = TEMP
     TEMP = B(K)
     B(K) = B(L)
     B(L) = TEMP
```

Рис. 8.15. Стр. 3. Отыскание наибольшего коэффициента и перестановка уравнений (практический пример 10).



```
1    5 6 7                FORTRAN STATEMENT
500  DO 300 I = KP1, N
     FACTOR = A(I, K) / A(K, K)
     A(I, K) = 0.0
     DO 301 J = KP1, N
301  A(I, J) = A(I, J) - FACTOR * A(K, J)
300  B(I) = B(I) - FACTOR * B(K)
     C(N) = B(N) / A(N, N)
     I = NM1
710  IP1 = I + 1
     SUM = 0.0
     DO 700 J = IP1, N
700  SUM = SUM + A(I, J) * C(J)
     C(I) = (B(I) - SUM) / A(I, I)
     I = I - 1
     IF (I) 800, 800, 710
800  DO 900 I = 1, N
900  PRINT 901, I, C(I)
901  FORMAT (I5, F15.7)
     STOP
     END
```

Рис. 8.15. Стр. 4. Исключение неизвестных, обратная подстановка

местить их в массив P. Эти величины используются сразу же при выводе нормальных уравнений.

При внимательном рассмотрении системы нормальных уравнений нетрудно заметить следующее: за исключением элемента, стоящего на пересечении первой строки и первого столбца степень x для всех остальных элементов равна сумме номера строки и номера столбца минус 2. Этот факт используется на второй странице программы для определения коэффициентов при неизвестных в нормальных уравнениях. Вслед за коэффициентами вычисляются постоянные члены. B(1) необходимо вычислить отдельно, потому что в общем случае неизвестно, каков будет результат возведения числа в нулевую степень в программе на ФОРТРАНе.

Конечно, нельзя утверждать, что все вычисления произведены наиболее эффективным образом. В действительности все эти операции можно было бы проделать гораздо быстрее, хотя при этом пришлось бы использовать больший объем памяти. В данном случае мы сочли такое усложнение неоправданным, так как основной интерес представляет не скорость работы программы, а сам способ наименьших квадратов.

Третья страница программы содержит операторы, с помощью которых решаются нормальные уравнения. Решение производится по методу исключения, и на третьей странице помещены операторы, производящие поиск наибольшего коэффициента и перестановку уравнений. Программа довольно точно соответствует блок-схеме, изображенной на фиг. 8.8, с одним исключением. Последний оператор, присваивающий I значение K + 1, не присутствует в явном виде, так как значение, равное K + 1, присваивается переменной I в начале следующей страницы с помощью оператора DO.

На последней странице программы приведены операторы, завершающие процесс исключения переменных и вычисляющие значения неизвестных путем обратной подстановки. Процесс исключения начинается с оператора 500 и кончается оператором 300. Нужно помнить, что оператор 300 заканчивает область действия двух операторов DO: оператора DO в начале третьей страницы (с индексом K) и оператора 500 (с индексом I).

Первый из этих операторов необходим для перебора всех уравнений с 1-го по (N — 1)-е; второй — для исключения K-го неизвестного из уравнений, начиная с (K + + 1)-го и кончая N-м.

Обратная подстановка начинается после оператора 300. Программа весьма точно соответствует блок-схеме рис. 8.6 с некоторыми очевидными изменениями, которые позволяют сделать ее более компактной.

Для проверки программы воспользуемся следующим приемом: зададим в качестве исходной информации несколько точек параболы с известными коэффициентами и посмотрим, получатся ли в результате вычислений те же коэффициенты. Возьмем в качестве исходного уравнения

$$y = 2 + 4x + x^2$$

и вычислим координаты нескольких точек на этой кривой

x	y
1	7
2	14
3	23
4	34
5	47

Примем в программе M = 2, и поскольку задано пять пар значений x и y, то в результате вычислений мы должны получить $c_1 = 2$, $c_2 = 4$ и $c_3 = 1$. Вычисленные значения довольно точно совпадают с исходными

$$c_1 = 1.9999947$$
$$c_2 = 4.0000044$$
$$c_3 = 0.9999993$$

Интересно, что получится, если попытаться провести через эти же точки кубическую параболу; первые три коэффициента должны остаться неизменными, а четвертый должен равняться нулю. В действительности получается

$$c_1 = 2.0001345$$
$$c_2 = 3.9998111$$

$$c_3 = 1.0000721$$
$$c_4 = -0.0000080$$

Вариация результатов оказалась немного больше того, что можно было ожидать. Попробуем сделать вычисления для многочлена четвертой степени

$$c_1 = 1.9959750$$
$$c_2 = 4.0075371$$
$$c_3 = 0.9954456$$
$$c_4 = 0.0010951$$
$$c_5 = -0.0000907$$

Этот результат не внушает уже никакого доверия. Ведь задача состоит в том, чтобы провести многочлен четвертого порядка через пять точек, а теоретически пять точек точно определяют его коэффициенты.

Попытаемся теперь провести многочлен пятого порядка через эти же точки. Здесь мы вообще не должны были бы получить никакого решения, так как для точного определения многочлена пятого порядка нужны шесть точек. Через пять точек можно провести бесконечно много многочленов пятого порядка; следует ожидать, что если мы попытаемся произвести такого рода вычисления с помощью нашей программы, то получится система нормальных уравнений с бесконечным множеством решений.

При решении такой системы уравнений методом исключения должна появиться строка из одних нулей, а при обратной подстановке попытка деления на нуль должна остановить работу ЭЦВМ. (В упражнении 36 эта ситуация демонстрируется для случая, когда квадратичная парабола проводится через две произвольные точки.) Все же попробуем сделать вычисления для многочлена пятой степени

$$c_1 = 2.0754113$$
$$c_2 = 3.8280203$$
$$c_3 = 1.1410277$$
$$c_4 = -0.0532048$$
$$c_5 = 0.0093776$$
$$c_6 = -0.0006245$$

В данном случае ошибки округления привели к тому, что все вычисления были произведены, хотя теоретически должно было бы получиться переполнение разрядной сетки ЭЦВМ (при делении на нуль) и остановка вычислений!

Возникают серьезные сомнения в том, что такой способ нахождения коэффициентов многочлена способен давать точные результаты. Чтобы убедиться, насколько ошибочным может быть результат, рассмотрим еще один пример. Возьмем уравнение

$$y = 40 + 10x + 5x^2 + 3x^3 + 2x^4 + x^5 + x^6$$

и вычислим значения y для $x = 1, 2, 3, \ldots, 50$. После этого попробуем для этих 50 пар значений найти уравнение многочлена 6-го порядка с помощью нашей программы. Первый коэффициент был напечатан в виде 1018375. 2890625; в действительности он мог бы оказаться еще больше, так как некоторые цифры с левой стороны могли оказаться отброшенными (вспомним, что спецификация поля в операторе FORMAT была написана в виде F15.7). Совершенно ясно, что этот, да и все другие коэффициенты, вычисленные машиной, не имеют никакого смысла.

Что-то в нашей программе совершенно разладилось. Может быть, неприятности связаны с суммированием? Или с решением системы нормальных уравнений? Или порочен сам метод наименьших квадратов? Попытаемся произвести некоторые контрольные вычисления, чтобы определить источник затруднений.

Возьмем то же самое уравнение, но вычислим y для значений $x = 1.0, 1.1, 1.2, \ldots, 5.0$. Так же, как и в предыдущем случае, будем находить для этих 50 пар значений коэффициенты многочлена 6-го порядка. На этот раз получаем следующие результаты:

$$c_1 = 41.3747716$$
$$c_2 = 1.4650151$$
$$c_3 = 19.4771159$$
$$c_4 = -7.3396831$$
$$c_5 = 5.5579804$$
$$c_6 = 0.4154199$$
$$c_7 = 1.0368103$$

Эти значения явно не внушают большого доверия, но по крайней мере порядки величин правильные. По-видимому, одной из причин затруднений в предыдущем примере были сильно отличающиеся друг от друга порядки чисел, стоящих в качестве коэффициентов в нормальных уравнениях. В самом деле, значение a_{11} равно 50, а значение a_{77} порядка 10^{21}. При столь различных числах неизбежны трудности того же рода, как возникавшие в свое время при вычислении синуса из ряда Тейлора (см. практический пример 3).

Ошибки возникают как при подсчете сумм, так и при решении системы уравнений методом исключения. Например, при возведении 49 в двенадцатую степень некоторые цифры в правой части числа неизбежно теряются, причем безвозвратно; они не восстанавливаются при исключении неизвестных, в результате которого из этих больших значений должны были бы появиться меньшие числа. Впрочем, даже если бы сумму и удалось вычислить точно, все равно ошибки округления при решении системы методом исключения не позволили бы получить точный результат.

Можно убедиться в том, что все неприятности происходят только от ошибок округления, производя вычисления с удвоенной точностью. При этом каждое число занимает две ячейки памяти и имеет примерно 16 десятичных значащих цифр. Произведя вычисление для исходного набора значений $x = 1, 2, 3, \ldots, 50$, получаем следующий результат:

$$
\begin{aligned}
c_1 &= 39.9945393 \\
c_2 &= 10.0036631 \\
c_3 &= 4.9993401 \\
c_4 &= 3.0000494 \\
c_5 &= 1.9999982 \\
c_6 &= 1.0000000 \\
c_7 &= 1.0000000
\end{aligned}
$$

Эти коэффициенты уже достаточно близки к исходным.

Обратите внимание, насколько сильно искажают результат ошибки округления и какие крайние меры приходится принимать для обеспечения хотя бы невысокой точности: ведь в данном случае довольно простая теория пред-

сказывает некоторые вполне точные значения, а вместе с тем для получения четырех достоверных цифр в ответе приходится производить вычисления с 16 значащими цифрами.

Читателям, желающим познакомиться более основательно с причинами неприятностей, можно порекомендовать превосходную книгу Ардена[1]).

Радикальным решением проблемы является использование ортогональных полиномов. Например, в упражнении 3 гл. 3 было показано, что полиномы Чебышева ортогональны на интервале $-1 \leqslant x \leqslant 1$ с некоторой весовой функцией. Если использовать $T_n(x)$ для построения системы нормальных уравнений, то недиагональные коэффициенты будут, вообще говоря, малы по сравнению с диагональными. Это обстоятельство позволяет получить более точные решения, иногда даже с помощью итерационного метода Гаусса — Зейделя.

В идеальном случае, конечно, недиагональные коэффициенты должны были бы вообще обратиться в нуль, и тогда решение стало бы тривиальным. Интересующиеся читатели могут найти некоторые сведения по этому вопросу в статье Форсайта[2]).

Примеры были подобраны с таким расчетом, чтобы обратить внимание читателей на печальные последствия, которыми грозит легкомысленное отношение к ошибкам округления. Однако мы не желали бы создать впечатление, что метод наименьших квадратов невозможно использовать из-за этих трудностей. Поэтому в заключение главы приведем простой пример практического использования этого метода.

В книге Уортинга и Геффнера[3]) приведена таблица экспериментальных данных по зависимости теплоемкости воды от температуры, причем теплоемкость при 15° С принята за единицу.

---

[1]) Arden B. W., An introduction to digital computing, Addison-Wesley, 1963.

[2]) Forsythe G. E., Generation and use of orthogonal polynomials for data fitting with a digital computer, *SIAM J.*, 5, 74—88 (июнь 1957).

[3]) Worthing A. G., Geffner J., Treatment of experimental data, Wiley, 1943, p. 268.

T, °C	c_p	T, °C	c_p
0	1.00762	55	0.99919
5	1.00392	60	0.99967
10	1.00153	65	1.00024
15	1.00000	70	1.00091
20	0.99907	75	1.00167
25	0.99852	80	1.00253
30	0.99826	85	1.00351
35	0.99818	90	1.00461
40	0.99828	95	1.00586
45	0.99849	100	1.00721
50	0.99878		

Построив точки на графике, видим, что они довольно близки к квадратичной параболе и что, по-видимому,

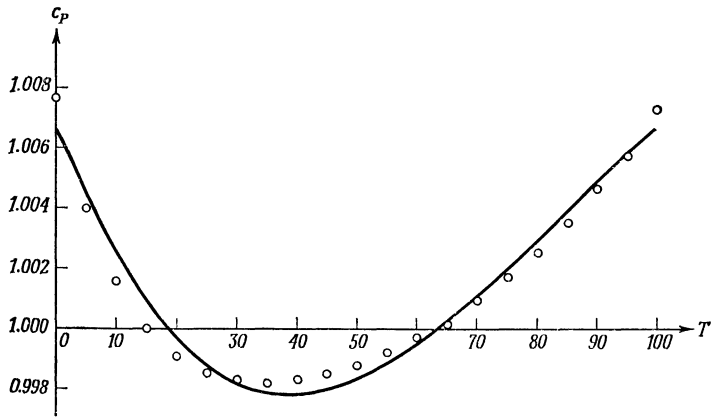


Рис. 8.16. Экспериментальные значения теплоемкости воды (кружки) и многочлен третьего порядка, проведенный по методу наименьших квадратов (практический пример 10).

кубическая парабола будет вполне хорошим приближением. Использовав нашу программу для нахождения коэффи-

циентов этой кубической параболы, получим

$$c_p = 1.006447 - 0.0004987884T + 0.000008459123T^2 - \\ - 0.00000003453391T^3$$

Ошибки округления в данном случае не доставляют никаких неприятностей. В этом можно убедиться, проделав вычисления с удвоенной точностью: коэффициенты изменяются очень ненамного, а сумма квадратов отклонений остается той же, что и при вычислениях с обычной точностью.

На рис. 8.16 изображены экспериментальные точки и кривая, вычисленная по методу наименьших квадратов. Мы видим, что отклонения положительного и отрицательного знака более или менее уравновешены, как и следовало ожидать. Сравнение вычисленной кривой с экспериментальными значениями показывает, что наибольшее отклонение имеет место при 0° С и равно 0.0012, все остальные отклонения существенно меньше.

Если ошибки такого порядка допустимы, то можно использовать эту кубическую формулу при расчетах, где требуется зависимость теплоемкости воды от температуры.

Упражнения

***1.** Решите следующую систему уравнений методом исключения. Перестановку уравнений не производите.

$$x - y + z = -4,$$
$$5x - 4y + 3z = -12,$$
$$2x + y + z = 11.$$

2. Решите следующую систему уравнений методом исключения. Перестановку уравнений не производите.

$$w + x + y + z = 10,$$
$$2w + 3x + y + 5z = 31,$$
$$-w + x + 5y + 3z = -2,$$
$$3w + x + 7y - 2z = 18.$$

***3.** Решите следующую систему уравнений методом исключения, используя перестановку уравнений и производя все вычисления с точностью до четырех значащих цифр.

$$2x + 6y - z = -12,$$
$$5x - y + 2z = 29,$$
$$-3x - 4y + z = 5,$$

4. Попробуйте решить следующую систему методом исключения. Что при этом происходит и в чем причина затруднений?

$$x+y+z=2,$$
$$2x-3y+z=11,$$
$$4x-y+3z=10.$$

5. Попробуйте решить следующую систему методом исключения. Что при этом происходит и в чем причина затруднений?

$$2x-3y+4z=8,$$
$$4x+2y-3z=-1,$$
$$6x+7y-10z=-10.$$

***6.** Решите следующую систему уравнений методом исключения, производя все действия с комплексными числами.

$$(2+3i)\,x+(2-i)\,y=2+i,$$
$$(4+6i)\,x+(3-6i)\,y=-2-5i.$$

7. В системе из упражнения 6 напишите $x=x_r+ix_i$ и $y=y_r+iy_i$. Раскройте скобки и приравняйте друг другу отдельно мнимую и действительную части каждого уравнения. Покажите, что в результате получается система из 4 уравнений с 4 неизвестными и что решение этой системы дает мнимые и действительные части решений системы из упражнения 6.

***8.** Решите следующую систему уравнений методом исключения. Все вычисления производите с точностью до 4 значащих цифр. Сначала решите систему, не переставляя уравнения, затем переставьте уравнения.

$$x+592y=437,$$
$$592x+4308y=2251.$$

9. Вспомним, что при решении уравнений (8.5), (8.6) и (8.7) методом Гаусса мы сначала исключали x_1 и получали следующую систему:

$$a_{11}x_1+a_{12}x_2+a_{13}x_3=b_1, \tag{8.5}$$
$$a'_{22}x_2+a'_{23}x_3=b'_2, \tag{8.9}$$
$$a'_{32}x_2+a'_{33}x_3=b'_3. \tag{8.10}$$

Затем мы вводили множитель

$$m'_3=\frac{a'_{32}}{a'_{22}}$$

и вычитали из уравнения (8.10) уравнение (8.9), умноженное на m'_3. При этом получалось

$$a''_{33}x_3=b''_3\,. \tag{8.11}$$

Покажите, что если ввести множитель

$$m_1' = \frac{a_{12}}{a_{22}'}$$

и вычесть из уравнения (8.5) уравнение (8.9), умноженное на m_1', то получится

$$a_{11}'x_1 + a_{13}'x_3 = b_1'. \qquad (*)$$

Найдите формулы для определения a_{11}', a_{13}' и b_1'.

Введем теперь множители

$$m_1'' = \frac{a_{13}'}{a_{33}''},$$

$$m_2'' = \frac{a_{23}'}{a_{33}''}.$$

Покажите, что если вычесть из уравнения (*) уравнение (8.10), умноженное на m_1'', из уравнения (8.9) — уравнение (8.10), умноженное на m_2'', то получится следующая система уравнений:

$$a_{11}''x_1 = b_1'',$$
$$a_{22}''x_2 = b_2'',$$
$$a_{33}''x_3 = b_3''.$$

Найдите формулы для определения a_{11}'', a_{22}'' и a_{33}''.

Заметим, что эти три уравнения можно решить *без обратноя подстановки*. Этот метод решения называется *методом исключений Гаусса — Жордана*.

10. Рассмотрим следующую систему уравнений:

$$\begin{aligned}
2x_1 - x_2 &= 1,\\
-x_1 + 2x_2 - x_3 &= 1,\\
-x_2 + 2x_3 - x_4 &= 1,\\
-x_3 + 2x_4 - x_5 &= 1,\\
-x_4 + 2x_5 - x_6 &= 1,\\
-x_5 + 2x_6 &= 1.
\end{aligned}$$

Покажите, что большинство коэффициентов этой системы так и останется нулевыми после приведения ее к треугольному виду методом исключения. Системы, подобные этой, называются *тридиагональными* из-за расположения коэффициентов. Такие системы часто возникают при решении уравнений в частных производных (см. гл. 11).

11. Предположим, что необходимо решить две системы линейных уравнений, причем левые части всех уравнений одной и другой систем попарно одинаковы, а правые части различны. Если для решения систем используется метод исключения, то будут ли две полученные в результате треугольные системы уравнений отличаться своими левыми частями? правыми частями?

Опишите, как можно усовершенствовать приведенную в этой главе программу для решения уравнений методом Гаусса, чтобы найти решения обеих систем, произведя всего один процесс исключения.

12. Метод, предложенный в упражнении 11, можно применить также и в том случае, когда имеются три системы уравнений с различными правыми частями. Используя этот метод, решите следующие три системы уравнений:

$$\begin{cases} x-y \quad\;\; = 1, \\ x+y+z = 0, \\ \quad y-z = 0; \end{cases}$$

$$\begin{cases} x-y \quad\;\; = 0, \\ x+y+z = 1, \\ \quad y-z = 0; \end{cases}$$

$$\begin{cases} x-y \quad\;\; = 0, \\ x+y+z = 0, \\ \quad y-z = 1. \end{cases}$$

Теперь пусть x_1, y_1, z_1 — решение первой системы, x_2, y_2, z_2 — решение второй системы, x_3, y_3, z_3 — решение третьей системы. Покажите, что решение системы

$$\begin{aligned} x-y \quad\;\; &= b_1, \\ x+y+z &= b_2, \\ \quad y-z &= b_3 \end{aligned}$$

дается формулами

$$\begin{aligned} x &= b_1x_1 + b_2x_2 + b_3x_3, \\ y &= b_1y_1 + b_2y_2 + b_3y_3, \\ z &= b_1z_1 + b_2z_2 + b_3z_3. \end{aligned}$$

Можно ли из этого примера сделать заключение о том, сколько систем уравнений необходимо решить, чтобы впоследствии иметь возможность непосредственно находить решение системы при любых значениях правых частей уравнений?

13. Используя метод, изложенный в разд. 8.4, уточните решение системы из упражнения 8, найденное без перестановки уравнений.

***14.** Приближенное решение следующей системы равно $x_1 = 2$, $x_2 = 3$, $x_3 = 4$. Уточните это решение, используя метод, изложенный в разд. 8.4.

$$\begin{aligned} 1.781x_1 + 3.008x_2 - 4.880x_3 &= -7.704 \\ 4.632x_1 - 1.064x_2 - 2.274x_3 &= -6.359 \\ -3.387x_1 + 9.814x_2 - 4.779x_3 &= 3.946 \end{aligned}$$

15. Приближенное решение следующей системы равно $x = 10$, $y = -3.99$. Уточните это решение, производя сначала вычисления

с точностью до 4 значащих цифр, затем с точностью до 6 значащих цифр.

$$234x + 546y = 156,$$
$$158x + 371y = 103.$$

16. В системе уравнений из упражнения 15 измените 371 на 371.2 и решите снова эту систему. Сравните относительное изменение коэффициента и относительное изменение решения y.

***17.** В следующей системе все a_{ij} заданы точно, но b_i заданы только с точностью до двух знаков после запятой. Найдите верхний предел для возможной ошибки в решении. Сначала вычислите этот предел для того порядка, в котором уравнения написаны в тексте, затем переставьте уравнения и снова определите верхний предел ошибки.

$$x + y = 2,$$
$$x + 2y = 3.$$

***18.** Рассмотрим следующую систему уравнений:

$$x - y + 4z = 6,$$
$$2x + 3y - z = 18,$$
$$3x + y + z = 19.$$

Решите эту систему, не используя перестановку уравнений (что, как мы убедились в упражнении 17, меняет верхние пределы ошибок), и найдите верхние пределы возможных ошибок в решении при следующих условиях:

а. Все a_{ij} заданы точно, все b_i заданы с точностью до двух знаков после запятой.

б. Все b_i заданы точно, все a_{ij} заданы с точностью до двух знаков после запятой.

в. Все a_{ij} и b_i заданы с точностью до двух знаков после запятой.

г. Все a_{ij} и b_i заданы с точностью до двух значащих цифр.

19. Для следующей системы уравнений сделайте то же, что и в упражнении 18,

$$2x + y + z = 7,$$
$$2x + 2y + 3z = 10,$$
$$-4x + 4y + 5z = 14.$$

***20.** Решите следующую систему уравнений итерационным методом Гаусса — Зейделя. Продолжайте итерационный процесс до тех пор, пока разница между последовательными приближениями x, y и z не станет меньше 0.02. Значит ли это, что приближенное решение отличается от точного решения не более чем на 0.02?

$$10x + 2y + 6z = 28,$$
$$x + 10y + 9z = 7,$$
$$2x - 7y - 10z = -17.$$

21. Решите следующую систему уравнений итерационным методом Гаусса — Зейделя. Продолжайте итерационный процесс до тех пор, пока разница между последовательными приближениями x, y и z не станет меньше 0.02. Сравните скорость сходимости итерационного процесса в этом упражнении со скоростью сходимости в упражнении 20. В чем причина различия?

$$20x+2y+6z=38,$$
$$x+20y+9z=-23,$$
$$2x-7y-20z=-57.$$

22. Рассмотрим систему из двух уравнений

$$a_1x+b_1y=c_1,$$
$$a_2x+b_2y=c_2.$$

Пусть k_1 — наклон прямой, соответствующей первому уравнению, и k_2 — наклон прямой для второго уравнения. Покажите, что:

а. Если $k_1>1$ и $-1<k_2<0$, то при решении системы итерационным методом Гаусса — Зейделя сходимость носит колебательный характер.

б. Если $k_1>1$ и $0<k_2<1$, то при решении системы итерационным методом Гаусса — Зейделя все последовательные приближения находятся с одной стороны от x и y.

в. Если $k_1<-1$ и $-1<k_2<0$, то при решении системы итерационным методом Гаусса — Зейделя все последовательные приближения также находятся с одной стороны от x и y.

23. Рассмотрим систему из двух уравнений

$$a_{11}x_1+a_{12}x_2=b_1,$$
$$a_{21}x_1+a_{22}x_2=b_2.$$

Предположим, что для решения этих уравнений применяется итерационный метод, описываемый следующими формулами:

$$x_1^{(k)}=\frac{1}{a_{11}}(b_1-a_{12}x_2^{(k-1)}),$$
$$x_2^{(k)}=\frac{1}{a_{22}}(b_2-a_{21}x_1^{(k-1)}).$$

а. Покажите, что необходимое и достаточное условие сходимости этого метода выглядит так:

$$\left|\frac{a_{12}a_{21}}{a_{11}a_{22}}\right|<1.$$

Сравните это условие с условием сходимости метода Гаусса — Зейделя.

б. Покажите, что метод Гаусса — Зейделя сходится вдвое быстрее только что предложенного метода.

в. Проверьте утверждение, сделанное в пункте б, произведя четыре итерации для системы уравнений

$$2x+y=2,$$
$$x-2y=-2$$

и сравнив результаты этих вычислений с результатами решения той же системы методом Гаусса — Зейделя в разд. 8.6.

г. Дайте геометрическую интерпретацию этого итерационного метода.

24. Какие из следующих систем уравнений приводимы? (См. примечание в разд. 8.6.)

а.
$$\begin{cases} -x_1 + 3x_3 + x_4 = 3, \\ 3x_1 + 2x_2 + x_3 - 2x_4 = 4, \\ 2x_1 + 4x_4 = 6, \\ x_3 - x_4 = 0. \end{cases}$$

б.
$$\begin{cases} -x_1 + 3x_3 + x_4 = 3, \\ 3x_1 + 2x_2 + x_3 - 2x_4 = 4, \\ 2x_2 + 4x_4 = 6, \\ x_3 - x_4 = 0. \end{cases}$$

в.
$$\begin{cases} x_2 + x_3 = 2, \\ x_1 - x_3 = 0, \\ 2x_1 + x_2 = 3. \end{cases}$$

г.
$$\begin{cases} x_1 + 2x_2 = 3, \\ x_1 + x_2 + x_3 = 3, \\ 2x_1 + x_2 = 3. \end{cases}$$

25. Решите следующую систему из четырех уравнений методом Гаусса — Зейделя:

$$\begin{aligned} x_1 + x_4 &= 2, \\ x_1 + 4x_2 - x_4 &= 4, \\ x_1 + x_3 &= 2, \\ x_3 + x_4 &= 2. \end{aligned}$$

Удовлетворяют ли коэффициенты уравнения неравенствам, гарантирующим сходимость метода? Почему метод все же расходится? Каково решение этой системы?

26. Рассмотрим снова программу, приведенную на рис. 8.15 в разд. 8.8. Предположим, что имеется одномерный массив NROW, содержащий 11 элементов. Перед началом вычислений каждому такому элементу было присвоено значение, равное его номеру, т. е. NROW (I) = I. Предположим, кроме того, что индексы при переменных в ФОРТРАНе могут в свою очередь иметь индексы, и рассмотрим следующее усовершенствование программы:

а. Когда при поиске наибольшего коэффициента установлено, что необходимо поменять местами уравнения L и K, то вместо

этого поменяйте местами NROW (L) и NROW (K), а сами уравнения оставьте без изменения.

б. На третьей и четвертой страницах рис. 8.15, там, где индекс, например I, относится к номеру уравнения, вместо I пишите NROW (I). Например, оператор 301 запишется в следующем виде:

```
301 A (NROW (I), J) = A (NROW (I), J) —FACTOR*A (NROW (K), J)
```

Приведите какой-нибудь небольшой пример и покажите, что если бы в ФОРТРАНе были допустимы индексы у индексов, то решение системы по методу исключения можно было бы производить, не переставляя уравнений.

Следует отметить, что некоторые алгебраические языки допускают наличие индексов при индексах, иногда за счет дополнительных затрат машинного времени. Многие ЭЦВМ имеют систему так называемой косвенной адресации, что существенно уменьшает эти дополнительные затраты времени.

***27.** Как уже указывалось, индексы при переменных в ФОРТРАНе не могут в свою очередь иметь индексов. Однако это затруднение можно обойти, если перед каждым оператором, где номер уравнения употребляется в качестве индекса, расположить один или несколько операторов, которые будут вычислять значение этого индекса по определенным правилам. Покажите на каком-нибудь примере, как это можно сделать.

28. Напишите программу, где наибольший коэффициент отыскивался бы среди строк и *столбцов*. При составлении программы используйте методику из упражнений 26 и 27, так чтобы не производить в действительности никаких перестановок. Покажите на каком-нибудь примере, что эта программа работает правильно.

29. Напишите программу для решения системы из шести уравнений методом Гаусса — Жордана (см. упражнение 9), предполагая, что заданы коэффициенты и свободные члены.

30. Усовершенствуйте программу, приведенную на рис. 8.15, таким образом, чтобы можно было решить с помощью одного лишь процесса исключения две системы уравнений с различными правыми частями. Правую часть второй системы задайте в виде одномерного массива D.

31. Программу из упражнения 30 усовершенствуйте следующим образом: всем элементам массива D присвойте перед началом вычислений значение $+1$ и все операции с массивом D при исключении неизвестных производите путем сложения абсолютных значений. При нахождении решений методом обратной подстановки также складывайте абсолютные значения и игнорируйте знак минус при делении. Тогда результат решения этой дополнительной системы будет равен Δ_i из разд. 8.5.

32. Составьте программу для решения системы из N уравнений итерационным методом Гаусса — Зейделя, взяв за основу блок-схему, изображенную на рис. 8.9.

33. Напишите программу для вычисления определителя N-го порядка. Используйте метод исключения, получите треугольную систему уравнений, затем вычислите произведение коэффициентов

на главной диагонали — это произведение и будет равно определителю системы. Если при исключении неизвестных приходится менять местами уравнения, введите некоторую целую переменную для подсчета количества перестановок. Если перестановки производились нечетное число раз, то измените знак вычисленного произведения диагональных элементов на обратный.

34. Покажите, что если с помощью метода наименьших квадратов искать уравнение кривой в виде

$$y = a,$$

то величина a будет равна среднему арифметическому из всех значений y_i.

35. Покажите, что если с помощью метода наименьших квадратов искать уравнение кривой в виде

$$y = a + bx,$$

причем исходная информация ограничена двумя точками x_1, y_1 и x_2, y_2, то решение системы нормальных уравнений приводит к уравнению прямой, проходящей через две точки.

36. Покажите, что если попытаться провести кривую второго порядка через две точки, то система нормальных уравнений будет иметь бесчисленное множество решений. Действительно, через две точки можно провести бесчисленное множество квадратичных парабол.

***37.** Предположите, что значения x в исходной информации являются целыми числами от 1 до N. Выведите упрощенную формулу для нормальных уравнений, в левых частях которых не приходилось бы вычислять никаких сумм; функциональную зависимость ищите в виде прямой линии $y = a + bx$.

***38.** Выведите нормальные уравнения для того случая, когда функциональная зависимость ищется в виде

$$y = ae^{bx}.$$

Используйте логарифмы.

39. Выведите нормальные уравнения для того случая, когда функциональная зависимость ищется в виде

$$y = y_0 e^{-h^2x^2}.$$

40. Выведите нормальные уравнения для того случая, когда функциональная зависимость ищется в виде

$$y = a + \frac{b}{x} + \frac{c}{x^2}.$$

Покажите, что можно сделать такие преобразования, чтобы нормальные уравнения стали линейными.

41. Выведите нормальные уравнения для того случая, когда функциональная зависимость ищется в виде

$$y = a\sqrt{1 + bx^2}.$$

Можете ли вы решить эти уравнения?

42. Обозначим

$$\Omega = \frac{\sum (y_i - y_i')^2}{n - m},$$

где y_i — исходное значение,

y_i' — значение ординаты кривой, подобранной методом наименьших квадратов,

n — количество точек исходной информации,

m — количество параметров в функциональной зависимости.

Тогда по Гауссу наилучшей функциональной зависимостью будет та, при которой Ω минимально.

Рассмотрим следующую исходную информацию:

x	y
0	0
1	20
2	40
3	50
4	70

Согласно критерию Гаусса, будет ли в данном случае лучше линейная или квадратичная функциональная зависимость?

ГЛАВА

9

ФУНКЦИИ, ПОДПРОГРАММЫ И ВСПОМОГАТЕЛЬНЫЕ ОПЕРАТОРЫ

9.1. ВВЕДЕНИЕ

Одним из основных стимулов к употреблению вычислительных машин является то, что с их помощью можно уменьшить затраты человеческого труда на производство вычислений. Грубо говоря, наше отношение к этому вопросу можно сформулировать следующим образом: «Зачем делать что-либо, если это может сделать машина?» Этот же принцип можно продолжить и применить к написанию программ. В настоящей главе мы покажем, как можно ответить на вопрос: «Зачем писать дважды некоторый участок программы, если достаточно сделать это один раз?»

Такую экономию труда при написании программ позволяют осуществить различного вида *функции* и *подпрограммы*, предусмотренные в ФОРТРАНе. Оказывается, что можно написать некоторый оператор или группу операторов однажды, а затем обращаться к этому оператору или к группе операторов всякий раз, когда требуется произвести определенные вычисления.

В ФОРТРАНе имеется несколько различных типов функций. Эти функции служат для различных целей и поразному формируются и используются. Мы рассмотрим их по очереди, начиная с того типа, который мы уже вкратце рассматривали,— с функций, предусмотренных в самой программе-трансляторе. Затем мы рассмотрим два вспомогательных оператора, которые связаны с двумя другими типами функций.

9.2. ФУНКЦИИ, ПРЕДУСМОТРЕННЫЕ В ПРОГРАММЕ-ТРАНСЛЯТОРЕ

Большинство вариантов ФОРТРАНа предусматривает несколько десятков функций, с помощью которых можно вычислять такие величины, как синусы, логарифмы и аб-

солютные значения. Набор этих функций зависит не только от применяемого варианта ФОРТРАНа и от устройства вычислительной машины, но даже и от особенностей данного ее экземпляра. Если на какой-то ЭЦВМ приходится очень часто производить расчеты траекторий баллистических ракет, то целесообразно предусмотреть в программе-трансляторе специальную функцию, задающую плотность воздуха в зависимости от высоты. Как правило, для каждой вычислительной машины можно указать такие функции, которые будут использоваться на ней чаще, нежели на других. В некоторых случаях в программе-трансляторе предусматривается вычисление и натуральных, и десятичных логарифмов, хотя они и могут быть легко получены из других. Каждый программист должен иметь полный список функций, предусмотренных на его ЭЦВМ, вместе с подробными пояснениями к ним, с такими, как точность функции, количество необходимого для ее вычисления машинного времени, форма представления аргументов (например, выражать ли угол в радианах или в градусах) и т. д.

Чтобы использовать такую функцию, достаточно написать в соответствующем месте программы ее наименование и перечислить вслед за ним ее аргументы. (Многие функции зависят от нескольких аргументов, например функция, с помощью которой отыскивается наименьшее из всех перечисленных чисел.) Наименования этих функций установлены заранее, и программист должен писать их точно так, как это предусмотрено в программе-трансляторе. Хотя наименование этих функций и не зависит от программиста, укажем, что оно может иметь от четырех до семи букв или цифр, всегда кончается буквой F и всегда начинается с буквы, причем с буквы X тогда и только тогда, когда значение функции целое.

Функции, предусмотренные в программе-трансляторе, бывают двух видов соответственно двум способам их связи с рабочей программой. *Открытые* функции требуют лишь очень небольшого количества машинных команд; эти команды вставляются в рабочую программу всякий раз, когда требуется вычислить такую функцию. *Замкнутые* функции в общем случае требуют гораздо большего числа машинных команд; эти функции вставляются в рабочую программу в одном-единственном месте, а рабочая про-

грамма каждый раз передает управление такому участку, когда необходимо вычислить функцию. Как правило, функции, предусмотренные в программе-трансляторе, являются замкнутыми.

9.3. АРИФМЕТИЧЕСКИЙ ОПЕРАТОР-ФУНКЦИЯ

Часто случается так, что по ходу выполнения программы необходимо многократно производить одну и ту же довольно простую последовательность вычислений. В этом случае целесообразно выделить такую последовательность вычислений в качестве особой функции. Она понадобится только для одной программы, так что нет смысла предусматривать новую функцию в программе-трансляторе (кроме всего прочего, это очень трудоемкий процесс). Взамен определяется функция, которую можно использовать только в данной программе. На другие программы эта функция не окажет никакого влияния.

Арифметический оператор-функция *определяется* путем написания одного арифметического оператора вида $a = b$, где a — наименование функции, b — некоторое выражение. Наименование, которое находится на усмотрении программиста, должно быть выбрано в соответствии с теми же правилами, что и наименования функций, предусмотренных в программе-трансляторе. Это наименование должно содержать от четырех до семи букв или цифр, должно оканчиваться на букву F и должно обязательно начинаться с буквы, причем оно должно начинаться с буквы X тогда и только тогда, когда значение функции является целым. Естественно, это наименование не должно совпадать с наименованием какой-либо функции, предусмотренной в трансляторе. Наименование функции ставится перед скобками, в которых перечисляются аргументы функции (или аргумент, если он один). Если количество аргументов больше одного, то внутри скобок эти аргументы отделяются друг от друга запятыми. Аргументы в *определении функции* не должны иметь индексов.

Правая часть этого оператора может быть любым выражением, не содержащим переменных с индексами. В этом выражении могут быть использованы переменные, не указанные при перечислении аргументов, а также другие функции (кроме нее самой).

Все определения арифметических операторов-функций должны находиться в программе до появления первого исполняемого оператора.

В качестве иллюстрации рассмотрим некоторую программу, где необходимо вычислять один из корней квадратного уравнения $aX^2 + bX + c = 0$ при различных значениях a, b и c. Чтобы производить эти вычисления, можно определить функцию, написав

```
ROOTF(A, B, C)=(-B+SQRTF(B**2-4.*A*C))/(2.*A)
```

С помощью программы-транслятора этот оператор будет преобразован в последовательность машинных команд для вычисления значения функции по трем значениям ее аргументов.

Этот оператор является *только определением* функции, сам по себе он не обусловливает никаких вычислений. Наименования переменных, использованных в качестве аргументов, являются «пустышками»[1]), в частности они могут быть такими же, как и наименования переменных в каком-нибудь другом месте программы. Наименования аргументов совершенно несущественны, за исключением правила, согласно которому соответствующие переменные являются целыми или действительными.

Арифметический оператор-функцию можно использовать в любом месте программы, написав там, где это нужно, ее наименование и подставив вместо аргументов подходящие выражения. В данном случае термин «подходящий» означает, в частности, что если в определении функции в качестве аргумента указана действительная переменная, то и выражение, подставляемое вместо этого аргумента, должно быть действительным. (То же относится и к целым аргументам.) Значения этих выражений подставляются вместо аргументов в тот участок программы, который образовался при переводе определения функции, и в результате вычисляется ее значение. Выражения, подставляемые вместо аргументов, *могут при необходимости иметь индексы.*

Предположим теперь, что необходимо вычислить значение функции ROOTF(A, B, C) при A = 16.9, B = R —

[1]) В оригинале книги «dummies».— *Прим. перев.*

— S и C = T + 6.9; к значению этой функции необходимо прибавить косинус X и присвоить переменной ANS значение, равное окончательному результату. Все эти вычисления можно произвести, написав в программе оператор.

```
ANS = ROOTF(16.9, R — S, T + 6.9) + COSF(X)
```

Предположим, что в дальнейшем в той же программе необходимо вычислить значение этой же функции при A = = DATA(I), B = DATA (I + 1) и C = 0.087, возведенное в куб, и присвоить результату наименование TEMP. Вычисления производятся оператором

```
TEMP = ROOTF(DATA(I), DATA(I + 1), 0.087)**3
```

Подчеркнем еще раз, что переменные A, B и C в определении функции не имеют никакого отношения к переменным с теми же самыми наименованиями, которые могут встретиться в других местах программы. Для иллюстрации предположим, что необходимо найти корень уравнения

$$22.97X^2 + AX + B = 0,$$

где A и B — некоторые переменные в программе. Корень можно найти с помощью следующего оператора:

```
VAL = ROOTF(22.97, A, B)
```

Переменные A и B, которые имеются в этом операторе, не имеют никакого отношения к переменным A и B, использованным для определения функции. В общем аргументы в определении функции не что иное, как «пустышки», определяющие лишь порядок подстановки фактических значений в участок рабочей программы, возникший при трансляции определения функции.

Хорошим примером употребления арифметического оператора-функции может служить некоторое видоизменение программы расчета светимости электрической лампочки (практический пример 8 из гл. 6). Вспомним, что в программе пять раз пришлось писать арифметический оператор для вычисления подинтегральной функции, меняя каждый раз аргумент. Теперь мы введем арифметический оператор-функцию в начале программы и будем обращаться к этому оператору всякий раз, когда потребуется.

Подинтегральная функция определялась формулой

$$E = \frac{1}{x^5\left(e^{\frac{1.432}{Tx}} - 1\right)}.$$

Ее легко записать в виде арифметического оператора-функции

```
EFFF(X) = 1.0/(X**5*(EXPF(1.432/T*X)) — 1.0))
```

Наименование функции было выбрано с таким расчетом, чтобы удовлетворить правилу, гласящему, что наименование функции должно содержать от четырех до семи букв или цифр и оканчиваться на букву F. Переменная X, как это и должно быть при определении оператора-функции, является «пустышкой», которая только определяет способ вычислений: когда в программе встречается выражение, где использована функция, то с ее аргументом производятся те же операции, что и над переменной X в ее определении.

Обратим внимание, что в определении функции использована переменная, не указанная в качестве аргумента,— переменная T. Как мы уже говорили, X является «пустышкой», поскольку является аргументом. А так как переменная T не является аргументом, то она не является и «пустышкой»; при вычислении функции эта переменная будет иметь то же самое значение, как и в любом другом месте программы. Как мы уже говорили, использование в определении функции переменных, не являющихся ее аргументами, вполне допустимо; как мы видим, при этом отпадает необходимость перечислять в качестве аргументов переменные, которые не изменяются при вызове функции из различных частей программы.

Полностью усовершенствованная программа приведена на рис. 9.1. Отметим некоторые изменения по сравнению с программой на рис. 6.7.

В гл. 11 было вкратце упомянуто, что наименование переменной не должно совпадать с наименованием функции или с наименованием функции без конечной буквы F. Это правило применимо не только к функциям, предусмотренным в программе-трансляторе, но и к арифметическим операторам-функциям. Поэтому наименование переменной EFF на рис. 6.7 нельзя использовать в программе

на рис. 9.1, так как функции присвоено наименование EFFF. Совершенно безразлично, изменить ли наименование функции или наименование переменной, лишь бы было

```
1    5 6 7  FORTRAN STATEMENT
       EFFF(X) = 1.0 / (X**.5 * (EXPF(1.432/(T*X)) - 1.0))
       READ 60, TEMP1, TEMP2, TMPINC, A, B, N
   60  FORMAT (5F10.0, I4)
       FN = N
       H = (B - A) / FN
       T = TEMP1
  100  SUM4 = 0.0
       SUM2 = 0.0
       I = 1
  120  FI = I
       X = A + FI * H
       SUM4 = SUM4 + EFFF(X)
       SUM2 = SUM2 + EFFF(X + H)
       IF (I - N + 3) 210, 320, 320
  210  I = I + 2
       GO TO 120
  320  EFFIC = 64.77 * H / 3.0 * (4.0*SUM4 + 2.0*SUM2
      1     + EFFF(A) + 4.0*EFFF(B - H) + EFFF(B)) / T**4
       PRINT 610, T, EFFIC
  610  FORMAT (2E20.8)
```

```
1    5 6 7  FORTRAN STATEMENT
       T = T + TMPINC
       IF (T - TEMP2) 100, 100, 200
  200  STOP
       END
```

Рис. 9.1. Усовершенствованный вариант программы рис. 6.7 с использованием арифметического оператора-функции и с другими незначительными изменениями.

удовлетворено правило. На рис. 9.1 функции присвоено наименование EFFF, а переменной — наименование EFFIC.

Несколько изменен метод вычисления последовательных значений X. Вместо того чтобы последовательно прибавлять по 2H (это привело бы к накоплению ошибок округления), значение переменной I преобразовывается в действительную форму и умножается на H.

Других изменений в программе нет.

9.4. ПОДПРОГРАММЫ FUNCTION И SUBROUTINE

При всей своей очевидной полезности для написания программ арифметический оператор-функция имеет два серьезных ограничения: его объем ограничен одним оператором и с его помощью можно вычислить всего одно значение. Подпрограммы FUNCTION и SUBROUTINE устраняют эти ограничения.

Но дело не только в этих двух ограничениях. Замечательно то, что эти два типа подпрограмм являются подпрограммами в полном смысле этого слова: их можно составлять и транслировать независимо от основной программы, частью которой они являются. Наименования переменных в этих подпрограммах полностью независимы от наименований переменных в основной программе и в других подпрограммах. Эти подпрограммы могут иметь все свои операторы DIMENSION и некоторые другие вспомогательные операторы, описанные ниже. Короче говоря, подпрограммы FUNCTION и SUBROUTINE совершенно не зависят от основной программы, и в то же время очень легко организовать «связь» между основной программой и этими подпрограммами. Это означает, что большую программу можно разделить на части и транслировать их независимо одну от другой. Эта возможность создает двойное удобство.

Во-первых, одну и ту же подпрограмму можно использовать с различными основными программами. Например, многие программы могут включать в себя решение системы линейных алгебраических уравнений методом исключения. Так как при решении систем уравнений приходится иметь дело с массивами информации и так как решение системы уравнений нельзя записать в виде одного оператора, то арифметический оператор-функцию в данном случае применить не удастся. Однако можно написать подпро-

грамму SUBROUTINE, с помощью которой можно проделать все нужные вычисления. Эту подпрограмму можно оттранслировать отдельно и использовать с любой основной программой. Единственное, что при этом необходимо,— это написать основную программу, учитывая правила использования подпрограммы.

Во-вторых, можно свободно транслировать различные участки программ независимо один от другого и ставить их на машину для вычислений. Это значит, что такие участки можно проверять по мере их написания, а это может явиться большим преимуществом. Кроме того, иногда очень трудно определить, работает ли полная программа должным образом, если нет уверенности в правильной работе подпрограмм. Отдельная трансляция позволяет отлаживать и проверять эти подпрограммы независимо и уже потом вставлять их в общую программу.

Таким образом, подпрограммы обладают тремя основными достоинствами. Первое из них то, о котором говорилось в введении к этой главе: оператор или группа операторов, расположенных в определенном месте программы, могут быть вызваны к действию обращением из любого другого места программы. Как уже указывалось, этим экономится труд программиста, так как не приходится дублировать участки программы, и экономится машинная память, которую занимали бы эти задублированные участки. Второе достоинство подпрограмм состоит в том, что их можно использовать с различными основными программами. Третье достоинство — возможность разделения большой программы на подпрограммы и независимой проверки подпрограмм. Независимо от причин их использования подпрограммы являются мощным средством языка FORTRAN.

Так же, как и в случае арифметического оператора-функции, необходимо тщательно соблюдать правила определения и правила использования подпрограммы. Чтобы *определить* те вычисления, которые будет производить подпрограмма FUNCTION, необходимо написать участок программы, содержащий необходимые операторы, перед этим участком программы расположить слово FUNCTION и наименование функции, а после участка программы поставить END. Наименование функции может иметь от одной

до шести букв или цифр, первой должна стоять буква, причем тогда и только тогда, когда значение функции целое, первой должна стоять одна из букв I, J, K, L, M, N; последней не должна быть буква F, если в наименовании содержится больше трех букв и цифр. (Обратите внимание на различие в наименовании арифметического оператора-функции и подпрограммы FUNCTION.) Наименование функции должно появиться по крайней мере один раз в подпрограмме в качестве переменной в левой части арифметического оператора или в списке переменных в операторе ввода. За наименованием функции следуют скобки, внутри которых перечисляются ее аргументы. Если количество аргументов больше одного, то внутри скобок они отделяются друг от друга запятыми.

Так же, как и в случае арифметического оператора-функции, переменные, стоящие в скобках после наименования, являются только «пустышками», образцом, согласно которому будут производиться вычисления в дальнейшем. В качестве аргументов можно употреблять переменные без индексов и наименования массивов. Внутри подпрограммы, однако, можно без всяких ограничений использовать переменные с индексами. Подпрограмма должна содержать по крайней мере один оператор RETURN по причинам, которые мы объясним ниже.

Для того чтобы *использовать* подпрограмму FUNCTION, необходимо только написать наименование функции там, где необходимо получить ее значение, и после него подставить в скобки подходящие выражения вместо аргументов. При этом обращение к подпрограмме производится следующим образом. Подпрограмма FUNCTION преобразовывается транслятором в последовательность машинных команд и записывается в памяти машины. Если в основной программе встречается наименование подпрограммы, то производится передача управления к ее началу. После того, как вычисления по подпрограмме закончены, управление должно быть передано обратно основной программе. Для этой цели и служит один или несколько операторов RETURN; по окончании действия подпрограммы этот оператор возвращает управление к месту ее вызова. (Вообще говоря, при использовании арифметического оператора-функции все делается точно так же; разница заключается

в том, что вся подпрограмма в этом случае состоит из одного оператора и не нужен специальный оператор, устанавливающий момент окончания вычислений.)

Рассмотрим простой пример применения подпрограммы FUNCTION. Предположим, что в некоторой программе часто необходимо вычислять функцию, приведенную на на рис. 1.3. Вычисление этой функции можно оформить в виде подпрограммы, приведенной на рис. 9.2, где функции присвоено наименование Y.

Если теперь необходимо вычислить значение этой функции для аргумента, равного GRS —6.8, разделить результат

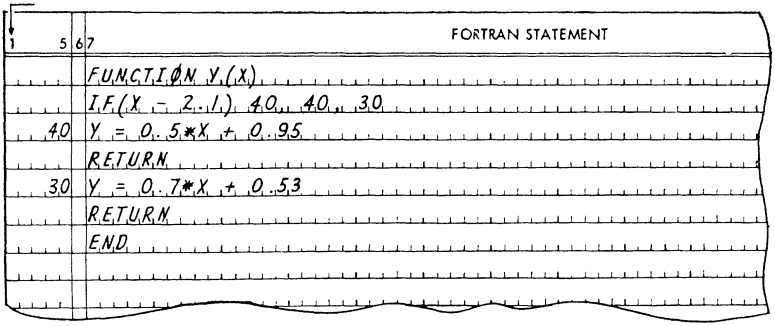


Рис. 9.2. Пример подпрограммы FUNCTION.

на 12.99 и обозначить частное через EWR, то надо вставить в основную программу следующий оператор:

$$EWR = Y(GRS - 6.8)/12.99$$

Если необходимо вычислить корень квадратный из значения функции для аргумента, равного корню квадратному из 1.0+ RHO, и обозначить результат через PDX, то можно написать такой оператор:

$$PDX = SQRTF(Y(SQRTF(1.0+RHO)))$$

Подпрограмма FUNCTION может иметь много аргументов, в том числе и массивы. Например, предположим, что необходимо найти произведение элементов двумерного массива (квадратной матрицы), лежащих на главной диагонали (т. е. таких элементов, у которых номер строки и номер

столбца одинаковы). Массивы, для которых подсчитывается это произведение, должны быть перечислены, как обычно, в операторе DIMENSION в основной программе, и массивы эти не должны превосходить некоторых максимальных размеров. Наименования массивов в подпрограмме и в списке аргументов функции по-прежнему будут «пустышками», но они все равно должны быть перечислены в операторе DIMENSION в подпрограмме. Предположим, что максимально возможный размер массивов составляет 10×10 элементов, но может быть и меньше. Поэтому то количество столбцов и строк, для которого нужно в действительности вычислять произведение, задается с помощью целой переменной. Подпрограмма приведена на рис. 9.3.

```
1    5 6 7   FORTRAN STATEMENT
        FUNCTION DIAGPR(A,N)
        DIMENSION A(10,10)
        DIAGPR = A(1,1)
        DO 69 I = 2,N
   69   DIAGPR = DIAGPR * A(I,I)
        RETURN
        END
```

Рис. 9.3. Другой пример подпрограммы FUNCTION.

Если теперь необходимо вычислить произведение диагональных элементов для некоторого массива DATA, размер которого задан значением переменной LAST, можно написать

```
DET = DIAGPR(DATA, LAST)
```

Чтобы найти квадрат произведения диагональных элементов некоторого массива X, размер которого задан значением переменной JACK, можно написать

```
EIG = DIAGPR(X, JACK)**2
```

Как видно из приведенных примеров, подпрограмма FUNCTION имеет все те же свойства, что и арифметический оператор-функция, но теперь можно использовать много

операторов, а не только один; кроме того, в подпрограмме можно употреблять все операторы языка ФОРТРАН, а не только арифметические операторы. Подпрограмма может обращаться к другим подпрограммам, но не сама к себе; две подпрограммы также не могут обращаться друг к другу[1]).

Согласно приведенному описанию, подпрограмма FUNCTION может вычислить всего одно значение, а именно то, которое соответствует ее наименованию. В действительности это не так — количество выходных величин может быть сколь угодно большим: любой из аргументов может представлять собой выходную величину. Чтобы понять, как это делается, рассмотрим следующее усложнение предыдущей подпрограммы. Предположим, что если произведение элементов массива, лежащих на главной диагонали, меньше или равно 100, то в основной программе необходимо перейти к оператору 12; если это произведение заключено между 100 и 1000, то нужно перейти к оператору 123; если же оно больше или равно 1000, то нужно перейти к оператору 1234. Все эти номера относятся к операторам в *основной* программе. Конечно, необходимые передачи управления можно было бы осуществить в самой основной программе с помощью операторов IF, но если эту процедуру придется часто повторять, то желательно, чтобы возможно бо́льшая часто повторяющихся действий производилась подпрограммой.

С этой целью изменим сначала наименование функции с DIAGPR на NDIAGP, так что теперь ее значение должно быть целым. После этого пишем усовершенствованную подпрограмму таким образом, что значение NDIAGP становится равным $-1, 0$ или $+1$ в зависимости от того, было ли произведение диагональных элементов меньше 100, между 100 и 1000 или больше 1000 соответственно. В число

---

[1]) Подпрограмма, обращающаяся сама к себе, называется *рекурсивной*. Рекурсивность допускается в АЛГОЛе, где это оказывается очень полезным для множества невычислительных приложений, например при написании программ-трансляторов, операциях с языками (например, с английским) или с математическими *символами*, безотносительно к их численным значениям. Иногда можно создать условия для рекурсивности в ФОРТРАНе путем «склеивания» аргументов в массивы.

аргументов функции мы включаем при этом DIAGPR и в ходе выполнения подпрограммы мы присваиваем этой переменной значение, равное произведению диагональных элементов. Усовершенствованная подпрограмма приведена на рис. 9.4.

```
1    5 6 7  FORTRAN STATEMENT
       FUNCTION NDIAGP (A, N, DIAGPR)
       DIMENSION A(10,10)
       DIAGPR = A(1,1)
       DO 69 I = 2, N
   69  DIAGPR = DIAGPR * A(I,I)
       IF (DIAGPR - 100.0) 68, 68, 67
   68  NDIAGP = -1
       RETURN
   67  IF (DIAGPR - 1000.0) 66, 65, 65
   66  NDIAGP = 0
       RETURN
   65  NDIAGP = 1
       RETURN
       END
```

Рис. 9.4. Пример подпрограммы SUBROUTINE.

Теперь предположим, что необходимо вычислить произведение диагональных элементов массива BETA, имеющего 8 строк и 8 столбцов. Это произведение должно стать новым значением переменной PRODCT, в зависимости от которого мы должны перейти к операторам 12, 123 или 1234, как было описано выше.

Оператор перехода запишется так:

```
IF (NDIAGP (BETA, 8, PRODCT)) 12, 123, 1234
```

В этом операторе встречается наименование функции FUNCTION с подходящими выражениями вместо аргументов. Поэтому будет вызвана соответствующая подпрограмма, в ходе выполнения которой будет вычислено значение функции и присвоено новое значение третьему аргументу (в нашем случае — переменной PRODCT). Когда

управление передается от подпрограммы обратно основной программе, то с помощью оператора IF выбирается тот или иной путь продолжения вычислений. В любом случае в ходе последующих вычислений можно использовать только что вычисленное значение PRODCT.

Подпрограмма SUBROUTINE почти полностью аналогична подпрограмме FUNCTION с тремя различиями:

1. SUBROUTINE не имеет никакого численного значения, связанного с ее наименованием. Все выходные параметры определяются через аргументы; число выходных параметров может быть любым.

2. К подпрограмме SUBROUTINE нельзя обратиться, просто написав ее наименование, так как с ним не связано никакое численное значение. Для обращения к SUBROUTINE используется оператор CALL, после которого пишется наименование подпрограммы и перечисляются аргументы.

3. Поскольку выход подпрограммы SUBROUTINE может быть комбинацией из целых и действительных чисел, то первая буква наименования не используется для указания целого или действительного. В остальном наименование SUBROUTINE подчиняется тем же правилам, что и наименование FUNCTION.

Во всем остальном подподграммы SUBROUTINE и FUNCTION совершенно аналогичны.

Основные свойства подпрограммы SUBROUTINE можно проиллюстрировать на следующем примере. Предположим, что в некоторой программе часто требуется находить наибольший по абсолютной величине элемент в заданной строке массива размерами 10 на 10. Таким образом, в качестве входных параметров для SUBROUTINE необходимо наименование массива и номер строки. Выходными величинами будут абсолютная величина наибольшего элемента в заданной строке и номер столбца, в котором находится этот элемент. Сама подпрограмма может быть составлена, например, так, как показано на рис. 9.5.

Теперь предположим, что необходимо найти наибольший элемент в третьей строке массива ZETA. Этот наибольший элемент нужно назвать DIVIS, и номер столбца, в котором он расположен, нужно назвать NCOL. Напишем оператор

```
CALL LARGE (ZETA,3, DIVIS, NCOL)
```

Этот оператор передает управление подпрограмме, в ходе выполнения которой вычисляются значения DIVIS и NCOL, а затем управление передается оператору, следующему за CALL. Если в дальнейшем в основной программе

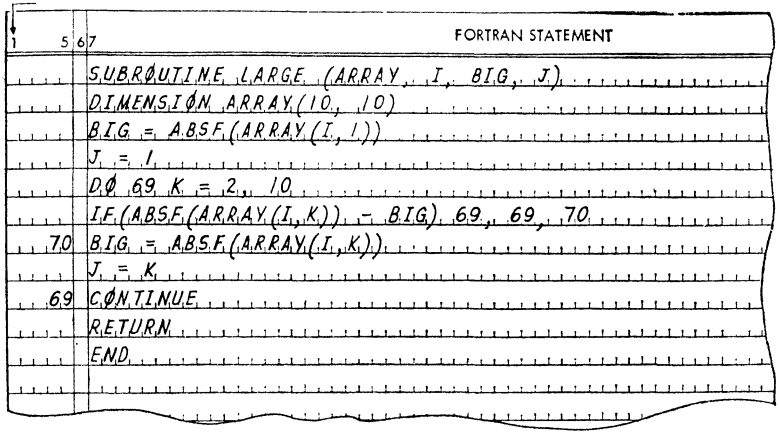


Рис. 9.5. Пример подпрограммы SUBROUTINE.

встретится необходимость найти наибольший элемент массива DETAIL, стоящий в (M + 2)-й строке, присвоить абсолютную величину этого элемента переменной SIZE, а номер столбца, где он находится,— переменной KWHICH, то все это можно сделать с помощью оператора

CALL LARGE (DETAIL, M + 2, SIZE, KWHICH)

Следует еще раз подчеркнуть независимость наименований переменных в основной программе и в подпрограммах. В основной программе может, например, встретиться оператор

CALL LARGE (ARRAY, I, BIG, J)

Если имеется такой оператор, то все переменные, стоящие в скобках, *должны быть определены в основной программе*. Наименования I в основной программе и в подпрограмме совершенно независимы друг от друга. Это и должно быть так: наименование I в подпрограмме показывает, что делать с величиной, приходящей из основной

Таблица 9.1

	Функции, предусмотренные в программе-трансляторе	Арифметические операторы-функции	Подпрограмма FUNCTION	Подпрограмма SUBROUTINE
Наименование	4—7 букв или цифр; первой должна быть буква, причем в тех и только тех случаях, когда значение функции целое; первой должна быть буква X; последней должна быть буква F	4—7 букв или цифр; первой должна быть буква, причем в тех и только тех случаях, когда значение функции целое; первой должна быть буква X; последней должна быть буква F	1—6 букв или цифр; первой должна быть буква, причем в тех и только тех случаях, когда значение функции целое; первой должна быть одна из букв I, J, K, L, M. N. Последней не должна быть буква F, если наименование содержит более трех букв или цифр	1—6 букв или цифр; первой должна быть буква; последней не должна быть буква F, если наименование содержит более трех букв или цифр
Способ определения	Предусмотрены в программе-трансляторе	Один арифметический оператор перед началом программы	Любое количество операторов после слова FUNCTION	Любое количество операторов после слова SUBROUTINE

Продолжение табл. 9.1

	Функции, предусмотренные в программе-трансляторе	Арифметические операторы-функции	Подпрограмма FUNCTION	Подпрограмма SUBROUTINE
Способ использования	Написать наименование там, где необходимо значение функции	Написать наименование там, где необходимо значение функции	Написать наименование там, где необходимо значение функции	Оператор CALL
Количество аргументов	Один или более в зависимости от определения	Один или более в зависимости от определения	Один или более в зависимости от определения	Любое количество, в том числе *и ни одного*, в зависимости от определения
Количество выходных значений	Одно	Одно	Одно соответствует наименованию функции, остальные могут быть указаны в качестве аргументов	Любое количество

программы, в то время как наименование I в основной программе показывает величину, которая должна быть вычислена в ней перед обращением к подпрограмме.

9.5. ТАБЛИЦА ОСНОВНЫХ ХАРАКТЕРИСТИК ФУНКЦИЙ И ПОДПРОГРАММ

В ФОРТРАНе предусмотрено всего четыре типа функций и подпрограмм: функции, предусмотренные в программе-трансляторе, арифметические операторы-функции, подпрограммы FUNCTION и подпрограммы SUBROUTINE. Их основные характеристики сведены в табл. 9.1.

9.6. ОПЕРАТОРЫ EQUIVALENCE И COMMON

Эти два неисполняемых оператора создают программисту определенные удобства при наименовании переменных и распределении их по ячейкам памяти ЭЦВМ.

Оператор EQUIVALENCE приводит к тому, что значения двух и даже более переменных могут находиться в одной и той же ячейке памяти. Польза от такого уплотнения памяти двоякая.

С одной стороны, это позволяет программисту добиться того, что две или более переменных будут означать одно и то же. Предположим, что при написании какой-то длинной программы программист непроизвольно изменил наименования переменных и что в окончательной программе X, X1 и RST6 должны означать одну и ту же переменную. Для этого нет необходимости переписывать всю программу, что является достаточно сложным делом, причем при переписывании программы легко допустить различные ошибки. Гораздо проще написать

```
EQUIVALENCE (X, X1, RST6)
```

и ошибка исправлена.

Другое приложение оператора EQUIVALENCE относится к тому случаю, когда можно записывать в одной и той же ячейке памяти переменные, которые не бывают нужны одновременно. Предположим, что при вводе информации появилась некоторая переменная I27, но затем больше не использовалась. В ходе вычислений по программе была

введена переменная NP1, примененная совместно с оператором DO, но после окончания цикла DO эта переменная больше не употреблялась. Впоследствии для той же цели была введена переменная JJM2, а при печати выходной информации была введена переменная NEXT1. При таком положении дел эти переменные никогда не бывают нужны одновременно, и нет смысла отводить им четыре независимые ячейки памяти. Если у программиста возникают затруднения с количеством ячеек памяти, то можно свести все четыре переменные в одну ячейку памяти, написав

EQUIVALENCE(I27, NP1, JJM2, NEXT1)

Того же самого результата можно было бы, конечно, добиться, изменив наименования переменных, но использовать EQUIVALENCE явно проще.

Эти два приложения EQUIVALENCE различны только внешне, сам оператор и его обработка с помощью программы-транслятора одинаковы в обоих случаях.

Один оператор EQUIVALENCE может установить эквивалентность любого количества наборов переменных. Например, если необходимо сделать эквивалентными переменные A и B и также сделать эквивалентными переменные X и Y, то достаточно написать

EQUIVALENCE (A, B), (X, Y)

Очень редко случается так, что нехватка места в памяти ЭЦВМ вынуждает устанавливать эквивалентность переменных без индексов. Наибольшая ценность оператора EQUIVALENCE состоит как раз в том, что с его помощью можно сделать эквивалентными целые массивы. К сожалению, правила использования оператора EQUIVALENCE для этой цели сильно отличаются в различных вариантах ФОРТРАНа. Поэтому наиболее целесообразно ознакомиться с этими правилами непосредственно для того варианта, который применен на его ЭЦВМ.

Оператор COMMON

Уже говорилось о том, что наименования переменных в подпрограммах независимы от наименований переменных в основной программе: наименование X в основной программе совершенно не обязано означать то же самое, что и

наименование X в подпрограмме. Однако если необходимо, чтобы эти два X означали одну и ту же величину, то и в основной программе и в подпрограмме необходимо поместить оператор

COMMON X

Этот оператор может выглядеть и иначе. Предположим, например, что имеется такая пара операторов

основная программа: COMMON X, Y, I
подпрограмма: COMMON A, B, J

Тогда X и A будут записаны в одной и той же ячейке памяти, то же относится к паре Y и B и к паре I и J. При этом не только экономятся ячейки памяти, но также устанавливается соответствие между переменными в основной программе и в подпрограммах. Например, если даже не упоминать X и A в качестве аргументов, то при изменении, например, в подпрограмме значения A точно таким же образом изменится значение X в основной программе.

Вероятно, наиболее частое использование оператора COMMON состоит в том, чтобы установить взаимное соответствие между переменными в основной программе и в одной или нескольких подпрограммах. В этом случае во всех подпрограммах и в основной программе ставятся одинаковые операторы COMMON. После этого те переменные, которые перечислены в операторе COMMON, можно употреблять в основной программе и в различных подпрограммах, даже не упоминая их при перечислении аргументов. Использование оператора COMMON иллюстрируется в нижеследующем практическом примере.

9.7. ПРАКТИЧЕСКИЙ ПРИМЕР 11: РЕШЕНИЕ КВАДРАТНЫХ УРАВНЕНИЙ С ПОМОЩЬЮ ПОДПРОГРАММ

В этом практическом примере мы проиллюстрируем некоторые способы применения подпрограмм и оператора COMMON, о которых мы говорили выше. С точки зрения собственно вычислений этот пример очень прост — решение квадратных уравнений производится с помощью все той же формулы из курса средней школы. Основное, на что следует обратить внимание при разборе этого примера,— организация программы и форматы ввода — вывода.

Некоторые приемы ввода и вывода информации нами еще до сих пор не рассматривались, хотя их описание и имеется в приложении 1. Эти приемы ввода и вывода рассмотрены в данном примере очень кратко; те из читателей, которые хотят обратить внимание в основном на программирование с помощью подпрограмм, могут свободно пропустить все подробности относительно ввода и вывода информации.

Программа составлена для того, чтобы решать квадратные уравнения вида $Ax^2 + Bx + C = 0$. С ее помощью можно вводить в ЭЦВМ коэффициенты большого количества таких уравнений (многие сотни), решать их и отпечатывать легко читаемую таблицу, где для каждого уравнения содержались бы коэффициенты и корни. Каждая перфокарта содержит коэффициенты двух уравнений, всего 6 чисел. Однако на выходной ленте каждое уравнение должно занимать отдельную строку. Корни могут быть вещественными и комплексными. Каждая страница выходной ленты должна иметь заголовок, страницы должны быть пронумерованы в процессе печати, на каждой странице должно быть напечатано ровно 20 строк выходной информации, строки должны следовать с двойным интервалом.

В программе следует использовать две подпрограммы SUBROUTINE. К каждой из них придется обращаться дважды за время обработки одной перфокарты. Этим самым мы избегаем дублирования команд в подпрограммах и упрощаем проверку и отладку всей программы. Основная программа содержит в себе ввод данных с перфокарт, печать заголовков для выходной информации, нумерацию страниц, подсчет строк и определение конца расчетов по пробной перфокарте, в которой $A = 0$. Очевидно, что при работе не может встретиться перфокарта с $A = 0$, так как при этом уравнение уже не является квадратным.

С помощью первой подпрограммы будут находиться решения уравнений, причем принимается во внимание, что если дискриминант $B^2 - 4AC$ отрицателен, то корни уравнения комплексные. Входными данными для этой подпрограммы, называемой SOLVE, являются три значения коэффициентов уравнения, поэтому три наименования коэффициентов должны быть перечислены в скобках в качестве аргументов. Выходными величинами являются дейст-

вительные и мнимые части двух корней уравнения, которым даны наименования X1REAL, X1IMAG, X2REAL и X2IMAG. Эти четыре переменные употребляются как в основной программе, так и в обеих подпрограммах. Поэтому они перечислены в операторах COMMON во всех трех местах, в результате чего отпала необходимость объявлять их аргументами подпрограмм.

С помощью второй подпрограммы, называемой OUTPUT, производится печатание коэффициентов и корней уравнения. Желательно печатать результаты так, чтобы с первого взгляда было видно, является ли корень вещественным или комплексным. Если корни вещественные, то место, предназначенное для печати мнимой части корня, должно остаться свободным; если корни чисто мнимые, то должно остаться свободным то место, которое предназначено для печати действительной части корня. Вспомним, что если коэффициенты уравнения — вещественные числа, то комплексные корни всегда будут взаимно сопряженными; не может быть такого случая, чтобы только один из корней был комплексным или чтобы у двух комплексных корней были различные действительные части.

Блок-схема основной программы показана на рис. 9.6. В этой довольно простой блок-схеме следует отметить одно нововведение: после того как обнаружена пробная карта и соответственно определен момент конца вычислений, управление передается не оператору STOP, а оператору CALL EXIT. Это сделано в предположении, что программа будет выполняться на ЭЦВМ, работающей под управлением программы-монитора. В данном случае оператор CALL EXIT возвращает управление монитору точно так же, как оператор RETURN в конце подпрограммы возвращает управление основной программе.

Блок-схема подпрограммы для нахождения корней уравнения показана на рис. 9.7. Процедура вычисления корней уравнения в этой подпрограмме находится где-то между абсолютно необходимым минимумом сложности, требующимся для различения вещественных и комплексных корней, и более сложными программами, в которых можно предусмотреть каждый частный случай. Минимально необходимо было бы производить вычисления комплексных корней, если дискриминант отрицателен, и производить вычисления

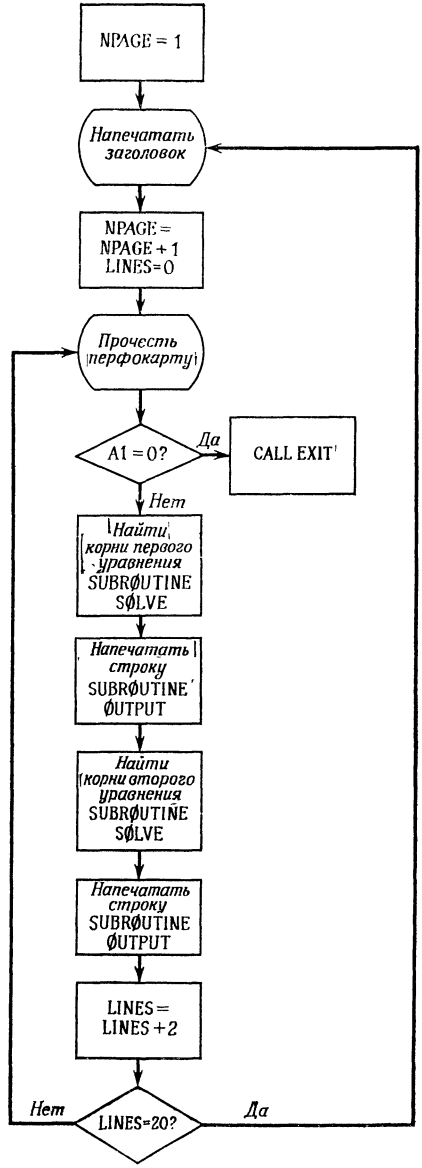


Рис. 9.6. Блок-схема основной программы для решения квадратных уравнений (практический пример 11).

вещественных корней, если дискриминант нуль или положителен. Так как оператор IF автоматически дает нам разделение на три ветви, то целесообразно использовать это преимущество и не вычислять корень квадратный из нуля при нулевом дискриминанте. Можно было бы, конечно, предусмотреть и другие частные случаи. Если C равно нулю, то оба корня вещественные, причем один равен нулю, а второй $-B/A$. Если B равно нулю, то формулы также несколько упрощаются. Если же и B, и C равны нулю,

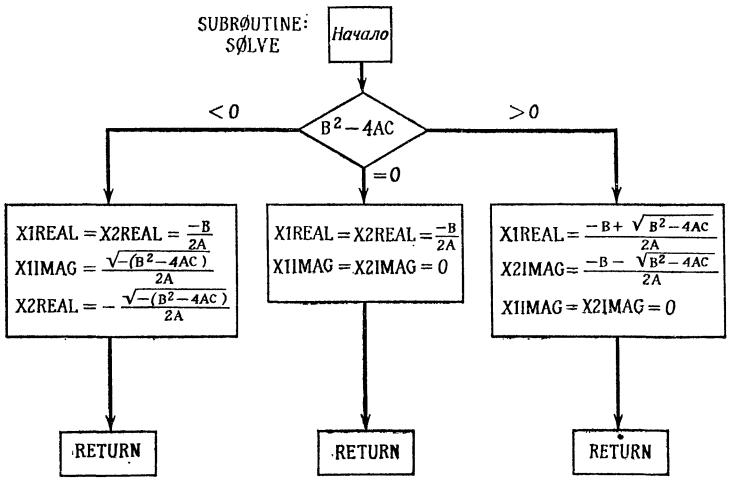


Рис. 9.7. Блок-схема подпрограммы для нахождения корней в практическом примере 11.

то, очевидно, и оба корня будут нулевыми, хотя и трудно представить себе задачу, где могло бы возникнуть такое уравнение.

Тем не менее всегда следует вовремя остановиться. Конечно, при учете различных частных случаев можно добиться некоторой экономии машинного времени, но только тогда, когда дополнительные затраты времени на их учет не перевесят всю экономию. И даже если машинное время все-таки экономится, усложнение программы при организации полной системы тестов может оказаться неоправданным с точки зрения программиста.

Блок-схема подпрограммы для печати результатов приведена на рис. 9.8. Эта подпрограмма, названная OUTPUT, также довольно проста. Заметим, что нет необходимости проверять оба мнимых корня на равенство нулю, так как они будут одновременно равны или не равны нулю. Для вещественных корней это утверждение несправедливо.

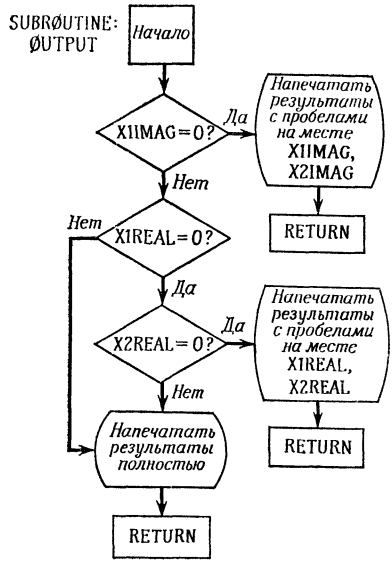


Рис. 9.8. Блок-схема подпрограммы для печати результатов в практическом примере 11.

Основная программа приведена на рис. 9.9. Она начинается с оператора COMMON, в котором перечисляются действительные и мнимые части двух корней уравнения. Точно такие же операторы COMMON имеются в обеих подпрограммах, так что эти четыре наименования неизвестных означают одно и то же во всех частях программы. Без операторов COMMON это, конечно, было бы не так.

Теперь мы присваиваем номеру страницы значение 1 и печатаем заголовок, который в основном состоит из названий столбцов будущей таблицы; единственной переменной в этой строке является номер страницы. Печатание

текста, состоящего из букв и цифр, осуществляется с помощью спецификации поля, которая еще до сих пор не рассматривалась — спецификация типа H[1]). При использовании этой спецификации мы сначала пишем целое число, которое означает количество знаков текста, затем букву H, затем сам текст. Например, при записи в операторе FORMAT вида 7HX2REAL на выходном листе бумаги

```
1    5 6 7  FORTRAN STATEMENT
       COMMON X1REAL, X1IMAG, X2REAL, X2IMAG
       NPAGE = 1
     5 PRINT 8, NPAGE
     8 FORMAT (1H1, 9X, 1HA, 14X, 1HB, 14X, 1HC, 11X,
     1   7HX1 REAL, 8X, 7HX1 IMAG, 8X, 7HX2 REAL, 8X,
     2   1,7HX2 IMAG      PAGE, I4///)
       NPAGE = NPAGE + 1
       LINES = 0
    14 READ 15, A1, B1, C1, A2, B2, C2
    15 FORMAT (6F10.0)
       IF (A1) 16, 17, 16
    17 CALL EXIT
    16 CALL SOLVE (A1, B1, C1)
       CALL OUTPUT (A1, B1, C1)
       CALL SOLVE (A2, B2, C2)
       CALL OUTPUT (A2, B2, C2)
       LINES = LINES + 2
       IF (LINES - 40) 14, 5, 5
       END
```

Рис. 9.9. Основная программа для практического примера 11.

будет напечатано слово X2REAL. Текст может состоять из любых знаков, которые может напечатать выходная печатная машина, включая и «знак» пробел.

Самая первая спецификация поля в операторе FORMAT под номером 8 имеет совсем другое назначение. Первый символ не печатается, а задает расстояние между строками

[1]) Названа в честь Германа Холлерита (Hollerith), который изобрел способ изображения букв алфавита на перфокартах в 1890 г.

выходной ленты: пробел означает обычный одинарный интервал между строками, нуль — двойной интервал, а единица определяет, что данная строка начнет печататься с новой страницы. Таким образом, когда мы пишем 1H1 в начале оператора FORMAT, то тем самым указываем, чтобы эта строка была напечатана уже на новой странице после соответствующего сдвига бумажной ленты. Следующая спецификация поля нам также незнакома; она обозначается буквой X. Такая спецификация задает пробел, и в данном случае 9X означает, что в этом месте на бумаге будет оставлено девять свободных позиций. То же самое можно было бы сделать с помощью спецификации типа H, оставив эти девять позиций в виде пробела при написании оператора FORMAT, но X несколько проще и при его употреблении меньше вероятность допустить ошибку. Далее в операторе FORMAT мы находим весь остальной текст для заголовка. Отметим, что пробел при печати с помощью спецификации типа H необходимо оставлять и учитывать его при подсчете числа, определяющего длину текста. Мы видим несколько примеров этого, в особенности в последней спецификации типа H, где оказалось проще оставить шесть пробелов, нежели вводить спецификацию 6X. Естественно, подобные решения находятся на усмотрении программиста.

Спецификация типа I знакома нам: с ее помощью печатается номер страницы, причем ему отводятся для печати четыре позиции. В конце оператора FORMAT имеется еще одно нововведение: две косые черты создают дополнительный интервал между заголовком и первой строкой результатов. (Как мы увидим в приложении 1, косая черта применяется не только для этой цели.)

Внешний вид заголовка, напечатанного в соответствии с этим оператором FORMAT, показан на рис. 9.12.

После прибавления единицы к счетчику страниц и присвоения нулевого значения счетчику строк мы вводим с перфокарты коэффициенты двух уравнений. В операторе READ всем шести коэффициентам соответствуют различные наименования. Первый коэффициент в каждой перфокарте проверяется и в случае равенства его нулю управление передается оператору CALL EXIT, что заканчивает вычисления по программе.

Теперь можно находить корни для первого набора коэффициентов, так что мы вызываем подпрограмму SUBROUTINE SOLVE, с помощью которой решается уравнение. Аргументами являются три первых коэффициента

```
1    5 6 7  FORTRAN STATEMENT
       SUBROUTINE SOLVE (A, B, C)
       COMMON X1REAL, X1IMAG, X2REAL, X2IMAG
       DISC = B**2 - 4.0 * A * C
       IF (DISC) 50, 60, 70
  50   X1REAL = -B / (2.0 * A)
       X2REAL = X1REAL
       X1IMAG = SQRTF(-DISC) / (2.0 * A)
       X2IMAG = -X1IMAG
       RETURN
  60   X1REAL = -B / (2.0 * A)
       X2REAL = X1REAL
       X1IMAG = 0.0
       X2IMAG = 0.0
       RETURN
  70   S = SQRTF(DISC)
       X1REAL = (-B + S) / (2.0 * A)
       X2REAL = (-B - S) / (2.0 * A)
       X1IMAG = 0.0
       X2IMAG = 0.0
       RETURN
       END
```

Рис. 9.10. Подпрограмма нахождения корней уравнения для практического примера 11.

первого уравнения. Когда вычисления по подпрограмме SOLVE заканчиваются, то переменным X1REAL, X1IMAG, X2REAL и X2IMAG присвоены некоторые значения, равные корням уравнения. Теперь мы вызываем подпрограмму для печатания результатов, также указывая в качестве ее аргументов только три коэффициента уравнения; четыре числа, соответствующие корням, переданы в эту подпрограмму с помощью оператора COMMON. Повторный вызов каждой из подпрограмм позволяет найти и напечатать решение второго уравнения, коэффициенты

которого содержались на той же перфокарте. После этого к счетчику числа строк прибавляется 2, проверяется, заполнена ли уже страница, и, если страница заполнена, на новой странице печатается заголовок.

Две подпрограммы, приведенные соответственно на рис. 9.10 и 9.11, довольно просты. В подпрограмме SOLVE введены две новые переменные DISC и S, чтобы не вычислять некоторых выражений многократно. Учтен

```
1    5 6 7  FORTRAN STATEMENT
      SUBROUTINE OUTPUT (A, B, C)
      COMMON X1REAL, X1IMAG, X2REAL, X2IMAG
      IF (X1IMAG) 90, 91, 90
   91 PRINT 95, A, B, C, X1REAL, X2REAL
   95 FORMAT (1H0, 1P4E15.4, 15X, 1PE15.4)
      RETURN
   90 IF (X1REAL) 100, 101, 100
  101 IF (X2REAL) 100, 102, 100
  102 PRINT 103, A, B, C, X1IMAG, X2IMAG
  103 FORMAT (1H0, 1P3E15.4, 15X, 1PE15.4, 15X, 1PE15.4)
      RETURN
  100 PRINT 110, A, B, C, X1REAL, X1IMAG, X2REAL, X2IMAG
  110 FORMAT (1H0, 1P7E15.4)
      RETURN
      END
```

Р и с. 9.11. Подпрограмма печати результатов для практического примера 11.

тот факт, что комплексные корни всегда являются комплексно-сопряженными; это также помогает избежать вычисления одного и того же выражения дважды.

Подпрограмма OUTPUT также довольно проста, но операторы FORMAT нужно рассмотреть внимательно. Пробелы в печати в случае чисто действительных и в случае чисто мнимых корней вводятся с помощью спецификации 15X. Заметим, что каждый оператор FORMAT начинается со спецификации типа 1HO. Как мы уже знаем, нуль в этой спецификации задает двойной интервал меж-

A	B	C	X1 REAL	X1 IMAG	X2 REAL	X2 IMAG	PAGE 3
1.0000E 00	-2.0000E 00	1.0000E 00	1.0000E 00		1.0000E 00		
1.0000E 00	-1.0000E 01	2.5000E 01	5.0000E 00		5.0000E 00		
1.0000E 00	-3.0000E 00	2.0000E 00	2.0000E 00		1.0000E 00		
2.0000E 00	-6.0000E 00	4.0000E 00	2.0000E 00		1.0000E 00		
1.0000E 00	1.0000E 00	-2.5500E 03	5.0000E 01		-5.1000E 01		
1.0000E 01	-2.0000E 01	1.0000E 01	1.0000E 00		1.0000E 00		
1.0000E 03	-2.0000E 03	1.0000E 03	1.0000E 00		1.0000E 00		
2.0000E-02	-4.0000E-02	2.0000E-02	1.0000E 00		1.0000E 00		
1.4320E 00	9.8760E 00	-4.5670E 00	4.3500E-01		-7.3316E 00		
8.8130E 00	-1.3104E 01	0.	1.4869E 00		0.		
2.3009E 01	1.9917E 01	0.	-5.1810E-09		-8.6562E-01		
1.0000E 00	0.	-1.0000E 00	1.0000E 00		-1.0000E 00		
1.0000E 00	0.	1.0000E 00		1.0000E 00		-1.0000E 00	
9.0000E 00	0.	3.6000E 01		2.0000E 00		-2.0000E 00	
1.0000E 00	2.0000E 00	5.0000E 00	-1.0000E 00	2.0000E 00	-1.0000E 00	-2.0000E 00	
6.3190E 00	4.3380E 00	2.3294E 01	-3.4325E-01	1.8891E 00	-3.4325E-01	-1.8891E 00	
-9.0000E 00	2.3000E 01	3.7000E 01	-1.1189E 00		3.6744E 00		
6.1000E 01	0.	8.7000E 01		1.1942E 00		-1.1942E 00	
6.1000E 01	2.0000E 00	8.7000E 01	-1.6393E-02	1.1941E 00	-1.6393E-02	-1.1941E 00	
6.1000E 01	1.5900E 02	8.7000E 01	-7.8145E-01		-1.8251E 00		

Рис. 9.12. Выходная печать при вычислениях по программам рис. 9.9, 9.10 и 9.11 (практический пример 11).

ду строками, в результате чего облегчается чтение таблицы результатов.

На рис. 9.12 показана одна страница выходных данных, которая могла бы быть напечатана с помощью нашей программы. Естественно, по сравнению с нормальными размерами таблица уменьшена.

Упражнения

*1. Определите арифметический оператор-функцию, с помощью которого можно было бы вычислить

$$\text{DENOMF (X)} = X^2 + \sqrt{1 + 2X + 3X^2}.$$

Затем используйте эту функцию, чтобы вычислить

$$\text{ALPHA} = \frac{6.9 + Y}{Y^2 + \sqrt{1 + 2Y + 3Y^2}},$$

$$\text{BETA} = \frac{2.1Z + Z^4}{Z^2 + \sqrt{1 + 2Z + 3Z^2}},$$

$$\text{GAMMA} = \frac{\sin Y}{Y^4 + \sqrt{1 + 2Y^2 + 3Y^4}},$$

$$\text{DELTA} = \frac{1}{\sin^2 Y + \sqrt{1 + 2\sin Y + 3\sin^2 Y}}.$$

2. Определите арифметический оператор-функцию, с помощью которого можно было бы вычислить

$$\text{SLGF (A)} = 2.549 \ln\left(A + A^2 + \frac{1}{A}\right).$$

Затем используйте эту функцию в вычислениях

$$\text{R} = X + \ln X + 2.549 \ln\left(X + X^2 + \frac{1}{X}\right),$$

$$\text{S} = \cos X + 2.549 \ln\left(1 + X + (1 + X)^2 + \frac{1}{1 + X}\right),$$

$$\text{T} = 2.549 \ln\left[(A - B)^3 + (A - B)^6 + \frac{1}{(A - B)^3}\right],$$

$$\text{U} = [B(I) + 6]^2 + 2.549 \ln\left[\frac{1}{B(I)} + \frac{1}{B(I)^2} + B(I)\right].$$

*3. Определите арифметический оператор-функцию, с помощью которого можно было бы вычислить

$$\text{S34F (X, A)} = \sqrt{X^2 - A^2}.$$

Затем используйте эту функцию для вычисления

$$\mathrm{SFK}=\frac{V\cdot\sqrt{V^2-R^2}}{2}-\frac{R}{2}\ln\left|V+\sqrt{V^2-R^2}\right|;$$

$$\mathrm{PSB}=\frac{[X(I)^2-B^2]^{7/2}}{7}+\frac{2B^2[X(I)^2-B^2]^{5/2}}{5}+\frac{B^4[X(I)^2-B^2]^{3/2}}{3}.$$

4. Определите арифметический оператор-функцию, с помощью которого можно было бы вычислить

$$\mathrm{SQUADF}(A,\ B,\ C,\ X)=\sqrt{AX^2+BX+C}.$$

Затем используйте эту функцию для вычисления

$$\mathrm{ETX}=\frac{4PZ+2Q}{(4PR-Q^2)\sqrt{PZ^2+QZ+R}},$$

$$\mathrm{AVP}=\sqrt{RY^2+SY+\sqrt{DY^2+FY+16}}.$$

***5.** Напишите подпрограмму FUNCTION, с помощью которой можно было бы вычислить

$$Y(X)=\begin{cases}1+\sqrt{1+X^2} & \text{при } X<0,\\ 0 & \text{при } X=0,\\ 1-\sqrt{1+X^2} & \text{при } X>0.\end{cases}$$

Затем напишите операторы для вычислений по следующим формулам, где использована обычная математическая запись «Y как функция $A+Z$» и т. д.

$$\mathrm{F}=2+Y(A+Z),$$

$$\mathrm{G}=\frac{Y[X(K)]+Y[X(K+1)]}{2},$$

$$\mathrm{H}=Y[\cos(2\pi X)]+\sqrt{1+Y(2\pi)}.$$

6. Напишите подпрограмму FUNCTION, с помощью которой можно было бы вычислить

$$\mathrm{RHO}(\mathrm{A},\ \mathrm{B},\ \mathrm{N})=\frac{A}{2\pi}\sum_{i=1}^{N}B_i,$$

где B — одномерный массив, содержащий не более 50 элементов (т. е. $N\leqslant 50$).

Затем используйте эту функцию для вычисления величины, равной произведению $1/2\pi$ на сумму первых 18 элементов некоторого массива A; назовите результат SOME.

***7.** Пусть имеется двумерный массив A, содержащий 20 строк и 20 столбцов. Напишите подпрограмму FUNCTION, с помощью которой можно было бы вычислить суммы абсолютных значений

элементов k-й строки массива A, за исключением A (K, K), т. е.

$$\text{SUMNR (A, K)} = \sum_{J \neq K} |A(K, J)|.$$

Указание: эта сумма равна также $\sum_{J=1}^{20} A(K, J) | - | A(K, K) |$.

8. Массив A содержит 20 строк и 20 столбцов. Напишите подпрограмму FUNCTION, с помощью которой можно было бы вычислить

$$\text{PD (A, I, J)} = \frac{A(I-1, J)+A(I+1, J)+A(I, J-1)+A(I, J+1)}{4}.$$

Затем используйте эту подпрограмму для вычисления

$$B_{ij} = (1-A)\, B_{ij} + A\, \frac{B_{i-1,\, j}+B_{i+1,\, j}+B_{i,\, j-1}+B_{i,\, j+1}}{4}.$$

Можно ли было бы произвести эти вычисления с помощью арифметического оператора-функции?

***9.** Одномерный массив A содержит не более 50 элементов. Напишите подпрограмму SUBROUTINE, с помощью которой можно было бы подсчитать среднее арифметическое первых N элементов массива и также количество тех элементов, которые оказались равными нулю. Назовите эту подпрограмму

AVERNZ (A, N, AVER, NZ)

Затем используйте эту подпрограмму для нахождения среднего из первых 20 элементов массива ZETA, причем среднее значение назовите ZMEAN, а количество нулевых элементов назовите NZCNT.

10. Напишите подпрограмму SUBROUTINE, с помощью которой можно было бы, имея значения переменных A, B, X и L, вычислить

$$R = \sqrt{A+BX+X^L},$$

$$S = \cos(2\pi X + A)\cdot e^{BX},$$

$$T = \left(\frac{A+BX}{2}\right)^{L+1} - \left(\frac{A-BX}{3}\right)^{L+1}.$$

***11.** Напишите пять арифметических операторов-функций, все с наименованием DEQF (X, Y), с помощью которых можно было бы вычислить

а. $y - 2e^{-x}$,

б. $2x + (x^2 - y)\operatorname{tg} x$,

в. $y^3 - \frac{y}{x}$,

г. $\cos x - \sin x - y$,

д. $\frac{2y}{x} + x^2 e^x$.

12. Напишите пять арифметических операторов-функций, все с наименованием SOLNF (X), с помощью которых можно было бы вычислить

а. $e^{-x}+e^{x}$,

б. $x^2+\cos x$,

в. $\frac{1}{2x^2}+2x$,

г. $\cos x+e^{-x}$,

д. $x^2\cdot(e^{x}-e^{1})$.

***13.** Напишите подпрограмму FUNCTION, с помощью которой можно было бы вычислить

$$\text{YNEXT (X, Y, H)} = Y+\frac{H}{6}(k_1+2k_2+2k_3+k_4),$$

где

$$k_1 = \text{DEQF (X, Y)},$$

$$k_2 = \text{DEQF}\left(X+\frac{H}{2},\ Y+\frac{Hk_1}{2}\right),$$

$$k_3 = \text{DEQF}\left(X+\frac{H}{2},\ Y+\frac{Hk_2}{2}\right),$$

$$k_4 = \text{DEQF}\,(X+H,\ Y+Hk_3).$$

В качестве арифметического оператора-функции возьмите любую из функций упражнения 11.

14. Напишите подпрограмму FUNCTION, с помощью которой можно было бы вычислить

$$\text{SQUARE (X, L)} = \sin x+\frac{1}{3}\sin 3x+\ldots+\frac{1}{L}\sin Lx.$$

При $L = 29$ вычислите и напечатайте SQUARE для всех значений X от нуля до 7 с шагом 0.2.

Это разложение прямоугольной волны с амплитудой $\pi/4$ и периодом 2π в ряд Фурье.

15. Напишите подпрограмму FUNCTION, с помощью которой можно было бы вычислять интеграл некоторой функции в пределах от A до B с разбиением на N интервалов. Аргументами подпрограммы должны быть величины A, B и N. Подинтегральная функция должна быть написана в виде арифметического оператора-функции и перфокарта с этим арифметическим оператором должна вставляться в подпрограмму FUNCTION перед транслированием подпрограммы.

16. Разработайте подпрограмму FUNCTION для нахождения корня уравнения четвертой степени. В качестве аргументов подпрограмма должна иметь коэффициенты уравнения, начальное приближение, допуск по точности решения, название корня и максимально допустимое количество итераций.

17. Напишите программу для нахождения коэффициентов кривой методом наименьших квадратов (рис. 8.15) в виде подпрограммы SUBROUTINE. Разработайте требования к основной программе, где можно было бы использовать эту подпрограмму.

ГЛАВА

10 ОБЫКНОВЕННЫЕ ДИФФЕРЕНЦИАЛЬНЫЕ УРАВНЕНИЯ

10.1. ВВЕДЕНИЕ

Уравнения, содержащие производную функции одной переменной, возникают во многих областях прикладной математики. Вообще говоря, любая физическая ситуация, где рассматривается степень изменения одной переменной по отношению с другой переменной, описывается дифференциальным уравнением, а такие ситуации, очевидно, встречаются весьма часто.

Рассмотрим простой пример. Пусть степень изменения y по отношению к x пропорциональна y:

$$\frac{dy}{dx} = y. \tag{10.1}$$

(Для обозначения дифференцирования мы будем часто употреблять штрих, это уравнение можно записать в виде: $y' = y$.) Решение этого уравнения хорошо известно:

$$y = ae^x,$$

где a — произвольная постоянная. При различных значениях постоянной a получается семейство кривых, которые все удовлетворяют уравнению (10.1). Собственно говоря, уравнение (10.1) является просто утверждением, что в каждой точке кривой значение самой функции равно значению производной. Если в дополнение к дифференциальному уравнению задать значение y для некоторого значения x, то можно определить постоянную a. Например, предположим, что решение уравнения (10.1) должно проходить через точку $x = 0$, $y = 1$, что обычно записывается в следующем виде:

$$y(0) = 1. \tag{10.2}$$

При этом легко найти, что постоянная a равна 1 и что из всего семейства кривых только одна удовлетворяет одновременно (10.1) и (10.2):

$$y = e^x.$$

Существует множество приемов для нахождения решений дифференциальных уравнений через элементарные или через специальные функции, например функции Бесселя. Мы вовсе не хотим сколько-нибудь умалить значение и важность классических методов решения дифференциальных уравнений, но тем не менее следует ясно представлять себе, что очень часто в практических задачах такие методы или вообще неприменимы, или приводят к таким сложным решениям, что затраты труда на их получение или на соответствующие расчеты превосходят все допустимые пределы. Например, внешне простое уравнение

$$y' = x^2 + y^2$$

не имеет элементарного решения. В большом количестве практических задач коэффициенты или функции в дифференциальном уравнении могут содержать существенные нелинейности или даже задаваться в виде таблиц экспериментальных данных (последний случай сразу снимает вопрос о возможности решения классическими методами).

Таким образом, по многим причинам мы вынуждены обратиться к методам решения, которые могут пригодиться тогда, когда классические методы не помогают. (Следует еще раз подчеркнуть, что факт существования задач, которые не могут быть разрешены классическими методами, не говорит о том, что на эти методы можно больше не обращать внимания. Те численные методы, которые мы рассмотрим в этой главе, не снимают с математика ответственности за правильное составление дифференциального уравнения; они не освобождают его от попыток решить это уравнение, например, через специальные функции, многие из которых широко используются; численные методы не имеют также отношения к тому факту, что опытный математик зачастую может изменить постановку задачи и тем самым облегчить ее решение. Короче говоря, существование численных методов не является оправданием для плохой математической постановки задачи.)

В этой главе мы будем в основном рассматривать методы решения одного обыкновенного дифференциального уравнения первого порядка (частных производных в этом уравнении не должно быть) с одним начальным условием:

$$y' = f(x, y), \tag{10.3}$$

$$y(x_0) = y_0. \tag{10.4}$$

Методы, которые мы здесь рассмотрим, легко обобщаются для системы уравнений первого порядка. (См., например, практический пример в разд. 10.8.) Более того, уравнения высших порядков можно свести к системе уравнений первого порядка. Например, уравнение второго порядка

$$y'' = g(y', y, x)$$

можно переписать в следующем виде:

$$z' = g(z, y, x),$$

$$y' = z,$$

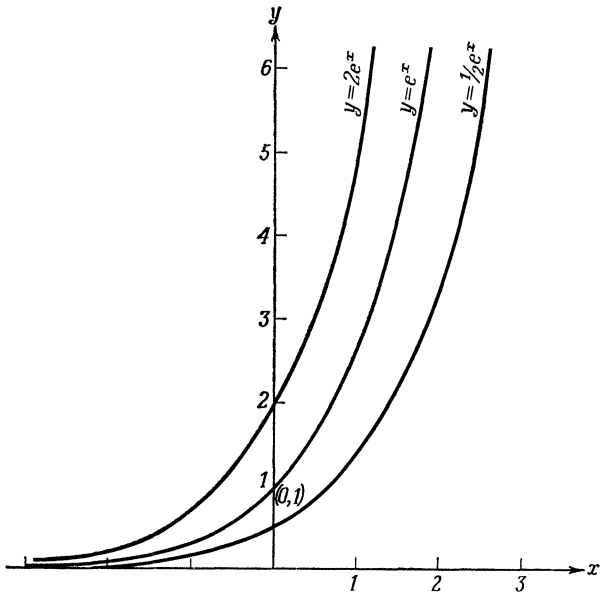
где z — новая зависимая переменная, определяемая вторым уравнением. Теперь получается система уравнений относительно y и z. Решение этой системы даст и функцию, и ее производную. Таким образом, методы этой главы имеют гораздо более широкое приложение, нежели можно предположить, исходя из простоты исходного уравнения.

Установим теперь, что имеется в виду под решением уравнения (10.3) с начальным условием (10.4) и что имеется в виду под его численным решением. Уравнение (10.3) можно рассматривать как определение кривой через ее производную в координатной плоскости xy. Мы знаем, как вычислить производную в каждой точке этой кривой через координаты этой точки x и y. В общем случае уравнению (10.3) удовлетворяет целое семейство кривых; начальное условие (10.4) позволяет выбрать из этого семейства одну определенную кривую, которая проходит через заданную точку. Например, уравнение (10.1) приводит к семейству кривых, некоторые из этих кривых показаны на рис. 10.1; начальное условие (10.2) позволяет выбрать одну из этих кривых (она показана на рисунке).

Решение найдено в виде функции $y = f(x)$. Чтобы найти численные значения функции, необходимо просто

подставить соответствующие значения x и вычислить y.

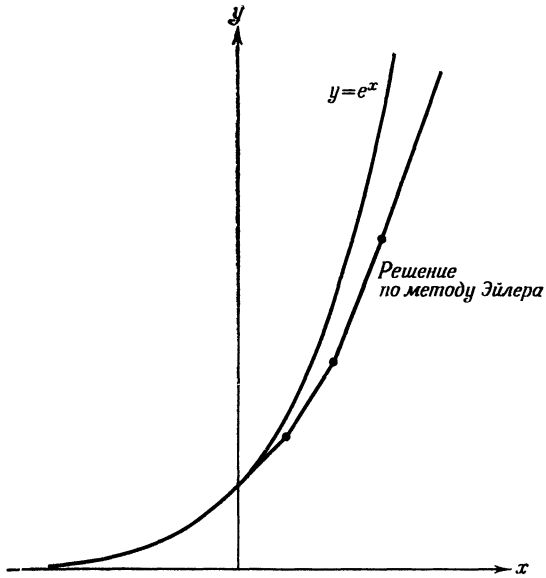
Грубо говоря, численное решение дифференциального уравнения находится следующим образом. Дифференциальное уравнение задает наклон кривой в любой точке как функцию от x и от y. В начальный момент мы знаем только одну точку, через которую проходит кривая, а именно x_0, y_0. Поэтому мы начинаем с этой точки, вычисляем нак-



Р и с. 10.1. Три кривых из семейства кривых, описываемых дифференциальным уравнением $y' = y$.
Начальное условие $y(0) = 1$ определяет конкретную кривую из этого семейства.

лон кривой при $x = x_0$ и $y = y_0$ и продвигаемся на некоторое малое расстояние вдоль получившейся касательной. Если шаг по x обозначить через h, то мы приходим в точку $x_1 = x_0 + h$ и получаем новое значение $y = y_1$. Продолжая эту процедуру дальше, мы получаем последовательность коротких отрезков прямой, которые, как мы надеемся, являются достаточно хорошим приближением к искомой функции.

Естественно, в этом простейшем методе численного решения дифференциальных уравнений имеется множество недостатков. (Этот метод известен, как метод Эйлера.) Мы пытаемся приблизить *кривую* отрезками *прямой*, что с



Р и с. 10.2. Графическое представление численного метода решения дифференциального уравнения.
Метод решения, описанный в тексте, известен под названием м етода Эйлера.

самого начала должно приводить к затруднениям. Достаточно часто будет возникать такая ситуация, когда последовательность отрезков прямых существенно отклонится от искомой кривой, как это показано на рис. 10.2. Эта проблема называется проблемой *устойчивости* метода и мы будем уделять ей большое внимание.

Совершенно ясно, что нужно каким-то образом учесть кривизну искомого решения, а не просто искать приближение в виде отрезков прямой, как это делается в методе Эйлера. В основном существует два широких класса методов,

1. *Одноступенчатые* методы, в которых используется только информация о самой кривой в одной точке и не производятся итерации. Одним из таких методов является решение уравнений с помощью рядов Тейлора, мы этот метод подробно рассматриваем, но как можно показать, он обычно довольно неудобен для практического использования. Практически удобные методы этого класса (а их существует множество) включают в себя методы Рунге—Кутта. Мы увидим, что эти методы являются прямыми (без итераций), что, казалось бы, должно привести к экономии машинного времени, но в действительности эти методы требуют многократных повторных вычислений функции. Кроме того, эти методы имеют тот недостаток, что при использовании их трудно оценивать допускаемую ошибку.

2. *Многоступенчатые* методы, в которых следующую точку кривой можно найти, не производя так много повторных вычислений функции, как при использовании одноступенчатых методов, где для достижения достаточной точности требуются итерации. Большинство методов этого класса называются методами прогноза и коррекции. Хотя и имеются некоторые трудности, связанные с использованием итерационной процедуры и с получением нескольких начальных точек решения, но они уравновешиваются тем фактом, что оценку ошибки при использовании этого метода легко получить в качестве побочного продукта вычислений.

Как и во многих других случаях, эти два класса методов придется сочетать разумным образом, учитывая их достоинства и недостатки.

К каждой из этих категорий относится множество методов, так как численное решение дифференциальных уравнений получило в последние годы исключительно сильное развитие. Те методы, которые мы будем рассматривать в этой главе, могут применяться для решения практических задач; кроме того, они могут подготовить читателя к знакомству с большинством других современных методов.

10.2. РЕШЕНИЕ С ПОМОЩЬЮ РЯДОВ ТЕЙЛОРА

Мы начинаем с метода, который теоретически пригоден для решения любых дифференциальных уравнений, но с вычислительной точки зрения не представляет почти

никакого практического интереса. Его ценность для нас заключается в том, что он даст нам некоторый эталон для сравнения различных практически удобных методов.

Напишем разложение функции $y(x)$ в ряд Тейлора в окрестности точки $x = x_m$. Другими словами, предположим, что процесс решения уже доведен до некоторой заданной точки, и нас интересует, что произойдет при переходе к следующей точке. Этим приемом мы будем часто пользоваться при рассмотрении различных методов решения.

$$y(x) = y_m + y'_m(x - x_m) + \frac{y''_m}{2}(x - x_m)^2 + \frac{y'''_m}{6}(x - x_m)^3 + \ldots, \quad (10.5)$$

где $y_m^{(j)}$ есть j-я производная функции $y(x)$ в точке $x = x_m$. Теперь предположим, что мы нашли приближенное решение уравнения для $m+1$ точек на оси x, т. е. для $x_0, x_1, x_2, \ldots, x_m$. (Вспомним, что решение в точке x_0 дается начальным условием 10.4.) Последовательные значения x_i расположены на расстоянии h друг от друга, т. е. $x_i = x_0 + ih$. Мы можем найти приближенное решение для точки x_{m+1}, подставив $x_{m+1} = x_m + h$ в формулу (10.5).

$$y_{m+1} = y_m + hy'_m + \frac{h^2}{2}y''_m + \frac{h^3}{6}y'''_m + \ldots. \quad (10.6)$$

Чем больше членов ряда мы возьмем для вычисления, тем точнее будет приближение. В любом случае необходимо вычислять различные производные функции $y(x)$. Из (10.3) имеем

$$y'_m = f(x_m, y_m). \quad (10.7)$$

Дифференцируя (10.3) по x, получаем

$$y'' = \frac{\partial}{\partial x}f(x, y) + f(x, y)\frac{\partial}{\partial y}f(x, y), \quad (10.8)$$

так что

$$y''_m = f_x + ff_y, \quad (10.9)$$

где частные производные по x и по y обозначены соответствующими индексами около буквы f:

$$f_x = \frac{\partial f}{\partial x}.$$

Предполагается, что все значения функции и ее производных вычисляются при $x = x_m$ и $y = y_m$.

При этом уравнение (10.6) приобретает вид

$$y_{m+1} = y_m + h\left(f + \frac{h}{2}(f_x + ff_y) + O(h^3), \qquad (10.10)$$

где $O(h^3)$ означает, что в следующие члены ряда h входит в степени не ниже третьей. Иначе говоря, если для нахождения приближенного решения уравнения (10.3) будет использована формула (10.10) без $O(h^3)$, то ошибка ограничения будет приблизительно равна Kh^3, где K — некоторая постоянная.

Нахождение решения с помощью ряда Тейлора является одноступенчатым методом, так как для вычисления y_{m+1} требуется информация только об одной предыдущей точке x_m, y_m.

С практической точки зрения трудность этого метода заключается в том, что может быть очень трудно (иногда даже просто невозможно) найти f_x и f_y. Кроме того, если попытаться получить лучшее приближение, т. е. меньшую ошибку ограничения, то необходимо вычислить y'''_m, которая равна

$$y'''_m = f_{xx} + 2ff_{xy} + f^2 f_{yy} + f_x f_y + ff_y^2.$$

Последующие производные становятся еще более сложными.

Таким образом, с точки зрения практических вычислений этот метод обычно неудобен. Однако если теперь мы будем сравнивать различные практически применяемые методы, то для их оценки у нас есть некоторый критерий — насколько тот или иной метод согласуется с разложением в ряд Тейлора. Некоторые методы будут согласоваться вплоть до членов порядка h, другие — вплоть до членов порядка h^4 и т. д. Этот критерий можно применить несмотря на то, что практические методы вообще не предусматривают вычисления производных от функции $f(x, y)$.

10.3. МЕТОДЫ РУНГЕ — КУТТА

Рассмотрение методов, применяемых на практике для решения дифференциальных уравнений, мы начнем с их широкой категории, известной под общим названием мето-

дов Рунге — Кутта. Как можно заранее предположить, различные методы этой категории требуют большего или меньшего объема вычислений и соответственно обеспечивают большую или меньшую точность. Для того чтобы рассмотреть сущность этих методов, рассмотрим сначала один из них, известный под названием метода Эйлера. Этот метод очень редко применяется на практике, но, как и в случае рядов Тейлора, является отправной точкой для дальнейшего изложения.

Методы Рунге — Кутта обладают следующими отличительными свойствами:

1. Эти методы являются одноступенчатыми: чтобы найти y_{m+1}, нужна информация только о предыдущей точке x_m, y_m.

2. Они согласуются с рядом Тейлора вплоть до членов порядка h^p, где степень p различна для различных методов и называется порядком метода.

3. Они не требуют вычисления производных от $f(x, y)$, а требуют только вычисления самой функции.

Именно благодаря третьему свойству методы Рунге — Кутта более удобны для практических вычислений, нежели ряд Тейлора. Однако, как и можно ожидать, для вычисления одной последующей точки решения нам придется вычислять функцию $f(x, y)$ несколько раз при различных значениях x и y Это та цена, которую приходится платить за право не вычислять никаких производных, но цена более чем умеренная.

Как обычно, рассмотрим сначала геометрическое построение и выведем некоторые формулы на основе геометрических аналогий. После этого мы подтвердим полученные результаты аналитически.

Предположим, что нам известна точка x_m, y_m на искомой кривой. Тогда мы можем провести прямую линию с тангенсом угла наклона

$$y'_m = f(x_m, y_m),$$

которая пройдет через точку x_m, y_m. Это построение показано на рис. 10.3, где кривая представляет собой точное, но, конечно, неизвестное решение уравнения, а прямая линия L_1 построена так, как это только что описано. Тогда следующей точкой решения можно считать ту, где прямая L_1

пересечет ординату, проведенную через точку $x = x_{m+1} = x_m + h$.

Уравнение прямой L_1 выглядит так:

$$y = y_m + y'_m (x - x_m),$$

но ведь

$$y'_m = f(x_m, y_m),$$

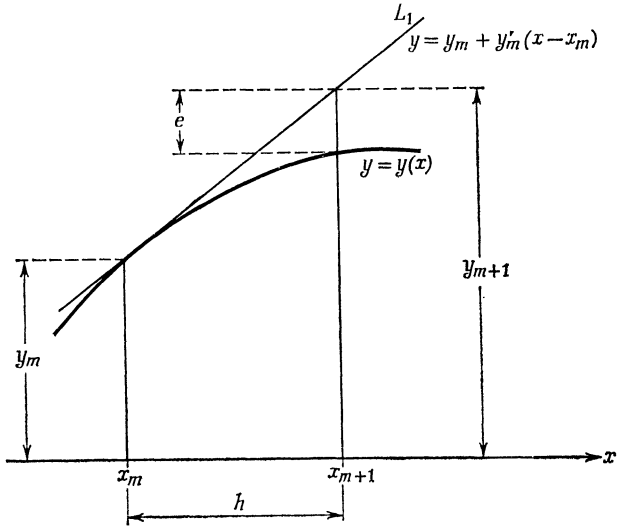
и, кроме того,

$$x_{m+1} = x_m + h,$$

так что

$$y_{m+1} = y_m + hf(x_m, y_m). \qquad (10.11)$$

Ошибка при $x = x_{m+1}$ показана в виде отрезка e. Очевидно, найденное таким образом приближенное решение



Р и с. 10.3. Геометрическое представление метода Эйлера.

согласуется с разложением в ряд Тейлора вплоть до членов порядка h, так что ошибка ограничения равна

$$e_T = Kh^2.$$

Заметим, что хотя на рис. 10.3 было показано, что точка y_m лежит точно на кривой, в действительности, конечно, y_m является приближенным значением и не лежит точно на кривой.

Формула (10.11) описывает *метод Эйлера*, один из самых старых и широко известных методов численного интегрирования дифференциальных уравнений.

Этот метод имеет довольно большую ошибку ограничения; кроме того, он очень часто оказывается неустойчивым — малая ошибка (происходящая от ограничения, округления или заложенная в исходных данных) увеличивается с ростом x. Поэтому мы больше не будем рассматривать этот метод, а перейдем к более точным. Нужно, однако, заметить, что, согласно нашему определению, метод Эйлера является методом Рунге — Кутта, а именно одним из методов Рунге — Кутта первого порядка, так как он согласуется с разложением в ряд Тейлора вплоть до членов порядка h.

Для вычисления значения y_{m+1} метод Эйлера использует наклон касательной только в точке x_m, y_m. Этот метод можно усовершенствовать множеством различных способов. Из этих способов мы здесь рассмотрим два, так называемые *исправленный метод Эйлера* и *модифицированный метод Эйлера*, а затем покажем, что оба они не что иное как два метода из семейства методов Рунге — Кутта второго порядка.

В исправленном методе Эйлера мы находим средний тангенс угла наклона касательной для двух точек: x_m, y_m и $x_m + h$, $y_m + hy'_m$. Последняя точка есть та самая, которая в методе Эйлера обозначалась x_{m+1}, y_{m+1}. Геометрически процесс нахождения точки x_{m+1}, y_{m+1} можно проследить по рис. 10.4. С помощью метода Эйлера находится точка $x_m + h$, $y_m + hy'_m$, лежащая на прямой L_1. В этой точке снова вычисляется тангенс угла наклона касательной, на рисунке этому значению соответствует прямая L_2. Усреднение двух тангенсов дает прямую $\bar{L}$. Наконец, через точку x_m, y_m мы проводим прямую L, параллельную $\bar{L}$. Точка, в которой прямая L пересечется с ординатой, восстановленной из $x = x_{m+1} = x_m + h$, и будет искомой точкой x_{m+1}, y_{m+1}.

Тангенс угла наклона прямой $\bar{L}$ и прямой L равен

$$\Phi(x_m, y_m, h) = \frac{1}{2}[f(x_m, y_m) + f(x_m + h, y_m + hy'_m)], \quad (10.12)$$

где

$$y'_m = f(x_m, y_m). \quad (10.13)$$

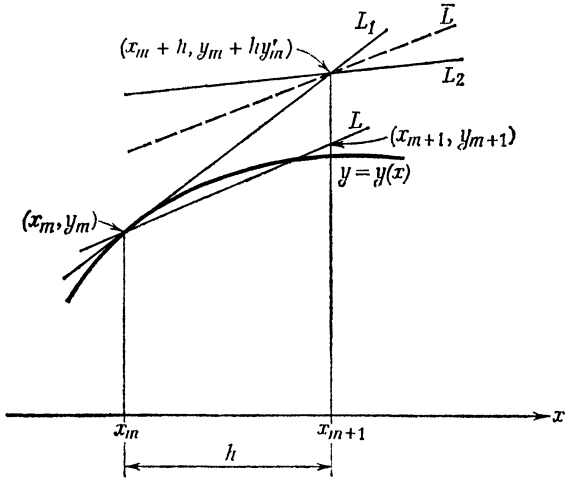
Уравнение линии L при этом записывается в виде

$$y = y_m + (x - x_m)\,\Phi(x_m, y_m, h),$$

так что

$$y_{m+1} = y_m + h\Phi(x_m, y_m, h). \qquad (10.14)$$

Соотношения (10.12), (10.13) и (10.14) описывают *исправленный метод Эйлера.*



Р и с. 10.4. Геометрическое представление исправленного метода Эйлера.

Чтобы выяснить, насколько хорошо этот метод согласуется с разложением в ряд Тейлора, вспомним, что разложение в ряд функции $f(x, y)$ можно записать следующим образом:

$$f(x, y) = f(x_m, y_m) + (x - x_m)\frac{\partial f}{\partial x} + (y - y_m)\frac{\partial f}{\partial y} + \ldots, \qquad (10.15)$$

где частные производные вычисляются при $x = x_m$ и $y = y_m$. Подставляя в формулу (10.15) $x = x_m + h$ и $y = y_m + hy'_m$ и используя выражение (10.13) для y'_m, получаем

$$f(x_m + h\ y_m + hy'_m) = f + hf_x + hff_y + O(h^2),$$

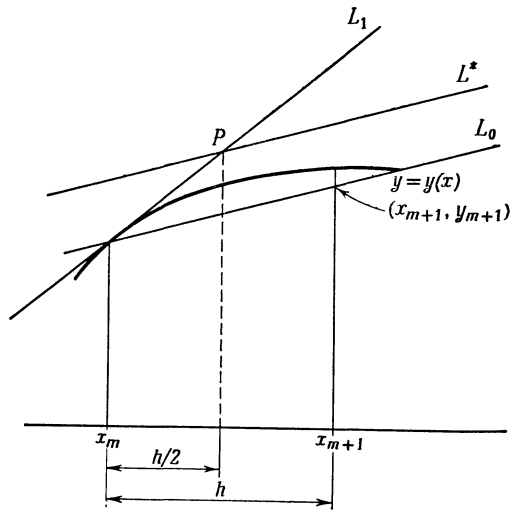
где снова функция f и ее производные вычисляются в точке x_m, y_m. Подставляя результат в (10.12) и производя необходимые преобразования, получаем

$$\Phi(x_m, y_m, h) = f + \frac{h}{2}(f_x + ff_y) + O(h^2).$$

Наконец, подставив последнее соотношение в (10.14), приведем к формуле, которую можно непосредственно сравнить с разложением в ряд Тейлора

$$y_{m+1} = y_m + hf + \frac{h^2}{2}(f_x + ff_y) + O(h^3).$$

Как видим, исправленный метод Эйлера согласуется с разложением в ряд Тейлора вплоть до членов степени h^2,



Р и с. 10.5. Геометрическое представление модифицированного метода Эйлера.

являясь, таким образом, методом Рунге — Кутта второго порядка. При использовании этого метода функцию $f(x, y)$ приходится вычислять дважды — в точке x_m, y_m и в точке $x_m + h$, $y_m + hy'_m$. В целях сравнения заметим, что для достижения той же самой степени точности метод разложения в ряд Тейлора потребовал бы вычисления значений трех различных функций: f, f_x и f_y.

В исправленном методе Эйлера усреднялись *наклоны* касательных. Можно пойти по другому пути и усреднять *точки* в следующем смысле. Рассмотрим рис. 10.5, где первоначальное построение сделано точно так же, как и на рис. 10.4 — через точку x_m, y_m проведена прямая L_1 с тангенсом угла наклона, равным $f(x_m, y_m)$. Но на этот раз мы берем точку, лежащую на пересечении этой прямой и ординаты $x = x_m + h/2$. На рисунке эта точка обозначена через P, а ее ордината равна $y = y_m + (h/2)\, y'_m$. Вычислим тангенс угла наклона касательной в этой точке

$$\Phi(x_m, y_m, h) = f\left(x_m + \frac{h}{2},\ y_m + \frac{h}{2} y'_m\right), \qquad (10.16)$$

где

$$y'_m = f(x_m, y_m). \qquad (10.17)$$

Прямая с таким наклоном, проходящая через P, обозначена через L^*. Вслед за тем, мы проводим через точку x_m, y_m прямую, параллельную L^*, и обозначаем ее через L_0. Пересечение этой прямой с ординатой $x = x_m + h$ и даст искомую точку x_{m+1}, y_{m+1}. Уравнение прямой L_0 можно записать в виде

$$y = y_m + (x - x_m)\, \Phi(x_m, y_m, h),$$

где Φ задается формулой (10.16). Поэтому

$$y_{m+1} = y_m + h\Phi(x_m, y_m, h). \qquad (10.18)$$

Соотношения (10.16), (10.17) и (10.18) описывают так называемый *модифицированный метод Эйлера* или *исправленный метод ломаных*. Мы предоставляем читателям самим доказать, что этот метод согласуется с разложением в ряд Тейлора (10.10) вплоть до членов степени h^2 и является поэтому еще одним методом Рунге — Кутта второго порядка.

Поскольку у нас теперь имеется два различных метода Рунге — Кутта второго порядка, то интересно выяснить, что эти методы имеют между собой сходного и можно ли их обобщить.

Заметим, что оба метода описываются формулами вида

$$y_{m+1} = y_m + h\Phi(x_m, y_m, h) \qquad (10.19)$$

и что в обоих случаях Φ имеет вид

$$\Phi(x_m, y_m, h) = a_1 f(x_m, y_m) + a_2 f(x_m + b_1 h, y_m + b_2 h y'_m), \tag{10.20}$$

где

$$y'_m = f(x_m, y_m). \tag{10.21}$$

В частности, для исправленного метода Эйлера

$$a_1 = a_2 = 1/2;$$
$$b_1 = b_2 = 1,$$

в то время как для модифицированного метода Эйлера

$$a_1 = 0; \quad a_2 = 1,$$
$$b_1 = b_2 = 1/2.$$

Формулы (10.19), (10.20) и (10.21) описывают некоторый метод типа Рунге — Кутта. Посмотрим, какого порядка метод можно рассчитывать получить в лучшем случае и каковы допустимые значения параметров a_1, a_2, b_1 и b_2.

Чтобы получить соответствие ряду Тейлора вплоть до членов степени h, в общем случае достаточно одного параметра. Чтобы получить согласование вплоть до членов степени h^2, потребуются еще два параметра, так как необходимо учитывать члены $h^2 f_x$ и $h^2 f f_y$. Так как у нас имеется всего четыре параметра, три из которых потребуются для согласования с рядом Тейлора вплоть до членов порядка h^2, то самое лучшее, на что здесь можно рассчитывать,— это метод второго порядка. С другой стороны, так как имеется четыре параметра, а условий, наложенных на эти параметры, всего три, то можно ожидать, что удастся построить множество методов второго порядка, варьируя один свободный параметр. Так это и есть на самом деле.

В разложении $f(x, y)$ в ряд (10.15) в окрестности точки x_m, y_m положим

$$x = x_m + b_1 h,$$
$$y = y_m + b_2 h f.$$

Тогда

$$f(x_m + b_1 h, y_m + b_2 h f) = f + b_1 h f_x + b_2 h f f_y + O(h^2),$$

где функция и производные в правой части равенства вычислены в точке x_m, y_m. Тогда (10.19) можно переписать в виде

$$y_{m+1} = y_m + h\,[a_1 f + a_2 f + h\,(a_2 b_1 f_x + a_2 b_2 f f_y)] + O(h^3).$$

Сравним эту формулу с разложением в ряд Тейлора (10.10). Если потребовать совпадения членов степени hf, то

$$a_1 + a_2 = 1.$$

Сравнивая члены, содержащие $h^2 f_x$, получаем

$$a_2 b_1 = \frac{1}{2}.$$

Сравнивая члены, содержащие $h^2 f f_y$, получаем

$$a_2 b_2 = \frac{1}{2}.$$

Так как мы пришли к трем уравнениям для определения четырех неизвестных, то одно из этих неизвестных можно задать произвольно, исключая, может быть, нуль, в зависимости от того, какой параметр взят в качестве произвольного. Положим, например,

$$a_2 = \omega \neq 0.$$

Тогда

$$a_1 = 1 - \omega,$$

$$b_1 = b_2 = \frac{1}{2\omega},$$

и соотношения (10.19), (10.20) и (10.21) сведутся к

$$y_{m+1} = y_m + h\left[(1-\omega)\,f(x_m, y_m) + \right.$$
$$\left. + \omega f\left(x_m + \frac{h}{2\omega},\ y_m + \frac{h}{2\omega} f(x_m, y_m)\right)\right] + O(h^3). \quad (10.22)$$

Это наиболее общая форма записи метода Рунге — Кутта второго порядка. При $\omega = 1/2$ мы получаем исправленный метод Эйлера, при $\omega = 1$ получаем модифицированный метод Эйлера. Для всех ω, отличных от нуля, ошибка ограничения равна

$$e_T = Kh^3. \quad (10.23)$$

Можно оценить верхний предел $|K|$. Согласно Ральстону[1]), наименьший верхний предел получается при $\omega = 2/3$.

Перед изучением методов Рунге — Кутта высших порядков рассмотрим снова пример (10.1)

$$y' = y$$

с начальным условием

$$y(0) = 1.$$

Как уже говорилось ранее, точное решение имеет вид

$$y = e^x.$$

Метод Рунге — Кутта второго порядка (10.22) для этого уравнения запишется в следующем виде:

$$y_{m+1} = y_m + h\left[(1-\omega)\, y_m + \omega\left(y_m + \frac{h}{2\omega}\, y_m\right)\right]$$

и для любого ω, отличного от нуля, сведется к

$$y_{m+1} = y_m\left(1 + h + \frac{h^2}{2}\right).$$

Нетрудно получить

$$y_{m+1} = \left(1 + h + \frac{h^2}{2}\right)^{m+1}.$$

Сумма, стоящая в скобках, представляет собой не что иное, как три первых члена разложения функции e^h в ряд Тейлора, и поэтому

$$y_{m+1} \approx e^{h(m+1)} = e^{x_{m+1}},$$

как и можно было ожидать.

Если принять $h = 0{,}1$, то получаются значения y, которые приведены в табл. 10.1. В той же таблице указаны значения ошибки $e^{x_i} - y_i$. Можно убедиться, что ошибка возрастает с ростом x. К тому времени, как x достигает значения 2 ($i = 20$), ошибка возрастет до 0.021. Заметим, что ошибка всегда положительна, так как

$$1 + h + \frac{h^2}{2} < e^h \quad \text{при} \quad h > 0.$$

---

[1]) Ralston A., Runge — Kutta methods with minimum error bounds, *Mathematics of Computation*, **16**, 431—437 (1962).

Таблица 10.1

i	x_i	y_i	Ошибка
1	0.1	1.105	0.000
2	0.2	1.221	0.000
3	0.3	1.349	0.001
4	0.4	1.491	0.001
5	0.5	1.648	0.001
6	0.6	1.821	0.001
7	0.7	2.012	0.002
8	0.8	2.223	0.003
9	0.9	2.456	0.004
10	1.0	2.714	0.004

Методы Рунге — Кутта третьего и четвертого порядков можно вывести совершенно аналогично тому, как это делалось при выводе методов первого и второго порядков[1]). Мы не будем воспроизводить здесь эти выкладки, а ограничимся тем, что приведем формулы, описывающие метод четвертого порядка, один из самых употребительных методов интегрирования дифференциальных уравнений. Этот метод применяется настолько широко, что в литературе по вычислениям на ЭЦВМ этот метод просто называется «методом Рунге — Кутта» без всяких указаний на тип или порядок. Этот классический метод Рунге — Кутта описывается системой следующих пяти соотношений:

$$y_{m+1} = y_m + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4), \tag{10.24}$$

где

$$k_1 = f(x_m, y_m), \tag{10.25}$$

$$k_2 = f\left(x_m + \frac{h}{2}, y_m + \frac{hk_1}{2}\right), \tag{10.26}$$

$$k_3 = f\left(x_m + \frac{h}{2}, y_m + \frac{hk_2}{2}\right), \tag{10.27}$$

$$k_4 = f(x_m + h, y_m + hk_3). \tag{10.28}$$

[1]) См. предыдущую сноску.

Ошибка ограничения для этого метода равна

$$e_T = Kh^5,$$

так что формулы (10.24) — (10.28) описывают метод четвертого порядка. В статье Ральстона, на которую мы уже ссылались, оценивается верхний предел K и выводятся формулы для метода четвертого порядка, у которого этот предел минимален.

Заметим, что при использовании этого метода функцию необходимо вычислять *четыре* раза.

Интересно сравнить этот метод, с помощью которого интегрируется функция от x и y, с методами из гл. 6, где интегрировались функции только от x. Для этого предположим, что f является функцией только от x, т. е.

$$f(x, y) = F(x)$$

и

$$y(x) = \int_{x_0}^{x} F(x)\,dx. \tag{10.29}$$

Пусть

$$p = \frac{h}{2},$$

и обозначим

$$F_j = F(x_0 + jp).$$

Но ведь

$$y_m = y(x_0 + mh) = y(x_0 + 2mp).$$

Обозначим

$$Y_j = y(x_0 + jp),$$

так что

$$y_m = Y_{2m}.$$

Тогда (10.24) запишется так:

$$Y_{2m+2} - Y_{2m} = \frac{p}{3}(F_{2m} + 4F_{2m+1} + F_{2m+2}),$$

где m может принимать значения 0, 1, 2, ... $n-1$. При этом

$$Y_2 - Y_0 = \frac{p}{3}(F_0 + 4F_1 + F_2),$$

$$Y_4 - Y_2 = \frac{p}{3}(F_2 + 4F_3 + F_4),$$

. . . . . .

$$Y_{2n} - Y_{2n-2} = \frac{p}{3}(F_{2n-2} + 4F_{2n-1} + F_{2n}).$$

Складывая все эти равенства, получаем

$$Y_{2n} - Y_0 = \frac{p}{3}(F_0 + 4F_1 + 2F_2 + 4F_3 + \ldots + 4F_{2n-1} + F_{2n}),$$

т. е. формулу Симпсона для вычисления интеграла (10.29), где $x = x_0 + 2nh$.

Таким образом, классическая формула Рунге — Кутта оказывается обобщением формулы Симпсона, причем обобщение состоит в том, что формула не ограничена теперь функциями только от x. По этой причине такую формулу часто называют формулой Рунге — Кутта — Симпсона.

Рассмотрим снова наш простой пример

$$y' = y, \quad y(0) = 1.$$

Для этого уравнения формулы (10.25)—(10.28) приведут к следующим соотношениям:

$$k_1 = y_m,$$

$$k_2 = y_m + \frac{h}{2} y_m,$$

$$k_3 = y_m + \frac{h}{2}\left(y_m + \frac{h}{2} y_m\right),$$

$$k_4 = y_m + h\left[y_m + \frac{h}{2}\left(y_m + \frac{h}{2} y_m\right)\right].$$

Подставляя в (10.24), получаем

$$y_{m+1} = \left(1 + h + \frac{h^2}{2} + \frac{h^3}{6} + \frac{h^4}{24}\right) y_m.$$

Сумма в скобках представляет собой теперь *пять* первых членов разложения функции e^h в ряд Тейлора.

10.4. АНАЛИЗ ОШИБОК, ВОЗНИКАЮЩИХ ПРИ ИСПОЛЬЗОВАНИИ МЕТОДОВ РУНГЕ — КУТТА

Мы уже отмечали, что ошибка ограничения при использовании метода Рунге—Кутта p-го порядка равна Kh^{p+1}, где K — некоторая постоянная. Верхние пределы для K при $p = 2$, 3 и 4 приведены в статье Ральстона. Вывод формул для этих верхних пределов является вовсе не простой задачей; более того, этот вывод требует параметров, которые отсутствуют в формулах (10.24) — (10.28). Один из серьезных недостатков методов Рунге — Кутта состоит в отсутствии простых способов оценить их ошибку. С другой стороны, те методы, которые мы рассмотрим в следующем параграфе, в большой мере привлекательны именно потому, что оценка ошибки получается просто в качестве побочного результата вычислений.

Все же без некоторой оценки ошибки ограничения трудно правильно выбрать величину шага интегрирования h. Грубое оценочное правило предложено Коллатцом[1]). Если

$$\frac{|k_2 - k_3|}{|k_1 - k_2|}$$

становится велико (больше нескольких сотых), то шаг интегрирования необходимо уменьшить[2]).

Мерсон показал, что оценку ошибки ограничения можно получить за счет того, что придется лишний раз вычислять функцию $f(x, y)$. Это может оказаться нецелесообразным, если функция $f(x, y)$ достаточно сложна. Заинтересованные читатели могут найти необходимые сведения в книге Фокса[3]).

Более точную оценку ошибки ограничения можно получить, используя экстраполяционный переход к пределу (см. гл. 6).

---

[1]) Collatz L., Numerische Behandlung von Differentialgleichungen, Springer Verlag, Berlin, 1951, S. 34. (Русский перевод: Коллатц Л., Численные методы решения дифференциальных уравнений, ИЛ, М., 1953.)

[2]) Ralston and Wilf, Mathematical methods for digital computers, Wiley, 1960, ch. 9.

[3]) Fox L., Numerical solution of ordinary and partial differential equations, Pergamon, New York, 1962.

Чтобы получить такую оценку, допустим что Y_m есть точное решение при $x = x_0 + mh$. Тогда для классического метода четвертого порядка

$$Y_m = y_m^{(h)} + Kh^5, \tag{10.30}$$

где верхний индекс (h) при y_m означает, что это приближение было вычислено с шагом h. Затем вычислим снова это решение, но на этот раз с шагом $h/2$, так что

$$Y_m = y_m^{(h/2)} + K\left(\frac{h}{2}\right)^5. \tag{10.31}$$

Вычитая (10.30) из (10.31), получаем

$$y_m^{(h)} - y_m^{(h/2)} = -\frac{15}{16}Kh^5,$$

и ошибка ограничения в формуле (10.29) равна

$$E_T = Kh^5 = \frac{16}{15}[y_m^{(h/2)} - y_m^{(h)}]. \tag{10.32}$$

Предполагая, что пятая производная от y практически постоянна, мы получаем довольно точную оценку ошибки. Очевидно, препятствием к применению этого метода является то, что решение необходимо вычислять *дважды*, а результат обычно не стоит затраченных усилий.

Аналогичный результат можно получить для метода Рунге — Кутта второго порядка, где ошибка ограничения равна Kh^3.

Даже если ошибка ограничения мала, все равно метод Рунге — Кутта может при неблагоприятных условиях дать совершенно ошибочные результаты. Такие ошибочные результаты могут, например, появиться потому, что малые ошибки — ошибки ограничения или округления — могут возрасти при вычислении решения для больших x.

Например, рассмотрим следующее уравнение:

$$y' = -10y$$

с начальным условием

$$y(0) = 1.$$

Точное решение этого уравнения равно

$$y = e^{-10x}.$$

Рассуждая так же, как и в разд. 10.3 перед вычислением табл. 10.1, придем к приближенной формуле

$$y_{m+1} = (1 - 10h + 50h^2)^m.$$

Все методы Рунге — Кутта второго порядка дадут именно эту формулу.

Но теперь заметим, что сумма, стоящая в скобках, при $h > 0.2$ становится больше 1. Следовательно, при возрастании m y неограниченно возрастает. С другой стороны, точное решение убывает, стремясь к нулю.

Это явление называется частичной неустойчивостью по Майерсу[1]). От прочих типов неустойчивости ее отличает то свойство, что она зависит от h. Те неустойчивости, которые мы будем рассматривать в разд. 10.6, не зависят от h.

Следует предупредить читателей, что подобная частичная неустойчивость может возникать при использовании методов Рунге — Кутта даже и в тех случаях, когда точные решения и не уменьшаются экспоненциально, как это имело место в только что рассмотренном примере.

10.5. МЕТОДЫ ПРОГНОЗА И КОРРЕКЦИИ

Отличительной чертой методов Рунге — Кутта является то, что при вычислении следующей точки x_{m+1}, y_{m+1} используется информация только о точке x_m, y_m, но не о предыдущих. В методах второго порядка и выше приходится вычислять значение функции в одной или в нескольких промежуточных точках. Это представляется нерациональным, поскольку, если процесс интегрирования уже продвинулся на несколько шагов, то к нашим услугам имеется дополнительная информация, для использования которой вообще не нужно вычислять никаких функций — информация о предыдущих точках решения. Поскольку в методах Рунге — Кутта информация о предыдущих точках решения не используется, а также поскольку для методов Рунге — Кутта отсутствуют достаточно простые способы оценки ошибки, то целесообразно было бы рассмотреть некоторые дополнительные методы решения дифференциальных уравнений. Мы, однако, увидим, что эти методы отличаются тем свойством, что с их помощью *нельзя начать* решение урав-

[1]) См. гл. 4 книги Фокса, указанной в предыдущей сноске.

нения, так как в них *не только возможно, но и необходимо* использовать информацию о предыдущих точках решения. Чтобы начать решение уравнения, имея только одну точку, определяемую начальным условием, или для того, чтобы изменить шаг h, необходим метод типа Рунге — Кутта. Поэтому, как мы уже говорили, приходится использовать разумное сочетание двух методов решения.

Те методы, которые мы собираемся сейчас рассматривать, известны под общим названием методов прогноза и коррекции. Как ясно из названия, вначале «предсказывается» значение y_{m+1}, а затем используется тот или иной метод для «корректировки» этого значения. Естественно, после этого можно использовать ту же самую формулу для вторичной корректировки однажды уже скорректированного значения y_{m+1}. Этот итерационный процесс можно повторять сколько угодно раз, но в дальнейшем мы увидим, что из соображений эффективности целесообразно уменьшать число итераций, выбирая должным образом шаг интегрирования.

Среди множества возможных формул прогноза и коррекции мы выберем по одному примеру, применимому ко многим практическим задачам. После усвоения основных идей, лежащих в основе методов прогноза и коррекции, у читателя не возникнет никаких затруднений с применением множества других формул, встречающихся в специальной литературе.

Для прогноза мы используем формулу второго порядка

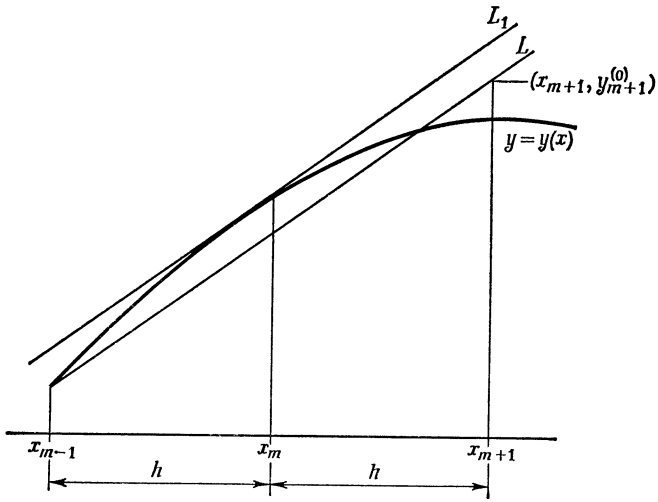
$$y_{m+1}^{(0)} = y_{m-1} + 2hf(x_m, y_m), \qquad (10.33)$$

где верхний индекс (0) означает исходное приближение к y_{m+1}, т. е. предсказанное значение. Непосредственно из написания формулы следует, что с ее помощью нельзя вычислить y_1. Это и понятно, так как для вычисления y_1 потребовалась бы точка, расположенная *перед* начальной точкой y_0. Чтобы начать решение с помощью метода прогноза и коррекции, часто используется метод Рунге — Кутта. К этому вопросу мы еще вернемся впоследствии.

Можно подумать, что если для предсказания точки x_{m+1}, y_{m+1} использовать метод Эйлера, то информация о точке x_{m-1}, y_{m-1} не потребуется. Это оказывается нецелесообразным, потому что при использовании метода Эйлера

слишком велика ошибка ограничения. Именно использование информации о предыдущих точках без вычисления значений функции позволяет классифицировать методы прогноза и коррекции как *многоступенчатые* методы решения дифференциальных уравнений.

Геометрически предсказание сводится к тому, что находится угол наклона касательной в точке x_m, y_m (прямая

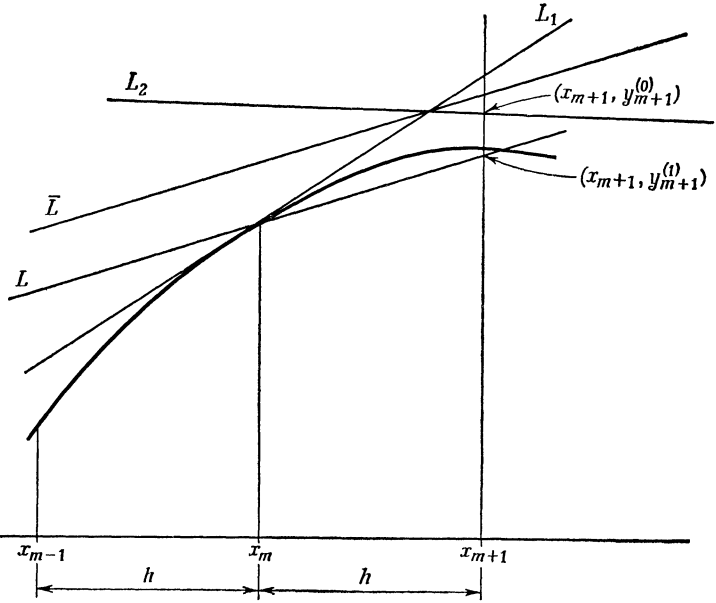


Р и с. 10.6. Геометрическое представление прогнозирующей формулы второго порядка, описанной в тексте.

L_1 на рис. 10.6). После этого через точку x_{m-1}, y_{m-1} проводится прямая L, параллельная L_1. Предсказанное значение y^0_{m+1} будет расположено там, где прямая L пересечется с ординатой $x = x_{m+1}$.

Теперь требуется некоторый метод коррекции предсказанного значения. Так как нам приближенно известна величина y_{m+1}, то можно вычислить наклон касательной в точке x_{m+1}, $y^{(0)}_{m+1}$. Эта касательная изображена на рис. 10.7 и обозначена L_2. Прямая L_1 на рис. 10.7 представляет собой то же самое, что и на рис. 10.6, и тангенс угла наклона ее равен $f(x_m, y_m)$. Мы усредняем тангенсы углов наклона линий L_1 и L_2 и получаем линию $\bar{L}$. Наконец, мы

проводим через точку x_m, y_m линию L, параллельную $\bar{L}$ и точка пересечения этой линии с ординатой $x = x_{m+1}$ дает новое приближение к y_{m+1}. Назовем это приближение



Р и с. 10.7. Геометрическое представление формулы коррекции второго порядка, описанной в тексте.

скорректированным значением $y^{(1)}_{m+1}$. Вычислить это скорректированное значение можно по формуле

$$y^{(1)}_{m+1} = y_m + \frac{h}{2}[f(x_m, y_m) + f(x_{m+1}, y^{(0)}_{m+1})].$$

Можно попытаться найти новое, по-видимому, еще лучшее приближение к y_{m+1}, используя значение $y^{(1)}_{m+1}$ и корректируя снова. В общем случае, i-е приближение к y_{m+1} вычисляется по формуле

$$y^{(i)}_{m+1} = y_m + \frac{h}{2}[f(x_m, y_m) + f(x_{m+1}, y^{(i-1)}_{m+1})] \quad (10.34)$$

для $i=1, 2, 3, \ldots$. Итерационный процесс прекращается, когда

$$|y_{m+1}^{(i+1)}-y_{m+1}^{(i)}|<\varepsilon \tag{10.35}$$

для некоторого положительного ε.

Внимательный читатель может заметить сходство между описанным здесь методом коррекции и исправленным методом Эйлера (см. формулы от (10.12) до (10.14)). Основная разница заключается в том, что в роли прогноза был использован метод Эйлера и отсутствовала итерационная процедура, здесь же используется более точная формула (10.38).

Естественно, возникает вопрос, удастся ли вообще когда-нибудь удовлетворить условию (10.35), иными словами, сходится ли процесс вообще?

Для ответа на этот вопрос заметим вначале, что имеет место равенство

$$y_{m+1}^{(i+1)}-y_{m+1}^{(i)}=\frac{h}{2}[f(x_{m+1}, y_{m+1}^{(i)})-f(x_{m+1}, y_{m+1}^{(i-1)})].$$

Используя теорему о среднем значении, получаем

$$y_{m+1}^{(i+1)}-y_{m+1}^{(i)}=\frac{h}{2}\left(\frac{\partial f}{\partial y}\right)[y_{m+1}^{(i)}-y_{m-1}^{(i-1)}], \tag{10.36}$$

где $\partial f/\partial y$ вычислено при $x=x_{m+1}$ и при некотором y, лежащем между $y_{m+1}^{(i+1)}$ и $y_{m+1}^{(i)}$.

Предположим теперь, что $\partial f/\partial y$ ограничена, т. е. можно найти такое M, что

$$\left|\frac{\partial f}{\partial y}\right|\leqslant M.$$

Тогда из (10.36) следует, что

$$|y_{m+1}^{(i+1)}-y_{m+1}^{(i)}|\leqslant\frac{hM}{2}|y_{m+1}^{(i)}-y_{m+1}^{(i-1)}|.$$

Аналогично

$$|y_{m+1}^{(i)}-y_{m+1}^{i-1}|\leqslant\frac{hM}{2}|y_{m+1}^{(i-1)}-y_{m+1}^{(i-2)}|.$$

Подставляя в предыдущее равенство, получаем

$$|y_{m+1}^{(i+1)}-y_{m+1}^{(i)}|\leqslant\left(\frac{hM}{2}\right)^2|y_{m+1}^{(i-1)}-y_{m+1}^{(i-2)}|.$$

Продолжая и дальше таким же образом, приходим к следующему результату:

$$|y_{m+1}^{(i+1)} - y_{m+1}^{(i)}| \leqslant \left(\frac{hM}{2}\right)^i |y_{m+1}^{(1)} - y_{m+1}^{(0)}|.$$

Таким образом, если выбрать величину шага h из условия

$$h < \frac{2}{M}, \tag{10.37}$$

то разница между последовательными скорректированными значениями стремится к нулю и процесс сходится.

Нужно четко представлять себе, что же именно было доказано. Мы доказали, что последовательные значения $y_{m+1}^{(i)}$ сходятся к *некоторому определенному* значению, но вовсе не обязательно к *точному решению* уравнения. Разница между тем и другим представляет собой ошибку ограничения, рассматриваемую в разд. 10.6.

В разд. 10.6 мы также рассмотрим вопрос о скорости сходимости итерационного процесса. Очевидно, что чем меньше h, тем скорее процесс будет сходиться.

Рассмотрим более внимательно формулу коррекции (10.34). Предположим, что f является функцией только от x, т. е.

$$f(x, y) = F(x),$$

так что

$$y(x) = \int_{x_0}^{x} F(x)\,dx. \tag{10.38}$$

Тогда

$$y_{m+1} = y_m + \frac{h}{2}[F(x_m) + F(x_{m+1})],$$

где мы опустили верхние индексы при y_{m+1}, так как первое же значение y_{m+1} будет точным (если пренебречь ошибками ограничения и округления).

Из этой формулы легко получить

$$y_n - y_0 = \frac{h}{2}[F(x_0) + 2F(x_1) + \ldots + 2F(x_{n+1}) + F(x_n)].$$

Это просто формула трапеций для вычисления интеграла (10.38), где $x = x_0 + nh$. Таким образом, рассмотренная нами формула коррекции является обобщением формулы трапеций, так же как классический метод Рунге — Кутта оказался обобщением формулы Симпсона. Естественно, исправленный метод Эйлера также является обобщением формулы трапеций.

Заметим, наконец, что формула коррекции

$$y_{m+1} = y_m + \frac{h}{2}[f(x_m, y_m) + f(x_{m+1}, y_{m+1})]$$

является *неявной* в том смысле, что y_{m+1} входит и в правую и левую ее части, причем слева стоит то значение, которое должно быть вычислено в качестве следующего приближения, а справа стоит предыдущее приближение. Все остальные формулы в этой главе были *явными*, т. е. значение y_{m+1} можно было вычислить через значения y_m и, может быть, y_{m-1}, а также через некоторые значения x. В следующей главе, посвященной дифференциальным уравнениям в частных производных, мы также будем проводить разделение формул на явные и неявные.

10.6. АНАЛИЗ ОШИБОК ПРИ ИСПОЛЬЗОВАНИИ МЕТОДОВ ПРОГНОЗА И КОРРЕКЦИИ

Анализ ошибок, возникающих при использовании методов прогноза и коррекции, удобно провести в несколько приемов. Сначала мы рассмотрим ошибки ограничения прогноза (10.33) и коррекции (10.34). Результаты этого анализа выразятся через величины, уже вычисленные в процессе нахождения решения. Затем мы рассмотрим вопрос выбора шага интегрирования h и, наконец, затронем вопросы устойчивости: возрастает ли ошибка (ограничения, округления или ошибка исходной информации) с ростом x.

Чтобы определить ошибку ограничения прогноза (10.33), вспомним, что ряд Тейлора для $y(x)$ в окрестности $x = x_m$ можно записать в следующем виде:

$$y(x) = y_m + y'_m(x - x_m) + \frac{y''_m}{2}(x - x_m)^2 + \frac{1}{6}(x - x_m)^3 y'''(\xi),$$

где ξ лежит между x и x_m. Полагая $x = x_{m+1}$, получаем

$$y_{m+1} = y_m + hy'_m + \frac{h^2}{2} y''_m + \frac{h^3}{6} y'''(\xi_1); \quad x_m \leqslant \xi_1 \leqslant x_{m+1}.$$

Аналогично, полагая $x = x_{m-1}$, получаем

$$y_{m-1} = y_m - hy'_m + \frac{h^2}{2} y''_m - \frac{h^3}{6} y'''(\xi_2); \quad x_{m-1} \leqslant \xi_2 \leqslant x_m.$$

Если вычесть одно из другого эти два равенства и заметить, что

$$\frac{y'''(\xi_1) + y'''(\xi_2)}{2} = y'''(\xi); \quad x_{m-1} \leqslant \xi \leqslant x_{m+1},$$

то тогда

$$y_{m+1} = y_{m-1} + 2hy'_m + \frac{h^3}{3} y'''(\xi).$$

Ошибка ограничения, таким образом, равна

$$E_T^{(n)} = \frac{h^3}{3} y'''(\xi); \quad x_{m-1} \leqslant \xi \leqslant x_{m+1}. \tag{10.39}$$

Если третья производная практически постоянна, то ошибка ограничения равна Kh^3, что и подразумевалось в разд. 10.5, где было сказано, что прогноз (10.33) относится к методам второго порядка.

Мы уже отмечали, что коррекция (10.34) является обобщением формулы трапеций. В гл. 6 было найдено выражение для ошибки ограничения формулы трапеций

$$E_T^{(\kappa)} = -\frac{h^3}{12} y'''(\eta); \quad x_{m-1} \leqslant \eta \leqslant x_{m+1}. \tag{10.40}$$

Как видим, здесь ошибка тоже пропорциональна h^3.

Тот факт, что ошибки ограничения и прогноза и коррекции обе имеют одинаковый порядок, позволяет предложить простой метод оценки y''', следовательно, также и оценки $E_T^{(\kappa)}$. Этот метод напоминает экстраполяционный переход к пределу, использованный в гл. 6 и в разд. 10.4.

Пусть Y_m — точное значение решения при $x = x_m$. Тогда

$$Y_m = y_m^{(0)} + \frac{h^3}{3} y'''(\xi)$$

и

$$Y_m = y_m^{(i)} - \frac{h^3}{12} y'''(\eta),$$

где $y_m^{(0)}$ и $y_m^{(i)}$ вычислены по формулам (10.33) и (10.34). Вычитая эти равенства одно из другого, получаем

$$0 = y_m^{(i)} - y_m^{(0)} - \frac{h^3}{12}[y'''(\eta) + 4y'''(\xi)].$$

Если предположить, что y''' практически постоянна на интервале $x_{m-1} \leqslant x \leqslant x_{m+1}$, то

$$\frac{5h^3}{12} y''' = y_m^{(i)} - y_m^{(0)}$$

и

$$E_T^{(к)} = -\frac{h^3}{12} y''' = \frac{1}{5}[y_m^{(0)} - y_m^{(i)}]. \qquad (10.41)$$

Величины, которые требуются для этой оценки, являются просто результатами вычислений по формулам (10.33) и (10.34). Таким образом, в противоположность методу Рунге — Кутта метод прогноза и коррекции дает в качестве побочного продукта вычислений легко определяемую оценку ошибки ограничения. Заметим, что эта оценка получилась такой простой потому, что ошибки ограничения прогноза и коррекции были одного порядка, в данном случае пропорциональны h^3. Желательно, поэтому, чтобы любая пара прогноз — коррекция обладала этим свойством.

Рассмотрим теперь вопрос выбора величины шага h. К сожалению, не существует формул для выбора шага интегрирования до начала решения задачи, за исключением, может быть, формулы (10.37), но, как правило, пользоваться ею довольно неудобно. Дело в том, что может быть весьма трудно оценить величину M, а если к тому же она сильно меняется на интервале интегрирования, то выбирать величину шага по формуле (10.37) было бы для большей части интервала очень неэкономично, так как h получилось бы слишком малым. Однако, как только вычисления начались, можно оценить ошибку ограничения по формуле (10.41). Если эта ошибка слишком велика, то мы уменьшаем величину шага (очень часто путем деления пополам). Если же, с другой стороны, ошибка гораздо меньше допустимой, то

мы увеличиваем величину шага (очень часто это делается путем удвоения).

Сделаем еще одно замечание по поводу выбора величины шага h. Из формулы (10.37) следует, что метод сходится, если

$$h < \frac{2}{M}.$$

Нам неизвестно значение M. Все же в любом случае очевидно, что чем меньше величина h, тем скорее сойдется итерационный процесс. Можно поставить вопрос следующим образом: если взять слишком малое h, то на каждую точку потребуется немного итераций, но количество точек будет велико; если же взять большое h, то на каждую точку затратится больше итераций, но само количество точек будет меньше. Этот вопрос весьма важен, так как при всех прочих равных условиях желательно составить программу так, чтобы затраты машинного времени были минимальными. Существуют веские эмпирические соображения[1]), согласно которым *оптимальное число итераций равно двум.* Под «оптимальностью» здесь понимается минимальный объем вычислений при заданной точности. Другими словами, шаг интегрирования нужно выбирать так, чтобы критерий сходимости (10.35) выполнялся после двух итераций.

Этот критерий легко запрограммировать. Мы подсчитываем количество итераций; если требуется более двух итераций, то мы уменьшаем величину шага; если достаточно одной итерации, то величину шага можно увеличить.

Мы ничего не говорили о том, что фактически происходит при изменении величины шага. Совершенно ясно, что при изменении шага необходимо принимать какие-то меры для восстановления дальнейшего хода вычислений, так как с новым шагом формулу прогноза уже нельзя более применять непосредственно. Конечно, при удвоении шага теоретически можно было бы вернуться на два шага назад (имеется в виду предыдущая величина шага) и начать работать с вдвое большим h, но практически всегда используется более непосредственный способ. На практике вычисления по формуле прогноза и коррекции в случае изменения шага

[1]) Hull T. E., Greemer A. L., Efficiency of predictor-corrector procedures, *J. ACM*, **10**, 291—301 (1963).

прекращаются и x_m, y_m принимается за новую начальную точку. Затем, начиная от этой точки, методом Рунге — Кутта ищется одна или несколько точек решения с новым шагом, а уже затем решение продолжается по методу прогноза и коррекции.

Наконец, обратимся к вопросу устойчивости, т. е. распространения и роста ошибки, возникшей в ходе вычислений.

Ошибка определяется формулой коррекции (10.34). Окончательное значение y_{m+1} (после бесконечно большого количества итераций) должно удовлетворять уравнению

$$y_{m+1} = y_m + \frac{h}{2}[f(x_m, y_m) + f(x_{m+1}, y_{m+1})], \quad (10.42)$$

если пренебречь ошибкой округления. Если же точное решение равно Y_m, то можно написать

$$Y_{m+1} = Y_m + \frac{h}{2}[f(x_m, Y_m) + f(x_{m+1}, Y_{m+1})] + e_m, \quad (10.43)$$

где e_m включает в себя ошибки ограничения и округления в формуле (10.42).

Если вычесть (10.42) из (10.43) и положить

$$\varepsilon_i = Y_i - y_i,$$

то

$$\varepsilon_{m+1} = \varepsilon_m + \frac{h}{2}\{[f(x_m, Y_m) - f(x_m, y_m)] + [f(x_{m+1}, Y_{m+1}) - f(x_{m+1}, y_{m+1})]\} + e_m.$$

На основании теоремы о среднем значении

$$\varepsilon_{m+1} = \varepsilon_m + \frac{h}{2}[f_y(x_{m+1}, \xi_{m+1}) \cdot \varepsilon_{m+1} + f_y(x_m, \xi_m) \cdot \varepsilon_m] + e_m,$$

где ξ_i лежит между y_i и Y_i. Поэтому

$$\varepsilon_{m+1} = \mu\varepsilon_m + \delta, \quad (10.44)$$

где

$$\mu = \frac{1 + \frac{hf_y}{2}}{1 - \frac{hf_y}{2}} \quad (10.45)$$

и

$$\delta = \frac{e_m}{1 - \frac{hf_y}{2}}. \quad (10.46)$$

При f_y опущены аргументы и, кроме того, мы предполагаем, что e_m не зависит от m.

Уравнение (10.44) называется *разностным уравнением*. Ниже мы рассмотрим, как оно решается относительно ε_m. Вспомним, что по формуле (10.37) для сходимости требовалось выполнение условия

$$\left|\frac{hf_y}{2}\right| < 1. \tag{10.47}$$

Предположим, что

$$f_y < 0. \tag{10.48}$$

Тогда, если h удовлетворяет условию (10.47), то

$$0 < \mu < 1.$$

Таким образом, ошибка ε_m, присутствовавшая в y_m, не возрастает в ε_{m+1}. Иначе говоря, ошибки не возрастают, и метод можно назвать *устойчивым* или, более точно, *абсолютно устойчивым*[1]).

С другой стороны, если

$$f_y > 0,$$

то

$$\mu > 1,$$

и метод становится неустойчивым, но это вовсе не означает, что его нельзя использовать. Мы увидим, что даже в этом случае *относительные* ошибки не возрастают.

Предположим, что f_y постоянна, т. е. дифференциальное уравнение имеет вид

$$y' = Ay,$$

где A — некоторая постоянная. Тогда

$$Y = ae^{Ax}$$

и

$$Y_m = ae^{A(x_0+mh)} = a^* e^{(hA)m}, \tag{10.49}$$

где a^* — новая постоянная, включающая в себя множитель e^{Ax_0}. Если $A > 0$, то Y_m возрастает экспоненциально

[1]) То есть абсолютная ошибка не возрастает.

с ростом m. Даже если абсолютная ошибка растет, то относительная ошибка ε_m/Y_m может и не возрастать. В действительности было бы даже неразумным требовать в данном случае, чтобы абсолютная ошибка была как-то ограничена.

Чтобы выяснить поведение относительной ошибки, заметим, что решение разностного уравнения (10.44) имеет вид

$$\varepsilon_m = a_0\mu^m + \frac{\delta}{1-\mu}, \tag{10.50}$$

где a_0 — произвольная постоянная. Читатель может проверить этот результат, подставив (10.50) в (10.44).

Теперь, если мы выполним деление в формуле (10.45) и подставим $f_y = A$, то получим

$$\mu = \left(1+\frac{hA}{2}\right)\left[1+\left(\frac{hA}{2}\right)+\left(\frac{hA}{2}\right)^2+\ldots\right] =$$

$$= 1 + hA + \frac{(hA)^2}{2} + \frac{(hA)^3}{4} + \ldots.$$

Этот ряд сойдется при $|hA| < 2$, что совпадает с условием сходимости (10.47). Но ведь

$$e^{hA} = 1 + hA + \frac{(hA)^2}{2} + \frac{(hA)^3}{6} + \ldots,$$

так что

$$\mu = e^{hA} + O(h^3).$$

Это означает, что $\mu = e^{hA}$ с тем же порядком точности, что и ошибка ограничения метода. Тогда из (10.50)

$$\varepsilon_m \approx a_0 e^{(hA)m} + \frac{\delta}{1-e^{hA}}. \tag{10.51}$$

Последнее слагаемое не зависит от m и при больших m им можно пренебречь.

Сравним (10.49) и (10.50):

$$\frac{\varepsilon_m}{Y_m} \approx \frac{a^*}{a_0} = \text{const}$$

и относительная ошибка остается постоянной.

Аналогично при $A < 0$ как Y_m, так и ε_m пропорциональны e^{-hA}. Поэтому и в этом случае относительная ошибка не возрастает.

Можно сказать поэтому, что метод, описанный в разд. 10.5, обладает *относительной устойчивостью.*

Необходимо помнить, что при рассмотрении вопроса об относительной устойчивости мы предполагали f_y постоянной. В большинстве практически интересных случаев это будет не так, но существуют убедительные эмпирические соображения, согласно которым в общем случае относительные ошибки в относительно устойчивых процессах не возрастают. То же самое можно сказать и об абсолютных ошибках в абсолютно устойчивых процессах.

Итак, метод прогноза и коррекции, описываемый формулами (10.33) и (10.34), является относительно устойчивым процессом. Более того, он является абсолютно устойчивым процессом, когда $f_y < 0$.

Можно сделать еще одно замечание. Согласно формуле (10.41)

$$Y_m - y_m^{(i)} = E_T^{(*)} = \frac{1}{5}\left(y_m^{(0)} - y_m^{(i)}\right).$$

Поэтому более точное приближение можно найти, введя окончательную поправку:

$$y_m = y_m^{(i)} + \frac{1}{5}\left(y_m^{(0)} - y_m^{(i)}\right). \qquad (10.52)$$

10.7. ДОСТИЖИМАЯ ТОЧНОСТЬ

Не представляется возможным дать в пределах этой книги сколько-нибудь подробный анализ влияния неточности параметров дифференциального уравнения (10.3) или начального условия (10.4), потому что число возможных типов дифференциальных уравнений очень велико. Однако можно обратить внимание читателей на те серьезные последствия, которые могут произойти от неточности исходной информации. Для этого рассмотрим следующий простой пример.

Рассмотрим дифференциальное уравнение[1])

$$y' = y - x.$$

При начальном условии

$$y(0) = 1$$

[1]) См. гл. 4 книги Фокса, указанной в сноске 3 на стр. 409.

точное решение имеет вид

$$y = x + 1.$$

Предположим теперь, что начальное условие задано с точностью 1%. Тогда решение уравнения может находиться в пределах от

$$y^{(1)} = 0.01e^x + x + 1$$

до

$$y^{(2)} = -0.01e^x + x + 1.$$

Легко убедиться, что при $x = 5$ ошибка в определении y может достигать $0.01\ e^5$, или около 30%, в то время как ошибка в начальном условии составляла всего 1%. Относительная ошибка быстро возрастает с ростом x.

Никакой численный метод не может дать решения этого уравнения с точностью выше 30% при $x = 5$, потому что эта ошибка уже заложена в исходной информации. Поэтому Мейерс называет ошибку подобного рода внутренней неустойчивостью.

10.8. СРАВНЕНИЕ МЕТОДОВ

В ходе предыдущего рассмотрения мы уже не раз отмечали сравнительные достоинства и недостатки одноступенчатых методов (методов Рунге — Кутта) и многоступенчатых методов (методов прогноза и коррекции). В этом разделе мы дадим сводку характеристик этих методов и предложим способ их сочетания, воспользовавшись достоинствами каждого из них.

Методы Рунге — Кутта

1. Так как в методах Рунге — Кутта используется информация только об очередной точке решения и не используется информация о ранее найденных точках, то с помощью этих методов можно начинать решение уравнения.

2. По той же самой причине, однако, при использовании этих методов приходится многократно вычислять функцию $f(x, y)$ и затрачивать на это много машинного времени.

3. Используя информацию только об очередной точке решения, эти методы позволяют очень легко менять величину шага h.

4. При использовании этих методов весьма трудно получить оценку для ошибки ограничения.

Методы прогноза и коррекции

Их свойства дополнительны к свойствам методов Рунге—Кутта.

1. Так как в этих методах используется информация о ранее вычисленных точках решения, то с их помощью нельзя начать решение уравнения.

2. Поскольку в этих методах вместо вычисления $f(x, y)$ используется информация о ранее вычисленных точках, то они более экономичны в смысле затрат машинного времени (за исключением, конечно, тех случаев, когда величина шага h слишком велика и требуется много итераций по формуле коррекции).

3. За исключением специальных случаев, не представляющих практического интереса, при любом изменении величины шага h приходится временно возвращаться к методам Рунге — Кутта.

4. В качестве побочного продукта вычислений получается хорошая оценка ошибки ограничения.

Глядя на эти взаимно дополняющие друг друга свойства двух типов методов, можно прийти к выводу, что наиболее целесообразной окажется комбинация из них. Мы предлагаем следующий путь сочетания двух методов.

1. Начать решение с помощью метода Рунге — Кутта, например, по формуле (10.22) и найти y_1.

2. Для вычисления следующих y_m использовать прогноз (10.33) и коррекцию (10.34).

3. Если для вычисления очередного значения y_i требуется более двух итераций или если ошибка ограничения, определяемая по формуле (10.41), слишком велика, необходимо уменьшить величину шага h (см. пункт 4). Если же, с другой стороны, ошибка ограничения слишком мала, то величину шага можно увеличить.

4. Чтобы изменить величину шага, примем последнее значение y_i, которое еще было вычислено достаточно точно, за исходное. Решение уравнения придется снова начать

с новой исходной точки методом Рунге — Кутта и только затем снова перейти к методу прогноза и коррекции.

5. В любом случае, когда с помощью формулы коррекции вычислено очередное значение $y_m^{(i)}$, окончательное значение y_m следует рассчитывать по формуле (10.52).

Естественно, существуют методы прогноза и коррекции более высоких порядков, обеспечивающие соответственно большую точность решения. Эти формулы можно найти в многочисленных книгах по численному анализу [1]), некоторые из них приведены в упражнениях. Следует, однако, учитывать, что для «запуска» этих методов требуются более точные методы Рунге — Кутта, такие, как метод четвертого порядка, описываемый формулами от (10.24) до (10.28).

10.9. ПРАКТИЧЕСКИЙ ПРИМЕР 12: ПОЛЕТ СВЕРХЗВУКОВОГО САМОЛЕТА

Рассмотрим задачу о вычислении траектории полета сверхзвукового турбореактивного самолета. Здесь мы опишем только основную постановку задачи, заинтересованные читатели могут найти необходимые подробности в книге Миеле [2]).

Предположим, что Земля плоская и что самолет имеет постоянный вес и постоянную площадь несущих поверхностей; при этом самолет можно рассматривать в виде точечной массы. Если траектория полета расположена в вертикальной плоскости, то на самолет действуют силы, изображенные на рис. 10.8, где через W обозначен вес, через T — тяга двигателя, через D — сопротивление воздуха и через L — подъемная сила.

Обозначим через V скорость самолета, через θ — угол, который траектория составляет с горизонтом, через Z — высоту.

---

[1]) См., например, гл. 5 книги Henrici P., Discrete variable methods in ordinary differential equations, Wiley, New York, 1962, или гл. 15 книги Hamming R. W., Numerical methods for scientists and engineers, McGraw-Hill, New York, 1962.

[2]) Miele A., Flight mechanics, v. 1, Theory of flight paths, Addison-Wesley, Reading, Mass., 1962.

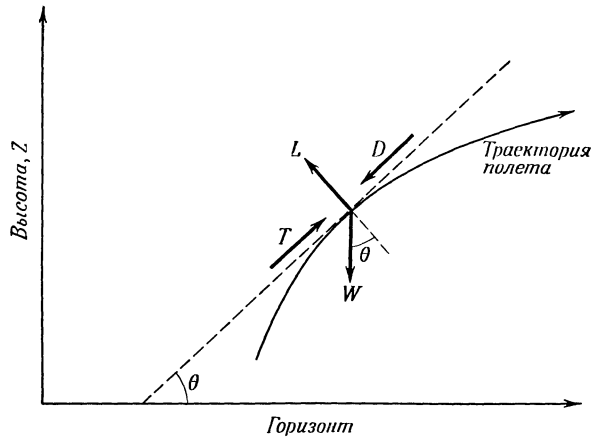
Уравнения движения самолета в направлениях тангенциальном и нормальном к траектории полета можно записать так:

$$\frac{dV}{dZ} = F(V,\theta); \tag{10.53}$$

$$\frac{d\theta}{dZ} = G(V,\theta), \tag{10,54}$$

при начальных условиях

$$V(0) = V_0; \qquad \theta(0) = \theta_0. \tag{10.55}$$



Р и с. 10.8. Силы, действующие на идеализированный самолет в практическом примере 12.

Функции F и G задаются формулами

$$F(V,\theta) = \frac{g(T - W\sin\theta - D)}{WV\sin\theta}; \tag{10.56}$$

$$D = \frac{1}{2} C_D \rho S V^2; \tag{10.57}$$

$$G(V,\theta) = \frac{g(L - W\cos\theta)}{WV^2\sin\theta}; \tag{10.58}$$

$$L = \frac{1}{2} C_L \rho S V^2. \tag{10.59}$$

Приведем таблицу обозначений коэффициентов и также типичные численные значения, которыми мы будем пользоваться для вычислений:

$$\left.\begin{array}{ll} g = 32.174\ \textit{фут/сек}^2 & \text{— ускорение свободного падения} \\ W = 20\,000 \text{ фунтов} & \text{— вес самолета} \\ C_D = 0.02 & \text{— коэффициент сопротивления} \\ C_L = 0.1 & \text{— коэффициент подъемной силы} \\ S = 160\ \textit{фут}^2 & \text{— площадь крыльев} \end{array}\right\} \quad (10.60)$$

Оставшиеся две величины, которые мы еще не определили, суть тяга T и плотность воздуха ρ. Мы будем предполагать, что тяга является функцией скорости и высоты. Плотность воздуха мы будем предполагать функцией только от высоты, хотя более точное рассмотрение должно было бы учитывать зависимость плотности от температуры. Нижеприведенные формулы являются полуэмпирическими, выведенными на основе экспериментальных данных

$$T = \left(1 + \frac{1}{2} e^{-(V \cdot 10^{-3} - 1.5)^2}\right)(10^4 - 0.27Z)\ \textit{фунтов}. \quad (10.61)$$

$$\left.\begin{array}{l} \rho = \rho_0 e^{-Z/2.36 \cdot 10^4}\ \textit{фунт} \cdot \textit{сек}^2/\textit{фут}^4, \\ \rho_0 = 2.3769 \cdot 10^{-3}\ \textit{фунт} \cdot \textit{сек}^2/\textit{фут}^4. \end{array}\right\} \quad (10.62)$$

Уравнения (10.53) и (10.54) мы будем решать одновременно. Мы будем принимать, что функции F и G определены соотношениями (10.56) — (10.62) и что начальные условия даны в виде (10.55). Процесс решения сводится к следующему:

1. Используется исправленный метод Эйлера для вычисления значений V_1, θ_1.

$$V_1 = V_0 + \frac{h}{2}[F(V_0, \theta_0) + F(V_0 + hF(V_0, \theta_0), \theta_0 + hG(V_0, \theta_0))];$$

$$\theta_1 = \theta_0 + \frac{h}{2}[G(V_0, \theta_0) + G(V_0 + hF(V_0, \theta_0), \theta_0 + hG(V_0, \theta_0))].$$

В уравнение для V входят и V и θ, и в уравнение для θ также входят оба неизвестных. Именно поэтому мы и гово-

рим, что эти два уравнения составляют систему. В правых частях уравнений, однако, V и θ появляются только с индексом нуль, т. е., иными словами, в правых частях уравнений при решении исправленным методом Эйлера используются предыдущие, заранее известные значения.

2. После того как с помощью исправленного метода Эйлера вычислены точки V_1 и θ_1, можно для дальнейших вычислений использовать метод прогноза и коррекции. Для $m = 1, 2, \ldots$ значения V_{m+1} и θ_{m+1} предсказываются с помощью формул

$$V_{m+1}^{(0)} = V_{m-1} + 2hF(V_m, \theta_m),$$
$$\theta_{m+1}^{(0)} = \theta_{m-1} + 2hG(V_m, \theta_m).$$

Эти формулы аналогичны (10.33).

3. Коррекция значений V_{m+1} и θ_{m+1} производится следующим образом:

$$V_{m+1}^{(i)} = V_m + \frac{h}{2}[F(V_m, \theta_m) + F(V_{m+1}^{(i-1)}, \theta_{m+1}^{(i-1)})],$$

$$\theta_{m+1}^{(i)} = \theta_m + \frac{h}{2}[G(V_m, \theta_m) + G(V_{m+1}^{(i-1)}, \theta_{m+1}^{(i-i)})].$$

Это аналогично (10.34). На данном этапе решения требуется, чтобы итерации производились для V и для θ по очереди, так как уравнения для V и для θ представляют собой единую систему. Если пытаться вычислить окончательное значение V, производя итерации только с уравнением для V, то в нем пришлось бы использовать только предсказанное значение θ, которое является всего лишь приближенным. Когда мы после этого произвели бы итерации с целью вычислить более точное значение θ, то, естественно, в результате получилось бы значение θ, отличное от того, которое было использовано для вычисления V. Короче говоря, уравнения представляют собой единую систему. (Внимательный читатель может заметить отчетливую параллель с итерационным методом Гаусса — Зейделя для решения системы линейных алгебраических уравнений, где также приходится производить итерации для всех уравнений по очереди. В самом деле, итерации только с одним уравнением не учитывают того, что эти уравнения представляют собой единую систему.)

При строгом рассмотрении вопроса необходимо было бы доказать, что такой процесс поочередных итераций сходится. Не приводя громоздких выводов отметим, что такой процесс поочередных итераций действительно сходится для достаточно широкого класса задач.

4. Ошибку ограничения мы оцениваем с помощью соотношений

$$E_T(V_{m+1}) = \frac{1}{5}(V_m^{(0)} - V_m^{(i)}),$$

$$E_T(\theta_{m+1}) = \frac{1}{5}(\theta_m^{(0)} - \theta_m^{(i)}).$$

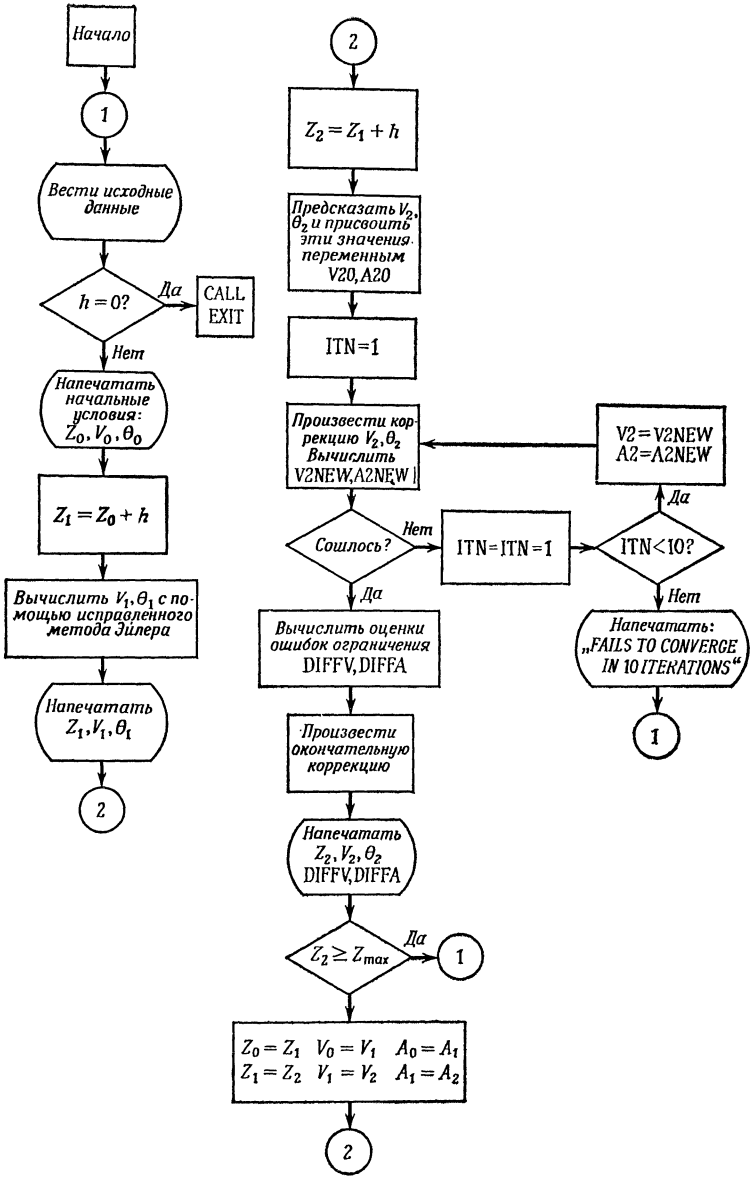
Оценки ошибки ограничения печатаются, а также используются для учета окончательной поправки в значениях V и θ согласно формуле (10.52).

Блок-схема программы решения задачи о полете самолета приведена на рис. 10.9. Эта блок-схема построена несколько по-другому, нежели большинство остальных в настоящей книге. В этой блок-схеме мы не старались показать все подробности вычислительного процесса, а попытались сделать возможно яснее сам принцип решения задачи. Основное внимание уделено не столько самим вычислительным операциям, сколько вынесению решений и передачам управления.

Программа начинается с ввода данных, относящихся к самолету и условиям полета. Условия полета описываются начальной и максимальной высотами Z_0 и $Z_{\max}$, начальным углом тректорни θ_0 и начальной скоростью V_0. Самолет описывается своим весом W, площадью крыльев S, коэффициентом подъемной силы C_L и коэффициентом сопротивления C_D. Наконец, для численного решения задачи требуется задать величину шага h и допуск для сходимости итерационного процесса TOLER. В процессе решения мы печатаем различные параметры, заголовок и начальные условия. Нулевое значение h сигнализирует об окончании колоды перфокарт.

Решение начинается с помощью исправленного метода Эйлера, при этом вычисляются V_1 и θ_1 и выводятся на печать.

Теперь можно перейти к методу прогноза и коррекции. На блок-схеме использованы следующие условные обо-



Р и с. 10.9. Блок-схема программы для решения системы двух дифференциальных уравнений полета (практический пример 12).

значения: Z_0 на блок-схеме и в программе соответствует x_{m-1} в тексте, Z_1 соответствует x_m, Z_2 соответствует x_{m+1}.

Нетрудно убедиться, что использование метода прогноза и коррекции в программе соответствует его описанию в тексте с очевидной разницей, что здесь мы имеем систему из двух уравнений, итерации которых нужно производить по очереди. В программу введен счетчик количества итераций, обозначенный через ITN. Это позволяет остановить работу программы в том случае, когда итерационный процесс расходится. Соотношение (10.41) используется для оценки ошибки ограничения, эти оценки затем печатаются. Соотношение (10.52) используется для ввода окончательных поправок в вычисленные значения V и θ. После печати очередной строки вычисленных значений мы проверяем, следует ли закончить расчеты, и если нет, то возвращаемся и снова используем метод прогноза и коррекции. Однако перед этим необходимо сначала присвоить переменной Z_0 то значение, которое имела до этого переменная Z_1, и соответственно присвоить переменной Z_1 значение, которое имела до этого переменная Z_2. Аналогичные операции следует проделать и с переменными V и θ. После того как внесены эти изменения, можно найти следующую точку решения с помощью метода прогноза и коррекции.

Программа, проведенная на рис. 10.10, точно соответствует блок-схеме и притом дополнена несколькими приемами, облегчающими программирование. Некоторым операторам в программе присвоены номера, без которых можно было бы обойтись; это сделано для того, чтобы программу было легче описать.

Четыре первых оператора в программе представляют собой арифметические операторы-функции, с помощью которых вычисляются плотность воздуха, тяга двигателя, а также правые части двух дифференциальных уравнений. Следует отметить, что в функции, предназначенные для вычисления правых частей, входят некоторые переменные, не являющиеся их аргументами, такие, как вес самолета или тяга двигателя.

Операторы с 5 по 105 включительно описывают процесс ввода исходной информации, печати параметров задачи и начальных условий. Можно заметить, что вместо операто-

```
      DENSTYF(Z) = 2.3769E-3 * EXPF(-Z/2.36E4)
      THRUSTF(V,Z)=(1.0+0.5*EXPF(-(V*0.001-1.5)**2))*(10000.0-0.27*Z)
      FFNCF(V, A) = GRAV * (T - WEIGHT * SINF(A)
     1    - 0.5 * CD * RHO * S * V**2) / (WEIGHT * V * SINF(A))
      GFNCF(V, A) = GRAV *(0.5*CL*RHO*S*V**2 - WEIGHT*COSF(A)) /
     1    (WEIGHT * V**2 * SINF(A))
      GRAV = 32.174
    5 READ INPUT TAPE 7, 1, Z0, ZMAX, THETA0, V0, WEIGHT, S, CL, CD,
     1   H, TOLER
    1 FORMAT (8F10.0/2F10.0)
      IF(H) 3, 3, 2
    3 CALL EXIT
    2 WRITE OUTPUT TAPE 10, 4, Z0, ZMAX, THETA0, V0, WEIGHT, S, CL,
     1 CD, H, TOLER
    4 FORMAT (4H1Z0=, F7.0, 7H  ZMAX=, F7.0, 9H  THETA0=, F6.2,
     1 5H  V0=, F6.0//8H WEIGHT=, F7.0, 5X, 10HWING AREA=, F5.0//
     2 18H LIFT COEFFICIENT=, F6.3, 5X, 17HDRAG COEFFICIENT=, F6.3//
     3 3H H=, F7.2, 5X, 10HTOLERANCE=, F7.4///9H ALTITUDE, 2X,
     4 8HVELOCITY,4X, 5HANGLE, 6X, 5HDIFFV, 5X,5HDIFFA, 3X,3HITN//)
      WRITE OUTPUT TAPE 10, 105, Z0, V0, THETA0
  105 FORMAT (F8.0, F11.2, F10.3, F10.5, F10.5, I4)
   14 Z1 = Z0 + H
      RHO = DENSTYF(Z0)
      T = THRUSTF(V0, Z0)
      A0 = THETA0 /57.29578
      F = FFNCF(V0, A0)
      G = GFNCF(V0, A0)
      VV = V0 + H * F
      AA = A0 + H * G
      RHO = DENSTYF(Z1)
      T = THRUSTF(VV, Z1)
      V1 = V0 + 0.5 * H * (F + FFNCF(VV, AA))
      A1 = A0 + 0.5 * H * (G + GFNCF(VV, AA))
      THETA = A1 * 57.29578
  114 WRITE OUTPUT TAPE 10, 105, Z1, V1, THETA
   16 Z2 = Z1 + H
      RHO = DENSTYF(Z1)
      T = THRUSTF(V1, Z1)
      F = FFNCF(V1, A1)
      G = GFNCF(V1, A1)
      V2 = V0 + 2.0 * H * F
      A2 = A0 + 2.0 * H * G
      V20 = V2
  116 A20 = A2
   17 RHO = DENSTYF(Z2)
      ITN = 1
   12 T = THRUSTF(V2, Z2)
      V2NEW = V1 + 0.5 * H * (F + FFNCF(V2, A2))
      A2NEW = A1 + 0.5 * H * (G + GFNCF(V2, A2))
      IF(ABSF((V2 - V2NEW)/V2NEW) - TOLER)  10, 11, 11
   10 IF(ABSF((A2 - A2NEW)/A2NEW) - TOLER) 20, 11, 11
   11 ITN = ITN + 1
      IF (ITN - 10) 19, 18, 18
   19 V2 = V2NEW
      A2 = A2NEW
      GO TO 12
   18 WRITE OUTPUT TAPE 10, 9876
 9876 FORMAT (35H FAILS TO CONVERGE IN 10 ITERATIONS)
  117 GO TO 5
   20 DIFFV = 0.2 * (V20 - V2NEW)
      DIFFA = 0.2 * (A20 - A2NEW)
      V2 = V2NEW + DIFFV
      A2 = A2NEW + DIFFA
      THETA = 57.29578 * A2
      DIFFA = 57.29578 * DIFFA
      WRITE OUTPUT TAPE 10, 105, Z2, V2, THETA, DIFFV, DIFFA, ITN
      IF (Z2 + 1.0 - ZMAX) 15, 5, 5
   15 A0 = A1
      A1 = A2
      Z0 = Z1
      Z1 = Z2
      V0 = V1
      V1 = V2
      GO TO 16
      END
```

Р и с. 10.10. Программа для решения системы двух дифференциальных уравнений полета, составленная согласно блок-схеме рис. 10.9. (практический пример 12).

ра READ использован оператор READ INPUT TAPE, а вместо оператора PRINT использован оператор WRITE OUTPUT TAPE. Причина, по которой использованы именно такие операторы, очевидна: вычисления по этой программе производились на большой ЭЦВМ, где вводимая и выводимая информация обычно записывается на магнитной ленте, а подготовка магнитной ленты с исходными данными и расшифровка ленты с результатами вычислений производится с помощью какой-нибудь малой машины. Во время подготовки входных данных и расшифровки результатов большая ЭЦВМ может производить вычисления по другим программам. Это сделано для того, чтобы избежать простоев в работе большой машины, неизбежных в противном случае из-за относительно малого быстродействия вводных и выводных устройств. Практически все большие ЭЦВМ работают таким образом.

Вывод параметров производится по довольно сложному оператору FORMAT, с помощью которого печатаются также обозначения различных переменных, контролируются интервалы между словами и расстояние между строками печати; последняя часть этого оператора служит для печатания заголовка. Читатели, уже ознакомившиеся с приложением 1, могут попытаться изучить этот оператор; тот, кто не интересуется форматами, может пропустить этот оператор. В операторе FORMAT под номером 105 имеются три спецификации поля, которые не используются предыдущим оператором вывода, а просто игнорируются. Эти три спецификации будут использованы в дальнейшем, когда тот же оператор FORMAT будет упомянут в другом операторе WRITE OUTPUT TAPE, где указаны еще три числа, предназначенные для печати.

Вычисления по исправленному методу Эйлера производятся с помощью операторов от 14 до 114. При этом на основе начальных условий определяется первая точка решения. Следует обратить внимание на необходимость перевода углов из градусов в радианы (для внутренних вычислений) и обратно (для вывода на печать). Так как в оба дифференциальных уравнения входят тяга двигателя и плотность воздуха, то их необходимо рассчитать перед вычислением правых частей.

С помощью операторов от 16 до 116 предсказываются значения V2 и A2. Эти предсказанные значения сохраняются под обозначениями V20 и A20; после того как сойдется процесс коррекции, можно будет сравнить предсказанное и скорректированное значения.

С помощью операторов от 17 до 117 производится коррекция значений V2 и A2, пока процесс не сойдется, если для этого требуется не более 10 итераций. Вычисление плотности воздуха не включено в этот цикл, поскольку она зависит только от высоты и вычисляется один раз перед началом цикла. С другой стороны, тяга двигателя зависит от скорости и поэтому должна рассчитываться заново при каждой очередной итерации; последовательно вычисляемые приближения для скорости отличаются друг от друга. Критерий сходимости процесса основан на вычислении относительной разности последующих приближений. Заметим, что для окончания итерационного процесса сойтись должны обе переменные V и θ. Только после этого начинается печатание результатов.

С помощью операторов от 20 до конца программы производится окончательная коррекция (10.52), осуществляется печатание результатов вычислений, проверяется, не окончен ли очередной цикл вычислений по высотам, и если нет, то управление передается оператору 15, с которого начинается определение следующей точки решения. Обратите внимание на слагаемое 1.0 в операторе IF, проверяющем окончание цикла по Z2: это слагаемое введено потому, что в результате многократного прибавления H и ошибки округления значение Z2 может оказаться немного меньше ZMAX, хотя и очень близко к нему. Естественно, мы предполагаем, что H при вычислениях по этой программе будет гораздо больше 1. Если ZMAX еще не достигнуто, то управление передается оператору 15 и нескольким идущим за ним, где происходит подготовка переменных для следующего шага вычислений. Если ZMAX уже достигнуто, то управление передается оператору 5 для ввода нового набора исходных данных. Наконец, при обнаружении на перфокарте значения $H = 0$ вычисления по программе оканчиваются.

Ранее указывалось, что можно составить программу, где величина шага автоматически уменьшается вдвое, если для вычисления очередной точки требуется слишком много

итераций, или удваивается, если количество итераций слишком мало. Чтобы избежать излишних усложнений, мы не стали вводить автоматическое изменение шага интегрирования, но тщательно составленная программа для решения задач подобного рода должна обязательно предусматривать автоматическое изменение шага. Будет полезно отметить некоторые детали, которые надо учитывать при введении в программу автоматического изменения шага интегрирования.

1. Если заранее не предусмотреть такой возможности, то может случиться, что условия уменьшения и увеличения величины шага выполняются в одной и той же точке. При этом мы возвращаемся туда, откуда начали, и можем повторять эти два взаимно уничтожающихся действия до бесконечности.

2. Многие дифференциальные уравнения имеют особенности, т. е. точки, в которых решение не существует. Мы увидим пример такой особенности ниже, в одном из вариантов решения уравнения полета. При приближении к такой точке может случиться так, что шаг начнет автоматически уменьшаться и уменьшаться, а особенность так и не будет достигнута. Поэтому в программе необходимо предусматривать ограничивающий счетчик количества уменьшений шага интегрирования.

3. Следует писать программы таким образом, чтобы по желанию можно было отказаться от автоматического выбора шага. Например, может потребоваться график, построенный на основании результатов вычисления; в этом случае для нас важнее постоянство шага интегрирования, нежели экономия машинного времени. Это относится в основном к увеличению шага интегрирования.

4. Если уменьшать шаг приходится сразу же после ввода в действие метода прогноза и коррекции, то как далеко нужно возвращаться назад? Если принять в качестве исходной ту точку, которая найдена с помощью исправленного метода Эйлера, то ее точность весьма сомнительна. Если возвращаться снова к начальным условиям, то что делать с той строкой результатов вычислений, которая уже напечатана?

Все эти трудности вполне преодолимы; в некоторых случаях бывает просто необходимо составлять программу,

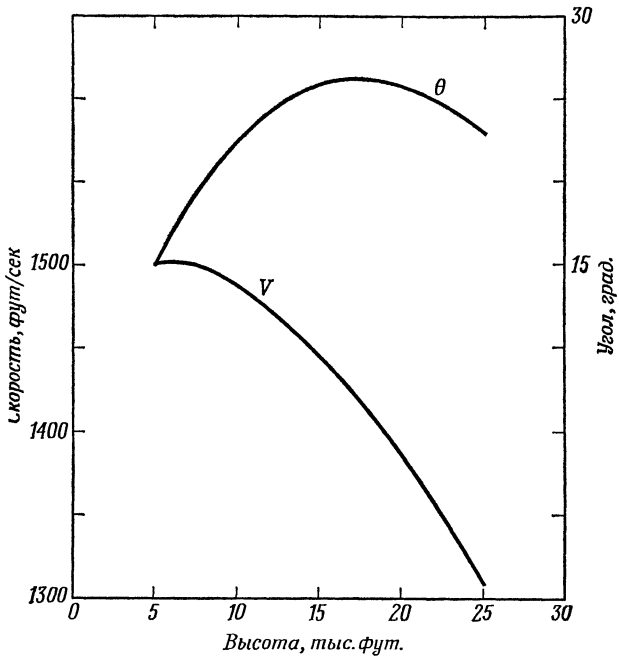
предусматривая в ней автоматический выбор шага интегрирования. В нашей задаче, однако, такое усложнение программы совершенно не оправдано.

```
ZO= 5000.  ZMAX= 25000.  THETAO= 15.00  VO= 1500.

WEIGHT= 20000.      WING AREA= 160.

LIFT COEFFICIENT= 0.100      DRAG COEFFICIENT= 0.020

H= 500.00      TOLERANCE= 0.0001

ALTITUDE  VELOCITY    ANGLE      DIFFV      DIFFA     ITN

  5000.    1500.00    15.000
  5500.    1501.29    16.146
  6000.    1501.69    17.175   -0.01588   -0.00195    2
  6500.    1501.38    18.108   -0.01127   -0.00118    2
  7000.    1500.49    18.959   -0.00818   -0.00094    2
  7500.    1499.10    19.738   -0.00611   -0.00071    2
  8000.    1497.29    20.452   -0.00469   -0.00057    2
  8500.    1495.11    21.109   -0.00367   -0.00046    2
  9000.    1492.60    21.712   -0.00299   -0.00037    1
  9500.    1489.79    22.267   -0.00247   -0.00031    1
 10000.    1486.71    22.777   -0.00199   -0.00026    1
 10500.    1483.38    23.245   -0.00164   -0.00022    1
 11000.    1479.81    23.674   -0.00135   -0.00019    1
 11500.    1476.04    24.065   -0.00112   -0.00016    1
 12000.    1472.05    24.420   -0.00093   -0.00014    1
 12500.    1467.88    24.741   -0.00078   -0.00012    1
 13000.    1463.52    25.030   -0.00065   -0.00010    1
 13500.    1458.98    25.286   -0.00054   -0.00008    1
 14000.    1454.27    25.512   -0.00045   -0.00007    1
 14500.    1449.39    25.709   -0.00037   -0.00006    1
 15000.    1444.34    25.876   -0.00030   -0.00005    1
 15500.    1439.14    26.014   -0.00024   -0.00004    1
 16000.    1433.78    26.124   -0.00019   -0.00003    1
 16500.    1428.27    26.206   -0.00014   -0.00002    1
 17000.    1422.60    26.261   -0.00010   -0.00001    1
 17500.    1416.78    26.287   -0.00007   -0.00000    1
 18000.    1410.81    26.287   -0.00003    0.00001    1
 18500.    1404.68    26.258    0.         0.00001    1
 19000.    1398.41    26.202    0.00002    0.00002    1
 19500.    1391.98    26.117    0.00005    0.00003    1
 20000.    1385.40    26.003    0.00007    0.00004    1
 20500.    1378.67    25.861    0.00010    0.00005    1
 21000.    1371.78    25.688    0.00012    0.00007    1
 21500.    1364.74    25.485    0.00014    0.00008    1
 22000.    1357.54    25.249    0.00016    0.00009    1
 22500.    1350.18    24.981    0.00018    0.00011    1
 23000.    1342.66    24.678    0.00020    0.00013    1
 23500.    1334.97    24.339    0.00021    0.00015    1
 24000.    1327.12    23.961    0.00023    0.00018    1
 24500.    1319.10    23.544    0.00025    0.00021    1
 25000.    1310.91    23.083    0.00026    0.00025    1
```

Рис. 10.11. Выходная печать для программы рис. 10.10 (практический пример 12).
Значения параметров указаны в заголовке.

На рис. 10.11 воспроизведена таблица результатов вычислений, напечатанная с помощью нашей программы. Значения параметров задачи размещены над таблицей.

Заметим, что $H = 500$ и допуск равен 0.0001. Так как в качестве допуска берется относительная разность последовательных приближений, то величина 0.0001 означает, что два последовательных приближения к значению скорости



Р и с. 10.12. Графическое представление данных рис. 10.11, показывающее зависимость скорости и угла от высоты (практический пример 12).

должны отличаться не более чем на 0,15 *фут/сек* и два последовательных приближения к значению угла — не более чем на 0.017° для высот около 6000 *фут*. Эти значения несколько больше оценок ошибок ограничения, так что не удивительно, что после первых шести строк для остальных потребовалась только одна итерация.

На рис. 10.12. построен график скорости и угла в зависимости от высоты. Мы видим, что сначала скорость слегка возрастает, затем начинает уменьшаться. Угол сначала

стабильно возрастает; это возрастание угла, естественно, отчасти ответственно за потерю скорости, так как чем круче траектория самолета по отношению к горизонту, тем больше становится влияние члена $W \sin\theta$ в формуле для силы, противодействующей движению самолета. Но постепенно с падением скорости и с достижением больших высот наступает момент, когда уменьшаются и тяга двигателя, и подъемная сила. Уменьшение тяги приводит к тому, что скорость самолета уменьшается еще сильнее, а снижение подъемной силы замедляет возрастание крутизны траектории. В конце концов угол перестает возрастать и начинает уменьшаться. (Необходимо помнить, что́ именно мы здесь строим на графике. Уменьшение угла вовсе не означает, что самолет пошел на снижение, это означает всего лишь, что уменьшилась крутизна траектории, оставаясь при этом положительной.)

Необходимо еще раз подчеркнуть, что рассматриваемая нами модель полета является в высшей степени идеализированной. В частности, мы совершенно не учитываем возможные манипуляции с рулями самолета и не принимаем в расчет изменения числа оборотов двигателя. В действительном полете, конечно, возможны компромиссные решения между скоростью и крутизной траектории.

Возвращаясь снова к численным методам, посмотрим, как отразится изменение шага интегрирования на результатах вычислений и как при этом изменятся ошибки ограничения и необходимое количество итераций. Рис. 10.13 представляет собой такую же таблицу выходных данных, как и рис. 10.11, с той разницей, что теперь $H = 1000$. Можно убедиться в том, что ошибки ограничения возросли, причем не в отношении два к одному, а гораздо сильнее. Так как мы производим окончательную коррекцию вычисленных значений на основе этих оценок ошибки ограничения, разница между значениями самих переменных, VELOCITY и ANGLE, довольно мала. Однако все же некоторая разница имеется, и это подтверждает, что мы имеем дело не с ошибками ограничения, а всего лишь с оценками этих ошибок. Если бы существовал способ точно определить ошибки ограничения, то можно было бы получать точные решения дифференциальных уравнений независимо от шага интегрирования.

Наряду с возрастанием ошибок ограничения увеличивается также количество итераций, необходимых для определения очередной точки. Если при $H = 500$ для большинства точек было достаточно одной итерации, то теперь для большинства точек требуется уже по две итерации.

Можно пойти еще дальше и попытаться произвести вычисления с шагом $H = 2000$. Результаты этих вычисле-

```
ZO= 5000.  ZMAX= 25000.  THETAO= 15.00  VO= 1500.

WEIGHT= 20000.      WING AREA= 160.

LIFT COEFFICIENT= 0.100      DRAG COEFFICIENT= 0.020

H=1000.00      TOLERANCE= 0.0001
```

ALTITUDE	VELOCITY	ANGLE	DIFFV	DIFFA	ITN
5000.	1500.00	15.000			
6000.	1501.66	17.181			
7000.	1500.43	18.961	-0.08774	-0.01163	2
8000.	1497.23	20.454	-0.04428	-0.00496	2
9000.	1492.53	21.713	-0.02823	-0.00362	2
10000.	1486.64	22.778	-0.01780	-0.00238	2
11000.	1479.75	23.674	-0.01203	-0.00171	2
12000.	1471.99	24.420	-0.00826	-0.00124	2
13000.	1463.45	25.029	-0.00576	-0.00091	2
14000.	1454.20	25.512	-0.00402	-0.00066	2
15000.	1444.28	25.875	-0.00286	-0.00046	1
16000.	1433.72	26.123	-0.00196	-0.00029	1
17000.	1422.54	26.259	-0.00114	-0.00015	1
18000.	1410.74	26.285	-0.00052	-0.00001	1
19000.	1398.35	26.200	0.00000	0.00013	1
20000.	1385.34	26.001	0.00044	0.00028	1
21000.	1371.72	25.686	0.00083	0.00045	1
22000.	1357.48	25.246	0.00107	0.00066	2
23000.	1342.60	24.675	0.00128	0.00093	2
24000.	1327.06	23.958	0.00153	0.00129	2
25000.	1310.85	23.078	0.00173	0.00179	2

Р и с. 10.13. То же, что на рис. 10.11, за исключением того, что $H = 1000$ (практический пример 12 .

ний приведены на рис. 10.14. Мы видим, что новое удвоение шага привело к тому, что ошибки ограничения возросли в 5—10 раз. Теперь все точки решения, кроме одной, пришлось определять с помощью нескольких итераций по формуле коррекции.

Если бы это вычисление пришлось воспроизводить много раз примерно с теми же параметрами, то целесообразнее всего было бы принять $H = 2000$. Мы видим, что в большинстве случаев достаточно двух итераций, а как уже отмечали ранее, две итерации являются оптимальным количеством с точки зрения минимизации затрат машинного времени при заданной точности. Ошибки ограничения лежат

в допустимых пределах, так же как и разности между значениями, вычисленными при величине шага, равной 2000, и более точными для меньшего шага.

Хотя этот практический пример и построен в значительной мере на основе идеализированной задачи, он все же может явиться хорошим пояснением того, что могло остаться неясным ранее. Очень редко случается так, чтобы вычисления на ЭЦВМ производились только для какого-нибудь одного набора исходных данных. Гораздо чаще ставится

```
ZO=  5000.  ZMAX= 25000.  THETAO= 15.00  VO= 1500.

WEIGHT= 20000.      WING AREA= 160.

LIFT COEFFICIENT= 0.100      DRAG COEFFICIENT= 0.020

H=2000.00      TOLERANCE= 0.0001

ALTITUDE  VELOCITY    ANGLE       DIFFV      DIFFA    ITN

  5000.    1500.00    15.000
  7000.    1500.34    19.011
  9000.    1492.20    21.729   -0.39923   -0.05955    3
 11000.    1479.41    23.688   -0.10399   -0.01357    2
 13000.    1463.09    25.039   -0.06668   -0.01030    2
 15000.    1443.91    25.882   -0.02309   -0.00482    2
 17000.    1422.16    26.264   -0.01347   -0.00221    2
 19000.    1397.97    26.202   -0.00316    0.00013    1
 21000.    1371.34    25.685    0.00340    0.00252    2
 23000.    1342.22    24.670    0.00833    0.00574    2
 25000.    1310.47    23.088    0.01190    0.01115    3
```

Р и с. 10.14. То же, что на рис. 10.11, за исключением того, что $H = 2000$ (практический пример 12).

задача рассчитать поведение какой-либо системы при различных изменяющихся условиях работы. Очень часто эти вычисления производятся для того, чтобы определить условия, в которых эта система будет работать оптимальным образом.

В случае самолета наиболее интересными характеристиками полета являются такие величины, как время подъема с одной высоты на другую, время полета на заданное расстояние на заданной высоте или на некоторой последовательности высот и т. д. Можно поставить, например, такую задачу: в ходе длительного полета общий вес самолета с топливом уменьшается; высота, на которой должен лететь самолет для того, чтобы расход топлива был минимальным, зависит от веса самолета. Как нужно менять высоту в ходе полета, чтобы добиться минимального расхода топлива?

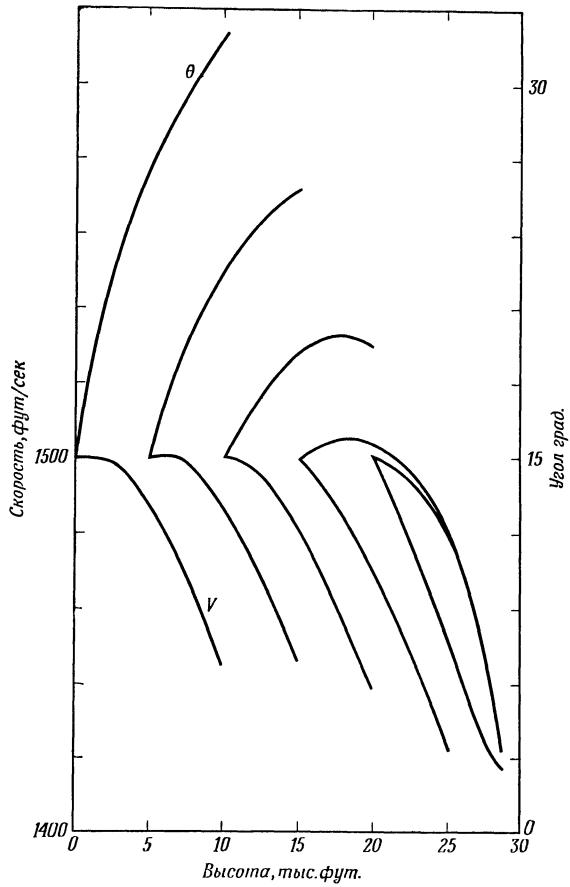
Решение подобных вопросов может занять сотни часов машинного времени. Вместе с задачами конструктивного расчета самого самолета и двигателя они составляют одну из важнейших областей применения ЭЦВМ.

Основное преимущество этого подхода состоит в том, что, как только удалось сформулировать математическое описание системы любого рода, в наших руках оказывается хороший инструмент для исследования разнообразных процессов, которые могут происходить в этой системе, а также для конструирования самой системы.

Чтобы приступить к расчету практически интересных задач, связанных с самолетом, нам потребовалось бы несколько дополнительных переменных — переменные, описывающие положение рулей самолета, количество оборотов двигателя, информация о зависимости потребления топлива от условий полета и т. д. При наших знаниях программирования вполне можно было бы ввести эти дополнительные зависимости, но это чересчур усложнило бы программы, в чем нет никакой необходимости для большинства читателей. Поэтому мы ограничимся здесь только двумя несложными примерами, основанными на уже рассмотренной постановке задачи.

Поставим вопрос: что произойдет, если самолет будет преодолевать перепад высот 10 000 *фут*, начиная с различных высот, но каждый раз с начальной скоростью 1500 *фут/сек* и с углом 15°? С помощью нашей программы легко произвести такой расчет. Все, что нужно сделать, это подготовить перфокарты с исходными данными, где Z0 и ZMAX отличались бы на 10 000 *фут*, значения Z0 проходили бы заданную последовательность начальных высот, а V_0 и θ_0 были бы равны соответственно 1500 *фут/сек* и 15°. Результаты вычислений для пяти высот изображены на рис. 10.15. Мы видим, что во всех случаях, независимо от того, начинается ли полет от уровня моря или от 20 000 *фут*, скорость спадает примерно по одному и тому же закону. Угол, напротив, очень сильно зависит от начальной высоты. Если полет начинается от уровня моря или от высоты 5000 *фут*, то после преодоления перепада высот 10 000 *фут* угол еще продолжает возрастать; если полет начинается от высоты 10 000 или 15 000 *фут*, то угол сначала возрастает, а затем начинает падать; если полет

начинается от 20 000 *фут*, то угол сразу начинает уменьшаться. Более того, при такой постановке задачи самолет



Р и с. 10.15. Графики зависимости скорости и угла от высоты при преодолении самолетом перепада высот 10 000 *фут* (практический пример 12).

Начальная скорость и начальный угол во всех случаях одинаковы, но начальные высоты различны.

не сможет подняться от 20 000 до 30 000 *фут* с начальной скоростью 1500 *фут/сек* и углом 15°. Физическая причина,

очевидно, заключается в том, что тяга двигателя сильно уменьшается с высотой. Математически трудность состоит в том, что самолет переходит в горизонтальный полет, а математическая формулировка задачи не предусматривает возможности горизонтального полета (из-за $\sin\theta$ в знаменателе обоих дифференциальных уравнений).

На рис. 10.16 приведены несколько последних строк результатов вычисления, напечатанных машиной во время

```
28000.    1419.91    5.845   -0.00013    0.00004   1
28050.    1419.51    5.657   -0.00014    0.00005   1
28100.    1419.14    5.463   -0.00017    0.00006   1
28150.    1418.78    5.260   -0.00020    0.00007   1
28200.    1418.44    5.048   -0.00026    0.00009   1
28250.    1418.13    4.825   -0.00033    0.00011   2
28300.    1417.84    4.591   -0.00042    0.00015   2
28350.    1417.60    4.342   -0.00054    0.00019   2
28400.    1417.39    4.078   -0.00074    0.00025   2
28450.    1417.24    3.794   -0.00103    0.00035   2
28500.    1417.16    3.485   -0.00150    0.00050   2
28550.    1417.16    3.145   -0.00235    0.00079   2
28600.    1417.28    2.762   -0.00409    0.00136   3
28650.    1417.60    2.314   -0.00832    0.00275   3
28700.    1418.25    1.751   -0.02315    0.00760   5
FAILS TO CONVERGE IN 10 ITERATIONS
```

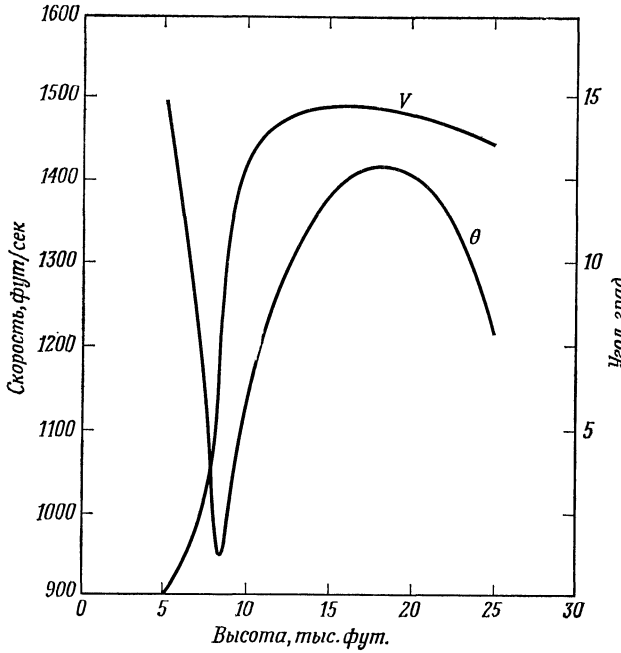
Р и с. 10.16. Несколько последних строк выходной печати (практический пример 12).

Рассчитывалась попытка преодоления самолетом перепада высот от 20 000 до 30 000 *фут*.

расчета полета от 20000 до 30000 *фут*. Мы видим, что по мере того, как $\sin\theta$ становится все меньше и меньше, ошибки ограничения возрастают. Количество необходимых итераций соответственно также растет, пока не наступает такой момент, когда при допуске 0.0001 не удается достичь сходимости за 10 итераций. Даже если уменьшить шаг и увеличить допуск, все равно не удастся провести решение существенно дальше: угол неуклонно уменьшается, стремясь к нулю.

Попробуем рассмотреть другую задачу: что произойдет со скоростью и с углом, если начинать полет с различной начальной скоростью, всегда преодолевая перепад высот от 5000 до 25 000 *фут* и начиная полет под углом 15°? На рис. 10.17 изображен график поведения V и θ для $V_0 = 900$ *фут/сек*. В самом начале полета скорость слишком мала, чтобы обеспечить достаточную подъемную силу. Малая скорость означает малую тягу, и угол быстро уменьшается. Но тяга все же достаточна для того, чтобы создать ускорение, и чем меньше становится угол, тем бо́льшая

доля тяги идет на создание ускорения. Таким образом, чем сильнее уменьшается угол, тем быстрее растет скорость самолета. На высоте около 8500 *фут* угол уменьшается примерно до 1.2°, в результате чего для нахождения нескольких точек потребовалось по 6 итераций, в то время как



Р и с. 10.17. Зависимость скорости и угла от высоты при тех же начальных условиях, что и для рис. 10.11, за исключением начальной скорости, равной 900 *фут/сек* (практический пример 12).

для ранее найденных точек было достаточно одной итерации. В этом режиме почти горизонтального полета практически вся тяга идет на создание ускорения, скорость быстро возрастает, и в результате возрастает подъемная сила. Возрастающая подъемная сила приводит в конце концов к возрастанию угла. Угол возрастает примерно до высоты 18 000 *фут*, после чего уменьшение скорости, тяги и плотности воздуха приводит к уменьшению подъемной силы.

Решение было вычислено только до высоты 25000 *фут*, но сомнительно, что при этих условиях самолет может забраться существенно выше.

Поведение самолета можно описать следующим образом: единственной причиной того, что самолет смог достигнуть высоты 25000 *фут*, является тот факт, что самолет почти в самом начале перешел в горизонтальный полет и находился в этом полете достаточно долгое время, накопив скорость. Возникает предположение, что если бы начальная скорость была немного больше, то самолет не перешел бы в горизонтальный полет и в результате не смог бы достигнуть высоты 25000 *фут*. Производя вычисления по программе с теми же данными, но только при начальной скорости 1000 *фут/сек*, мы убеждаемся, что дело именно так и обстоит. Теперь, вместо того чтобы перейти в горизонтальный полет и быстро набрать скорость, самолет медленно лезет вверх и угол траектории уменьшается понемногу, но стабильно. В результате при высоте 24200 *фут* угол уменьшается до 1.2° и итерационный процесс перестает сходиться. Увеличив допуск до 0.0003 и уменьшив шаг до 5 футов, можно продолжить решение до 24320 *фут*, но потом итерационный процесс опять перестает сходиться. Совершенно очевидно, что решение представляет собой горизонтальный полет или даже снижение самолета.

При начальной скорости 1100 *фут/сек* поведение самолета снова меняется. При $H = 100$ итерационный процесс не сходится для высоты 18400 *фут*, но скорость в это время растет настолько быстро, что возникает желание попытаться продолжить решение с меньшим шагом. Действительно, при решении с шагом 5 *фут* можно определить, что при высоте 18550 *фут* угол уменьшается до 0.087°, а затем снова начинает возрастать. Угол возрастает вплоть до высоты 23000 *фут*, после чего он начинает уменьшаться.

При $V_0 = 1200$ *фут/сек* общая картина поведения самолета такая же, как и при 1000 *фут/сек*: решение нельзя продолжить дальше 21500 *фут*. При $V_0 = 1300$ *фут/сек* и более запас подъемной силы достаточен, чтобы вызвать крутой рост угла с самого начала полета, так что когда впоследствии угол начинает уменьшаться, все же возможно достигнуть высоты 25000 *фут*, не переводя самолет в горизонтальный полет.

Чему нас научил этот практический пример? Надеемся, что часть необходимых выводов уже сделана, а остальные не заставят себя ждать.

1. Многие из самых интересных с практической точки зрения дифференциальных уравнений совершенно невозможно решать классическими методами. Например, эмпирический вид функции тяги двигателя таков, что практически безнадежно искать аналитическое решение в этом случае.

2. Нужно уметь находить некоторый компромисс между чересчур упрощенной и очень усложненной программой. Например, мы могли бы ввести в программу автоматическое удвоение шага интегрирования и автоматическое деление этого шага пополам. Иногда такое усложнение программы было бы необходимо, в нашем же случае это усложнение совершенно не оправдано.

3. Эффективнее и полезнее всего применять ЭЦВМ тогда, когда необходимо варьировать параметры задачи и определять влияние различных условий или их наборов на рабочие характеристики системы. Иногда можно даже запрограммировать ЭЦВМ на поиски оптимальных условий. Эта область прикладной математики в настоящее время бурно развивается [1]).

Упражнения

1. Пусть задано уравнение $y' = 0.04y$ и начальное условие $y(0) = 1000$.

а. Вычислите $y(1)$, используя метод Эйлера с $h = 1$.

б. Вычислите $y(1)$, используя метод Эйлера с $h = 0.5$.

в. Вычислите $y(1)$, используя метод Эйлера с $h = 0.25$.

г. Вычислите $y(1)$, используя решение уравнения $y = 1000\ e^{0.04t}$.

д. Проинтерпретируйте пункты а — г в терминах банковских операций.

***2.** Степень радиоактивности пропорциональна количеству остающегося радиоактивного вещества. Дифференциальное уравнение записывается поэтому в следующем виде: $y' = -ky$, где знак минус отражает тот факт, что радиоактивность убывает с течением времени. Предположим, что $k = 0.01$ и что в начальный момент имеется 100 *г* радиоактивного вещества. Сколько вещества останется в момент $t = 100$?

---

[1]) См., например, Bellman R. E., Dreyfus E., Applied dynamic programming, Princeton University Press, 1962.

Решение этого уравнения имеет вид $y = 100\ e^{-kt}$, точный ответ равен 36.788 *гр*. Найдите решение численно с помощью:

а. Метода Эйлера, $h = 25$.
б. Метода Эйлера, $h = 10$.
в. Метода Эйлера, $h = 5$.
г. Метода Эйлера, $h = 1$.
д. Исправленного метода Эйлера, $h = 20$.
е. Исправленного метода Эйлера, $h = 10$.
ж. Модифицированного метода Эйлера, $h = 20$.
з. Модифицированного метода Эйлера, $h = 10$.
и. Метода Рунге — Кутта четвертого порядка, $h = 100$.
к. Метода Рунге — Кутта четвертого порядка, $h = 50$.
л. Метода прогноза и коррекции второго порядка, описанного в книге, с $h = 20$, причем известно, что $y(20) = 81.8731$ точно.
м. Метода прогноза и коррекции с $h = 10$, если известно, что $y(10) = 90.4837$ точно.

3. Тело с начальной массой 200 *кг* ускоряется постоянной силой в 2000 ньютон. Масса тела уменьшается на 1 *кг/сек*. Если в момент $t = 0$ тело находилось в покое, найдите его скорость через 50 *сек*. Дифференциальное уравнение записывается в виде $dV/dt = 2000/(200 - t)$, решение этого уравнения

$$V = 2000 \log [200/(200 - t)],$$

так что $V(50) = 575.36$. Решите уравнение численно, используя:

а. Метод Эйлера, $h = 10$.
б. Метод Эйлера, $h = 5$.
в. Метод Эйлера, $h = 2$.
д. Модифицированный метод Эйлера, $h = 10$.
е. Метод Рунге — Кутта четвертого порядка, $h = 10$.
ж. Метод прогноза и коррекции, описанный в книге, $h = 10$.

4. Предположите, что тело, описанное в упражнении 3, испытывает сопротивление воздуха, равное удвоенному значению скорости. Дифференциальное уравнение запишется теперь в виде $dV/dt = (2000 - 2V)/(200 - t)$, и если тело покоится при $t = 0$, то решение равно $V = 10t - t^2/40$, так что $V(50) = 437.5$. Решите уравнение численно, используя:

а. Метод Эйлера, $h = 10$.
б. Исправленный метод Эйлера, $h = 10$,
в. Модифицированный метод Эйлера, $h = 10$.
г. Метод Рунге — Кутта четвертого порядка, $h = 10$.
д. Метод прогноза и коррекции, описанный в книге, $h = 10$.

5. Дано уравнение $y' = 4 - 2x$ и начальное условие $y(0) = 2$. Решите это уравнение, используя исправленный метод Эйлера и модифицированный метод Эйлера, принимая $h = 0.5$ и продолжая решение вплоть до $x = 5$. Сравните результат с решением $y = -x^2 + 4x + 2$. Можно ли ожидать, что метод Эйлера также даст точный результат? Для какого типа уравнения можно получить точный результат с помощью метода Эйлера?

6. Дано уравнение $y' = -x/y$ и начальное условие $y(0) = 20$. Вычислите $y(24)$, используя:

а. Модифицированный метод Эйлера, $h = 2$.

б. Метод Рунге — Кутта четвертого порядка, $h = 4$. Что при этом происходит?

7. Дано уравнение $y' = (6y + 2x + 1)/x$ и начальное условие $y(1) = -17/30$. Вычислите $y(4)$, используя описанный в книге метод прогноза и коррекции с $h = 1$. Произведите те же вычисления, взяв в качестве начального условия $y(1) = -16/30$.

8. Дано уравнение $y' = xy^2 + 3xy$ и начальное условие $y(0) = -0.5$. Вычислите $y(3)$, используя описанный в книге метод прогноза и коррекции с $h = 0.5$.

9. Дано уравнение $y' = \cos x - \sin x - y$ и начальное условие $y(0) = 2$. Вычислите решение уравнения вплоть до $x = 10$, используя описанный в книге метод прогноза и коррекции с $h = 0.5$. Покажите, что полученное решение обладает абсолютной устойчивостью. Аналитически решение уравнения можно записать в виде $y = \cos x + e^{-x}$.

10. Дано уравнение $y' = 2y/x + x^2e^x$ и начальное условие $y(1) = 0$. Вычислите решение уравнения вплоть до $x = 5$, используя описанный в книге метод прогноза и коррекции с $h = 0{,}2$. Покажите, что хотя абсолютная ошибка и возрастает, относительная ошибка не увеличивается. Аналитическое решение уравнения равно $y = x^2(e^x - e)$.

***11.** Начертите блок-схемы и напишите программы для решения дифференциальных уравнений с помощью: метода Эйлера, исправленного метода Эйлера, модифицированного метода Эйлера, метода Рунге — Кутта четвертого порядка, метода прогноза и коррекции, описанного в книге. В каждой программе первым оператором должен быть арифметический оператор-функция, определяющий дифференциальное уравнение; этот оператор можно заменять при решении различных уравнений. В каждой программе необходимо предусмотреть ввод с перфокарт исходных значений x_0 и y_0 (начальных условий), h (величины шага) и XLAST (конечного значения x). В программе для решения уравнений методом прогноза и коррекции необходимо предусмотреть нахождение первой точки решения с помощью метода Рунге — Кутта четвертого порядка. Изменения шага интегрирования в программе предусматривать не следует, но требуется ввести счетчик количества итераций для остановки вычислений, если процесс не сошелся на протяжении 10 итераций.

12. Усовершенствуйте программу упражнения 11 для метода прогноза и коррекции таким образом, чтобы при вычислении каждой очередной точки решения печаталась оценка ошибки ограничения.

13. Усовершенствуйте программу упражнения 11 для метода прогноза и коррекции таким образом, чтобы величина шага интегрирования автоматически увеличивалась или уменьшалась вдвое в зависимости от счетчика количества итераций.

14. Покажите, что общая формула для метода Рунге — Кутта (10.22), примененная к уравнению $y' = -y$ с начальным условием $y(0) = 1$, приводит к соотношению

$$y_m = \left(1 - h + \frac{h^2}{2}\right)^m \approx e^{-x_m}.$$

Можно ли утверждать, что ошибка при использовании этого приближения не изменяет знака?

15. Для уравнения $y' = ky$ можно преобразовать формулу коррекции (10.34) таким образом, что она станет явной относительно y_{m+1}. Покажите, что найденное таким образом выражение для y_{m+1} согласуется с рядом Тейлора вплоть до членов порядка h^2 включительно.

16. Рассмотрим следующую совокупность прогноза и коррекции:

$$y^{(0)}_{m+1} = y_m + \frac{h}{2}[y'_m + f(x_m + h,\ y_m + hy'_m)]$$

и

$$y^{(i+1)}_{m+1} = y_m + \frac{h}{2}[y'_m + f(x_{m+1},\ y^{(i)}_{m+1})],$$

где $y'_m = f(x_m, y_m)$.

а. Как называется этот прогноз?

б. Каковы порядки этих двух формул?

в. Каково преимущество этого прогноза перед (10.33)?

г. Каков недостаток этого прогноза по сравнению с (10.33)?

***17.** Сведите уравнение $y'' + y = F(x)$ к системе двух уравнений первого порядка. Покажите, что правая часть каждого уравнения не зависит от дифференцируемой переменной в левой части. Означает ли это, что при использовании формулы коррекции не потребуется производить итераций?

18. Сведите уравнение $y''' + a(x, y)\, y'' + b(x, y)\, y' = c(x, y)$ к системе трех уравнений первого порядка и опишите численный метод ее решения.

19. Если в качестве прогноза использовать метод Эйлера

$$y^0_{m+1} = y_m + hf(x_m, y_m),$$

то подходящей коррекцией при этом будет, например,

$$y^{(i)}_{m+1} = y_m - hf(x_{m+1}, y^{(i-1)}_{m+1}).$$

а. Покажите, что ошибка ограничения прогноза имеет вид

$$E^n_T = \frac{h^2}{2} y''(\xi),\ x_m \leqslant \xi \leqslant x_{m+1},$$

а ошибка ограничения коррекции

$$E^k_T = -\frac{h^2}{2} y''(\eta),\ x_m \leqslant \eta \leqslant x_{m+1}.$$

Предполагая, что y'' практически постоянна, найдите оценку E^k_T, используя значения $y^{(0)}_{m+1}$ и $y^{(i)}_{m+1}$. Выведите формулу для окончательной поправки, аналогичную (10.52).

б. Покажите, что итерационный процесс сходится, если

$$h < \frac{1}{M},$$

где

$$\left|\frac{\partial f}{\partial y}\right| < M.$$

в. Покажите, что метод обладает абсолютной и относительной устойчивостью при $df/dy < 0$, но не обладает ни той, ни другой при $df/dy > 0$.

20. Рассмотрим уравнение второго порядка $y'' + ay' + by = c$, где a, b и c являются функциями от x. Предположим, что начальные условия заданы в виде $y(0) = K_0$ и $y(1) = K_1$. Предложите численный метод решения для этого случая и вычислите $y(x)$ для $0 < x \leqslant 1$. Указание: задайте произвольно значение $y(h)$ и проверьте, не приведет ли оно к $y(1) = K_1$. Если нет, измените его в нужную сторону.

21. Покажите, что разностное уравнение

$$\varepsilon_{n+1} + 2a\varepsilon_n + \varepsilon_{n-1} = 0$$

имеет решение

$$\varepsilon_n = c_1\lambda_1^n + c_2\lambda_2^n,$$

где

$$\lambda_1 = -a + \sqrt{a^2 - 1};$$

$$\lambda_2 = -a - \sqrt{a^2 - 1},$$

а c_1 и c_2 — произвольные постоянные. Используя этот результат, покажите, что прогноз (10.33) не обладает устойчивостью.

* **22.** Рассмотрим общую формулу

$$y_{m+1} = A_0 y_m + A_1 y_{m-1} + hB_0 y'_m,$$

которую называют обычно прогнозом типа Милна [1]) и которая характеризуется тем, что в правой части членов с y на один больше, чем с y'.

а. Выпишите уравнения, которым удовлетворяют A_0, A_1 и B_0, если прогноз должен давать точные результаты для $y = 1$, $y = x$ и $y = x^2$.

б. Покажите, что решение этих уравнений приводит к формуле прогноза (10.33).

в. Применив формулу прогноза к функции $y = x^3$, покажите, что ошибка ограничения имеет порядок h^3.

23. Рассмотрим формулу

$$y_{m+1} + A_0 y_m + h(B_0 y'_m + B_1 y'_{m-1}).$$

Она обычно называется прогнозом типа Адамса — Башфорта [2]) и характеризуется тем, что в правой части членов с y' на один больше, чем с y.

---

[1]) См. разд. 15.6 книги Хемминга (сноска 2 на стр. 427).

[2]) См. разд. 15.7 книги Хемминга.

а. Выпишите уравнения, которым удовлетворяют A_0, B_1 и B_0, если прогноз должен давать точные результаты для $y = 1$, $y = x$ и $y = x^2$.
б. Покажите, что в результате решения этих уравнений получается следующая формула:

$$y_{m+1} = y_m + \frac{h}{2}(3y'_m - y'_{m-1}).$$

в. Применяя формулу прогноза к функции $y = x^3$, покажите, что ошибка ограничения имеет порядок h^3. Таким образом, этот прогноз можно использовать совместно с коррекцией (10.34).

24. Рассмотрим формулу коррекции

$$y_{m+1} = a_0 y_m + h(b_{-1}y'_{m+1} + b_0 y'_m).$$

а. Почему эта формула является формулой коррекции, а не прогноза?
б. Выпишите уравнения, которым удовлетворяют a_0, b_{-1} и b_0, если коррекция должна давать точные результаты для $y = 1$, $y = x$ и $y = x^2$.
в. Покажите, что в результате решения уравнений получается формула коррекции (10.34).
г. Применяя эту формулу к функции $y = x^3$, покажите, что ошибка ограничения имеет порядок h^3.

***25.** Рассмотрим формулу прогноза

$$y_{m+1} = A_0 y_m + A_1 y_{m-1} + A_2 y_{m-2} + h(B_0 y'_m + B_1 y'_{m-1}).$$

а. Относится ли она к типу Милна или Адамса — Башфорта?
б. Выпишите уравнения, которым удовлетворяют A_0, A_1, A_2, B_0 и B_1, если формула должна давать точные результаты для $y = 1$, $y = x$, $y = x^2$, $y = x^3$ и $y = x^4$.
в. Покажите, что решение этих уравнений приводит к следующей формуле прогноза:

$$y_{m+1} = -9y_m + 9y_{m-1} + y_{m-2} + 6h(y'_m + y'_{m-1}).$$

г. Какой степени h пропорциональна ошибка ограничения?

26. Рассмотрим формулу прогноза

$$y_{m+1} = A_0 y_m + A_1 y_{m-1} + h(B_0 y'_m + B_1 y'_{m-1} + B_2 y'_{m-2}).$$

а. Относится ли она к типу Милна или Адамса — Башфорта?
б. Выпишите уравнения, которым удовлетворяют A_0, A_1, B_0, B_1, B_2, если формула должна давать точные результаты для $y = 1$, $y = x$, $y = x^2$, $y = x^3$ и $y = x^4$.
в. Решите эти уравнения относительно A_0, A_1, B_0, B_1 и B_2.
г. Применяя эту формулу к функции $y = x^5$, покажите, что ошибка ограничения имеет порядок h^5.

* 27. Рассмотрим коррекцию

$$y_{m+1} = a_0 y_m + a_1 y_{m-1} + h\,(b_{-1} y'_{m+1} + b_0 y'_m + b_1 y'_{m-1}).$$

а. Является эта формула явной или неявной?

б. Выпишите уравнения, которым удовлетворяют a_0, a_1, b_{-1} b_0, b_1, если формула должна давать точные результаты для $y = 1$, $y = x$, $y = x^2$, $y = x^3$ и $y = x^4$.

в. Почему нельзя решить такую систему уравнений относительно перечисленных неизвестных?

г. Решите четыре первых уравнения относительно a_0, b_{-1}, b_0 и b_1, выражая неизвестные через a_1.

д. Покажите, что ошибка ограничения имеет порядок h^4, если $a_1 \neq 1$.

е. Покажите, что ошибка ограничения имеет порядок h^5, если $a_1 = 1$.

Замечание. Несмотря на полученный результат, значение $a_1 = 1$ не является оптимальным, хотя при таком выборе ошибка ограничения и минимальна. Дело в том, что этот выбор a_1 в сильной мере влияет на устойчивость метода. Для некоторых дифференциальных уравнений приходится выбирать $a \neq 1$ с целью обеспечить необходимую устойчивость.

ГЛАВА

11

УРАВНЕНИЯ В ЧАСТНЫХ ПРОИЗВОДНЫХ

11.1. ВВЕДЕНИЕ И НЕКОТОРЫЕ ОПРЕДЕЛЕНИЯ

Дифференциальные уравнения в частных производных составляют в настоящее время одну из наиболее быстро развивающихся отраслей численного анализа. Области науки и техники, где рассматриваются уравнения в частных производных, весьма многочисленны и важны; к ним относятся, например, ядерная физика, аэродинамика и т. п. Кроме того, возможности современных ЭЦВМ позволяют ставить на повестку дня также и задачи, решение которых просто немыслимо без использования вычислительных машин.

В данной книге, конечно, невозможно дать полное описание предмета. Однако мы рассмотрим здесь общий подход к решению уравнения в частных производных и проанализируем также некоторые численные методы, которые могут оказаться полезными во многих случаях[1]).

Наше рассмотрение будет ограничено линейными дифференциальными уравнениями второго порядка с двумя независимыми переменными. Эти уравнения можно записать в следующем виде:

$$Au_{xx} + Bu_{xy} + Cu_{yy} + Du_x + Eu_y + Fu = G, \qquad (11.1)$$

где A, B, C, D, E, F, G являются функциями только от независимых переменных x и y. Зависимой переменной

---

[1]) Для серьезного изучения мы рекомендуем превосходную книгу Forsythe G. E., Wasow W. R., Finite difference methods for partial differential equations, Wiley, New York, 1960. (Русский перевод: Вазов В., Форсайт Дж., Разностные методы решения дифференциальных уравнений в частных производных, ИЛ, 1963.)

является u, индексы при u означают частные производные, например

$$u_{xy} = \frac{\partial^2 u}{\partial x\, \partial y}.$$

Вспомним, что обыкновенное дифференциальное уравнение (10.1) имело целое семейство решений, причем нужное решение выбиралось с помощью начального условия (10.2). Точно так же в случае уравнения в частных производных ему должна сопутствовать некоторая дополнительная информация, позволяющая выбрать определенное решение. Но теперь, поскольку у нас имеются две независимые переменные, условие выбора одного конкретного решения должно задаваться вдоль какой-то кривой в плоскости xy. Это условие может быть наложено на функцию u, на ее производную или на функцию и на ее производную совместно. В некоторых случаях кривая, вдоль которой задано условие, будет замкнутой, в других случаях незамкнутой; это зависит от типа уравнения.

Мы будем подразделять уравнения на три типа (имеются в виду уравнения второго порядка):

1. Уравнение называется *эллиптическим*, если $B^2 - 4AC < 0$.

2. Уравнение называется *параболическим*, если $B^2 - 4AC = 0$.

3. Уравнение называется *гиперболическим*, если $B^2 - 4AC > 0$.

Уравнение может принадлежать к нескольким типам в зависимости от значений коэффициентов. Уравнение

$$yu_{xx} + u_{yy} = 0$$

является эллиптическим при $y > 0$, параболическим при $y = 0$ и гиперболическим при $y < 0$. Те уравнения, которые мы будем рассматривать в этой главе, имеют постоянные коэффициенты и поэтому принадлежат каждое к одному какому-нибудь типу. Такие уравнения часто встречаются на практике.

В следующих разделах мы рассмотрим примеры каждого из трех типов уравнений с соответствующими граничными условиями и разработаем для них практические методы решения.

11.2. РАЗНОСТНЫЕ УРАВНЕНИЯ

Классическое определение производной функций одной переменной записывается в виде

$$\frac{dy}{dx} = \lim_{h \to 0} \frac{y(x+h) - y(x)}{h}.$$

Естественно, в ЭЦВМ мы не можем произвести предельного перехода. С другой стороны, мы можем придать h некоторое малое, хотя и ненулевое значение и попытаться проверить, что приближение получается достаточно точным (проблема точности) и что ошибка не возрастает в ходе процесса вычислений (проблема устойчивости).

Этот метод сводится к тому, что мы *производную заменяем разностью.* Хотя мы и не употребляли этой формулировки, но именно таким способом мы решали в гл. 10 обыкновенные дифференциальные уравнения. Здесь мы будем пытаться применить тот же самый метод к уравнениям в частных производных, приближенно заменяя производные разностями.

Поскольку теперь имеются две независимые переменные, то обе они должны участвовать в разностном уравнении. Мы, однако, начнем с того, что рассмотрим разности только в направлении x.

Вспомним, что разложение функции $u(x, y_0)$ в ряд Тейлора в окрестности точки x_0, y_0 можно записать в виде

$$u(x, y_0) = u(x_0, y_0) + (x - x_0) u_x(x_0, y_0) + \frac{(x-x_0)^2}{2} u_{xx}(\xi, y_0),$$

где ξ лежит между x и x_0. Если теперь положить $x = x_0 + h$, то после некоторых преобразований получаем

$$u_x(x_0, y_0) - \frac{u(x_0+h, y_0) - u(x_0, y_0)}{h} = -\frac{h}{2} u_{xx}(\xi, y_0).$$

Другими словами, если попытаться приближенно представить u_x с помощью

$$u_x(x_0, y_0) = \frac{u(x_0+h, y_0) - u(x_0, y_0)}{h}, \qquad (11.2)$$

то ошибка ограничения будет равна

$$E_T = -\frac{h}{2} u_{xx}(\xi, y_0); \quad x_0 \leqslant \xi \leqslant x_0 + h.$$

Равенство (11.2) было получено с помощью подстановки в ряд Тейлора $x = x_0 + h$; результат называется *правой разностью*. Можно получить и другое равенство, называемое *левой разностью*, подставляя в ряд Тейлора $x = x_0 - h$. При этом получаем

$$u_x(x_0, y_0) = \frac{u(x_0, y_0) - u(x_0 - h, y_0)}{h}. \qquad (11.3)$$

Нам потребуются впоследствии и правая и левая разности, как мы сейчас убедимся при выводе разностной формулы для u_{xx}. Сначала нам придется написать разностное приближение для u_{xx} через u_x, а затем заменить u_x подходящими разностными приближениями. Напишем приближение для u_{xx}, используя правую разность

$$u_{xx}(x_0, y_0) = \frac{u_x(x_0 + h, y_0) - u_x(x_0, y_0)}{h}. \qquad (11.4)$$

Если в эту формулу подставить теперь правые разности для u_x, то весь окончательный результат окажется как бы «сдвинутым» вправо. Чтобы скомпенсировать этот эффект, используем левые разности для u_x. Левая разность для $u_x(x_0, y_0)$ задается формулой (11.3) и, кроме того,

$$u_x(x_0 + h, y_0) = \frac{u(x_0 + h, y_0) - u(x_0, y_0)}{h} \qquad (11.5)$$

(естественно, это выражение полностью совпадает с правой разностью для $u_x(x_0, y_0)$). Подставляя (11.3) и (11.5) в (11.4), получаем

$$u_{xx}(x_0, y_0) = \frac{u(x_0 + h, y_0) - 2u(x_0, y_0) + u(x_0 - h, y_0)}{h^2}. \qquad (11.6)$$

Это весьма важный результат, которым мы будем ниже неоднократно пользоваться. Отметим симметрию формулы относительно x_0, y_0.

Чтобы определить ошибку ограничения, возникающую при замене производных разностями, вспомним, что

$$u(x, y_0) = u(x_0, y_0) + (x - x_0)\, u_x(x_0, y_0) + \frac{(x - x_0)^2}{2} u_{xx}(x_0, y_0) + \frac{(x - x_0)^3}{6} u_{xxx}(x_0, y_0) + \frac{(x - x_0)^4}{24} u_{xxxx}(\xi, y_0).$$

Теперь положим $x = x_0 + h$; $x = x_0 - h$, а затем сложим два получившихся равенства. При этом получается, что ошибка ограничения равна

$$E_T = -\frac{h^2}{12} u_{xxxx}(\xi, y_0), \quad x_0 - h \leqslant \xi \leqslant x_0 + h.$$

Мы рассматривали только производные в направлении x. Совершенно аналогичный анализ можно провести для производных в направлении y, где величину шага по y мы обозначим через k:

$$u_{yy}(x_0, y_0) = \frac{u(x_0, y_0 - k) - 2u(x_0, y_0) + u(x_0, y_0 + k)}{k^2}. \qquad (11.7)$$

Ошибка ограничения равна

$$E_T = -\frac{k^2}{12} u_{yyyy}(x_0, \eta), \quad y_0 - k \leqslant \eta \leqslant y_0 + k.$$

Теперь мы имеем разностные выражения для u_x, u_{xx} и u_{yy}. Разностное выражение для u_y полностью аналогично (11.2), а вывод выражения для u_{xy} мы предоставляем читателям (см. упражнение 5). Используя эти выражения, можно полностью переписать дифференциальное уравнение в частных производных, получив из него уравнение в конечных разностях. Например, общеизвестное уравнение Лапласа

$$u_{xx} + u_{yy} = 0$$

можно переписать в виде

$$\frac{u(x_0 + h, y_0) - 2u(x_0, y_0) + u(x_0 - h, y_0)}{h^2} + \\ + \frac{u(x_0, y_0 - k) - 2u(x_0, y_0) + u(x_0, y_0 + k)}{k^2} = 0.$$

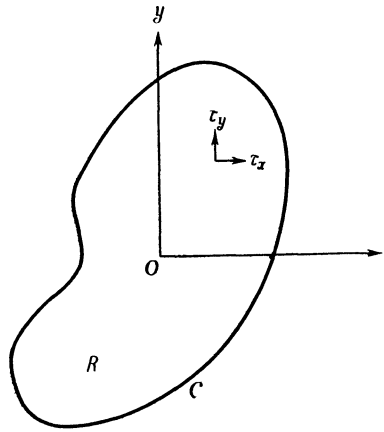
В двух следующих разделах мы рассмотрим методы численного решения этого разностного уравнения.

11.3. ЭЛЛИПТИЧЕСКИЕ УРАВНЕНИЯ

Типичным примером задачи, приводящей к эллиптическому уравнению, является, например, расчет напряжений, возникающих при упругом кручении длинного цилиндрического стержня. Сечение стержня может быть любой формы, например, как на рис. 11.1. Мы будем считать, что

сечение стержня ограничено кривой C; область, ограниченную этой кривой, мы будем обозначать через R.

Предположим, что ось z параллельна оси цилиндра и проходит через центр тяжести сечения O. Таким образом, сечение цилиндра лежит в плоскости xy. Наконец, обозначим угол кручения на единицу длины через θ, т. е. угол



Р и с. 11.1. Сечение цилиндра в задаче об упругом кручении.

поворота плоскости $z = z_0$ относительно плоскости $z = 0$ будет равен $z_0 \cdot \theta$.

Единственными ненулевыми напряжениями в этой задаче являются напряжения сдвига τ_x и τ_y в направлениях осей x и y в плоскости xy. (Индексы здесь *не означают* дифференцирования.) Если определить функцию ψ с помощью соотношений

$$\tau_x = \frac{E\theta}{2(1+\nu)} \frac{\partial \psi}{\partial y}, \qquad \tau_y = -\frac{E\theta}{2(1+\nu)} \frac{\partial \psi}{\partial x}, \tag{11.8}$$

где E — модуль Юнга и ν — коэффициент Пуассона для материала стержня, то функция ψ является решением уравнения

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = -2 \tag{11.9}$$

внутри области R, а на границе C этой области

$$\psi = \text{const}.$$

Поскольку интересующие нас физические величины зависят только от производных ψ, то величина константы не играет в данном случае никакой роли. Обычно принимают

$$\psi = 0 \tag{11.10}$$

на кривой C.

Уравнение (11.9) называется *уравнением Пуассона*. Его часто записывают в виде

$$\Delta\psi = -2$$

или

$$\nabla^2\psi = -2.$$

Многие физические задачи приводят к уравнению Пуассона. Распределение потенциалов (или электрических напряжений) на проводящей плоскости при задании потенциала на границе также удовлетворяет уравнению Пуассона. Наконец, задача о стационарных потоках тепла в двумерном теле также сводится к уравнению Пуассона, как мы убедимся при рассмотрении практического примера 13 разд. 11.9.

После такого краткого объяснения происхождения эллиптических уравнений обратимся к выводу разностных схем их решения. Сам процесс решения будет рассмотрен в следующем разделе.

Рассмотрим классическую задачу Дирихле

$$\frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2} = 0 \tag{11.11}$$

в некоторой области R и

$$\psi = f(x, y) \tag{11.12}$$

на границе этой области, которой является кривая C. Уравнение (11.11), как уже об этом говорилось ранее, представляет собой *уравнение Лапласа*, частный случай уравнения Пуассона.

Ради простоты будем предполагать, что кривая C представляет собой отрезки прямых, параллельных осям x и y.

Конечно, можно было бы и не ограничиваться таким узко специальным случаем и попытаться разработать программы для кривых более сложной формы, но, так как мы будем рассматривать только основы методов решения, такой большой общности нам не потребуется.

В частности, мы будем рассматривать прямоугольник ширины A и высоты B. Мы убедимся далее, что обобщение для случая любой области, ограниченной линиями, параллельными осям x и y, достаточно просто (в частности, именно такой случай рассматривается в практическом примере), хотя и возникают некоторые затруднения в смысле программирования.

Разделим сначала ширину прямоугольника A на n интервалов, каждый размером $h = A/n$; точно таким же образом разделим высоту B на m частей размерами $k = B/m$. Внутри области получаются при этом $(n - 1)(m - 1)$ пересечений (углов) сетки. Мы запишем разностное соотношение для каждой внутренней точки и решим после этого получившуюся систему уравнений.

Прежде всего договоримся о том, в каком порядке нумеровать точки пересечения. Мы начинаем нумерацию в горизонтальном направлении слева направо от нуля, крайняя правая точка будет при этом n-й. Аналогично в вертикальном направлении мы нумеруем точки снизу вверх от нуля до m. Узел с индексами i, j будет i-м слева и j-м снизу. Например, на рис. 11.2 узел P имеет индексы 3,2, узел Q — индексы 2,8.

Пусть начало координат совпадает с точкой 0,0. Обозначим

$$u(ih,\ jk) = u_{i,\,j}.$$

Аналогично запишем

$$f(ih,\ jk) = f_{i,\,j}.$$

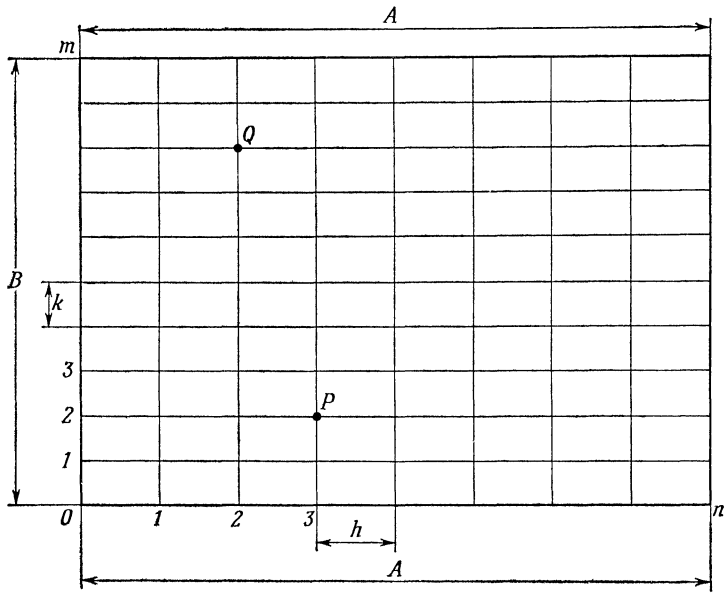
При этой системе обозначений граничное условие (11.12) можно записать в виде

$$\begin{aligned}
&u_{i,\,0} = f_{i,\,0}; \quad i = 0,\ 1,\ 2,\ \ldots,\ n,\\
&u_{i,\,m} = f_{i,\,m}; \quad i = 0,\ 1,\ 2,\ \ldots,\ n,\\
&u_{0,\,j} = f_{0,\,j}; \quad j = 0,\ 1,\ 2,\ \ldots,\ m,\\
&u_{n,\,j} = f_{n,\,j}; \quad j = 0,\ 1,\ 2,\ \ldots,\ m.
\end{aligned} \tag{11.13}$$

Пусть теперь точка i, j будет точкой x_0, y_0 в (11.6) и (11.7). Если теперь обозначить $\lambda = k/h$, то дифференциальное уравнение (11.11) сведется к разностному уравнению вида

$$\lambda^2 u_{i+1,\,j} + \lambda^2 u_{i-1,\,j} + u_{i,\,j+1} + u_{i\;\,j-1} - \\ -2(1+\lambda^2)\,u_{i,\,j} = 0 \qquad (11.14)$$

для $i = 1, 2, 3, \ldots, n-1$ и $j = 1, 2, 3, \ldots, m-1$.

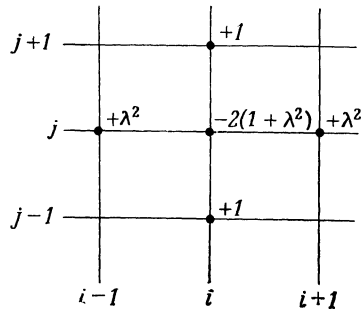


Р и с. 11.2. Построение сетки в прямоугольной области.

При $\lambda = 1$, т. е. при одинаковых величинах интервалов разбиения в горизонтальном и вертикальном направлениях, это соотношение означает, что значение $u_{i,j}$ является средним арифметическим из четырех соседних с ним.

Важно то, что мы теперь получили систему линейных алгебраических уравнений. Всего имеется $(m-1)(n-1)$ уравнений относительно $(m+1)(n+1)$ неизвестных. После того как $2(m+n)$ неизвестных будут исключены с помощью граничного условия (11.13), остается точно

$(m-1)(n-1)$ уравнений относительно $(m-1)(n-1)$ неизвестных. Методы решения таких систем с помощью ЭЦВМ мы рассматривали в гл. 8. Однако перед тем, как браться за эту систему уравнений, посмотрим на нее внимательнее и постараемся найти некоторые ее свойства, которые могли бы облегчить процесс вычисления решения. Этим мы займемся в следующем разделе.



Заметим, что уравнение (11.14) можно представить схематически, начертив пять узлов разностного уравнения (11.14) и обозначив около каждого из них соответствующий коэффициент. Этот рисунок называется трафаретом[1]). Трафарет геометрически иллюстрирует разностную аппроксимацию дифференциального уравнения.

Заметим, наконец, что мы здесь обходим один очень важный вопрос, решение которого выходит за рамки настоящей книги. Ранее мы указывали, что при $h \to 0$ и $k \to 0$ разностное уравнение приближается к дифференциальному уравнению. Однако нас интересует в основном совсем другой вопрос: приближается ли при $h \to 0$ и $k \to 0$ решение разностного уравнения к решению дифференциального уравнения? Для эллиптических уравнений на этот вопрос можно дать положительный ответ, хотя мы и не будем воспроизводить здесь доказательство. Ниже мы, однако, увидим, что в случае параболических и гиперболических уравнений придется соблюдать некоторые ограничения, которые обеспечили бы такого рода сходимость.

---

[1]) В оригинале «stencil».— *Прим. перев.*

11.4. РЕШЕНИЕ ЭЛЛИПТИЧЕСКОГО РАЗНОСТНОГО УРАВНЕНИЯ

Запишем подробно некоторые из уравнений (11.14). Для удобства примем $\lambda = 1$, т. е. $h = k$, но общность выводов нетрудно проверить при любом $\lambda > 0$. Начнем с $i = 1$, $j = 1$ и при неизменном j пройдем значения $i = 1, 2, \ldots, i = 1, 2, \ldots, n - 1$:

$$
\begin{aligned}
4u_{1,1} - u_{2,1} - u_{1,2} &= f_{1,0} + f_{0,1};\\
-u_{1,1} + 4u_{2,1} - u_{3,1} - u_{2,2} &= f_{2,0};\\
-u_{2,1} + 4u_{3,1} - u_{4,1} - u_{3,2} &= f_{3,0};\\
&\ldots\ldots\ldots\ldots\ldots\ldots\\
-u_{n-2,1} + 4u_{n-1,1} - u_{n-1,2} &= f_{n-1,0} + f_{n,1}.
\end{aligned}
$$

Теперь увеличим j до 2 и снова пройдем значения $i = 1, 2, \ldots, n - 1$:

$$
\begin{aligned}
-u_{1,1} + 4u_{1,2} - u_{2,2} - u_{1,3} &= f_{0,2};\\
-u_{2,1} - u_{1,2} + 4u_{2,2} - u_{3,2} - u_{2,3} &= 0;\\
-u_{3,1} - u_{2,2} + 4u_{3,2} - u_{4,2} - u_{3,3} &= 0;\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
-u_{n-2,1} - u_{n-3,2} + 4u_{n-2,2} - u_{n-1,2} - u_{n-2,3} &= 0;\\
-u_{n-1,1} - u_{n-2,2} + 4u_{n-1,2} - u_{n-1,3} &= f_{n,2}.
\end{aligned}
$$

Продолжим и дальше таким же образом, увеличивая каждый раз j и проходя значения $i = 1, 2, \ldots, n - 1$. Конечное значение j будет равно $m - 1$.

Система обладает двумя важными свойствами, которые помогают выбрать для нее метод решения. Свойства эти следующие:

1. Подавляющая часть коэффициентов системы равна нулю.

2. В каждом уравнении один из коэффициентов равен $+4$. Если в уравнении имеется пять коэффициентов, отличных от нуля, то сумма остальных четырех коэффициентов равна -4, если же количество ненулевых коэффициентов меньше пяти, то сумма остальных равна -2 или -3.

Таким образом, в этой системе выполнены условия сходимости итерационного метода Гаусса — Зейделя (на основе ее второго свойства). Первое же свойство системы делает решение методом исключения весьма непривлекательным: исходная система с большим количеством равных нулю коэффициентов превратится после исключения неизвестных в полную треугольную систему. Учитывая, что m и n часто бывают довольно большими (несколько сотен), решение системы уравнений с помощью метода исключения нельзя признать целесообразным.

Поучительно написать некоторые уравнения в том виде, в котором с ними будут производиться итерации. Обозначая верхними индексами порядковый номер итерации и полагая, как обычно, $u_{i,j}^{(0)} = 0$ для всех i, j, получим следующий порядок решения системы уравнений:

$$u_{1,1}^{(1)} = \frac{1}{4}[f_{1,0} + f_{0,1} + u_{2,1}^{(0)} + u_{1,2}^{(0)}],$$

$$u_{2,1}^{(1)} = \frac{1}{4}[f_{2,0} + u_{1,1}^{(1)} + u_{3,1}^{(0)} + u_{2,2}^{(0)}],$$

$$u_{3,1}^{(1)} = \frac{1}{4}[f_{3,0} + u_{2,1}^{(1)} + u_{4,1}^{(0)} + u_{3,2}^{(0)}].$$

. . . . . .

$$u_{n-1,1}^{(1)} = \frac{1}{4}[f_{n-1,0} + f_{n,1} + u_{n-2,1}^{(1)} + u_{n-1,2}^{(0)}],$$

$$u_{1,2}^{(1)} = \frac{1}{4}[f_{0,2} + u_{1,1}^{(1)} + u_{2,2}^{(0)} + u_{1,3}^{(0)}],$$

$$u_{2,2}^{(1)} = \frac{1}{4}[u_{2,1}^{(1)} + u_{1,2}^{(1)} + u_{3,2}^{(0)} + u_{2,3}^{(0)}],$$

. . . . . .

Хотя вид у этой таблицы получился весьма устрашающий, но в действительности все вычисления очень легко запрограммировать для ЭЦВМ. При программировании возникает только одно небольшое затруднение, да и то скорее даже не затруднение, а неудобство. Дело в том, что в ФОРТРАНе не допускаются индексы, равные нулю. В этой главе мы используем общепринятые математические обозначения, но вообще вполне можно было бы начинать нумерацию с 1, и тогда затруднение отпало бы. Можно сделать и по-другому — написать все формулы в общепри-

нятом математическом виде, а затем при составлении программы сделать несложные изменения в индексации. Так мы и поступим.

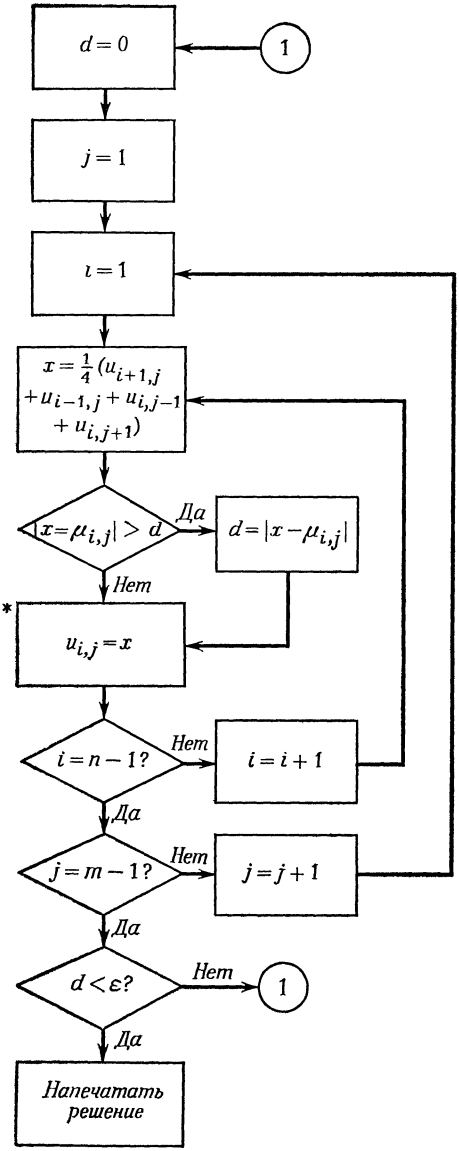
Основная схема решения системы уравнений следующая. Введем в программу двумерный массив, состоящий из $m + 1$ строк и $n + 1$ столбцов. Строки нуль и m, а также столбцы нуль и n представляют собой граничные условия, т. е. они задаются перед решением системы уравнений. Иногда эта исходная информация вводится с перфокарт, в других случаях граничные условия могут вычисляться по каким-либо формулам. После этого начинаются собственно вычисления; схема расчетов содержит два цикла DO: один из этих циклов имеет в качестве индексного параметра j, и значение j изменяется от 1 до m — 1, другой цикл имеет в качестве индексного параметра i, и значение i изменяется от 1 до n — 1. Каждая пара значений i, j определяет узел, в котором уравнение (11.14) решается относительно $u_{i,\,j}$, и только что вычисленное значение сравнивается с тем, которое было вычислено в результате предыдущей итерации. Наибольшая разность значений неизвестных, полученных при последовательных итерациях, сравнивается с допуском ε, чтобы определить, сошелся процесс или нет. Если процесс не сошелся, то производится очередная итерация.

Все эти вычисления, кажется, проще сделать, нежели описать. Весь процесс решения эллиптических разностных уравнений изображен на рис. 11.3, где приведена блок-схема программы. Программа для решения эллиптического уравнения при несколько более сложной форме границы имеется в практическом примере 13 в разд. 11.9.

Метод Гаусса — Зейделя в применении к эллиптическим разностным уравнениям называется *методом Либмана* или *методом последовательных смещений*.

Вспомним теперь (гл. 5), что бывает полезно увеличить или уменьшить очередную поправку при расчете очередного приближения к корню уравнения. Мы также отмечали в разд. 8.6, что аналогичный метод может оказаться полезным при решении систем линейных алгебраических уравнений. В действительности это справедливо и для метода Либмана.

Чтобы можно было увеличивать или уменьшать очередные поправки, изменим на блок-схеме содержание блока,



Р и с. 11.3. Блок-схема программы для решения эллиптических разностных уравнений методом Либмана.

Прямоугольник, отмеченный звездочкой, можно видоизменить для того, чтобы получить экстраполированный метод Либмана. Предполагается, что граничные значения и допуск ε вычислены с помощью отдельного участка программы.

отмеченного звездочкой, следующим образом:

$$u_{i,j} = \omega x + (1 - \omega) u_{i,j},$$

где x — значение $u_{i,j}$, вычисленное из уравнения (11.14). Параметр ω называется *ускоряющим множителем*. В общем случае мы будем требовать, чтобы значение ω лежало в пределах от 1 до 2. Если $\omega = 1$, то мы имеем обычный метод Либмана, при $\omega > 1$ мы имеем *ускоренный метод Либмана*. При правильном выборе ω можно получить потрясающую экономию машинного времени — до 20-кратной! В практическом примере из разд. 11.9 приведен график количества итераций в зависимости от ω, необходимых для решения некоторой конкретной системы. Для случая, когда область вычислений близка к квадрату и числа m и n велики, оптимальным значением ω будет приблизительно 1.9.

Мы рассматривали метод Либмана пока что только для случая уравнения Лапласа. Вообще говоря, любое эллиптическое уравнение без членов, содержащих u_{xy}, приводит к системе разностных уравнений, удовлетворящей условиям сходимости.

Все сказанное до сих пор относилось к линейным уравнениям. Вопрос о сходимости для нелинейных уравнений разработан весьма слабо. Нужно отметить, однако, что многократно делались успешные попытки решения квазилинейных[1]) уравнений с помощью экстраполированного метода Либмана. Общий подход в этом случае сводится к тому, что задается наугад некоторое значение ω и наблюдается скорость сходимости интерационного процесса. Затем, изменяя ω и наблюдая вызываемые этим изменения скорости сходимости, можно подобрать такое его значение, при котором итерационный процесс сходится быстрее всего. Удавалось получить сходимость даже для уравнений с членом типа u_{xy}, хотя в этом случае нет никаких теоретических оснований ожидать сходимости.

Имеется много других способов решения разностных уравнений (11.14). Наиболее часто используются *линейная релаксация* (см. упражнение 31), *блочная релаксация* и *метод чередующихся направлений* (см. упражнение 32). Часто они оказываются более эффективными, нежели метод Либмана.

[1]) Квазилинейными называются уравнения, линейные по u_{xx}, u_{yy} и x_{xy}, но нелинейные по u_x, u_y, u, x, y.

Наконец, можно использовать экстраполяционный переход к пределу, как мы это делали в других случаях. Если $U_{i,j}$ — точное, а $u_{i,j}$ — приближенное решение, то

$$U_{i,j} = u_{i,j} + K_{i,j}h^2$$

при условии, что четвертая производная практически постоянна. При другой величине интервала p приближенное решение будет равно $v_{i,j}$, так что

$$U_{i,j} = v_{i,j} + K_{i,j}p^2$$

и

$$K_{i,j} = \frac{v_{i,j} - u_{i,j}}{h^2 - p^2}.$$

11.5. ГИПЕРБОЛИЧЕСКИЕ УРАВНЕНИЯ

Как и ранее, изучение этого типа дифференциальных уравнений в частных производных мы начнем с краткого рассмотрения соответствующей физической задачи.

Имеется струна длиной L, натянутая между двумя точками оси x, точкой $x = 0$ и точкой $x = L$. Натяжение струны равно T. Если отклонить струну от положения равновесия и отпустить, то она начнет колебаться. Смещение каждой точки струны относительно положения равновесия зависит не только от координаты x этой точки, но и от времени t. Отклонение струны от положения равновесия описывается уравнением гиперболического типа, которое приводится без вывода

$$u_{xx} - a^2 u_{tt} = 0$$

для $0 \leqslant x \leqslant L$ и $t > 0$. Коэффициент a учитывает физические характеристики

$$a^2 = \frac{w}{Tg}.$$

Здесь w — вес струны на единицу длины, T — натяжение, g — ускорение силы тяжести. Это уравнение обычно называется *волновым уравнением*. Для простоты мы будем принимать $a = 1$, так что наша задача выразится следующим образом:

$$u_{xx} - u_{tt} = 0, \quad 0 \leqslant x \leqslant L, \quad t > 0. \tag{11.15}$$

Такая формулировка задачи вовсе не означает потери общности, так как простая замена переменных сводит любое волновое уравнение к виду (11.15).

Поскольку концы струны закреплены, то имеем

$$u(0,\ t)=u(L,\ t)=0;\quad t\geqslant 0. \tag{11.16}$$

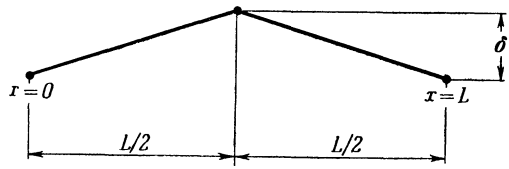
Начальными условиями являются начальное смещение

$$u(x,\ 0)=f(x);\quad 0\leqslant x\leqslant L. \tag{11.17}$$

и начальная скорость

$$u_t(x,\ 0)=g(x);\quad 0<x<L. \tag{11.18}$$

Например, если мы оттянем струну за середину, как показано на рис. 11.4, и отпустим ее без придания ей



Р и с. 11.4. Возможные начальные условия для колеблющейся струны, описываемой волновым уравнением.

начальной скорости, то начальные условия запишутся в виде

$$f(x)=\begin{cases}\dfrac{2\delta}{L}x; & 0\leqslant x\leqslant \dfrac{L}{2},\\[2ex] \dfrac{2\delta}{L}(L-x); & \dfrac{L}{2}\leqslant x\leqslant L,\end{cases}$$

$$g(x)=0;\qquad 0<x<L.$$

Заметим, что мы назвали эти условия *начальными*, а не *граничными*. В действительности если задать для гиперболического уравнения граничные условия, то полученная задача не будет иметь единственного решения. Подобные задачи называются *некорректно поставленными*. Очень важно, чтобы дополнительные условия должным образом соответствовали каждому типу уравнений.

Чтобы найти разностные уравнения, соответствующие (11.15), воспользуемся снова равенствами (11.6) и (11.7),

причем вместо y будем писать t. Снова начертим сетку, но теперь эта сетка простирается бесконечно в направлении положительных значений t; мы можем искать решение для сколь угодно далекого момента времени. В направлении x мы примем шаг сетки равным h, в направлении t — равным k. Поэтому интервал L разделяется на $n = L/h$ малых интервалов h, а в направлении t может быть сколь угодно много интервалов k.

Если обозначить

$$u_{i,j} = u(ih,\ jk)$$

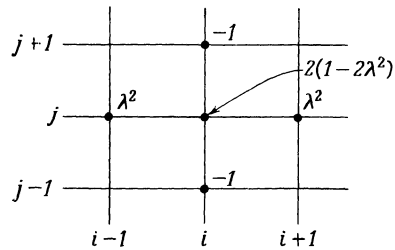
и

$$\lambda = \frac{k}{h},$$

то разностное уравнение запишется в виде

$$u_{i,j+1} = 2(1-\lambda^2)u_{i,j} + \lambda^2(u_{i+1,j} + u_{i-1,j}) - u_{i,j-1} \quad (11.19)$$

для $i = 1, 2, \ldots, n$ и для $j = 1, 2, \ldots$. Трафарет при этом выглядит так:



Граничное условие (11.16) легко записать в виде

$$u_{0,j} = u_{n,j} = 0; \quad j = 1, 2, 3, \ldots. \quad (11.20)$$

Начальное условие (11.17) можно написать в виде

$$u_{i,0} = f(ih); \quad i = 1, 2, \ldots, n-1. \quad (11.21)$$

Чтобы записать в разностном виде начальное условие (11.18), можно использовать равенство (11.2), откуда

$$\frac{u_{i,1} - u_{i,0}}{k} = g(ih).$$

После этого, используя (11.21), получаем

$$u_{i,1} = f(ih) + kg(ih). \quad (11.22)$$

11.6. РЕШЕНИЕ ГИПЕРБОЛИЧЕСКОГО РАЗНОСТНОГО УРАВНЕНИЯ

Заметим теперь, что (11.21) и 11.22) дают значения u для первых двух строк: $j = 0$ и $j = 1$. Подставляя $j = 1$ в (11.19), получим

$$u_{i,2} = 2(1 - \lambda^2) u_{i,1} + \lambda^2 (u_{i+1,1} + u_{i-1,1}) - u_{i,0}.$$

Все слагаемые в правой части этого уравнения включают значения u только из первых двух строк сетки; но ведь все эти значения известны из начальных условий. Поэтому в последнем уравнении имеется только одно неизвестное и все значения функции, соответствующие третьей строке, можно вычислить в явном виде. После этого можно вычислять значения функции в четвертой строке, исходя из значений во второй и в третьей строках, и т. д. столько раз, сколько потребуется. При этом даже не приходится решать систем уравнений.

Таким образом, (11.19) представляет собой *явную* схему решения волнового уравнения. Для сравнения вспомним, что при решении эллиптического уравнения (11.14) представляло собой схему, где в каждом уравнении было более чем по одному неизвестному; таким образом, этот метод можно было назвать *неявным*.

Рассмотрим теперь вопросы сходимости и устойчивости метода. Мы не будем приводить здесь доказательств, а ограничимся только формулировкой окончательных результатов. Можно утверждать, что решение (11.19) сходится к решению (11.15) (имеется в виду, что при $h \to 0$ и $k \to 0$ решение разностного уравнения асимптотически приближается к решению дифференциального уравнения), если

$$\lambda < 1,$$

или, что то же самое,

$$k < h. \tag{11.23}$$

Это условие является достаточным для сходимости; но оно не является необходимым. Другими словами, существуют уравнения и величины интервалов, при которых (11.23) не выполняется, но все же получается правильный результат. Все дело в том, что тогда нельзя гарантировать сходимость. В общем случае, конечно, желательно обеспе-

чить сходимость наверняка, и поэтому мы будем требовать соблюдения условий (11.23).

Таким образом, как только выбрана величина интервала разбиения h в направлении x, то появляется ограничение на величину интервала по времени[1]). Если необходимо произвести вычисления для большого отрезка t, то может потребоваться большое количество шагов по времени.

Другим столь же важным обстоятельством является то, что при $\lambda > 1$ метод становится неустойчивым, как в абсолютном, так и в относительном смысле. Это означает, как и для обыкновенных дифференциальных уравнений, что любые ошибки возрастают в ходе вычисления решения.

Поэтому при решении уравнения (11.15) явными методами условие (11.23) обязательно должно выполняться. Отличительная особенность всех явных методов заключается в том, что при их использовании должно соблюдаться некоторое условие типа (11.23), обеспечивающее сходимость и устойчивость метода.

Существуют также *неявные* методы решения гиперболических уравнений, не подверженные неустойчивости. Неявные методы и соответствующие способы нахождения решений мы обсудим в разд. 11.8, посвященном решению параболических уравнений; все основные идеи этого раздела можно без труда обобщить на случай гиперболических уравнений.

11.7. ПАРАБОЛИЧЕСКИЕ УРАВНЕНИЯ

В качестве примера физической задачи, приводящей к уравнению этого типа, рассмотрим процесс теплопередачи по длинному стержню, лежащему вдоль оси x от $x = 0$ до $x = L$. Предположим, что в точке $x = 0$ температура поддерживается на уровне T_0, а в точке $x = L$ температура поддерживается на уровне T_L. Предположим также, что в момент времени $t = 0$ распределение температуры вдоль стержня задавалось функцией $f(x)$. Тогда распределение температуры вдоль стержня во все последующие моменты времени дается решением уравнения

$$u_{xx} = au_t, \tag{11.24}$$

[1]) Разумеется, мы предполагаем, что все величины измеряются в совместимых единицах.

где u — температура стержня в данной точке в данный момент времени, а постоянная $a = c\rho/k$, где c — теплоемкость материала стержня, ρ — плотность материала стержня и k — его теплопроводность. Для простоты положим $a = 1$, так что уравнение сведется к следующему виду:

$$u_{xx} = u_t. \tag{11.25}$$

Граничными условиями для этого уравнения являются

$$\begin{aligned} u(0, t) &= T_0, \\ u(L, t) &= T_L, \end{aligned} \tag{11.26}$$

и начальным условием является

$$u(x, 0) = f(x). \tag{11.27}$$

Уравнение (11.25) представляет собой параболическое дифференциальное уравнение в частных производных, известное под названием *уравнения теплопередачи* или *уравнения диффузии*.

Многие другие важные задачи также приводят к параболическим уравнениям. Например, уравнения, которыми описываются длинные линии связи, так называемые телеграфные уравнения, также являются параболическими.

При записи в разностной форме граничные условия (11.26) запишутся так:

$$\begin{aligned} u_{0, j} &= T_0; \quad j = 1, 2, \ldots, \\ u_{n, j} &= T_L; \quad j = 1, 2, \ldots . \end{aligned} \tag{11.28}$$

Начальное условие (11.27) запишется в виде

$$u_{i, 0} = f(ih). \tag{11.29}$$

Чтобы преобразовать в разностную форму (11.25), снова представим себе сетку, охватывающую область $0 \leqslant x \leqslant L$ и $t > 0$ с интервалом разбиения h в направлении x и интервалом разбиения k в направлении t. Используя (11.2) и (11.6), получаем

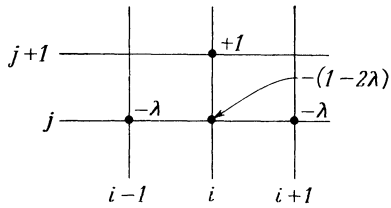
$$u_{i, j+1} = \lambda u_{i+1, j} + (1 - 2\lambda) u_{i, j} + \lambda u_{i-1, j}, \tag{11.30}$$

где

$$\lambda = \frac{k}{h^2}, \tag{11.31}$$

и индекс i изменяется от 1 до $n-1$, а индекс j — от 1 до ∞. Обратите внимание на новое определение λ.

Трафарет этого метода можно представить себе так:



Верхний предел ошибки ограничения выражается соотношением

$$|T_{i,j}| \leqslant \frac{\lambda(6\lambda-1)}{12} Mh^4, \tag{11.32}$$

где

$$|u_{xxxx}| < M$$

при условии $\lambda \neq \frac{1}{6}$, так как тогда это выражение обращается в нуль. При $\lambda = \frac{1}{6}$ верхний предел ошибки ограничения выглядит так:

$$|T_{i,j}| \leqslant \frac{N}{810} h^6, \tag{11.33}$$

где

$$|u_{xxxxxx}| < N.$$

Очевидно, что с точки зрения наименьшей ошибки ограничения целесообразно было бы выбирать

$$k = \frac{1}{6} h^2. \tag{11.34}$$

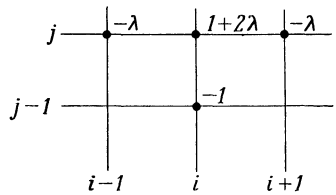
Это не всегда оказывается удобным.

У читателя могло сложиться впечатление, что преобразовать дифференциальное уравнение в разностное можно только единственным способом. Дело, однако, обстоит вовсе не так. Оставшуюся часть этого раздела мы посвятим рассмотрению других способов построения разностных схем для параболического уравнения и укажем на их преимущества и недостатки. Конкретно мы рассмотрим два способа.

Тот метод, который привел к уравнению (11.30), был основан на использовании правых разностей (11.2). Если использовать левые разности (11.3), то разностное уравнение запишется в следующем виде:

$$-\lambda u_{i+1,\,j}+(1+2\lambda)\,u_{i,\,j}-\lambda u_{i-1,\,j}=u_{i,\,j-1}, \qquad (11.35)$$

где λ означает то же, что и в (11.31). Трафарет этого метода выглядит так:



Верхний предел ошибки ограничения дается формулой

$$|T_{i,\,j}|\leqslant\frac{\lambda\,(6\lambda+1)}{12}\,Mh^4, \qquad (11.36)$$

где по-прежнему

$$|u_{xxxx}|<M.$$

Этот предел больше, нежели (11.32); кроме того, нельзя свести ошибку ограничения к $O\,(h^6)$ специальным выбором параметра λ.

Рассмотрим еще один способ построения разностного уравнения, эквивалентного (11.25). Вспомним, что, согласно (11.6), в точке $i,\ j$

$$u_{xx}\approx\frac{u_{i+1,\,j}-2u_{i,\,j}+u_{i-1,\,j}}{h^2}.$$

Аналогично в точке $i,\ j+1$

$$u_{xx}\approx\frac{u_{i+1,\,j+1}-2u_{i,\,j+1}+u_{i-1,\,j+1}}{h^2}.$$

Усредняя эти два приближения, получим

$$u_{xx}\approx\frac{1}{2h^2}\,(u_{i+1,\,j+1}-2u_{i,\,j+1}+u_{i-1,\,j+1}+$$
$$+u_{i+1,\,j}-2u_{i,\,j}+u_{i-1,\,j}).$$

Если теперь для вычисления u_t воспользоваться правыми разностями, то

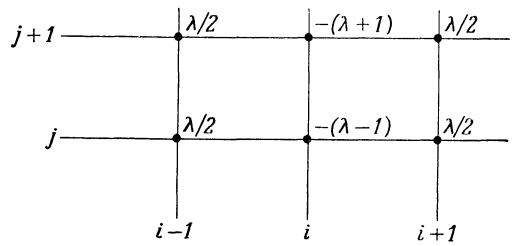
$$u_t \approx \frac{1}{k}(u_{i,\,j+1} - u_{i,\,j}),$$

и разностное уравнение, эквивалентное (11.25), запишется в виде

$$\frac{\lambda}{2} u_{i-1,\,j+1} - (\lambda+1)\,u_{i,\,j+1} + \frac{\lambda}{2} u_{i+1,\,j+1} = \\ = -\frac{\lambda}{2} u_{i-1,\,j} + (\lambda-1)\,u_{i,\,j} - \frac{\lambda}{2} u_{i-1,\,j}. \qquad (11.37)$$

Этот способ построения разностного уравнения часто называют *методом Кранка — Никольсона.*

Трафарет для (11.37) имеет вид



Верхний предел ошибки ограничения дается формулой

$$|T_{i,\,j}| \leqslant \frac{\lambda}{12} Mh^4,$$

где

$$|u_{xxxx}| < M. \qquad (11.38)$$

Этот верхний предел в $6\lambda + 1$ раз меньше, чем (11.36). Поскольку предел ошибки ограничения (11.38) меньше, чем (11.36), то по этой характеристике разностное уравнение (11.37) предпочтительнее, нежели (11.35).

11.8. РЕШЕНИЕ ПАРАБОЛИЧЕСКОГО РАЗНОСТНОГО УРАВНЕНИЯ

В предыдущем разделе мы рассмотрели три различных способа разностного представления дифференциального уравнения $u_{xx} = u_t$, а именно формулы (11.30), (11.35)

и (11.37). Рассмотрим теперь методы решения каждого из этих разностных уравнений и сопоставим их достоинства и недостатки. Во всех случаях будем считать, что граничные и начальные условия заданы в виде (11.28) и (11.29).

Первый способ, определяемый формулой (11.30), представляет собой *явную* систему уравнений для $u_{i,j+1}$ в том же самом смысле, в котором (11.19) являлось явной системой для случая гиперболического уравнения. Имея на основании начальных и граничных условий первую строку решения, мы можем вычислить вторую строку, $j = 1$, непосредственно из (11.30), положив $j = 0$. Вычислив вторую строку, мы можем таким же образом вычислить третью строку, положив на этот раз $j = 1$. Таким же способом можно продолжать вычислять решение столь далеко по оси времени, сколь в этом есть необходимость.

Эта ситуация полностью аналогична той, с которой мы столкнулись в разд. 11.6 при рассмотрении гиперболических уравнений. Поэтому естественно ожидать, что для параболических уравнений должны возникнуть те же вопросы сходимости и устойчивости. Так это и есть в действительности: процесс вычисления решения сходится и устойчив, если

$$\lambda < \frac{1}{2},$$

или, что то же самое, при

$$k < \frac{h^2}{2}.$$

Тем самым накладываются довольно серьезные ограничения на выбор шага по времени, гораздо более серьезные, нежели в случае гиперболического уравнения. Именно это и заставляет искать возможности решения уравнения другими способами, в частности (11.35) и (11.37).

И (11.35), и (11.37) представляют собой неявные методы решения, аналогичные тому, который был применен для эллиптических уравнений (11.14). Рассмотрим сначала (11.35). Мы можем выписать уравнения для первой строки решения $j = 1$, используя начальные и граничные условия

(11.28) и (11.29):

$$
\begin{aligned}
(1+2\lambda)\,u_{1,1}-\lambda u_{2,1} &= f(h)+\lambda T_0,\\
-\lambda u_{1,1}+(1+2\lambda)\,u_{2,1}-\lambda u_{3,1} &= f(2h),\\
-\lambda u_{2,1}+(1+2\lambda)\,u_{3,1}-\lambda u_{4,1} &= f(3h),\\
&\cdots\cdots\cdots\cdots\cdots\cdots\cdots\\
-\lambda u_{n-2,1}+(1+2\lambda)\,u_{n-1,1} &= f(nh-h)+\lambda T_L.
\end{aligned}
$$

Всего получается $n-1$ линейных уравнений с $n-1$ неизвестными для $i=1, 2, \ldots, n-1$. Эти уравнения можно решить методом исключения; кроме того, что подавляющее большинство коэффициентов равно нулю, сам вид системы делает решение методом исключения довольно простой задачей. Дело в том, что единственные ненулевые коэффициенты расположены в трех диагональных рядах симметрично относительно главной диагонали. Система уравнений этого типа называется *тридиагональной*. Можно решать эту систему и с помощью итерационного метода Гаусса — Зейделя, так как диагональный коэффициент в каждом уравнении равен $1+2\lambda$, а сумма недиагональных равна -2λ, что обеспечивает сходимость. Для ускорения сходимости можно воспользоваться экстраполяцией.

Найдя решение для первой строки, т. е. для $t=k$, $j=1$, можно записать систему уравнений для второй строки и снова получить тридиагональную систему из $n-1$ уравнений. После решения этой системы можно переходить к третьей строке и так делее. Для вычисления одной строки решения *каждый раз* приходится решать систему из $n-1$ линейных алгебраических уравнений.

Поскольку трудоемкость этого способа гораздо больше, чем для явных формул (11.30), можно задать себе вопрос, имеет ли смысл вообще когда-либо прибегать к такому способу? Ответ на этот вопрос заключается в том, что (11.35) *устойчиво и сходится для всех* $\lambda > 0$. Поэтому, пользуясь методом (11.35), можно брать гораздо большую величину шага по времени. Попытаемся определить, можно ли действительно добиться экономии машинного времени.

Приблизительно можно считать, что при использовании метода (11.35) затраты машинного времени на вычисление одной строки решения будут вдвое больше, чем при использовании метода (11.30). Если взять $\lambda=1$, то общая затрата

времени на расчет решения до определенного момента будет вдвое меньше. Правда, нужно учесть, что ошибка ограничения круто возрастает с ростом шага по времени, так что определяющим фактором может явиться именно максимально допустимая ошибка.

Последняя разностная формула (11.37) является другим неявным методом и для вычисления одной строки также требует решения тридиагональной системы из $n - 1$ уравнений. Затраты машинного времени при использовании этой формулы будут, следовательно, такими же, как и для (11.35). Этот метод также устойчив для всех λ.

Заметим, что при удвоении λ в формуле (11.37) ошибка ограничения удваивается, в то время как в (11.35) ошибка ограничения учетверялась. Поэтому метод (11.37) (метод Кранка — Никольсона) предпочтительнее, нежели (11.35).

Нужно отметить, однако, что для других параболических уравнений этот метод устойчив не при всех λ и поэтому иногда приходится прибегать к (11.35).

Существует еще много различных разностных методов и явных, и неявных. В этой главе были рассмотрены только самые простые из них. Как обычно, мы заканчиваем указанием, что изложенные в этой главе методы позволяют решать множество практически важных задач, а также служат хорошим введением для последующего знакомства с более полными источниками.

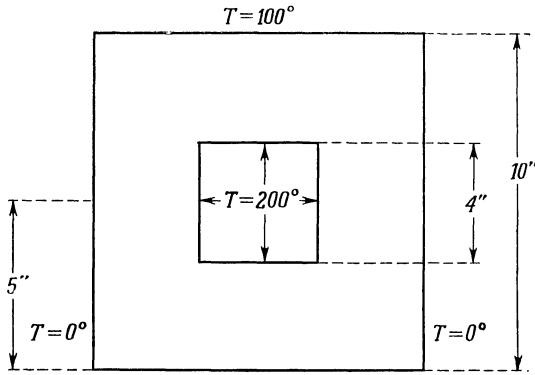
11.9. ПРАКТИЧЕСКИЙ ПРИМЕР 13: РАСПРЕДЕЛЕНИЕ ТЕМПЕРАТУРЫ В ТРУБЕ КВАДРАТНОГО СЕЧЕНИЯ

Рассмотрим длинную квадратную трубу с квадратным отверстием, по которой течет горячая жидкость. Труба наполовину погружена в ледяную ванну, так что температура нижней половины поверхности трубы равна 0° С. Верхняя плоскость трубы находится при постоянной температуре 100° С. Кроме того, мы предполагаем, что температура наружной поверхности трубы линейно изменяется от 0° С до 100° С на участке между ледяной ванной и верхней плоскостью трубы. Жидкость внутри трубы имеет температуру 200° С. Наружный размер трубы равен 10 дюймам, внутренний размер равен 4 дюймам. Поперечное сечение трубы изображено на рис. 11.5.

Распределение температуры в теле трубы удовлетворяет уравнению в частных производных

$$u_{xx} + u_{yy} = au_t,$$

где постоянная a учитывает теплоемкость, теплопроводность и плотность материала трубы. В данной задаче мы будем принимать, что жидкость течет в течение достаточно долгого времени для того, чтобы все переходные процессы успели



Р и с. 11.5. Сечение трубы в практическом примере 13.

закончиться, т. е. тепловой режим трубы стал стационарным. Тогда $u_t = 0$ и уравнение приобретает следующий вид:

$$u_{xx} + u_{yy} = 0.$$

Этому уравнению удовлетворяет распределение температур внутри трубы, а распределение температур на границах трубы задано граничными условиями. Таким образом, наша задача представляет собой задачу Дирихле для уравнения Лапласа, рассмотренную в параграфах 11.3 и 11.4.

Естественно, имеется в виду двумерная задача, т. е., например, не учитывается понижение температуры жидкости вдоль оси трубы. Иначе говоря, мы будем вычислять распределение температуры только для одного какого-то сечения трубы.

Предположим, что по наружному размеру в направлениях x и y сделано разбиение на 60 интервалов, так что

$h = k = 0{,}16667$ дюйма. Тогда граничные условия запишутся так:

$$u_{i,0} = 0; \qquad i = 0, 1, \ldots, 60;$$

$$u_{0,j} = u_{60,j} = \begin{cases} 0; & j = 0, 1, \ldots, 30, \\ \frac{100}{30}(j - 30); & j = 31, 32, \ldots, 60; \end{cases}$$

$$u_{i,60} = 100; \qquad i = 1, 2, \ldots, 60;$$

$$u_{i,42} = u_{i,18} = 200; \qquad i = 18, 19, \ldots, 41, 42;$$

$$u_{42,j} = u_{18,j} = 200; \qquad j = 18, 19, \ldots, 41, 42.$$

Разностные уравнения выглядят так:

$$u_{i,j} = \frac{1}{4}(u_{i-1,j} + u_{i,j-1} + u_{i+1,j} + u_{i,j+1})$$

при

$$i = 1, 2, \ldots, 59; \quad j = 1, 2, \ldots, 17,$$

и

$$i = 1, 2, \ldots, 17; \quad 43, \ldots, 59; \quad j = 18, 19, \ldots, 42,$$

и

$$i = 1, 2, \ldots, 59; \quad j = 43, 44, \ldots, 59.$$

Мы можем решить эти уравнения ускоренным методом Либмана и найти температуру во всех внутренних точках трубы.

Программа, приведенная на рис. 11.6, полностью соответствует блок-схеме, приведенной на рис. 11.3 и граничным условиям. То обстоятельство, что нельзя использовать нулевые индексы, учтено прибавлением 1 к каждому из них. Поэтому точки нумеруются не от 0 до 60, а от 1 до 61, индексы для граничных условий в полости трубы проходят значения от 19 до 43, а не от 18 до 42 и т. д.

С помощью первого оператора вводятся значения переменных OMEGA (ускоряющий множитель), EPS (критерий сходимости) и MAXIT (максимально допустимое количество итераций). С помощью последующих операторов вплоть до оператора 5 присваиваются начальные и граничные значения согласно тому, как это сделано в уравнениях. Чтобы сэкономить машинное время и не вычислять многократно

некоторых выражений, введены переменные A и B. Формула ускоренного метода Либмана переписана для этой программы в следующем виде:

$$u_{i,j} = u_{i,j} + \omega\left[\frac{1}{4}(u_{i+1,j} + u_{i,j+1} + u_{i-1,j} + u_{i,j-1}) - u_{ij}\right] =$$

$$= \frac{\omega}{4}(u_{i+1,j} + u_{i,j+1} + u_{i-1,j} + u_{i,j-1}) + (1-\omega)u_{ij}.$$

```
      READ INPUT TAPE 7, 100, OMEGA, EPS, MAXIT
  100 FORMAT (2F10.0, I3)
      DIMENSION U(61, 61)
      DO 1 I = 1, 61
      DO 1 J = 1, 61
    1 U(I, J) = 0.0
      DO 2 J = 32, 61
      FJ = J
      BOUND = 100.0 * (FJ - 31.0) / 30.0
      U(1, J) = BOUND
    2 U(61, J) = BOUND
      DO 3 I = 2, 60
    3 U(I, 61) = 100.0
      DO 4 I = 19, 43
      U(I, 43) = 200.0
    4 U(I, 19) = 200.0
      DO 5 J = 19, 43
      U(43, J) = 200.0
    5 U(19, J) = 200.0
      ITN = 1
      A = OMEGA / 4.0
      B = 1.0 - OMEGA
   10 D = 0.0
      DO 14 J = 2, 60
      DO 14 I = 2, 60
      IF (I - 19) 50, 11, 11
   11 IF (43 - I) 50, 12, 12
   12 IF (J - 19) 50, 13, 13
   13 IF (43 - J) 50, 14, 14
   50 UNEW = A*(U(I+1,J) + U(I-1,J) + U(I,J+1) + U(I,J-1)) + B*U(I,J)
      RESID = ABSF (UNEW - U(I,J))
      IF (RESID - D) 15, 15, 16
   16 D = RESID
   15 U(I,J) = UNEW
   14 CONTINUE
      IF (D - EPS) 60, 61, 61
   61 ITN = ITN + 1
      IF (ITN - MAXIT) 10, 10, 62
   62 WRITE OUTPUT TAPE 10, 600, MAXIT
  600 FORMAT (21H FAILS TO CONVERGE IN, I5, 10HITERATIONS)
   60 CALL EXIT
      END
```

Р и с. 11.6. Программа для вычисления распределения температуры в трубе квадратного сечения (практический пример 13).

Уравнение Лапласа решается с помощью экстраполированного метода Либмана.

Совершенно ясно, для чего вводятся новые переменные: нет никакого смысла многократно делить ω на 4 и многократно вычитать ω из 1. Остальная часть программы точно соответствует блок-схеме с добавлением операторов IF,

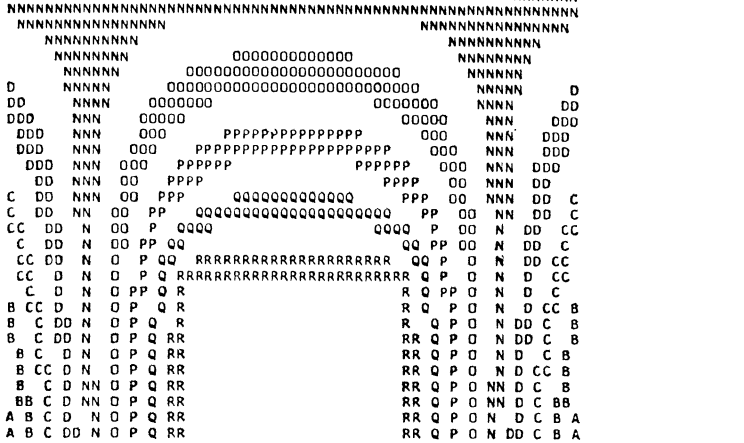


Рис. 11.7. Карта изотерм для трубы из практического примера 13
Карта была построена с помощью специальной программы.

которые служат для того, чтобы решение не вычислялось для полости трубы или на границе этой полости.

Процесс вывода результатов вычислений здесь не показан. Конечно, возможно напечатать вычисленные значения температуры для всех 2856 внутренних точек, но пытаться понять что-нибудь из этой массы чисел совершенно безнадежно.

В данном случае хорошим решением явилась бы печать карты изотерм (линий постоянной температуры) в сечении трубы с помощью ЭЦВМ. Такая карта приведена на рис. 11.7. Рисунок получился не квадратным, поскольку строки печатаются с плотностью по 6 на дюйм, а по горизонтали буквы имеют плотность 10 букв на дюйм, но это уже трудность второстепенная. Эта карта напечатана с помощью особой программы ЭЦВМ, которая также была составлена в основном на ФОРТРАНе. Карте соответствует таблица 11.1, в которой приведена температура, изображаемая различными изотермами. Например, из этой таблицы ясно, что символ *A* соответствует диапазону температур от 0° С до 11.7° С. Конечно, если требуется более подробная картина, то можно вывести на печать все значения температуры для внутренних точек трубы, хотя мы здесь и не будем воспроизводить эту операцию.

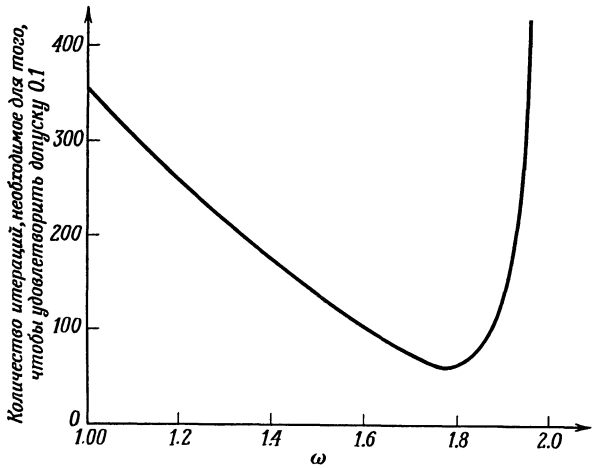
Таблица 11.1

Символ	Диапазон температуры в °С
A	0—11.7
B	23.6—35.3
C	47.1—58.7
D	70.6—82.3
N	94.2 –105.8
O	117.9—129.4
P	141.4—152.9
Q	164.8—176.3
R	188.3—200.0

Общая информация такого типа, как в карте изотерм, часто является главной целью начальной фазы исследований с помощью ЭЦВМ. Нередко в подобных случаях тре-

буется многократно решать одну и ту же задачу, варьируя значение некоторого параметра; в результате общая зависимость распределения температуры от этого параметра наглядно видна на диаграммах.

Наконец, попробуем оценить, насколько эффективным может оказаться метод ускорения. На рис. 11.8 построен график зависимости количества итераций, необходимого для



Р и с. 11.8. Количество итераций, необходимое для сходимости вычислений по программе рис. 11.6, как функция ускоряющего множителя ω (практический пример 13).

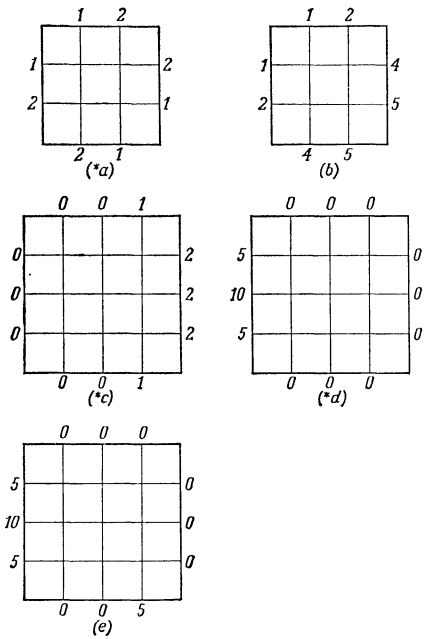
того, чтобы решение сошлось, от ускоряющего множителя ω. Заметим, что оптимальным значением для ω является приблизительно 1.8, а при отходе от этого оптимального значения в ту или другую сторону количество требуемых итераций увеличивается. Для ω = 1, т. е. при использовании обычного метода Либмана, количество требуемых итераций примерно в семь раз больше, нежели при оптимальном ω.

Упражнения

З а м е ч а н и е. Весьма полезно произвести решение нескольких примеров уравнений в частных производных с карандашом и бумагой или с клавишной вычислительной машинкой. Однако

если читатель решит не ограничиваться только самыми простыми упражнениями, то потребуется прибегнуть к помощи ЭЦВМ. Для решения численных примеров можно воспользоваться соответствующими модификациями программ, написанных для упражнений 17—22, а также программой, приведенной на рис. 11.6.

*1. Решите эллиптическое уравнение $u_{xx} + u_{yy} = 0$ для следующих квадратных сеток. Граничные условия принять такими, как они изображены на рисунках; для решения использовать метод Либмана и разностные уравнения (11.6) и (11.7); итерации производить до тех пор, пока разности между последовательными значениями функции для всех точек не станут меньше 0.005. Можно использовать методику ускорения сходимости итерационного процесса.



*2. Сетка в упражнении 1 (*d) имеет по три внутренних точки в каждом направлении. Если предположить, что в действительности граничные условия по левому вертикальному краю представляют собой линейное возрастание от нуля до 10, то можно получить более точное решение, взяв для этой задачи сетку с семью внутренними точками в каждом направлении. Граничные значения с левой стороны должны быть равны 2.5, 5.0, 7.5, 10.0, 7.5, 5.0, 2.5; все остальные должны быть по-прежнему равны нулю.

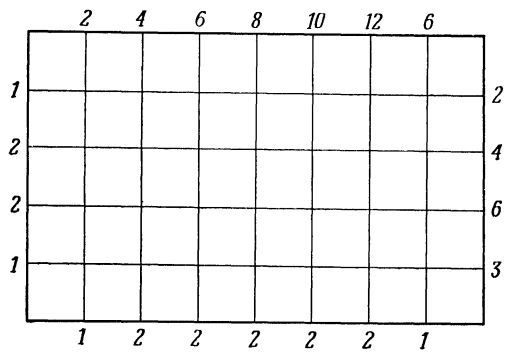
Решите упражнение 1 (**d*) с точностью 0.0001, затем решите описанное здесь упражнение также с точностью 0.0001. Сравните вычисленные значения для совпадающих точек сетки. Чем объясняется различие?

3. Разделите сетку, описанную в предыдущем упражнении, еще раз надвое, чтобы количество внутренних точек сетки в каждом направлении было равно 15. Граничные условия по левому вертикальному краю задайте числами 1.25, 2.5, 3.75, 5.0, 6.25, 7.5, 8.75, 10.0, 8.75, 7.5, . . ., 2.5, 1.25, остальные должны быть по-прежнему равны нулю.

Решите это упражнение с точностью 0.0001 и сравните решение с результатом для упражнения 2. (Указание: при решении системы уравнений необходимо пользоваться методом ускорения, если желательно получить точное решение менее чем за 100 итераций при нулевом начальном приближении.)

4. Решите задачи, аналогичные упражнениям 1-г, 2 и 3. Разница в условии задач состоит в том, что на этот раз ненулевые граничные значения расположены по правому вертикальному краю. Естественно, решения будут представлять собой зеркальные отражения решений исходных задач, но теперь количество итераций будет иным. Почему?

5. Рассмотрим следующую сетку и граничные условия



а. Используя обыкновенный метод Либмана ($\omega = 1$), решите систему уравнений последовательно с точностью 0.01, 0.001, 0.0001, 0.00001.

*б. Означает ли тот факт, что последовательные приближения при очередных итерациях отличаются друг от друга не более чем на 0.01, что сами эти последовательные приближения отличаются не более чем на 0.01 от точного решения системы разностных уравнений? Проверьте, сравнивая решение, вычисленное с точностью 0.01, и решение, вычисленное с точностью 0.001.

*в. При заданной точности нахождения решения 0.0001 попытайтесь решить систему при $\omega = 0.9, 1.0, \ldots, 1.9$.

Сравните количество итераций, необходимых для решения системы с различными ω.

*г. Измените граничное значение внизу в центре с 2.0 до 5.0 и решите систему разностных уравнений с новым граничным условием с точностью 0.0001, используя выбранный вами ускоряющий множитель. Сравните это решение с тем, которое было получено до изменения граничного условия. Влияет ли изменение значения на границе на все значения функции внутри области или только на некоторые?

6. Решите гиперболическое (волновое) уравнение $U_{xx} = U_{tt}$ явным методом (11.19). В направлении x возьмите 21 точку и присвойте им индексы от 1 до 21, примите $h = 1.0$. В качестве граничных условий возьмите

$$U(1, J) = U(21, J) = 0.0; \quad J = 1, 2, \ldots .$$

Для задания начальных условий используйте ситуацию, изображенную на рис. 11.4, причем стрелу прогиба δ примите равной 2.0, то есть

$$U(I, 1) = 0.2(I - 1) \quad \text{для} \quad I = 2, 3, \ldots, 11;$$
$$U(I, 1) = 0.2(21 - I) \quad \text{для} \quad I = 12, 13, \ldots, 20.$$

Для того чтобы можно было начать вычислять решение, примите $U(I, 2) = U(I, 1)$, $I = 2, 3, \ldots, 20$. Используйте $\lambda = 0.2$ и продолжите решение вплоть до $J = 250$.

7. Решите задачу, поставленную в упражнении 6, приняв:

а. $\lambda = 0.1$ $j_{\max} = 400$;
б. $\lambda = 0.4$ $j_{\max} = 100$;
в. $\lambda = 0.8$ $j_{\max} = 50$;
г. $\lambda = 1.0$ $j_{\max} = 20$.

***8.** Решите ту же задачу, что и в упражнении 6, приняв в качестве начального условия

$$U(I, 1) = U(I, 2) = 2 \sin \frac{\pi (I - 1)}{20}; \quad I = 2, 3, \ldots, 20.$$

Учитывая, что в начальный момент струна имеет форму синусоидальной полуволны, можно ли предсказать, каков будет вид решения для последующих моментов времени? Возьмите решение для достаточно удаленной точки времени, когда уже пройдет полный период колебаний, например $j = 202$, и для этой точки времени постройте график решения как функцию от I. Аналогично постройте график положения средней точки струны ($I = 11$) как функцию от времени.

9. Решите ту же задачу, что и в упражнении 8, приняв в качестве начальных условий:

а. $U(I, 1) = U(I, 2) = 2 \sin \dfrac{2\pi (I - 1)}{20}$;

б. $U(I, 1) = U(I, 2) = 2 \sin \dfrac{3\pi (I - 1)}{20}$.

10. Решите ту же задачу, что и в упражнении 6, приняв в качестве начальных условий:

$$U(2, 1) = U(8, 1) = 0.5,$$
$$U(3, 1) = U(7, 1) = 1.0,$$
$$U(4, 1) = U(6, 1) = 1.5,$$
$$U(5, 1) = 2.0,$$
$$U(I, 1) = 0.0; \quad I = 9, 10, \ldots, 20.$$

Можно ли ожидать, что такая форма струны будет периодически повторяться, так же как периодически повторялась синусоидальная форма? Опишите качественно поведение струны при этих начальных условиях.

***11.** Решите ту же задачу, что и в упражнении 6, изменив одно значение в начальных условиях:

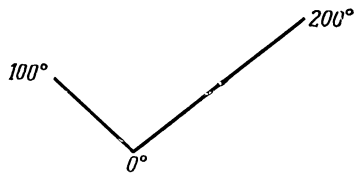
$$U(13, 1) = U(13, 2) = 0.4.$$

Сравните результат с решением упражнения 6. Влияет ли изменение начального условия в одной точке на все точки решения или только на некоторые из них?

***12.** Решите параболическое уравнение $U_{xx} = U_x$ явным методом (11.30). Возьмите 21 точку в направлении x с индексами от 1 до 21 и примите $h = 1.0$. В качестве граничных условий примите

$$U(1, J) = 100.0; \quad J = 1, 2, \ldots$$
$$U(21, J) = 200.0; \quad J = 1, 2, \ldots .$$

В качестве начального условия рассмотрите распределение температуры вдоль стержня



Это распределение температуры можно записать в виде

$$U(I, 1) = 10(11 - I); \quad I = 2, 3, \ldots, 11,$$
$$U(I, 1) = 20(I - 11); \quad I = 12, 13, \ldots, 20.$$

Примите $\lambda = 0.2$ и вычислите решение вплоть до $j = 200$. К какому виду должно стремиться распределение температуры по истечении достаточно длительного времени?

13. Решите ту же задачу, что и в упражнении 12, приняв

$$\lambda = 0.1; \quad j_{\max} = 400;$$
$$\lambda = 0.4; \quad j_{\max} = 100;$$
$$\lambda = 0.8; \quad j_{\max} = 50;$$
$$\lambda = 1.0; \quad j_{\max} = 20.$$

14. Решите ту же задачу, что и в упражнении 12, изменив одно начальное значение:

$$U(13,\ 1)=200.$$

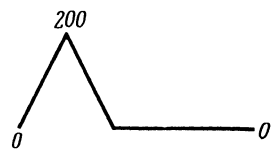
Примите $\lambda=0,\ 2$ и вычислите решение вплоть до $j=200$. Сравните результат с результатом решения упражнения 12. Оказывает ли изменение одного начального значения влияние на все точки решения или только на некоторые?

15. Решите ту же задачу, что и в упражнении 12, приняв граничные и начальные условия по следующим схемам и формулам:

а)

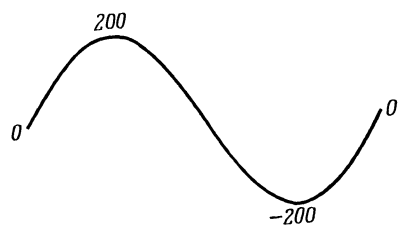
$$U(I,\ 1)=0.0; \qquad I=2,\ 3,\ \ldots,\ 11,$$
$$U(I,\ 1)=20\,(I-11); \qquad I=12,\ 13,\ \ldots,\ 20,$$

б)



$$U(2,\ 1)=U(8,\ 1)=50;$$
$$U(3,\ 1)=U(7,\ 1)=100;$$
$$U(4,\ 1)=U(6,\ 1)=150;$$
$$U(5,\ 1)=200.$$
$$U(I,\ 1)=0.0; \qquad I=9,\ 10,\ \ldots,\ 20.$$

в)



$$U(I,\ 1)=200\sin\frac{2\pi\,(I-1)}{20}; \qquad I=2,\ 3,\ \ldots,\ 20.$$

16. Решите ту же задачу, что и в упражнении 12, используя неявный метод (11.35); при этом требуется решить систему уравнений из разд. 11.8. Примите следующие параметры:

а. $\lambda = 0.8$; $j_{max} = 100$;

б. $\lambda = 1.0$; $j_{max} = 100$;

в. $\lambda = 2.0$; $j_{max} = 80$;

г. $\lambda = 5.0$; $j_{max} = 50$;

д. $\lambda = 10.0$; $j_{max} = 50$.

***17.** Напишите программу для решения задачи из упражнения 5. В программе следует использовать обыкновенный метод Либмана ($\omega = 1.0$), максимальное количество итераций не должно превосходить 50. Граничные условия необходимо вводить с перфокарт, причем на каждой карте должно содержаться значение I, значение J и соответствующее им граничное значение U (I, J). Конец массива исходной информации должен определяться по пробной карте с $I = 0$.

18. Видоизмените программу из упражнения 17 следующим образом: максимальное число внутренних точек сетки в каждом направлении не должно превышать 50. Перед началом работы программы с перфокарты должна быть введена информация, определяющая точность вычислений для итерационного процесса, значение ускоряющего множителя ω, максимально допустимое количество итераций и максимально возможные значения индексов в обоих направлениях.

***19.** Напишите программу для решения гиперболического уравнения из упражнения 6. В целях упрощения составьте программу так, как будто в вашем распоряжении есть ЭЦВМ с неограниченным объемом памяти. Образуйте двумерный массив, содержащий 21 элемент в направлении I и 500 — в направлении J. Введите с перфокарты значение λ и максимально требуемое количество шагов в направлении J. Затем с помощью программы (для этого потребуется несколько операторов) присвойте нужным элементам массива начальные и граничные значения. Предусмотрите, чтобы эту группу операторов можно было заменить для решения задач с различными начальными и граничными условиями. После этого в программе должны быть предусмотрены два цикла, находящиеся один внутри другого: внешний цикл с индексом J (время) и внутренний цикл с индексом I (координата x); вычисления можно производить по формуле (11.19). Удобнее всего принять для текущего момента времени индекс J, для предыдущего — индекс $J - 1$, а для момента времени, находящегося двумя шагами ранее,— индекс $J - 2$ (отметьте разницу с обозначениями в (11.19)). Как всегда, все индексы должны начинаться с 1, а не с 0, так как в ФОРТРАНе нулевые индексы не допускаются.

20. Модифицируйте программу из упражнения 19 таким образом, чтобы избежать больших затрат машинной памяти. Образуйте три одномерных массива по 21 элементу каждый. Их наименования могут быть, например: UJP1 (I), UJ (I) и UJM1 (I). Элементам массивов UJ (I) и UJM1 (I) присваиваются начальные значения

(из начальных условий). После этого вычисляется одна строка решения, то есть 21 новое значение элементов массива UJP1 (I); вычисленные значения печатаются. После этого можно переставить все элементы массива UJP1 (I) в массив UJ (I), а элементы массива UJ (I) — в массив UJM1 (I) и снова начать вычисления значений UJP1 (I) по значениям UJ (I) и UJM1 (I).

21. Напишите программу для решения параболического уравнения из упражнения 12. Программа может быть построена точно так же, как и в упражнении 19 (или 20), за исключением того что значения, соответствующие начальным условиям, присваиваются только тем элементам, которые обозначают нулевой момент времени, и, конечно, вместо разностной формулы (11.19) используется формула (11.30).

22. Напишите программу для решения параболического уравнения с помощью неявного метода (11.35) с системой уравнений из разд. 11.8.

При решении этой системы не потребуется образовывать двумерный массив коэффициентов; решение тридиагональной системы уравнений методом исключения можно представить в виде рекуррентной формулы (см. упражнение 23). Если же уравнения решаются методом итераций, то фактические вычисления еще проще: коэффициенты уравнений равны просто λ и $1 + 2\lambda$, эти уравнения можно записать в виде арифметических операторов. Если учесть еще, что T_0 и T_L являются в действительности значениями U на границе $C(I = 1$ и $I = 21)$, то первое и последнее уравнения можно решать так же, как и остальные; поэтому вся итерационная формула сведется к одному арифметическому оператору, индексы в котором будут управляться оператором DO. Способы определения момента, когда итерационный метод сойдется, и остановки вычислений при отсутствии сходимости могут быть такими же, как в блок-схеме на рис. 8.9 или в программе на рис. 8.15.

23. Рассмотрим некоторую тридиагональную систему уравнений, причем будем предполагать, что эта система имеет единственное решение

$$\begin{aligned}
b_1x_1 + c_1x_2 &= d_1,\\
a_2x_1 + b_2x_2 + c_2x_3 &= d_2,\\
a_3x_2 + b_3x_3 + c_3x_4 &= d_3,\\
&\vdots\\
a_{n-1}x_{n-2} + b_{n-1}x_{n-1} + c_{n-1}x_n &= d_{n-1},\\
a_nx_{n-1} + b_nx_n &= d_n.
\end{aligned}$$

а. Покажите, что решение этой системы можно найти с помощью следующих соотношений:

$$\begin{aligned}
x_n &= d_n^*,\\
x_r &= d_r^* - c_r^*x_{r+1}; \quad r = n-1, \ldots, 1,
\end{aligned}$$

где

$$c_1^* = \frac{c_1}{b_1}; \quad d_1^* = \frac{d_1}{b_1};$$

$$c_r^* = \frac{c_r}{b_r - a_r c_{r-1}^*}; \quad r = 2, \ldots, n,$$

$$d_r^* = \frac{d_r - a_r d_{r-1}^*}{b_r - a_r c_{r-1}^*}; \quad r = 2, \ldots, n.$$

б. Предположим, что для некоторого $r = 2, \ldots, n$

$$b_r - a_r c_{r-1}^* = 0.$$

Покажите, что тогда

$$x_{r+1} = \frac{d_r - a_r d_{r-1}^*}{c_r}$$

при условии, что $c_r \neq 0$. Рассмотрите случай $c_r = 0$. Покажите также, что задача сводится к решению

двух систем уравнений: одной — из r уравнений с r неизвестными и другой — из $(n - r - 1)$ уравнений с $(n - r - 1)$ неизвестными.

Что можно сказать о решении этих двух уменьшенных систем, если $c_k \neq 0$ для $k = r, r + 1, \ldots, n$?

24. Решите систему из пяти тридиагональных уравнений

$$\begin{aligned}
2x_1 - x_2 &= 2,\\
-x_1 + 2x_2 - x_3 &= 0,\\
-x_2 + 2x_3 - x_4 &= 0,\\
-x_3 + 2x_4 - x_5 &= 0,\\
-x_4 + 2x_5 &= 1.
\end{aligned}$$

25. Покажите, что если написать разностное приближение первой производной в виде

$$u_x = \frac{u_{i+1,\,j} - u_{i-1,\,j}}{2h},$$

то ошибка ограничения будет пропорциональна h^2. Это приближение называется *центральной разностью*. Отметим сходство центральной разности и формулы прогноза (10.33).

26. Выведите формулу

$$u_{xy} = \frac{u_{i+1,\,j+1} - u_{i+1,\,j-1} - u_{i-1,\,j+1} + u_{i-1,\,j-1}}{4hk}$$

и покажите, что ошибка ограничения при использовании этой формулы имеет порядок h^2. (Указание: дважды используйте центральные разности.)

***27.** Вместо правой разности (11.2) используйте центральную разность из упражнения 25 совместно с формулой (11.6) и получите новое разностное приближение к уравнению теплопроводности $u_{xx} = u_t$. Начертите трафарет метода.

Этот метод обычно называется методом Ричардсона. Он неустойчив при всех значениях $\lambda = k/h^2$.

28. В разностной формуле из упражнения 27 замените $u_{i,\,j}$ средним из $u_{i,\,j+1}$ и $u_{i\,,j-1}$, т.е.

$$u_{i,\,j}=\frac{u_{i,\,j+1}+u_{i,\,j-1}}{2}$$

и напишите новую аппроксимацию уравнения теплопроводности. Начертите трафарет метода.

Этот метод называется методом Дюфорта — Франкеля. Он сходится и устойчив при всех значениях $\lambda = k/h^2$.

Если начальные и граничные условия заданы в виде (11.26) и (11.27), а для решения уравнения используется метод Дюфорта — Франкеля, можно ли сразу приступать к вычислению $u_{i,\,1}$ для $i = 1, 2, \ldots, n-1$? Что нужно сделать для того, чтобы можно было начать вычисления?

***29.** На основе формулы (11.6) можно написать следующие три равенства:

$$u_{xx}^{(j+1)}=\frac{u_{i+1,\,j+1}-2u_{i,\,j+1}+u_{i-1,\,j+1}}{h^2},$$

$$u_{xx}^{(j)}=\frac{u_{i+1,\,j}-2u_{i,\,j}+u_{i-1,\,j}}{h^2},$$

$$u_{xx}^{(j-1)}=\frac{u_{i+1,\,j-1}-2u_{i,\,j-1}+u_{i-1,\,j-1}}{h^2}.$$

а. Напишите такие же равенства для u_{yy} в точках

$$i+1,\ j;\ i,\ j;\ i-1,\ j.$$

б. Предположите, что $h=k$, и пусть также

$$u_{xx}=\frac{1}{12}u_{xx}^{(j+1)}+\frac{5}{6}u_{xx}^{(j)}+\frac{1}{12}u_{xx}^{(j-1)},$$

$$u_{yy}=\frac{1}{12}u_{yy}^{(i+1)}+\frac{5}{6}u_{yy}^{(i)}+\frac{1}{12}u_{yy}^{(i-1)}.$$

Найдите на основе этих формул разностную аппроксимацию уравнения Лапласа

$$u_{xx}+u_{yy}=0.$$

Для уравнения Лапласа это приближение обеспечивает весьма высокую точность, однако для того случая, когда правая часть уравнения отлична от нуля, это приближение ничуть не точнее, нежели (11.14).

в. Начертите трафарет метода.

г. Можно ли для решения этой системы уравнений использовать метод Гаусса — Зайделя?

30. Хорошо известный вариационный принцип гласит, что если различные функции $u\,(x, y)$ имеют непрерывные первые производные внутри замкнутой области R и удовлетворяют граничному

условию $u(x, y) = f(x, y)$ на границе этой области C, то та из функций, которая минимизирует интеграл

$$I = \iint_R (u_x^2 + u_y^2)\, dx\, dy,$$

является решением задачи Дирихле

$$u_{xx} + u_{yy} = 0 \quad \text{внутри } R,$$
$$u(x, y) = f(x, y) \quad \text{на кривой } C.$$

Замените u_x и y_y правыми разностями. Замените интегрирование суммированием, а $dx\, dy$ произведением hk.

Если I минимально, то все первые частные производные I по u_{ij} должны обратиться в нуль. Покажите, что обращение этих производных в нуль эквивалентно системе разностных уравнений (11.14)

Что изменилось бы, если бы вместо правых разностей для u_x и u_y были взяты левые разности?

*31. В разд. 11.4 решение эллиптических разностных уравнений получилось следующим образом: оно вычислялось в отдельности для каждого узла из отдельного уравнения (метод Либмана).

Вместо этого мы можем попытаться решать одновременно несколько уравнений относительно нескольких неизвестных. В частности, можно написать систему уравнений для целой строки; при этом получается $n - 1$ уравнений относительно $n - 1$ неизвестных. Неизвестными будут значения $u_{i,j}$ для этой строки, $i = 1, 2, \ldots, n - 1$ при некотором фиксированном j. Значения u_{ij} во всех остальных строках принимаются равными своим последним приближениям.

Затем эти $n - 1$ уравнений решаются относительно $n - 1$ неизвестных и вычисляются новые приближения для $u_{1,j}, u_{2,j}, \ldots, u_{n-1,j}$. После этого составляется система из $n - 1$ уравнений для следующей строки сетки (с номером $j + 1$) и процесс повторяется. Итерация заканчивается, когда новые приближения будут вычислены для всех $m - 1$ строк.

а. Покажите, что для нахождения нового приближения в каждой строке сетки необходимо решить тридиагональную систему уравнений. Если таким образом перебирать по очереди все строки, то процесс вычислений называется *релаксацией по строкам*. Этот способ вычислений часто оказывается более эффективным, нежели метод Либмана, хотя для одной итерации приходится решать $m - 1$ тридиагональных систем уравнений.

б. Предположите, что сетка представляет собой квадрат, разбитый в каждом направлении на четыре равных интервала. Пусть все граничные значения равны $+1$. Выпишите три тридиагональные системы уравнений, которые необходимо решать при каждой итерации. Что можно сказать об уравнениях для первой и для третьей строк?

32. Вместо релаксации по строкам, описанной в упражнении 31, можно использовать релаксацию по столбцам, т. е. написать уравнения для вычисления значений функции во всех внутренних узлах, лежащих на одной вертикали, и решить получившуюся тридиагональную систему. Каждая такая система будет состоять из $m-1$ уравнений с $m-1$ неизвестными u_{ij}, $j=1, 2, \ldots, m-1$ при фиксированном i.

Весьма эффективную вычислительную схему можно построить следующим образом: во время нечетных итераций производить релаксацию по строкам, а во время четных итераций — по столбцам. Напишите системы уравнений, которые придется решать при использовании такой вычислительной схемы.

Этот метод называется *методом чередующихся направлений*. Он был разработан Писманом, Рэшфордом, Дугласом и другими.

33. Покажите, что волновое уравнение

$$u_{xx}=a^2u_{tt}$$

можно свести к виду

$$u_{yz}=0,$$

если сделать следующую замену переменных:

$$y=ax-t,$$
$$z=ax+t.$$

***34.** а. Напишите разностные уравнения (11.30) для уравнения теплопроводности $u_{xx}=u_t$ и положите $k=h^2$. Опишите процесс нахождения значений $u_{i,\,j+1}$ через значения $u_{i,j}$ и начертите трафарет для этого случая.

б. Разностное уравнение (а) неустойчиво. Чтобы убедиться в этом, положите $h=0{,}2$ и $k=0{,}04$. Введите модель ошибки в исходных данных, положив $u_{i,\,0}=0$ для $i\neq 10$ и $u_{10,0}=\varepsilon$. Проследите за процессом распространения этой ошибки, вычисляя решение вплоть до $t=0{,}16$ (на четыре шага вперед).

Заметьте, что на некоторые точки решения, например на точку $i=7$, $j=1$, наша модель ошибки совершенно не влияет независимо от величины ε. Это явление характерно для параболических и гиперболических уравнений, но не для эллиптических. См. также упражнения 5 г, 11, 14 и 36.

35. а. Напишите разностные уравнения (11.30) для уравнения теплопроводности $u_{xx}=u_t$ и положите $k=h^2/2$. Опишите процесс нахождения значений $u_{i,\,j+1}$ через значения $u_{i,\,j}$ и начертите трафарет для этого случая.

б. Разностное уравнение (а) устойчиво. Убедитесь в этом, положив $h=0.2$ и $k=0.02$. Введите модель ошибки в исходных данных, положив $u_{i,0}=0$ для $i\neq 10$ и $u_{i,0}=\varepsilon$ для $i=10$. Проследите за процессом распространения этой ошибки, вычисляя решение вплоть до $t=0.16$ (на восемь шагов вперед).

в. Какова сумма ошибок для всех i, вычисленных для определенного момента времени j?

г. Сравните этот результат с результатом упражнения 34. Какова максимальная ошибка при $t = 0.16$ в том и в другом случае?

36. а. Напишите разностную схему (11.14) для уравнения Лапласа и положите $h = k$. Опишите процесс вычисления $u_{i,\,j}$ через значения в соседних узлах.

б. Введите модель ошибки в граничных условиях, предположив для этого, что сетка представляет собой квадрат, разбитый в горизонтальном и вертикальном направлении на 6 интервалов ($i, j = 0, 1, \ldots, 6$). Положите все граничные значения равными нулю, за исключением $u_{0,3}$ (середина нижней границы). Положите $u_{0,3} = \varepsilon$. Проследите за влиянием этой ошибки на значения $u_{i,\,j}$ внутри квадрата.

в. Влияет ли ошибка в граничном условии в одной точке на значения функции в одной или в нескольких точках внутри квадрата? Сравните влияние ошибки на решение в этом упражнении с влиянием аналогичных ошибок в упражнениях 34 и 35.

г. В каком узле внутри квадрата ошибка вычисления функции $u_{i,\,j}$ максимальна?

37. Предположим, что используется метод релаксации по строкам (упражнение 31) с десятиточечной разностной аппроксимацией (упражнение 29). Какого типа будут те системы уравнений, которые придется решать для каждой строки? Будут ли затраты машинного времени больше, чем в том случае, если используются более простые пятиточечные уравнения (11.14), как это рассматривалось в упражнении 31?

38. Покажите, что если аппроксимировать $u_x(x_0, y_0)$ выражением

$$u_x = \frac{u(x_0+h, y_0) - u(x_0+h, y_0)}{2h},$$

то ошибка ограничения равна

$$E_T = -\frac{h^2}{6} u_{xxx}(\xi, y_0),$$

где

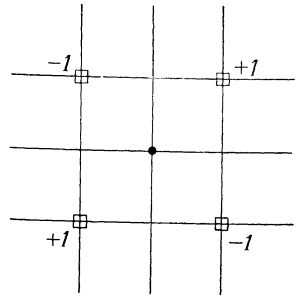
$$x_0 - h \leqslant \xi \leqslant x_0 + h.$$

Отметьте аналогию с формулой прогноза (10.33) для обыкновенных дифференциальных уравнений.

39. Используя результат упражнения 38, покажите, что разностную аппроксимацию для смешанной второй производной можно записать в виде

$$u_{xy} = \frac{u(x_0+h, y_0+k) - u(x_0-h, y_0+k) - u(x_0+h, y_0-k) + u(x_0-h, y_0-k)}{4hk}.$$

Трафарет в данном случае будет выглядеть так:



Разлагая значения $u(x, y)$, стоящие в правой части равенства, в ряд Тейлора в окрестности точки x_0, y_0, покажите, что при использовании этого приближения ошибка ограничения приблизительно равна

$$E_T = h^2 u_{xxxy} + k^2 u_{xyyy}$$

плюс члены, содержащие h и k в более высоких степенях.

40. Покажите, что уравнение

$$u_{xx} - a^2 u_{tt} = 0$$

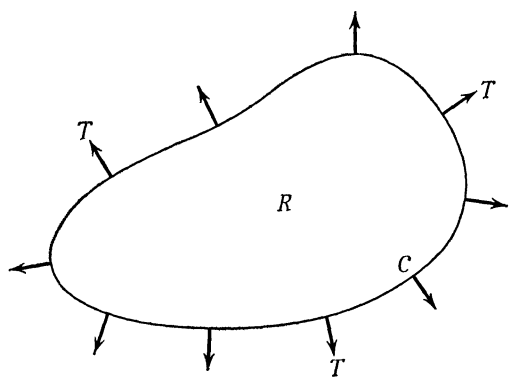
можно свести к волновому уравнению

$$u_{xx} - u_{yy} = 0$$

заменой переменных

$$ay = t.$$

41. Рассмотрим плоскую упругую мембрану, равномерно растягиваемую во все стороны с усилием T на единицу длины



Предположим также, что перпендикулярно плоскости мембраны приложено однородное давление p. Тогда прогиб мембраны удовлетворяет уравнению

$$\frac{\partial^2 z}{\partial x^2}+\frac{\partial^2 z}{\partial y^2}=-\frac{p}{T} \quad \text{внутри } R,$$

$z=0$ на границе мембраны C.

Покажите, что эту задачу можно свести к задаче о кручении цилиндра (11.9) и (11.10).

ПРИЛОЖЕНИЕ 1

СВОДКА МЕТОДОВ ВВОДА И ВЫВОДА ИНФОРМАЦИИ В ФОРТРАНе

Основная цель этого приложения заключается в том, чтобы снабдить читателей некоторой дополнительной информацией о методах ввода — вывода информации. Однако, чтобы это приложение можно было использовать также в качестве справочника, в нем приводятся и те сведения о методах ввода и вывода, которые уже изложены в основном тексте.

П.1.1. ОСНОВНЫЕ СВЕДЕНИЯ

Ввод и вывод информации из вычислительной машины можно произвести только при том условии, когда в программе содержатся следующие сведения об операциях ввода — вывода.

1. Определено устройство, с помощью которого вводится или выводится информация. Это делается путем использования соответствующего оператора ввода — вывода: PRINT, READ INPUT TAPE и т. д.

2. Определены переменные, которым должны быть присвоены новые значения или значения которых должны быть выведены на выводное устройство. Это делается путем перечисления переменных в операторе ввода — вывода.

3. Определен порядок, в котором переменные должны вводиться или выводиться. Он определяется самим списком, где перечислены переменные.

4. Определен формат, в котором информация появляется во вводном устройстве или в котором информация должна быть передана в выводное устройство. Формат определяется специальным оператором FORMAT, этому оператору присваивается номер и на этот номер должна иметься ссылка в операторе ввода — вывода.

Все эти вопросы уже обсуждались ранее и применение операторов ввода — вывода было проиллюстрировано в предыдущих главах книги. В этом приложении мы приведем более подробные и полные сведения о методах ввода и вывода информации. Эти сведения могут использоваться как для упрощения программ, так и для повышения эффективности вычислительной машины.

П.1.2. СПИСОК ПЕРЕМЕННЫХ В ОПЕРАТОРЕ ВВОДА — ВЫВОДА

Самым простым типом списка является такой, где переменные просто перечислены в том порядке, в котором они должны получить новые значения или быть выведены из ЭЦВМ. Как мы увидим далее, существуют и более сложные типы списков, но в любом случае соблюдается основной принцип «сканирования»: первому численному значению соответствует первое наименование переменной и первая спецификация поля в операторе FORMAT и т. д.

Рассмотрим теперь некоторые усложнения списка переменных. Прежде всего отметим, что целая переменная, содержащаяся в списке, может быть использована в том же списке в качестве индекса. При этом необходимо соблюдать два правила.

1. В тех случаях, когда значения целых переменных вводятся и используются в качестве индекса в операторе ввода, сами эти целые переменные должны располагаться в списке *до того*, как они появляются в качестве индексов. «До» для списка переменных означает «слева».

2. В этом же случае между переменной, значение которой вводится, и той переменной, где это только что введенное значение используется в качестве индекса, должна стоять левая скобка (дополнительно к той скобке, которая ставится при индексах). Для случая «простого» списка это означает, что наименование переменной с индексами должно быть поставлено в скобки.

Эту возможность часто используют, например, для того, чтобы ставить на нужные места элементы массивов, которые появляются на входе в случайной последовательности. Предположим, что элементы некоторого двумерного массива отперфорированы по одному на перфокарте и на той же пер-

фокарте имеются оба значения их индексов. Порядок расположения чисел на перфокарте следующий: I, J, A(I, J). Тогда ввод информации можно осуществить с помощью следующей системы операторов:

```
READ 400, I, J, (A(I, J))
400 FORMAT (I 2, I 2, E 14.7)
```

Наименование переменной A (I, J) заключено в операторе READ в скобки, как это требуется согласно правилу 2. Таким образом при вводе каждого элемента массива сразу точно определяется, в какую ячейку памяти этот элемент должен быть записан, т. е. указывается, что это за элемент.

Когда с помощью операторов ввода — вывода необходимо передавать полностью некоторые массивы или даже их части, то зачастую нет необходимости перечислять в списке все элементы. Чтобы передать массив полностью, достаточно поставить в списке его наименование без всяких индексов. Естественно, наименование этого массива должно содержаться где-либо в программе в операторе DIMENSION; тогда в списке переменных оператора ввода — вывода может отсутствовать соответствующая индексная информация. Все элементы массива обычно записываются в одном и том же формате, так что оператор FORMAT может содержать всего одну спецификацию поля. Если, например, нужно напечатать весь массив A, то можно написать

```
PRINT 21, A
21 FORMAT (E 20. 7)
```

где размеры массива *A* могут быть как угодно велики. В этом случае используется одно свойство оператора FORMAT: когда во время передачи информации достигается закрывающая скобка оператора FORMAT, то процесс сканирования начинается снова (в данном случае с самого начала). В этом примере указана всего одна спецификация поля; она и будет использоваться снова и снова, пока не будет передан весь массив A.

Заметим также, что при передаче массива только что описанным способом перебор его отдельных элементов производится в «естественной» последовательности, описанной

в гл. 7: первый индекс меняется наиболее быстро, последний — наиболее медленно.

В тех случаях, когда нужно передать только часть массива или когда желательно нарушить «естественный» порядок следования его элементов, возможно избежать их перечисления. Вместо этого можно организовать список таким образом, что он будет напоминать оператор DO, хотя самого слова DO там и не будет. Например, если написать

```
READ 200, (A(1, J) J = 1,10)
200 FORMAT (10F 7.0)
```

то с помощью этих операторов 10 чисел будут введены с перфокарт и эти значения будут присвоены первым 10 элементам первой строки массива A. Те же самые значения могли стать первыми 10 элементами первого столбца массива A, если бы оператор ввода выглядел так:

```
READ 200, (A(I, 1), I = 1,10)
```

Заметим, что переменная (или переменные) с индексами, а также информация о пределах изменения индексов должны быть заключены в скобки.

В операторе ввода — вывода можно организовывать индексные циклы один внутри другого, как в случае оператора DO. Предположим, например, что необходимо напечатать 60 чисел: первые 12 чисел должны быть первыми 12 элементами первой строки массива RESULT, следующие 12 чисел должны быть первыми 12 числами третьей строки этого же массива, следующие 12 чисел должны быть первыми 12 элементами пятой строки и т. д. Чтобы напечатать эти числа, можно составить следующую систему операторов:

```
PRINT 91, ((RESULT(I, J), J = 1,12), I = 1, 9, 2)
91 FORMAT (E 20.7)
```

Можно было бы, конечно, сделать то же самое и непосредственно с помощью оператора DO:

```
    DO 462 I = 1,9,2
    DO 462 J = 1,12
462 PRINT 91, A(I, J)
91 FORMAT (E 20.7)
```

Эти две формы записи совершенно эквивалентны. Таким образом, понятия внутреннего и внешнего цикла непосредственно переносятся на случай операторов ввода — вывода. Следует отметить, что в операторе ввода — вывода может быть не более трех циклов один внутри другого.

Мы убедились, что процесс сканирования может быть сложнее, нежели в простом списке, где только перечислены переменные. Тем не менее последовательность, в которой передаются значения переменных, полностью определяется списком. Следует отметить, что процесс сканирования в списке переменных идет синхронно с аналогичным процессом в списке спецификаций поля оператора FORMAT: каждый раз, когда очередь доходит до новой переменной в списке, соответственно выбирается новая спецификация поля в операторе FORMAT. Те методы индексации списка переменных и усовершенствования оператора FORMAT, которые мы рассмотрим ниже, несколько усложняют эту картину, но не меняют основной идеи одновременного сканирования в списке переменных и в списке спецификаций поля.

Индексацию можно организовать таким образом, чтобы целые переменные, встречающиеся ранее в списке, были впоследствии использованы в качестве индексных параметров. При такой индексации скобки, заключающие переменные с индексами и индексные параметры, должны удовлетворять правилу 2, приведенному ранее в этом разделе.

П.1.3. ОПЕРАТОР FORMAT

Роль оператора FORMAT состоит в том, чтобы описать, как организована информация на входе ЭЦВМ или как она должна быть организована на выходе. Для каждого передаваемого элемента информации необходимо предусмотреть в операторе FORMAT спецификацию поля, в которой указывается, какого вида информация передается и как она выглядит на входе или будет выглядеть на выходе. Все описание оператора FORMAT удобно разбить на два раздела, в первом мы рассмотрим различные спецификации поля и области их применения, а во втором — порядок расположения этих спецификаций поля в операторе FORMAT, совместимый с процессом сканирования.

Мы рассмотрим шесть типов спецификаций поля. В каждом случае обозначение спецификации поля состоит из:

1) буквы I, F, E, H, X или A, определяющей вид передаваемой информации или некоторые сведения относительно того, как с этой информацией следует обращаться;

2) целого числа, определяющего количество колонок перфокарты или позиций печатающего устройства, занимаемых данным элементом информации.

Спецификации типа F и E должны иметь еще второе число, определяющее позицию десятичной точки.

Чтобы не повторяться далее, отметим некоторые общие свойства спецификаций типов I, F и E.

Если у числа имеется знак *при вводе* информации, то он должен быть расположен слева от числа, т. е. знак числа должен занимать первую же непустую колонку поля. Использование знака плюс произвольно: при отсутствии знака число считается положительным. Если внутри числа имеются пробелы, то при вводе они считаются нулями.

При выводе информации число будет напечатано или пробито в правой части поля. Это относится к тому случаю, когда в спецификации поля предусмотрено больше позиций, чем это требуется для данного числа. Если в спецификации предусмотрено меньше позиций, чем требуется для данного числа, то при выводе информации будут потеряны его знак и старшие цифры, причем никакого указания на то, что предусмотренное поле оказалось слишком «коротким», не дается. Знак плюс при выводе информации не печатается и не пробивается.

Для всех шести типов спецификаций можно сделать так, чтобы одна и та же спецификация применялась к нескольким последовательным полям. Для этого достаточно написать перед спецификацией поля число повторений.

Спецификация типа I (целое число)

В общем виде эта спецификация пишется как Iw. Буква I определяет перевод двоичного целого числа внутри ЭЦВМ в десятичное целое число, которое будет выведено из машины, или перевод десятичного целого числа, пробитого на перфокарте, в двоичное число внутри ЭЦВМ. Общее

число позиций, занимаемых числом, включая знак и все пробелы, равно w. Использование десятичной точки в спецификации типа I не допускается.

Спецификация типа F (действительное число с фиксированной запятой)

В общем виде эта спецификация пишется как F$w.d$. Буква F означает, что двоичное число с плавающей запятой, записанное в ячейке памяти ЭЦВМ, будет преобразовано в десятичное число без порядка (при выводе информации); обратно, десятичное число без порядка будет переведено в двоичное число с плавающей запятой (при вводе информации). Полное число позиций, занимаемых полем, включая знак, десятичную точку и все пробелы, равно w. Если десятичная точка при вводе информации не пробита, то количество позиций, находящихся после десятичной точки, считается равным d. При выводе информации десятичная точка будет расположена так, что после нее останется d позиций.

Перфорация десятичной точки при вводе находится на усмотрении программиста. Если десятичная точка имеется, то при вводе будет принята во внимание действительно пробитая десятичная точка, а число d будет игнорировано. Ниже приведены некоторые примеры чисел и показано, в каком виде они вводятся в машину, со спецификацией F 10.6.

Пробито на перфокарте	Введено в ЭЦВМ
+12345678	+12.345678
1234.5678	+1234.5678
—1.2345678	—1.2345678
012345678	+12.345678
—1.2	—1.2
+1234567	+1.234567
123	+0.000123

При *выводе* появится d десятичных цифр после десятичной точки.

Спецификация типа E (нормализованная форма записи действительного числа)

В общем виде эта спецификация пишется как E$w.d$. Буква E определяет перевод числа с плавающей запятой, записанного в ячейке памяти ЭЦВМ, в число с десятичным порядком (при выводе) и обратный перевод числа с десятичным порядком в двоичное число с плавающей запятой (при вводе). Общее количество позиций поля равно w, причем сюда включаются позиции, необходимые для знака, десятичной точки, порядка и пробелов. Количество значащих цифр мантиссы после десятичной точки равно d (позиции, занимаемые порядком, при этом не учитываются).

Использование десятичной точки при вводе произвольно. Если десятичная точка пробита, то при вводе будет принята во внимание действительно пробитая десятичная точка, а число d будет игнорировано.

Порядок в общем случае записывается в виде E $\pm$ *ее*, точно так же, как и для константы с плавающей запятой в каком-либо операторе программы. Для упрощения пробивки перфокарт допускаются некоторые сокращения.

В случае, когда показатель степени положительный, знак плюс можно опускать или заменять пробелом; порядок при этом запишется в виде E *ее* или E*ее*. Если первая цифра в показателе степени равна нулю, то ее можно опустить. Если порядок записан со знаком плюс или минус, то можно опустить букву E. Поэтому, например, для показателя степени плюс два все нижеследующие формы записи допустимы и эквивалентны: E + 02, *E*02, *E* 02, *E* + 2, *E*2, +02, +2.

Приведем другой пример. Все нижеследующие числа будут преобразованы в одно и то же число, если производить ввод с помощью спецификации поля E 14.7 (напоминаем, что если десятичная точка пробита при вводе, то число d игнорируется):

+1.2345678 E 03 1234.5678 E 0

12345678.E —4 +0.12345678 +4

При выводе информация обычно печатается в виде $\pm 0.nnE\pm$ *ее* (знаки плюс обычно заменяются пробелами).

Количество значащих цифр после десятичной точки определяется числом *d*.

Совместно со спецификацией типа E можно использовать *масштабный коэффициент*. При этом спецификация пишется в виде *s*P*n*E*w.d*, где P означает, что использован масштабный коэффициент, *s* — сам этот коэффициент, *n* — количество повторений спецификации поля. Использование масштабного коэффициента приводит к тому, что положение десятичной точки сдвигается вправо на *s* позиций, а порядок соответственно уменьшается на *s*.

При вводе масштабный коэффициент не влияет на передачу информации, если использована спецификация типа E. Если задать спецификацию типа F, то масштабный коэффициент влияет на передачу информации и при вводе и при выводе, но такое его использование нельзя считать общепринятым, скорее это исключение из правила. Когда используется масштабный коэффициент, он влияет на все стоящие после него спецификации поля в операторе FORMAT (имеются в виду спецификации типов E и F). Поэтому, если только одна спецификация должна иметь отличный от нуля масштабный коэффициент, то перед следующей спецификацией необходимо поставить нулевой коэффициент.

Отметим, наконец, что для вывода результатов вычислений на печать в большинстве случаев наиболее целесообразной является спецификация 1PE 20.7. При использовании такой спецификации десятичная точка будет располагаться между первой и второй значащими цифрами, будут напечатаны все необходимые значащие цифры и останется достаточное количество пробелов, облегчающих чтение выходных результатов.

Спецификация типа H (Холлерита)

В общем виде эта спецификация записывается как *w*H. При использовании этой спецификации *w* знаков, непосредственно следующих за буквой H, будут напечатаны или пробиты там, где это требуется, исходя из положения спецификации типа H в операторе FORMAT. Спецификация типа H отличается от других тем, что ей не должна соответствовать никакая переменная в списке оператора ввода — вывода. Вместо этого происходит вывод самого текста, который расположен в операторе FORMAT после

буквы H. В этом тексте можно использовать любой символ, используемый в ЭЦВМ, включая даже «символ» пробела. Это единственный случай, когда пробел не игнорируется. В операторе вывода не требуется никаких указаний на то, что в связанном с ним операторе FORMAT имеется текст, который будет напечатан с помощью спецификации типа H. Когда при сканировании в списке спецификаций оператора FORMAT встречается спецификация типа H, то печатается текст, следующий за буквой H, но никакая переменная из списка в операторе вывода не передается на выходное устройство. Иногда бывает даже так, что в операторе FORMAT нет никаких других спецификаций поля, кроме H. При этом в операторе вывода не должно быть никакого списка переменных, а вся выводимая информация состоит только из текста.

Очень широко спецификация типа H используется для того, чтобы задавать расстояние между строками при печати. Если список спецификаций в операторе FORMAT начинается со спецификации типа H, то первый символ не печатается, а определяет расстояние между строками печати. Когда в качестве этого символа использован пробел, то расстояние между строками будет нормальным, одинарным. Если в качестве этого символа использован нуль, то расстояние между этой строкой и предыдущей будет двойным. Если же в качестве этого символа использована единица, то перед печатанием данной строки бумага будет продвинута к началу очередной страницы[1]). То же самое можно сказать и о случае, когда выводимая информация записывается на магнитную ленту с помощью оператора WRITE OUTPUT TAPE, так как затем информация с этой магнитной ленты переписывается на бумагу с помощью какой-нибудь ЭЦВМ.

Спецификация типа X (пробел)

В общем виде эта спецификация пишется как wX. Если такая спецификация применена при вводе, то w колонок

[1]) Если в качестве этого символа использован знак плюс, т. е. если спецификация написана в виде 1H+, то та строка, которая должна печататься, будет расположена поверх предыдущей строки без интервала. Это бывает полезно при печати графиков.— *Прим. перев.*

перфокарты будут пропущены. Предполагается, что эти колонки пустые и что спецификация типа X используется просто для того, чтобы не делать соседние поля длиннее, нежели это необходимо. Однако если в этих колонках и было что-нибудь пробито, то они все равно будут игнорированы.

Если эта спецификация применена при выводе, то между теми полями, где поставлена спецификация типа X, на выходном бланке появятся w пробелов.

Спецификация типа A (алфавитно-цифровые переменные)

В общем виде эта спецификация пишется как Aw. Чтобы она имела смысл, переменная, значение которой передается с ее помощью, должна быть алфавитно-цифровой. Если в данном варианте ФОРТРАНа допускаются алфавитно-цифровые переменные, то наименования их могут быть такими же, как и для действительных переменных, но в данном случае переменным присваиваются алфавитно-цифровые «значения». Присвоить переменной алфавитно-цифровое значение можно с помощью спецификации типа A при вводе информации, а также с помощью арифметического оператора, правая часть которого представляет собой спецификацию поля типа H. Например, чтобы присвоить алфавитно-цифровой переменной HEAD значение X REAL, достаточно написать оператор

```
HEAD = 6HX REAL
```

Между словами X и REAL должен стоять пробел, этот пробел учитывается при подсчете количества символов. В значении алфавитно-цифровых переменных (не в наименовании) могут встречаться любые символы, имеющиеся в распоряжении ЭЦВМ: буквы, цифры, знаки пунктуации, символы арифметических операций и так далее.

Использование алфавитно-цифровых переменных варьируется хотя бы потому, что различные ЭЦВМ имеют различные ограничения на количество символов, из которых может составляться значение переменной. Обычно работа с алфавитно-цифровыми переменными подробно описана в инструкциях к конкретной ЭЦВМ с учетом примененного на ней варианта ФОРТРАНа.

П.1.4. ДОПОЛНИТЕЛЬНЫЕ ПРИЕМЫ ПОСТРОЕНИЯ ОПЕРАТОРА FORMAT

Точно так же, как можно повторять какую-либо спецификацию поля, написав перед ней число, означающее количество повторений, можно повторять и целую группу спецификаций поля. Для этого группа спецификаций заключается в скобки и перед скобками пишется число, означающее количество повторений. Например, предположим, что на некоторой перфокарте имеется восемь чисел, пробитых по очереди в форматах I 2 и F 10.0. Тогда оператор FORMAT можно написать в виде 4 (I 2, F 10.0) и числа будут введены в ЭЦВМ. Это не то же самое, что написать 4 I 2, 4 F 10.0, так как в этом последнем случае имелось бы в виду наличие на перфокарте четырех чисел, написанных в формате I 2, а затем четырех чисел, написанных в формате F 10·0. Использовать такую группировку спецификаций поля можно только однократно, иными словами, скобки внутри скобок не допускаются.

Если список переменных в операторе ввода—вывода охватывает больше, нежели одну перфокарту или одну строку печати на выходном бланке, то для отделения перфокарт или строк печати друг от друга используется косая черта (/). Например, предположим, что необходимо прочесть две перфокарты с помощью одного оператора READ; на первой перфокарте пробито четырехзначное целое число, на второй — шесть действительных чисел в нормализованном виде. Оператор FORMAT можно написать так:

FORMAT (I4/6E 14.0)

Можно составить оператор FORMAT так, что первая перфокарта будет прочитана в каком-либо одном формате, а все последующие в другом формате. Например, предположим, что первая перфокарта содержит целое число и действительное число в нормализованном виде, все же остальные перфокарты — по два целых числа и по одному действительному числу в нормализованном виде. Тогда для того, чтобы первая карта была введена согласно формату I 4, E 14.0, а все последующие — согласно формату 2 I 4, E 14.0, достаточно ту группу спецификаций поля, которая соответст-

вует этим последующим перфокартам, заключить в скобки. Оператор FORMAT при этом запишется так:

```
FORMAT (I4, E 14.0/(2I4, E14.0))
```

Косая черта в общем случае означает конец одной строки или перфокарты и начало следующей. Закрывающиеся скобки оператора FORMAT означают конец строки или перфокарты. Если в операторе FORMAT имеются поставленные подряд несколько косых черт, то соответствующее количество перфокарт будет игнорировано при вводе или соответствующее количество строк на выходном бланке останется пустым (печатающее устройство просто подаст бумагу вперед на несколько интервалов). Пропуск n перфокарт при вводе или n строк печати при выводе достигается постановкой $n + 1$ косых черт подряд.

Подведем некоторые итоги тому, что было написано в этом приложении о процессе сканирования в списках переменных и спецификаций поля. Список переменных в операторе ввода — вывода определяет переменные, значения которых должны быть переданы, и их последовательность (принимая во внимание различного рода индексацию). Соответствующий оператор FORMAT определяет длину и характер поля, занимаемого каждым значением переменной, а также длину каждой перфокарты или строки печати, если их больше одной. Синхронно со сканированием в списке переменных происходит сканирование (слева направо) в списке спецификаций поля оператора FORMAT, при этом для каждой переменной отыскивается соответствующая спецификация. При сканировании в списке спецификации поля, конечно, принимаются в расчет разного рода числа, определяющие количество повторений той или иной спецификации. Если в списке спецификаций поля встречается спецификация типа H, то соответствующий текст печатается, но значения переменных из списка в операторе ввода — вывода не передаются, иными словами, на время печати текста, имеющегося в спецификации типа H, сканирование в списке переменных приостанавливается. Процесс прекращается только тогда, когда будут переданы все значения переменных, перечисленных в операторе ввода — вывода, но если после этого в операторе FORMAT еще останется спецификация типа H, то соответствующий ей текст будет

напечатан, хотя бы все значения переменных и были уже переданы. Если список спецификаций в операторе FORMAT исчерпан, а список переменных в операторе ввода — вывода еще не закончен, то передача переменных будет продолжена, а сканирование в списке спецификаций поля начнется снова с первой спецификации внутри последней группы скобок в операторе FORMAT.

Некоторые приемы построения оператора FORMAT из числа описанных в этом приложении уже применялись в книге, точнее, в нескольких программах, приведенных в последних ее главах. Эти программы приведены на на рис. 9.9, 9.11, 10.10 и в ответе к упражнению 19 гл. 11.

П.1.5. ОПЕРАЦИИ С МАГНИТНОЙ ЛЕНТОЙ

Весьма важным свойством современных ЭЦВМ является широкое применение магнитной ленты. Используется магнитная лента двояким образом.

Первое, наиболее распространенное применение магнитной ленты состоит в ускорении операций ввода и вывода. По сравнению со скоростью работы арифметического устройства машины операции ввода и вывода протекают, как правило, недопустимо медленно. (Перфокарты, например, вводятся со скоростью 10-15 карт в секунду, печать происходит со скоростью 8—12 строк в секунду.—*Прим. перев.*) Поэтому в настоящее время общепринято использовать при операциях ввода — вывода магнитную ленту для уменьшения времени простоя ЭЦВМ. В случае ввода это делается следующим образом: информация пробивается на перфокартах и переписывается на магнитную ленту с помощью специализированного устройства, не связанного с ЭЦВМ. Во время этой работы ЭЦВМ может производить некоторые другие вычисления, не имеющие отношения к нашей задаче. Когда же исходные данные и программа записаны на магнитную ленту, то информацию с этой ленты можно ввести в машину со скоростью, примерно в сто раз большей по сравнению с перфокартами.

Аналогично результаты вычислений можно записать на магнитной ленте вместо непосредственного вывода их на печать. После того как вычисления по программе закончены, ленту с выведенной информацией можно снять с ЭЦВМ

и поставить на специализированное устройство для печати. Результаты выводятся на печать, а ЭЦВМ в это время продолжает вычисления по другим программам.

Все эти операции, кажется, труднее описать, чем сделать. В действительности самому программисту не приходится иметь дело с магнитной лентой, все эти операции происходят без его участия. Использование магнитной ленты для ускорения операций ввода — вывода приводит к существенному увеличению производительности вычислительных машин.

Другая область использования магнитных лент состоит в том, что на магнитной ленте хранятся промежуточные результаты во время решения задачи. Например, иногда приходится иметь дело с очень большими массивами, которые невозможно разместить сразу в оперативной памяти ЭЦВМ. В таких случаях можно записывать промежуточные результаты на магнитную ленту по мере их получения и считывать их с магнитной ленты, когда они снова требуются для дальнейшего хода вычислений.

В зависимости от того, с какой целью применяется магнитная лента, имеются соответственно два типа операторов, описывающих операции с ней. Дело в том, что если лента используется только для хранения промежуточных результатов, то процесс записи и считывания информации может быть построен гораздо проще, нежели в том случае, когда эту ленту надлежит использовать с другим оборудованием, например с печатающим устройством.

Для считывания информации с ленты, полученной в результате переписи информации с перфокарт, используется оператор READ INPUT TAPE. Этот оператор почти аналогичен оператору READ, с одним исключением: необходимо указать номер того ленточного блока, куда установлена данная магнитная лента. Номер этого блока указывается перед номером оператора FORMAT. Например,

```
READ INPUT TAPE 5, 169, I, A, X1, SEG
```

Цифра 5 определяет, что лента установлена на блоке номер 5, во всех остальных отношениях этот оператор полностью аналогичен обычному оператору READ. Требования к оператору FORMAT, к индексации остаются теми

же самыми, и процесс сканирования происходит точно так же, как и в случае оператора READ.

Оператор WRITE OUTPUT TAPE точно так же полностью аналогичен оператору PRINT. Типичный пример записи этого оператора

WRITE OUTPUT TAPE 6, 401, (X(I), Y(I), I = 1,6)

Выбор конкретного ленточного блока определяется в основном конструкцией и режимом работы ЭЦВМ.

Для хранения на магнитных лентах промежуточных результатов вычислений используются операторы READ TAPE и WRITE TAPE. Отсутствие в этих операторах слов INPUT и OUTPUT заставляет предположить, что магнитные ленты, записанные таким образом, отличаются от тех лент, которые используются для ввода и вывода информации. В самом деле, при использовании операторов READ TAPE и WRITE TAPE не требуется использование оператора FORMAT. Типичный пример записи такого оператора

WRITE TAPE 8, ARRAY

Предполагается, что ARRAY является большим массивом; наименование его написано без индексов, так что весь он будет записан на магнитную ленту. Ленту, записанную таким образом, *нельзя использовать для вывода этого массива на печать*, потому что информация записана на ленте в том самом виде, в каком она присутствовала в ячейках памяти машины, без преобразования в пригодную для печати форму. Единственное, для чего годится записанная таким образом лента, это для того, чтобы впоследствии снова считать эту информацию, когда она потребуется для продолжения вычислений. Считывание информации производится с помощью оператора READ TAPE.

Имеются еще три оператора, используемые при работе с магнитной лентой, хотя никакая информация с помощью этих операторов и не передается. Оператор BACKSPACE n, где n по-прежнему является номером блока, отводит ленту назад на одну зону записи. Под зоной здесь имеется в виду либо информация, соответствующая одной перфокарте (строке печати), либо информация, записанная с помощью оператора WRITE TAPE. С помощью оператора END

FILE *n* на ленте записывается сигнал, по которому заканчивается работа печатающего устройства в преобразователе лента — печать. С помощью оператора REWIND *n* лента возвращается к своему началу. Обычно этот оператор ставится в конце программы, чтобы обслуживающему персоналу ЭЦВМ оставалось только снять ленту после окончания работы программы.

Следует отметить, что запись новой информации на ленту приводит к затиранию того, что было записано на ней ранее.

ПРИЛОЖЕНИЕ 2

НЕКОТОРЫЕ УПОТРЕБИТЕЛЬНЫЕ МАТЕМАТИЧЕСКИЕ ФОРМУЛЫ

А. Полиномы Чебышева

$$
\begin{aligned}
T_0(x) &= 1,\\
T_1(x) &= x,\\
T_2(x) &= 2x^2 - 1,\\
T_3(x) &= 4x^3 - 3x,\\
T_4(x) &= 8x^4 - 8x^2 + 1,\\
T_5(x) &= 16x^5 - 20x^3 + 5x,\\
T_6(x) &= 32x^6 - 48x^4 + 18x^2 - 1,\\
T_7(x) &= 64x^7 - 112x^5 + 56x^3 - 7x,\\
T_8(x) &= 128x^8 - 256x^6 + 160x^4 - 32x^2 + 1,\\
T_9(x) &= 256x^9 - 576x^7 + 432x^5 - 120x^3 + 9x,\\
T_{10}(x) &= 512x^{10} - 1280x^8 + 1120x^6 - 400x^4 + 50x^2 - 1,\\
T_{11}(x) &= 1024x^{11} - 2816x^9 + 2816x^7 - 1232x^5 + 220x^3 - 11x.
\end{aligned}
$$

Б. Степени x как функции $T_n(x)$

$$
\begin{aligned}
1 &= T_0,\\
x &= T_1,\\
x^2 &= \frac{1}{2}(T_0 + T_2),\\
x^3 &= \frac{1}{4}(3T_1 + T_3),\\
x^4 &= \frac{1}{8}(3T_0 + 4T_2 + T_4),\\
x^5 &= \frac{1}{16}(10T_1 + 5T_3 + T_5),\\
x^6 &= \frac{1}{32}(10T_0 + 15T_2 + 6T_4 + T_6),
\end{aligned}
$$

$$x^7=\frac{1}{64}(35T_1+21T_3+7T_5+T_7),$$

$$x^8=\frac{1}{128}(35T_0+56T_2+28T_4+8T_6+T_8),$$

$$x^9=\frac{1}{256}(126T_1+84T_3+36T_5+9T_7+T_9),$$

$$x^{10}=\frac{1}{512}(126T_0+210T_2+120T_4+45T_6+10T_8+T_{10}),$$

$$x^{11}=\frac{1}{1024}(462T_1+330T_3+165T_5+55T_7+11T_9+T_{11}).$$

В. Формулы для экономизации степенных рядов

$$x=T_1,$$

$$x^2=\frac{1}{2}(1+T_2),$$

$$x^3=\frac{1}{4}(3x+T_3),$$

$$x^4=\frac{1}{8}(8x^2-1+T_4),$$

$$x^5=\frac{1}{16}(20x^3-5x+T_5),$$

$$x^6=\frac{1}{32}(48x^4-18x^2+1+T_6),$$

$$x^7=\frac{1}{64}(112x^5-56x^3+7x+T_7),$$

$$x^8=\frac{1}{128}(256x^6-160x^4+32x^2-1+T_8),$$

$$x^9=\frac{1}{256}(576x^7-432x^5+120x^3-9x+T_9),$$

$$x^{10}=\frac{1}{512}(1280x^8-1120x^6+400x^4-50x^2+1+T_{10}),$$

$$x^{11}=\frac{1}{1024}(2816x^9-2816x^7+1232x^5-220x^3+11x+T_{11}).$$

Г. Приближения для некоторых элементарных функций

1. $\sin\left(\frac{\pi x}{2}\right)=C_1x+C_3x^3+C_5x^5+C_7x^7;\quad -1\leqslant x\leqslant 1,$

где

$$C_1 = \quad 1.570790988$$
$$C_3 = -0.645892663$$
$$C_5 = \quad 0.079433971$$
$$C_7 = -0.004332882$$

Ошибка ограничения: $|e_T| \leqslant 0.6 \cdot 10^{-6}$.

2. $$e^x = 1 + \cfrac{x}{-\frac{x}{2} + \left(\frac{k_0 + k_1 x^2}{1 + k_2 x^2}\right)}; \quad -\ln\sqrt{2} \leqslant x \leqslant \ln\sqrt{2},$$

где

$$k_0 = 1.00000\,00020\,967$$
$$k_1 = 0.09997\,43507\,186$$
$$k_2 = 0.01664\,11490\,538$$

Ошибка ограничения: $|e_T| \leqslant 10^{-10}$.

3. $$\ln(1+x) = k_0 + \cfrac{x}{k_1 + \cfrac{x}{k_2 + \cfrac{x}{k_3 + \cfrac{x}{k_4 + \cfrac{x}{k_5}}}}};$$

$0 \leqslant x \leqslant 1$,

где

$$k_0 = 0.00000\,00894$$
$$k_1 = 1.00000\,91365$$
$$k_2 = 2.00058\,59000$$
$$k_3 = 3.03119\,32666$$
$$k_4 = 1.07877\,48225$$
$$k_5 = 8.89527\,84060$$

Ошибка ограничения: $|e_T| \leqslant 10^{-7}$.

4. $$\operatorname{tg}\left(\frac{\pi x}{4}\right) = x\left[k_0 + \cfrac{x^2}{k_1 + \cfrac{x^2}{k_2 + \cfrac{x^2}{k_3}}}\right]; \quad -1 \leqslant x \leqslant 1,$$

где

$$k_0 = \quad 0.78539\,80289$$
$$k_1 = \quad 6.19223\,44479$$

$$k_2 = -0.65458\,87679$$
$$k_3 = 491.00139\,34779$$

Ошибка ограничения: $|e_T| \leqslant 0.2 \cdot 10^{-7}$.

5. $$\operatorname{arc\,tg} x = x \left[k_0 + \cfrac{x^2}{k_1 + \cfrac{x^2}{k_2 + \cfrac{x^2}{k_3 + \cfrac{x^2}{k_4}}}} \right]; \quad -1 \leqslant x \leqslant 1,$$

где

$$k_0 = 0.99999\,752$$
$$k_1 = -\ 3.00064\,286$$
$$k_2 = -\ 0.55703\,890$$
$$k_3 = -17.03715\,998$$
$$k_4 = -\ 0.20556\,880$$

Ошибка ограничения: $|e_T| \leqslant 0.2 \cdot 10^{-6}$.

Д. Абсциссы и весовые коэффициенты при интегрировании по методу Гаусса

	μ	A
$n = 2$	$\pm$ 0.57735 02692	1.00000 00000
$n = 3$	$\pm$ 0.77459 66692	0.55555 55556
	0.00000 00000	0.88888 88889
$n = 4$	$\pm$ 0.86113 63116	0.34785 48451
	$\pm$ 0.33998 10436	0.65214 51549
$n = 5$	$\pm$ 0.90617 98459	0.23692 68851
	$\pm$ 0.53846 93101	0.47862 86705
	0.00000 00000	0.56888 88889
$n = 6$	$\pm$ 0.93246 95142	0.17132 44924
	$\pm$ 0.66120 93865	0.36076 15730
	$\pm$ 0.23861 91861	0.46791 39346

ОТВЕТЫ К УПРАЖНЕНИЯМ

Во многих случаях к одному и тому же упражнению есть несколько ответов. Те ответы, что приведены здесь, как правило, являются «наилучшими», но, вообще говоря, довольно трудно сказать, каков должен быть критерий «качества» ответа в каждом случае. Иногда существуют несколько одинаково «хороших» ответов, например безразлично, писать ли $C = A + B$ или $C = B + A$. Короче говоря, приведенные здесь ответы верны, но необязательно единственно возможны. Если можно показать, что другой ответ также является правильным, то при прочих равных условиях необходимо принять и этот ответ.

Глава 1

Раздел 1.4.

1. 256.; 2.56; —43 000.; 1.0 E + 12; 4.92 E — 7; —10.0; —1.E — 16.

3. 12,345.0 (запятую использовать нельзя); +234 (нет десятичной точки); 1.6 E 63 (слишком велико для большинства ЭЦВМ); 1 E — 7 (нет десятичной точки).

5. Да

6. —234. (десятичную точку использовать нельзя); 23,400 (запятую использовать нельзя); 1 E 12 (нельзя использовать букву E); +100000000000000 (слишком велико для большинства ЭЦВМ).

Раздел 1.5.

1. Целые: I, IJK; LARGE; KAPPA

Действительные: G; GAMMA; BT07TH, ZCUBED; CDC 160; DELTA; A1P4; ALGOL

Неприемлемы: GAMMA421 (слишком много символов); IJK* (* использовать нельзя); J79—12 (— использовать нельзя); R (2) 19 (скобки использовать нельзя) ZSQUARET (слишком много символов); 12AT7 (не начинается с буквы); 2N173 (не начинается с буквы); EPSILON (слишком много символов); A1.4 (десятичную точку использовать нельзя): FORTRAN (слишком много символов).

Раздел 1.6

1. а. X+Y**3
г. A+B/C
е. A+B/(C+D)
з. ((A+B)/(C+D))**2+X**2
к. 1.+X+X**2/2.+X**3/6.
л. (X/Y)** (G—1.0)

2. б. (X+2.0)/(Y+4.0) Константы можно написать в любом другом эквивалентном виде, например: (X+2.)/(Y+4.)
д. ((X+A+3.1416)/(2.0*Z))**2
ж. (X/Y)** (R—1.0)
к. A+X* (B+X* (C+D*X))

Раздел 1.8

1. а. 13.0; действительное.
б. Нуль; действительное.
д. 4; целое.
е. 4.0; действительное.
л. 1.3333332; действительное.
о. 8.0; действительное, может получиться 7.9999999 или 8.0000001
п. 5; целое.

3. а. DELTA=BETA+2.0
в. C=SQRTF (A**2+B**2) или C=SQRTF (A*A+B*B)
г. R=1.414214
ж. Y=COSF (2.*X)*SQRTF (X/2.0)
з. G=G+2.0

4. а. AREA=2.*P*R*SINF (3.1416/P)
в. ARC=2.0*SQRTF (Y**2+1.3333*X**2)
д. S=—COSF (X)** (P+1.0)/(P+1.0)
е. G=0.5 LOGF ((1.+SINF (X))/(1.—SINF (X)))

Предпочтительно написать два оператора, чтобы не вычислять дважды синус:

S=SINF (X)
G=0.5 LOGF ((1.+S)/(1.—S))
и. E=X*ATANF (X/A)—A/2.*LOGF (A**2+X**2)
м. Q=(2./(3.1416*X))**0.5*SINF (X)

Поскольку

$$\left(\frac{2}{\pi x}\right)^{\frac{1}{2}} = \sqrt{\frac{2}{\pi}} \Big/ \sqrt{x} = 0.7978/\sqrt{x},$$

то оператор запишется более компактно и вычисления потребуют меньше машинного времени.

н. Q=0.7978/SQRTF (X)*SINF (X)
о. Y=2.5065*X** (X+1.)*EXPF (—X)

Раздел 1.9

```
1.      READ 94,A,B,C
   94 FORMAT (3F10.0)
      RADICL = SQRTF (B**2 — 4*.A*C)
      X1 = (—B + RADICL)/(2.*A)
      X2 = (—B — RADICL)/(2.*A)
      PRINT 93,A,B,C,X1,X2
   93 FORMAT (5E20.8)
3.      READ 97, A, E, H, P
   97 FORMAT (4F10.0)
      X = E*H*P/(SINF (A)*(H**4/16 + H**2*P**2))
      PRINT 98, A, E, H, P, X
   98 FORMAT (5E20.8)
5.        READ 300, A, X, S
  300 FORMAT (3F10.0)
      Y = SQRTF (X**2 — A**2)
      Z = X*S/2. — A**2/2. *LOGF (ABSF (X + S))
      PRINT 50,A,X,S,Y,Z
   50 FORMAT (5E20.8)
```

Раздел 1.10

```
1.        IF (X — Y) 11, 11, 12
   11 BIG = Y
      GO TO 13
   12 BIG = X
   13 Продолжение программы.
4. 400 IF (THETA — 6.2832) 402, 401, 401
   401 THETA = THETA — 6.2832
      GO TO 400
   402 Продолжение.
5.        IF (ABSF (XREAL) — 1.) 16, 82, 82
   16 IF (ABSF (XIMAG) — 1.) 81, 82, 82
6.        IF (SQRTF (XREAL**2 + XIMAG**2) — 1.) 187, 459, 459
10а.      IF (X — 0.999)67,63,21
   21 IF (X — 1.001) 63, 63, 67
10б.      IF (ABSF (X — 1.) — 0.001) 63, 63, 67
```

Раздел 1.11

```
1.        READ 20, ANNERN
          IF (ANNERN — 2000.00) 40, 40, 50
   40 TAX = 0.00
      GO TO 100
   50 IF (ANNERN — 5000.00) 60, 60, 70
   60 TAX = 0.02* (ANNERN — 2000.00)
      GO TO 100
   70 TAX = 60.00 + 0.05* (ANNERN — 5000.00)
  100 PRINT 30,ANNERN, TAX
```

```
      30 FORMAT (2E20.8)
      20 FORMAT (F10.0)
         STOP
         END
3.       X=1.0
      61 Y=16.7*X+9.2*X**2—1.02*X**3
         PRINT 62, X, Y
      62 FORMAT (2E20.8)
         IF (X—9.9) 63, 64, 64
      63 X=X+0.1
         GO TO 61
      64 STOP
         END
```

4. По первому методу достаточно заменить оператор IF в программе из упражнения 3 следующим оператором:

```
IF (ABSF (X—9.9)—1.E—4) 63, 64, 64
```

По второму методу количество прохождений цикла подсчитывается с помощью целой переменной. Программа для вычисления 90 значений Y приведена ниже.

```
         X=1.0
         N=1
      61 Y=16.7*X+9.2*X**2—1.02*X**3
         PRINT 62,X,Y
      62 FORMAT (2E20.8)
         IF (N—90) 63, 64, 64
      63 X=X+0.1
         N=N+1
         GO TO 61
      64 STOP
         END
```

Глава 2

1. Напишем $e=iR$,

где e — вольты,
i — амперы,
R — омы,

и положим соответствующие абсолютные ошибки равными e_e e_i, e_R. Тогда из условий задачи имеем

$$e_R=10\% \text{ от } 10 \text{ ом}=\pm 1,$$

$$e_i=\pm 0.1 \text{ а}.$$

Тогда

$$e_e=Re_i+ie_R=10\cdot(\pm 0.1)+2.0\cdot(\pm 1)$$

и

$$|e_e|\leqslant 10\cdot 0.1+2.0\cdot 1=3 \text{ в},$$

поэтому

$$e = 20 \pm 3 \text{ в}.$$

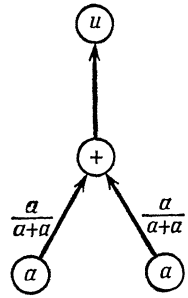
Относительная ошибка составляет

$$\frac{e_e}{e} \leqslant \left| \frac{\pm 3}{20} \right| = 0.15.$$

Таким образом

$$\left| \frac{e_e}{e} \right| \leqslant \left| \frac{e_i}{i} \right| + \left| \frac{e_R}{R} \right| = \left| \frac{\pm 0.1}{2} \right| + |\pm 10\%| = 0.05 + 0.1 = 0.15$$

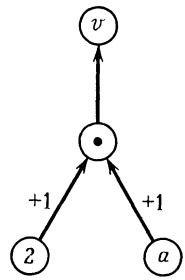
5. $u = a + a$



$$\frac{e_u}{u} = \frac{a}{a+a} i_a + \frac{a}{a+a} i_a + r = i_a + r.$$

$$\left| \frac{e_u}{u} \right| \leqslant |i_a| + |r| \leqslant 10^{-t+1}.$$

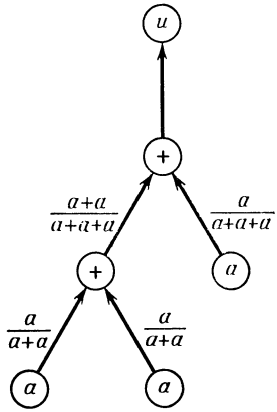
$$v = 2a.$$



$$\frac{e_v}{v} = +1\cdot i_2 + 1\cdot i_a + r = +1\cdot 0 + 1\cdot i_a + r.$$

$$\left|\frac{e_v}{v}\right| \leqslant |i_a| + |r| \leqslant 10^{-t+1}.$$

6. $u = a + a + a.$

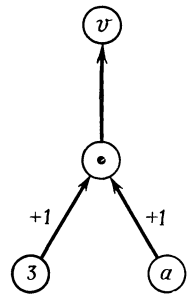


$$\frac{e_u}{u} = \frac{a+a}{a+a+a}\left(\frac{a}{a+a}\,i_a + \frac{a}{a+a}\,i_a + r_1\right) +$$

$$+\frac{a}{a+a+a}\,i_a + r_2 = \frac{1}{3}\,i_a + \frac{1}{3}\,i_a + \frac{2}{3}\,r_1 + \frac{1}{3}\,i_a + r_2.$$

$$\left|\frac{e_u}{u}\right| \leqslant |i_a| + \left|\frac{2}{3}\,r_1\right| + |r_2| \leqslant \frac{8}{3}\cdot 5\cdot 10^{-t}$$

$v = 3a.$



$$\frac{e_v}{v} = +1\cdot i_3 + 1\cdot i_a + r = 1\cdot 0 + i_a + r.$$

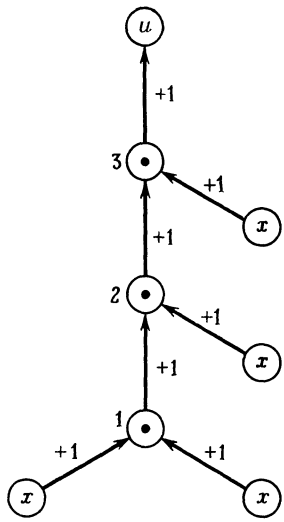
$$\left|\frac{e_v}{v}\right| \leqslant |i_a| + |r| < 2\cdot 5\cdot 10^{-t}.$$

Если $a = 0.6992$, то $a+a = 1.3984$, что округляется до 1.398, и $(a+a)+a = 1.0972$, что округляется до $2.097 = u$. С другой стороны, $3a = 2.0976$, что округляется до $2.098 = v$.

8. $$u = x\cdot(x\cdot(x\cdot x)).$$

$$\frac{e_u}{u} = i_x + i_x + r_1 + i_x + r_2 + i_x + r_3.$$

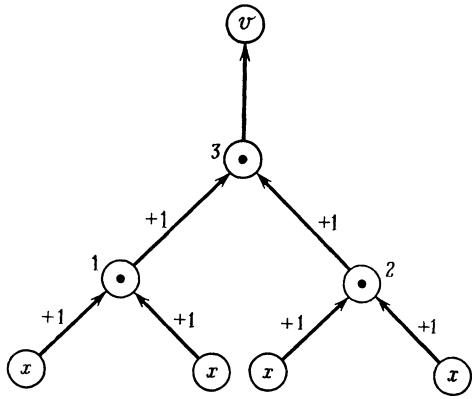
$$\left|\frac{e_u}{u}\right| \leqslant |4i_x| + |r_1| + |r_2| + |r_3| \leqslant 7\cdot 5\cdot 10^{-t}.$$



$$v = (x^2)^2.$$

$$\frac{e_v}{v} = i_x + i_x + r_1 + i_x + i_x + r_2 + r_3.$$

$$\left|\frac{e_v}{v}\right| \leqslant |4i_x| + |r_1| + |r_2| + |r_3| \leqslant 7\cdot 5\cdot 10^{-t}.$$



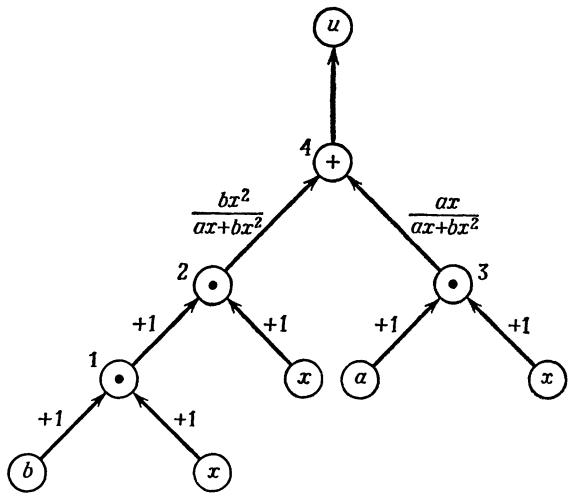
11. $u = ax + bx^2$.

Считая, что ошибки исходной информации равны нулю,

$$\frac{e_u}{u} = \frac{bx^2}{ax+bx^2}(r_1+r_2) + \frac{ax}{ax+bx^2} r_3 + r_4.$$

$$\left|\frac{e_u}{u}\right| \leqslant \left|\frac{a}{a+bx} r_3\right| + \left|\frac{bx}{a+bx} r_1\right| + \left|\frac{bx}{a+bx} r_2\right| + |r_4|.$$

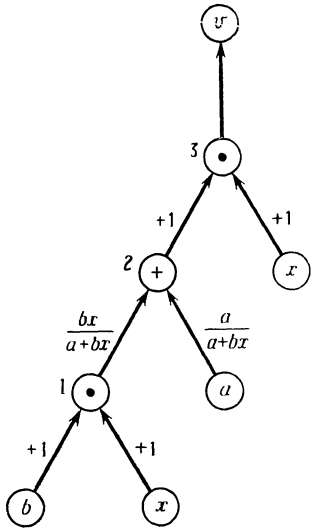
$$\left|\frac{e_u}{u}\right| \leqslant 5\cdot 10^{-t} + \left|\frac{bx}{a+bx}\right| \cdot 5\cdot 10^{-t} + 5\cdot 10^{-t}.$$



$$v = x \cdot (a + bx).$$

$$\frac{e_v}{v} = \frac{bx}{a+bx} \cdot r_1 + r_2 + r_3.$$

$$\left|\frac{e_v}{v}\right| \leqslant \left|\frac{bx}{a+bx}\right| \cdot 5 \cdot 10^{-t} + 5 \cdot 10^{-t} + 5 \cdot 10^{-t}.$$



Пусть символ $\simeq$ означает округленный результат. Тогда

$$bx = 0.29885994 \simeq 0.2989$$
$$bx^2 = 0.12858678 \simeq 0.1286$$
$$ax = 0.32802750 \simeq 0.3280$$
$$ax + bx^2 = 0.4566 = u$$
$$a + bx = 1.0614 \simeq 0.1061 \cdot 10^1$$
$$x(a + bx) = 0.4564422 \simeq 0.4564 = v$$

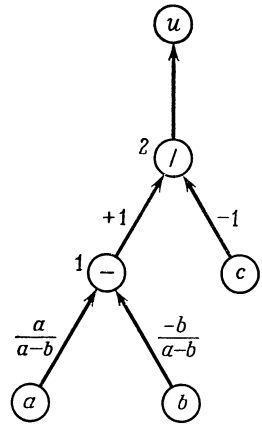
Точный ответ равен 0.456597

16. $$u = \frac{a-b}{c}.$$

Считая, что ошибки исходной информации равны нулю,

$$\frac{e_u}{u} = r_1 + r_2.$$

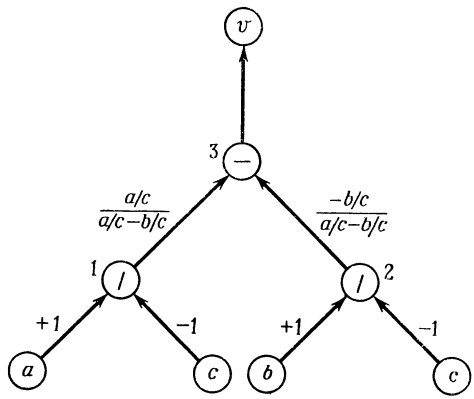
$$\left|\frac{e_u}{u}\right| \leqslant |r_1| + |r_2| \leqslant 10^{-t+1}.$$



$$v=\frac{a}{c}-\frac{b}{c};\quad \frac{e_v}{v}=\frac{a/c}{a/c-b/c}\,r_1-\frac{b/c}{a/c-b/c}\,r^2+$$

$$+r_3=\frac{a}{a-b}\,r_1-\frac{b}{a-b}\,r_2+r_3.$$

$$\left|\frac{e_v}{v}\right|\leqslant\left|\frac{a}{a-b}\,r_1\right|+\left|\frac{b}{a-b}\,r_2\right|+|r_3|.$$



Если $a\approx b$, то последнее выражение приближенно равно

$$\frac{2a}{a-b}\cdot 5\cdot 10^{-t}+5\cdot 10^{-t};$$

это выражение может стать весьма большим, если $a-b$ мало.

$$u=\frac{a-b}{c}=\frac{(0.41-0.36)}{0.70}=\frac{0.05}{0.70}=0.071428\simeq 0.71\cdot 10^{-1}.$$

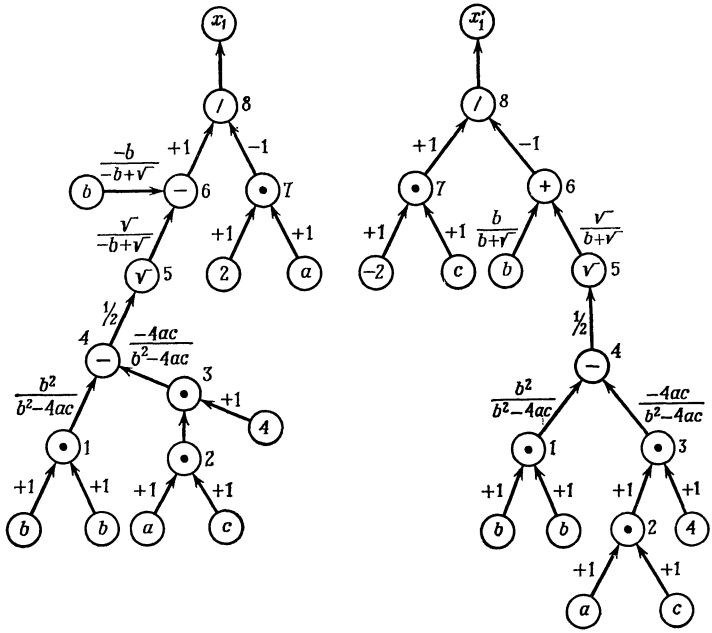
$$v=\frac{a}{c}-\frac{b}{c}=\frac{0.41}{0.70}-\frac{0.36}{0.70}=0.5857-0.51428\simeq$$

$$\simeq 0.59-0.51=0.80\cdot 10^{-1}.$$

18.

$$x_1=\frac{-b+\sqrt{b^2-4ac}}{2a};$$

$$x_1'=\frac{-2c}{b+\sqrt{b^2-4ac}}.$$



$$\frac{e_{x_1}}{x_1}=\frac{\sqrt{b^2-4ac}}{-b+\sqrt{b^2-4ac}}\left[\frac{1}{2}\left(\frac{b^2}{b^2-4ac}r_1-\frac{4ac}{b^2-4ac}\times\right.\right.$$

$$\left.\left.\times(r_2+r_3)+r_4\right)+r_5\right]+r_6-r_7+r_8;$$

при $b^2\gg 4ac$ приближенно получаем

$$\left|\frac{e_{x_1}}{x_1}\right|\leqslant\frac{b}{-b+\sqrt{b^2-4ac}}\left(\frac{|r_1|+|r_4|}{2}+|r_5|\right);$$

это выражение может стать весьма большим.

$$\frac{e_{x_1'}}{x_1'}=\frac{-\sqrt{b^2-4ac}}{b+\sqrt{b^2-4ac}}\left[\frac{1}{2}\left(\frac{b^2}{b^2-4ac}r_1-\frac{4ac}{b^2-4ac}(r_2+r_3)+\right.\right.$$
$$\left.\left.+r_4\right)+r_5\right]-r_6+r_7+r_8;$$

при $b^2\gg 4ac$ приближенно получаем

$$\left|\frac{e_{x_1'}}{x_1'}\right|\leqslant\frac{1}{2}\left[\frac{1}{2}(|r_1|+|r_4|)+|r_5|\right]+|r_6|+|r_7|+|r_8|.$$

$$x_1=\frac{-0.4002+\sqrt{0.16016004-0.00032}}{2}\simeq$$
$$\simeq\frac{-0.4002+\sqrt{0.1602-0.0003}}{2}\simeq$$
$$\simeq\frac{-0.4002+0.3999}{2}=-0.1500\cdot 10^{-3}.$$

$$x_1'=\frac{-0.00016}{0.4002+0.3999}=0.2000\cdot 10^{-3}=\text{точный корень.}$$

Глава 3

1. а. 4 члена (до $x^7/7!$ включительно); 6 членов.

б. 6 членов (до $x^9/9$ включительно); 13 членов.

6. $a_0=a_2=a_3=a_4=0;\quad a_1=1.$

8. а. $a_0=\frac{2}{\pi};\quad a_1=0;\quad a_2=\frac{-4}{3\pi};\quad a_3=0.\quad a_4=\frac{-4}{15\pi}.$

б. $\sqrt{1-x^2}=\frac{2}{15\pi}(23-4x^2-16x^4).$

в. При $x=0.5$ ряд Чебышева дает 0.891268, точное значение равно 0.866025; ряд Тейлора записывается в следующем виде: $\sqrt{1-x^2}=1-x^2-3x^4$, что при $x=0.5$ дает 0.5625

10. $T_0^*(x)=1;\quad T_1^*(x)=2x-1;$

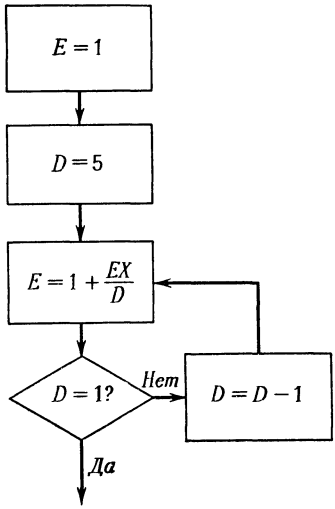
$T_2^*(x)=8x^2-8x+1;\quad T_3^*(x)=32x^3-48x^2+18x-1.$

12. $$e^x\simeq 1+\left(1+\frac{7}{64\cdot 7!}\right)x+\frac{x^2}{2!}+$$
$$+\frac{1}{3!}\left(1-\frac{56}{64\cdot 7!}\right)x^3+\frac{x^4}{4!}+\frac{1}{5!}\left(1+\frac{112}{64\cdot 7!}\right)x^5+\frac{x^6}{6!}.$$

19. а. EX = 1.0 + X* (1.0 + X* (0.5 + X*(0.166667 + X*(0.0416667 + X* (0.00833333 + X*0.00138889))))).

б. EX = 1.0 + X* (1.0 + X/2.0* (1.0 + X/3.0*(1.0 + X/4.0*(1.0 + X/5.0* (1.0 + X/6.0))))).

20.




```
      E = 1.0
      D = 5.0
   29 E = 1.0 + E*X/D
      IF (D — 1.0) 34, 34, 47
   47 D = D — 1.0
      GO TO 29
   34 Продолжение программы
```

26. SINE = —3.66519143* (X — 11.47221432/ (X + 8.03055026/X))

28. $b_1 = \frac{\pi}{2}$; $b_3 = -\frac{31}{294}\left(\frac{\pi}{2}\right)^3$;

$c_2 = \frac{3}{49}\left(\frac{\pi}{2}\right)^2$; $c_4 = \frac{11}{5880}\left(\frac{\pi}{2}\right)^4$,

31.
$$k_1 = a;$$
$$k_2 = \frac{1}{b - ad};$$
$$k_3 = \frac{(b-ad)^2}{(b-ad)\,d - c};$$
$$k_4 = \frac{(b-ad)\,d - c}{(b-ad)\,c}.$$

Глава 5

1. 0.7071. Нет, потому что для x, близкого к $+0.5$, $f'(x)$ больше 1.

4.
$$x_{n+1} = x_n - \frac{x_n^3 - c}{3x_n^2} = \frac{2x_n^3 + c}{3x_n^2}.$$

7. Поскольку интересующий нас корень $x \neq 0$, разделим уравнение на x для упрощения: $0.1x - \log x = 0$. Если теперь перейти к натуральным логарифмам для взятия производной, то итерационная формула запишется в виде

$$x_{n+1} = x_n - \frac{x_n - 10 \log_{10} x_n}{1 - \dfrac{4.3429}{x_n}}.$$

Корень равен $x = 1.3713$.

9. 1.100; -2.300; 2.673

12. $x = 2.753$; $|\varepsilon| \simeq 0.127$

21.
```
      X=A
   26 XNEW=(X**2+A)/(2.0*X)
      IF (ABSF (X-XNEW)-0.0001) 21, 24, 24
   24 X=XNEW
      GO TO 26
   21 SQRTA=XNEW
```

22.
```
      READ8, X0, A0, A1, A2, A3, A4, A5, A6
    8 FORMAT (8F10.0)
      FXO=A0+X0* (A1+X0* (A2+X0* (A3
     +X0* (A4+X0* (A5+X0*A6)))))
      PRINT9, A0, A1, A2, A3, A4, A5, A6, X0, FX0
    9 FORMAT (9F15.6)
```

33. $x = 1.932$; $y = 0.517$; да.

Глава 6

9. а. 1.00018

б. 1.08193

в. 1.00491

г. 1.58190

Для повышения точности результата лучше интегрировать в обратную сторону (от 10 к нулю), так как при этом уменьшается ошибка округления.

10. а.
```
      GAUSS = 5.0*EXPF (-5.0)* (0.17132449
     *(EXPF (5.0*0.93246951)
     + EXPF (-5.0*0.93246951)) + 0.36076157
     *(EXPF (5.0*0.66120939) + EXPF (-5.0*0.66120939))
     + 0.46791393*(EXPF (5.0*0.23861919)
     + EXPF (-5.0*0.23861919)))
```

б.
```
      I = 1
      SUM = EXPF (-10.0)
   23 X = I
      SUM = SUM + 2.0*EXPF (-10.0 + X)
      I = I + 1
      IF (I - 10) 23, 39, 39
   39 TRAP = 0.5* (SUM + 1.0)
```

в.
```
      I = 1
      SUM = EXPF (-10.0)
   23 X = I
      SUM = SUM + 4.0*EXPF (-10.0 + X)
      SUM = SUM + 2.0*EXPF (-9.0 + X)
      I = I + 2
      IF (I - 9) 23, 39, 39
   39 SIMP = 0.33333333* (1.0 + SUM + 4.0*EXPF (-1.0))
```

г.
```
      I = 0
      RECT = 0.
   23 X = I
      RECT = RECT + EXPF (-10.0 + X)
      I = I + 1
      IF (I - 10) 23, 39, 39
```

39 Продолжение программы.

13. На фиг. 6.7 замените операторы 32—61 следующими операторами:

```
32 END1 = 1.0/ (A**5* (EXPF (1.432/ (T*A)) — 1.0))
   END2 = 1.0/ (B**5* (EXPF (1.432/T*B)) — 1.0))
   BMH = 1.0/ ((B — H)**5* (EXPF(1.432/T* (B — H))) 1.0))
   SIMP = 64.77*H/3.0* (4.0* (SUM4 + BMH) +
        + 2.0*SUM2 + END1 + END2)/T**4
   TRAP = 64.77*H/2.0* (2.0* (SUM4 + SUM2 +
        + BMH) + END1 + END2)/T**4
   PRINT 61, T, SIMP, TRAP
61 FORMAT (3E20.8)
```

29. а. 17, 43

б. 7, 30, 53

в. 4, 20, 40, 56

Глава 7

1.
```
DIMENSION X (3)
DIST = SQRTF (X (1)**2 + X (2) **2 + X (3) **2)
```

3.
```
DIMENSION A (2, 2), B (2, 2), C (2, 2)
C (1, 1) = A (1, 1) *B (1, 1) + A (1, 2) *B (2, 1)
C (1, 2) = A (1, 1) *B (1, 2) + A (1, 2) *B (2, 2)
C (2, 1) = A (2, 1) *B (1, 1) + A (2, 2) *B (2, 1)
C (2, 1) = A (2, 1) *B (1, 2) + A (2, 2) *B (2, 2)
```

5. Без оператора DO:

```
      DIMENSION A (30), B (30)
      I = 1
      D = 0.0
   23 D = D + (A (I) — B (I)) **2
      I = I + 1
      IF (I — 30) 23, 23, 39
   39 D = SQRTF (D)
```

С оператором DO:

```
      DIMENSION A (30), B (30)
      D = 0.0
      DO 23 I = 1, 30
```

```
      23 D=D+(A(I)-B(I))**2
         D=SQRTF(D)
```

6. Без оператора DO:

```
         DIMENSION X(50), DX(49)
         I=1
      23 DX(I)=X(I+1)-X(I)
         I=I+1
         IF(I-49)23, 23, 39
      39 Продолжение программы
```

С оператором DO:

```
         DIMENSION X(50), DX(49)
         DO 23 I=1, 49
      23 DX(I)=X(I+1)-X(I)
```

9.

```
DIMENSION Y(50)
S=Y(I)+U*(Y(I+1)-Y(I-1))/2.0
      +U**2*(Y(I+1)-2.0*Y(I)+Y(I-1))/2.0
```

11. Без оператора DO:

```
         DIMENSION A(7), B(7)
         READ 21, A
      21 FORMAT(7F10.0)
         READ 21, B
         SUM=0.0
         I=1
      23 SUM=SUM+A(I)*B(I)
         I=I+1
         IF(I-8)23, 39, 39
      39 ANORM=SQRTF(SUM)
         PRINT 22, ANORM
      22 FORMAT(E20.8)
```

С оператором DO:

```
         DIMENSION A(7), B(7)
         READ 21, A
      21 FORMAT(7F10.0)
         READ 21, B
         SUM=0.0
         DO 23 I=1, 7
      23 SUM=SUM+A(I)*B(I)
         ANORM=SQRTF(SUM)
         PRINT 22, ANORM
      22 FORMAT(E20.8)
```

13.
```
      DIMENSION M (20)
      DO 23 I = 1, 20
   23 M (I) = I*M (I)
```

14.
```
      DIMENSION R (40), S (40), T (40)
      DO 32 I = 1, M
   32 T (I) = R (I) + S (I)
```

16.
```
      DIMENSION F (50)
      MM 1 = M — 1
      DO 43 I = 2, MM 1
   42 F (I) = (F (I — 1) + F (I) + F (I + 1))/3.0
```

17.
```
      DIMENSION B (50)
      BIGB = B (1)
      NBIGB = 1
      DO 40 I = 2, 50
      IF (BIGB — B (I)) 60, 40, 40
   60 BIGB = B (I)
      NBIGB = I
   40 CONTINUE
```

20.
```
      DIMENSION A (15, 15), X (15), B (15)
      DO 61 I = 1, 15
      B (I) = 0.0
      DO 61 J = 1, 15
   61 B (I) = B (I) + A (I, J) *X (J)
```

22.
```
      DIMENSION RST (20, 20)
      DPROD = RST (1, 1)
      DO 1 I = 2, 20
    1 DPROD = DPROD *RST (I, I)
```

23.
```
      DO 67 I = 100, 300
      X = I
      X = X/100.0
      Y = 41.926 *SQRTF (1.0 + X*X) + X** 0.333333 *EXPF (— X)
   67 PRINT 68, X, Y
   68 FORMAT (2 E 20.8)
```

27. Возможно много вариантов. Вот один из них

```
      SUM = 0.0
      DO 93 I = 12, 48, 2
      X = I
      X = X/10.0
   93 SUM = SUM + SQRTF (1.0 + X** 2) *EXPF (X)
      TRAP = 0.1* (SQRTF (2.0)* EXPF (— 1.0) + 2.0* SUM
                  + SQRTF (26.0)* EXPF (— 5.0))
```

Другой вариант

```
      SUM = 0.0
      DO 93 I = 2, 20
      X = I
      X = 0.8 + 0.2*X
   93 SUM = SUM + SQRTF (1.0 + X** 2)* EXPF (— X)
      TRAP = 0.1* (SQRTF (2.0)* EXPF (— 1.0) + 2.0* SUM
     + SQRTF (26.0)* EXPF (— 5.0))
```

31. Замените оператор 41 и всесмь следующих за ним операторов на

```
      B = A (1)
      C = B
      DO 40 I = 2, 4
      B = A (I) + Z*B
   40 C = B + Z*C
      B = A (5) + Z*B
```

34.

```
      DIMENSION GRAPH (41)
      X = 0.0
  100 S = SINF (X)
      I = 20.0* (S + 1.05)

      GRAPH (I) = 1HX
      PRINT 101, GRAPH
  101 FORMAT (1H , 41 A 1)
      GRAPH (I) = 1H
      X = X + 0.2
      IF (X — 7.0) 100, 100, 102
  102 STOP
      END
```

Глава 8

1. $x = 3, \quad y = 6, \quad z = -1.$

3. $x = 3.000, \quad y = -2.000, \quad z = 6.000$

6. $x = 1 + i, \quad y = 2 - i.$

8. Без перестановки уравнений $x = -1.400$, $y = 0.7406$

С перестановкой уравнений $x = -1.590$, $y = 0.7409$

14. $x = 2.222, \quad y = 3.333, \quad z = 4.444$

17. Без перестановки $\delta_x = 2.5 \cdot 10^{-2}$; $\delta_y = 1 \cdot 10^{-2}$.

С перестановкой $\delta_x = 1.5 \cdot 10^{-2}$; $\delta_y = 1 \cdot 10^{-2}$.

Разница свидетельствует о том, что эти числа представляют собой верхние границы возможных ошибок; в действительности ошибка

может быть меньше своей верхней границы. Ясно также, что это не *минимальные* границы, которые были бы значительно точнее.

18. а. $\delta_x = 3.1 \cdot 10^{-2}$; $\delta_y = 1.8 \cdot 10^{-2}$; $\delta_z = 0.8 \cdot 10^{-2}$
б. $\delta_x = 28.4 \cdot 10^{-2}$; $\delta_y = 16.3 \cdot 10^{-2}$; $\delta_z = 7.6 \cdot 10^{-2}$
в. $\delta_x = 31.5 \cdot 10^{-2}$; $\delta_y = 18{,}1 \cdot 10^{-2}$; $\delta_z = 8.4 \cdot 10^{-2}$
г. $\delta_x = 12.0$; $\delta_y = 6.9$; $\delta_z = 3.2$

20. $x = 1.003$; $y = -2.990$; $z = 3.994$

27.
```
I I = NROW (I)
KK = NROW (K)
A (I I, J) = A (I I, J) — FACTOR*A (KK, J)
```

37. Поскольку

$$\sum_{i=1}^{N} i = \frac{N(N-1)}{2},$$

$$\sum_{i=1}^{N} i^2 = \frac{N(N+1)(2N+1)}{6},$$

то имеем

$$Na + \frac{N(N-1)}{2} b = \sum y;$$

$$\frac{N(N-1)}{2} a + \frac{N(N+1)(2N+1)}{6} b = \sum xy.$$

38.

$$N \log a + \sum x_i b = \sum \log y_i;$$

$$\sum x_i \log a + \sum x_i^2 b = \sum (x_i \log y_i).$$

Глава 9

1.
```
DENOMF (X) = X**2 + SQRTF (1.0 + 2.0*X + 3.0*X**2)
ALPHA = (6.9 + Y)/DENOMF (Y)
BETA = (2.1*Z + Z**4)/DENOMF (Z)
GAMMA = SINF (Y)/DENOMF (Y**2)
DELTA = 1.0/DENOMF (SINF (Y))
```

3.
```
S 34 F (X, A) = SQRTF (X**2 — A**2)
SFK = 0.5* (V*S 34 F (V, R) —
          — R**2*LOGF(ABSF (V + S 34 F (V, R))))
PSB = S 34 F (X (I), B) **7/7.0 + 2.0*B**2*
        S 34 F (X (I), B) **5/5.0
      + B**4*S 34 F (X (I), B) **3/3.0
```

В обоих случаях лучше использовать два оператора: один — для вычисления функции, другой — для вычисления выражения.

```
5.  FUNCTION Y (X)
    IF (X) 10, 11, 12
 10 Y = 1.0 + SQRTF (1.0 + X*X)
    RETURN
 11 Y = 0.0
    RETURN
 12 Y = 1.0 — SQRTF (1.0 + X*X)
    RETURN
    END
    F = 2.0 + Y (A + Z)
    G = (Y (X (K)) + Y (X (K + 1)))/2.0
    H = Y (COSF (6.2832*X)) + SQRTF (1.0 + Y (6.2832*X))
7.  FUNCTION SUMNR (A, K)
    DIMENSION A (20, 20)
    SUMNR = 0.0
    DO 69 I = 1, 20
 69 SUMNR = SUMNR + ABSF (A (K, I))
    SUMNR = SUMNR — ABSF (A (K, K))
    RETURN
    END
9.  SUBROUTINE AVERNZ(A, N, AVER, NZ)
    DIMENSION A(50)
    AVER = 0.0
    NZ = 0
    DO 19 I = 1, N
    AVER = AVER + A(I)
    IF(A(I))19, 18, 19
 18 NZ = NZ + 1
 19 CONTINUE
    AN = N
    AVER = AVER/AN
    RETURN
    END
    CALL AVERNZ(ZETA, 20, ZMEAN, NZCNT)
11. DEQF(X, Y) = Y — 2.0*EXPF( — X)
    DEQF(X, Y) = 2.0*X + (X**2 — Y)*SINF(X)/COSF(X)
    DEQF(X, Y) = Y**3 — Y/X
    DEQF(X, Y) = COSF(X) — SINF(X) — Y
    DEQF(X, Y) = 2*Y/X + X**2*EXPF(X)
```

13.
```
FUNCTION YNEXT(X, Y, H)
DEQF(X, Y) = Y — 2.0*EXPF(—X)
AK1 = DEQF(X, Y)
AK2 = DEQF(X + H/2.0, Y + H*AK1/2.0)
AK3 = DEQF(X + H/2.0, Y + H*AK2/2.0)
AK4 = DEQF(X + H, Y + H*AK3)
YNEXT = Y + H/6.0*(AK1 + 2.0*(AK2 + AK3) + AK4)
RETURN
END
```

Глава 10

2.

а. 31.64
б. 34.87
в. 35.85
г. 36.64
д. 37.07
е. 36.85
ж. 37.07
з. 36.85
и. 37.50
к. 36.82
л. 36.69
м. 36.76

11. Метод Эйлера

```
      DEQF(X, Y) =
      READ 16, X, Y, H, XLAST
   16 FORMAT(4F 10.0)
      PRINT 17, X, Y, H
   17 FORMAT(1P3E20.7)
   20 Y = Y + H*DEQF(X, Y)
      X = X + H
      PRINT 17, X, Y
      IF(X — XLAST)20, 21, 21
   21 STOP
      END
```

Для исправленного метода Эйлера замените оператор 20 следующими двумя операторами:

```
   20 YP = DEQF(X, Y)
      Y = Y + H/2.0*(YP + DEQF(X + H, Y + H*YP))
```

Для модифицированного метода Эйлера замените оператор 20 следующими двумя операторами:

```
20 YP=DEQF(X, Y)
   Y=Y+H*DEQF(X+H/2.0, Y+H*YP/2.0)
```

Метод Рунге—Кутта четвертого порядка:

```
   DEQF(X, Y)=
   READ 16, X, Y, H, XLAST
16 FORMAT(4F10.0)
   H2=H/2.0
   PRINT17, X, Y, H
17 FORMAT(1P3E20.7)
20 AK1=DEQF(X, Y)
   AK2=DEQF(X+H2, Y+H2*AK1)
   AK3=DEQF(X+H2, Y+H2*AK2)
   AK4=DEQF(X+H, Y+H*AK3)
   Y=Y+H/6.0*(AK1+2.0*(AK2+AK3)+AK4)
   X=X+H
   PRINT 17, X, Y
   IF(X—XLAST)20, 21, 21
21 STOP
   END
```

Метод прогноза-коррекции, описанный в книге:

```
   DEQF(X, Y)=
   READ 16, X, Y, H, XLAST, TOLER
16 FORMAT(5F10.0)
   PRINT 17, X, Y, H, TOLER
17 FORMAT(1P4E20.7)
   X0=X
   Y0=Y
   H2=H/2.0
   AK1=DEQF(X0, Y0)
   AK2=DEQF(X0+H2, Y0+H2*AK1)
   AK3=DEQF(X0+H2, Y0+H2*AK2)
   AK4=DEQF(X0+H, Y0+H*AK3)
   X1=X0+H
   Y1=Y0+H/6.0*(AK1+2.0*(AK2+AK3)+AK4)
   PRINT17, X1, Y1
40 X2=X1+H
   Y2=Y0+2.0*H*DEQF(X1, Y1)
   ITN=1
```

```
41 Y2NEW=Y1+H2*(DEQF(X1, Y1)+DEQF(X2, Y2))
   IF(ABSF(Y2—Y2NEW)—TOLER)68, 67, 67
67 IF(ITN—10)66, 64, 64
66 Y2=Y2NEW
   ITN=ITN+1
   GO TO 41
64 PRINT88
88 FORMAT (35 H FAILS TO CONVERGE IN 10 ITERATIONS)
   STOP
68 PRINT 17, X2, Y2
   IF(X—XLAST)20, 21, 21
20 X0=X1
   Y0=Y1
   X1=X2
   Y1=Y2
   GO TO 40
21 STOP
   END
```

17. Положим $z=y'$. Тогда

$$z'=F(x)-y,$$
$$y'=z.$$

Итерации по формуле коррекции все же требуются, так как значение y в правой части первого уравнения получено по формуле прогноза и не может быть более точным, чем значение z.

22. а.
$$\begin{aligned} A_0+A_1&=1,\\ -A_1+B_0&=1,\\ A_1&=1. \end{aligned}$$

25. а. Милна.

б.
$$\begin{aligned} A_0+A_1+A_2&=1,\\ -A_1-2A_2+B_0+B_1&=1,\\ A_1+4A_2-2B_1&=1,\\ -A_1-8A_2+3B_1&=1,\\ A_1+16A_2-4B_1&=1. \end{aligned}$$

г. Пятого.

27. а. Неявная.

б.
$$\begin{aligned} a_0+a_1&=1,\\ -a_1+b_{-1}+b_0+b_1&=1,\\ a_1+2b_{-1}-2b_1&=1,\\ -a_1+3b_{-1}+3b_1&=1,\\ a_1+4b_{-1}-4b_1&=1. \end{aligned}$$

в. Потому что уравнения вырожденны. В действительности вырожденны последние три уравнения относительно a_1, b_1 и b_{-1}.

г. $a_0 = 1 - a_1$,

$$b_{-1} = \frac{5 - a_1}{12},$$

$$b_0 = \frac{2a_1 + 2}{3},$$

$$b_1 = \frac{5a_1 - 1}{12}.$$

Глава 11

1. а.

	1	2	
1	1.333	1.667	2
2	1.667	1.333	1
	2	1	

б.

	0	0	1	
0	0.1875	0.5000	1.875	2
0	0.2500	0.6250	1.2500	2
0	0.1875	0.5000	1.875	2
	0	0	1	

в.

	0	0	0	
5	2.634	1.250	0.491	0
10	4.286	1.875	0.714	0
5	2.634	1.250	0.491	0
	0	0	0	

2.

	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2.5000	1.8731	1.3400	0.9281	0.6247	0.4041	0.2405	0.1120	0.0000
5.0000	3.6525	2.5588	1.7479	1.1668	0.7513	0.4461	0.2074	0.0000
7.5000	5.1782	3.4950	2.3380	1.5436	0.9882	0.5851	0.2717	0.0000
10.0000	6.0654	3.9052	2.5656	1.6814	1.0730	0.6344	0.2945	0.0000
7.5000	5.1782	3.4951	2.3380	1.5436	0.9883	0.5851	0.2718	0.0000
5.0000	3.6526	2.5589	1.7480	1.1670	0.7514	0.4461	0.2075	0.0000
2.5000	1.8731	1.3401	0.9282	0.6248	0.4042	0.2406	0.1120	0.0000
	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	

Разница объясняется ошибками ограничения при приближении дифференциального уравнения к разностным. Величина ошибки ограничения зависит от h и от k; увеличив вдвое количество интервалов, мы тем самым уменьшили вдвое h и k.

5. б. Вовсе нет. В действительности, если известно, что последовательные приближения отличаются друг от друга меньше чем на ε, очень трудно, если не невозможно, сказать, насколько приближение

близко к точному значению. Нужно, однако, отметить, что в практически важных случаях эта проблема редко стоит так остро.

в.	ω	Необходимое количество итераций
	0.9	44
	1.0	37
	1.1	30
	1.2	24
	1.3	18
	1.4	17
	1.5	20
	1.6	26
	1.7	34
	1.8	53
	1.9	Не сошлось после 90 итераций

г. Влияет на все значения функции внутри области. Для эллиптических уравнений это всегда справедливо.

8. Струна колеблется в виде синусоиды в течение неопределенно долгого времени. Через нулевое положение она проходит в виде прямой линии; она достигает максимального отклонения в отрицательном направлении, и в этот момент отрицательные величины отклонения очень близки по величине к исходным положительным значениям. При построении графика положения середины струны как функции времени получается косинусоида.

11. Повлияет только на некоторые точки решения. Представьте себе график решения, где ось x расположена горизонтально, а ось t — вертикально. Выберите масштабы по осям одинаковыми; в нашем случае $\lambda = 0.2$, так что один интервал по оси x должен быть равен пяти интервалам по оси t. Теперь проведите две линии под углом 45° из точки возмущения, в нашем случае из точки 13,1. На те точки решения, которые лежат вне этих линий, возмущение не подействует. Эти линии называются *характеристиками*.

12. По прошествии бесконечного времени решение будет представлять собой линейное возрастание от 100° до 200°. После 20 шагов по времени ($\lambda = 0.2$, $j = 101$) распределение температуры будет следующим:

I	Температура (в градусах)
1	100.0
2	94.44
3	89.17
4	84.46
5	80.56
6	77.72
7	76.13
8	75.95
9	77.31
10	80.27

11	84.87
12	91.08
13	98.85
14	108.07
15	118.61
16	130.32
17	143.03
18	156.55
19	170.69
20	185.14
21	200.00

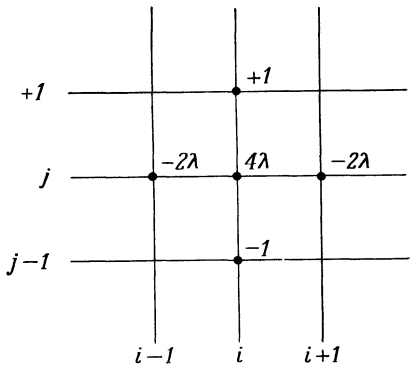
17.
```
      DIMENSION U (6,9)
   11 READ 13, I, J, VAL
   13 FORMAT (2I1, F10.0)
      IF(I) 15, 15, 14
   14 U (I, J) = VAL
      GO TO 11
   15 ITN = 1
   17 D = 0.0
      DO 18 I = 2, 5
      DO 18 J = 2, 8
      UNEW = 0.25*(U (I + 1,J) + U (I — 1,J) + U (I, J + 1) +
             U (I,J — 1))
      RESID = ABSF (UNEW — U(I, J))
      IF (RESID — D) 18. 18, 19
   19 D = RESID
   18 U(I, J) = UNEW
      IF (D — 0.001)20, 27, 27
   27 ITN = ITN + 1
      IF (ITN — 50)17, 17, 55
   55 PRINT 56
   56 FORMAT (35 HFAILS TO CONVERGE IN
                50 ITERATIONS)
      CALL EXIT
   20 PRINT 21, (((I,J , U(I, J)), J = 1,9), I = 1,6)
   21 FORMAT (5(2I3, F14.6))
      CALL EXIT
      END
```

В операторе вывода в этой программе применены некоторые приемы, рассмотренные в приложении I. Результаты вычислений можно было бы напечатать и каким-либо другим подходящим образом.

19.
```
      DIMENSION U (21,500)
   13 READ3, FLAMBD, MAXJ
    3 FORMAT (F10.0,I3)
      PRINT 12, FLAMBD
   12 FORMAT (8H1LAMBDA = , F10.4)
      B = FLAMBD*FLAMBD
      A = 2.0* (1.0 — B)
      DO 6I = 2,11
      FI = I
```

```
      U (I, 1) = 0.2* (FI — 1.0)
    6 U (I, 2) = U(I, 1)
      DO 7I = 12,20
      FI = I
      U (I, 1) = 0,2* (21.0 — FI)
    7 U (I, 2) = U (I, 1)
      DO 8J = 1,500
      U (1, J) = 0.0
    8 U (21, J) = 0.0
      DO 99J = 1,2
      PRINT39, J
   39 FORMAT (1H0, I10)
   99 PRINT40, (U (I,J), I = 1,21)
   40 FORMAT (11F11.6/10F11.6)
      DO 19J = 3, MAXJ
      DO 9I = 2,20
    9 U (I, J) = A*U (I, J — 1) + B* (U (I + 1, J — 1) +
      U (I — 1, J — 1)) — U (I, J — 2)
      PRINT 39, J
   19 PRINT 40, (U (I.J), I = 1,21)
      CALL EXIT
      END
```

27. $u_{i,\,j+1} - u_{i,\,j-1} = 2\lambda(u_{i-1,\,j} - 2u_{i,\,j} + u_{i+1,\,j})$.



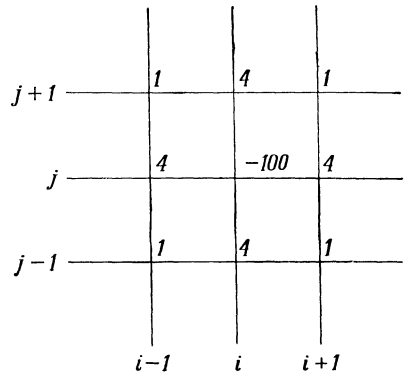
29. а. $u_{yy}^{(i+1)} = \dfrac{u_{i+1,\,j+1} - 2u_{i+1,\,j} + u_{i+1,\,j-1}}{k^2}$;

$$u_{yy}^{(i)} = \frac{u_{i,\,j+1} - 2u_{i,\,j} + u_{i,\,j-1}}{k^2};$$

$$u_{yy}^{(i-1)} = \frac{u_{i-1,\,j+1} - 2u_{i-1,\,j} + u_{i-1,\,j-1}}{k^2}.$$

б. $u_{i+1,\,j+1}+u_{i+1,\,j-1}+u_{i-1,\,j-1}+u_{i-1,\,j+1}+$
$+4\,(u_{i+1,\,j}+u_{i,\,j-1}+u_{i-1,\,j}+u_{i,\,j+1})=100u_{i,\,j}$.

в.



г. Да.

31. а. $-u^{(n)}_{i-1,\,j}+4u^{(n)}_{i,\,j}-u^{(n)}_{i+1,\,j}=u^{(n-1)}_{i,j+1}+u^{(n)}_{i,\,j-1}$,

где n — порядковый номер итерации и где вычисляются значения для j-й строки. В правых частях уравнений j-я строка не участвует и члены их можно считать постоянными.

б. Для 1-й строки

$$\begin{aligned} 4u^{(n)}_{1,1}-u^{(n)}_{2,1} &= 2+u^{(n-1)}_{1,2},\\ -u^{(n)}_{1,1}+4u^{(n)}_{2,1}-u^{(n)}_{3,2} &= 1+u^{(n-1)}_{2,2},\\ -u^{(n)}_{2,1}+4u^{(n)}_{3,1} &= 2+u^{(n-1)}_{3,2}, \end{aligned}$$

Для 2-й строки

$$\begin{aligned} 4u^{(n)}_{1,2}-u^{(n)}_{2,2} &= 1+u^{(n-1)}_{1,3}+u^{(n)}_{1,1},\\ -u^{(n)}_{1,2}+4u^{(n)}_{2,2}-u^{(n)}_{3,2} &= u^{(n-1)}_{2,3}+u^{(n)}_{2,1},\\ -u^{(n)}_{2,2}+4u^{(n)}_{3,2} &= 1+u^{(n-1)}_{3,3}+u^{(n)}_{3,1}. \end{aligned}$$

Для 3-й строки

$$\begin{aligned} 4u^{(n)}_{1,3}-u^{(n)}_{2,3} &= 2+u^{(n)}_{1,2},\\ -u^{(n)}_{1,3}+4u^{(n)}_{2,3}-u^{(n)}_{3,3} &= 1+u^{(n)}_{2,2}\\ -u^{(n)}_{2,3}+4u^{(n)}_{3,3} &= 2+u^{(n)}_{3,2}. \end{aligned}$$

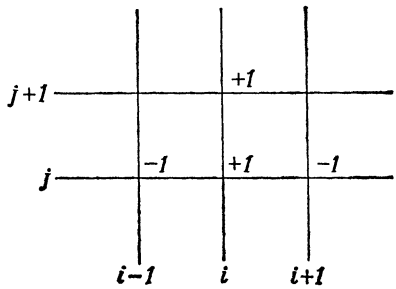
Заметим, что

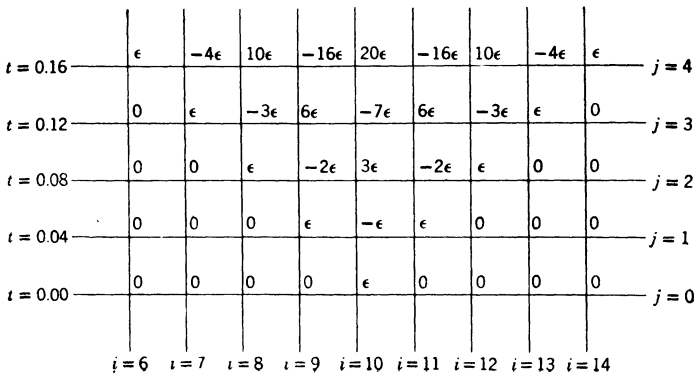
$$u_{i,3}^{(n)} = u_{i,1}^{(n-1)},$$

так что для каждой итерации достаточно решить всего две системы уравнений. Доказательство этого факта предоставляем читателям.

34. а. $u_{i,\,j+1} = u_{i+1,\,j} - u_{i,\,j} + u_{i-1,\,j}$.

Для нахождения $u_{i,\,j+1}$ складываются два значения, расположенные внизу по диагонали, и вычитается значение, расположенное непосредственно внизу.





ДОПОЛНЕНИЕ

СВОДКА ОСНОВНЫХ ПРАВИЛ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ ФОРТРАН

Б. М. Наймарк

Д. 1. ОСНОВНЫЕ СИМВОЛЫ ЯЗЫКА ФОРТРАН

Любая программа, составленная на языке ФОРТРАН, состоит в конечном счете из следующих основных символов:

Цифры: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Прописные буквы английского алфавита: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z.

Специальные символы: «+» (знак сложения), «—» (знак вычитания), «*» (знак умножения), «/» (знак деления), «**» (знак возведения в степень), «·» (десятичная точка), «,» (запятая), «(» (левая скобка), «)» (правая скобка). Все перечисленные символы входят в любой вариант языка ФОРТРАН. В некоторых вариантах встречаются дополнительные символы (например, знак «$» или отношения «<», «>», «≤» и т. п.).

Из основных символов строятся все конструкции языка. Ниже дается перечень этих конструкций.

Д.2. ЧИСЛА

В языке ФОРТРАН используются два типа чисел: целые и действительные. Целые числа составляются из десятичных цифр, количество которых обычно не превышает пяти. В некоторых вариантах языка максимальное количество цифр может быть бóльшим (в зависимости от типа применяемой машины). Перед целым числом может стоять знак «+» или «—». Абсолютная величина целого числа не может превышать некоторой степени двойки (различной для различных машин). Например, в ФОРТРАНе для IBM 709/7090 целые числа не могут превосходить 2^{17}, а в ФОРТРАНе для UNIVAC 1108 они не могут превышать 2^{15}.

Примеры целых чисел: 3; +1; —2854.

Действительные числа могут записываться различными способами. Первый способ записи: действительное число составляется из десятичных цифр и обязательно десятичной точки. Десятичная точка может располагаться в начале числа, в его конце или между двумя любыми его цифрами. Допустимое количество цифр различно в различных вариантах языка. Чаще всего действительное число составляется не более чем из 8 цифр. Перед числом может стоять знак «+» или «—». Примеры действительных чисел в данной форме записи: 35.; 476.0; .00678; 1.5

Вторая форма записи чисел включает десятичный порядок, обозначаемый буквой E. Число представляется в виде AEB, где A — действительное число, представленное в первой форме, а B — целое. Смысл такой записи состоит в том, что фактическая величина числа равна $A \times 10^B$. Абсолютное значение такого числа может колебаться между 10^{-p} и 10^p, причем p различно в различных вариантах языка. Например, в ФОРТРАНе для IBM 709/7090 действительное число в такой форме записи должно находиться в пределах между 10^{-38} и 10^{38}.

Числа в первой форме записи называются числами с фиксированной запятой, а во второй форме — числами с плавающей запятой. Примеры чисел с плавающей запятой: .675E4 (0.675×10^4); 34.215E7 (34.215×10^7); —0.0573E9 (-0.0573×10^9); —56.453E—12 (-56.453×10^{-12}).

Числа и с фиксированной и с плавающей запятой могут представляться с обычной или удвоенной точностью. Удвоенная точность означает, что число может содержать вдвое больше десятичных цифр, чем в случае обычной точности. Например, число π, представленное с обычной точностью с фиксированной запятой, выглядит как 3.1415926 (и точнее представить его уже нельзя). В случае удвоенной точности можно написать 3.1415926536.

Д.3. ПЕРЕМЕННЫЕ БЕЗ ИНДЕКСОВ

Переменная представляется в программе своим названием. Название переменной составляется из букв и цифр и начинается с буквы. Число символов в названии переменной не должно превышать 6.

Переменные могут быть двух типов: целые переменные и действительные переменные. Целые переменные могут принимать лишь значения, представляемые целыми числами. Название целой переменной должно начинаться с одной из букв I, J, K, L, M, N.

Действительные переменные могут принимать значения, представляемые действительными числами. Название целой переменной не должно начинаться с буквы I, J, K, L, M или N; оно может начинаться с любой другой буквы.

Примеры названий целых переменных: I2; J35; K; N; IND23; MARR. Примеры действительных переменных: D2; PSI; DELTA; Q32; PRESS; GE01.

Не допускается названий переменных, которые совпадают с названиями функций без последней буквы F. Например, если в программе используется функция под названием PRESF, то в той же программе не должно быть переменной PRES.

Д.4. ИНДЕКСЫ

Пусть u — название целой переменной, а c и b — целые числа. Индексом может являться выражение одного из следующих типов: u, c, $u + c$, $u - c$, $c*u$, $c*u + b$, $c*u - b$. Примеры индексов: J, 5, K3 + 2, K5 — 2, 3*J + 7, 12*M — 10.

Данное определение индексов подходит почти для всех вариантов ФОРТРАНа. Однако в некоторых случаях оно обобщается. Например, в ФОРТРАНе для машины UNIVAC 1108 индексом может являться выражение

$$u_1^* u_2^* \ldots {}^* u_n + (\text{или} -) v_1^* v_2^* \ldots {}^* v_m + (\text{или} -) \ldots + \\ + (\text{или} -) w_1^* w_2^* \ldots {}^* w_q,$$

где $u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_m, \ldots, w_1, w_2, \ldots, w_q$ — целые переменные или целые числа. Очевидно, индекс может принимать лишь целые значения.

Д.5. ПЕРЕМЕННЫЕ С ИНДЕКСАМИ

Переменная с индексами представляется в следующем виде:

$$u(v_1, v_2, \ldots, v_n),$$

где u — название переменной, а $v_1, v_2, \ldots, v_n$ — индексы. Переменные с индексами также могут быть целыми и действительными; их названия должны начинаться соответствующим образом. Различные варианты ФОРТРАНа допускают различное количество индексов. Чаще всего переменная не может иметь более трех индексов; однако иногда допускается до 7 индексов (как в ФОРТРАНе для UNIVAC 1108).

Примеры переменных с индексами: KSI (I1, 2*I21+ 3), DELTA (3, 5, J), PRES (7*J1 + 10).

Смысл переменной с индексами состоит в том, что она описывает целый массив значений (целых или действительных), причем каждое конкретное значение индексов соответствует определенному элементу массива. Элементы массива располагаются друг за другом в таком порядке, что первый индекс изменяется быстрее, а последний — медленнее.

Д.6. ВЫРАЖЕНИЯ

Выражением языка ФОРТРАН называется любая последовательность чисел, названий переменных (с индексами или без них) и функций, разделенных знаками арифметических операций, запятыми и скобками. При этом должны выполняться следующие условия.

а. Так как числа, переменные и функции могут быть как целыми, так и вещественными, то составленные из них выражения также могут принадлежать к одному из этих типов. Однако выражения не должны быть смешанными. Переменные или числа одного типа допускаются в выражении другого типа в одном из следующих случаев: 1) целые выражения могут содержать действительные величины лишь в качестве аргументов функций; 2) действительные выражения могут содержать целые величины лишь в качестве аргументов функций, индексов и показателей степени.

б. Числа, переменные и переменные с индексами являются простейшими выражениями, тип которых совпадает с типом числа или типом, обусловленным названием соответствующей величины. Примеры простейших выражений: 3.025, .53E7; PSI5 (I); OMEGA.

в. Функции являются выражениями того типа, который обусловлен их названием; тип аргументов указывается в определении функции. Например, если OMEGF (A, B) является действительной функцией, то OMEGF (C, D) представляет собой действительное выражение; при этом тип величин C и D должен совпадать с типом величин соответственно A и B.

г. При возведении выражения в степень его тип не изменяется; однако нельзя возводить целое выражение в действительную степень.

Замечание. Выражение вида A**B**C не допускается. Необходимо записывать его либо в виде A**(B**C), либо в виде (A**B)**C в зависимости от потребности.

д. Знак «+» или «—», записанный перед выражением, не изменяет его типа.

е. При заключении выражения в скобки его тип не изменяется.

ж. Из простых выражений можно составлять более сложные, соединяя простые выражения знаками арифметических операций. При этом необходимо следовать двум правилам: 1) нельзя записывать подряд два или более символов арифметических операций; 2) соединять можно лишь выражения одного типа.

з. Если в выражении отсутствуют скобки, то вычисления производятся согласно следующим правилам старшинства: сначала выполняется возведение в степень, затем умножение и деление и после этого сложение и вычитание. Если в выражении имеются скобки, то вначале производятся вычисления внутри скобок. Например, выражение A + B/C + D**E*F — G интерпретируется как A + (B/C) + + ((D**E)*F) — G.

Выше были описаны арифметические выражения, т. е. такие, которые принимают численные значения (целые или действительные). В ФОРТРАНе допускаются также логические (или булевы) выражения, принимающие лишь два значения: TRUE (истинно) и FALSE (ложно). Простейшими булевыми выражениями являются сами значения TRUE и FALSE. Более сложными булевыми выражениями являются конструкции вида

$$A \alpha B$$

где A и B — арифметические выражения, а α — операция отношения. Операции отношения обозначаются по-разному в различных вариантах языка ФОРТРАН. Например, имеются следующие обозначения: .EQ. (равно), .NE. (не равно), .GT. (больше, чем), .LT. (меньше, чем), .GE. (больше или равно), .LE. (меньше или равно). Если после вычисления обоих арифметических выражений окажется, что отношение справедливо, то булево выражение принимает значение TRUE; в противном случае это булево выражение принимает значение FALSE.

Булевы выражения указанных типов можно объединять в более сложные с помощью логических операций и скобок. К логическим операциям относятся: .AND. («И», или логическое умножение), .OR. («ИЛИ», или логическое сложение) и некоторые другие. В различных вариантах языка ФОРТРАН логические операции могут обозначаться различным образом. Логические операции определяются с помощью таблиц истинности. Например, операция .AND. определяется следующим образом. Пусть имеются два выражения A и B, которые могут принимать значения TRUE и FALSE. Тогда выражение A. AND. B принимает значение в соответствии с таблицей

A / B	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	FALSE

Д.7. ФУНКЦИИ

Функции являются важнейшими конструкциями языка ФОРТРАН. В ФОРТРАНе имеются 4 типа функций: а) библиотечные функции; б) встроенные функции; в) арифметические операторы-функции и г) функции-подпрограммы. Указанные типы функций различаются по способу их наименования, определения и обращения.

а. *Библиотечные функции.* Библиотечной функцией является конструкция вида α (β), где α — название, обязательно заканчивающееся буквой F, содержащее от 4

до 7 букв или цифр и начинающееся с буквы. β является аргументом функции. В качестве β можно использовать число, переменную с индексом или без индекса или вообще любое арифметическое выражение. Это выражение не должно содержать обращений к той функции, в аргумент которой оно входит. Обычно в ФОРТРАНе уже имеются следующие библиотечные функции: LOGF (X) (натуральный логарифм числа X); SINF (X) (синус угла X в радианах); COSF (X) (косинус числа X в радианах); EXPF (X) (экспонента); SQRTF (X) (корень квадратный из числа X); ATANF (X) (арктангенс числа X); TANHF (X) (тангенс гиперболический). В различных вариантах языка ФОРТРАН возможны дополнительные библиотечные функции (арксинус, арккосинус и т. п.). Кроме того, следуя определенным правилам (описанным в руководствах), можно добавлять библиотечные функции.

Смысл библиотечной функции состоит в том, что при ее появлении в каком-либо выражении вычисляется ее значение для тех значений аргументов, которые получились к данному моменту вычислений по программе.

б. *Встроенные функции.* Встроенной функцией называется конструкция вида $\alpha(\beta_1, \beta_2, \ldots, \beta_n)$. Значение самой функции может быть как целым, так и действительным. То же самое относится и ко всем ее аргументам $\beta_1, \beta_2, \ldots, \beta_n$. Название функции α обязательно заканчивается буквой F. Если значение функции является целым, то ее название должно начинаться с буквы X, в противном случае — с любой другой буквы. В различных вариантах ФОРТРАНа имеются различные наборы встроенных функций. Правила наименования встроенных функций также могут изменяться. Ниже приводится примерная таблица наиболее часто встречающихся встроенных функций. В ней через β, β_1, β_2 или $\beta_1, \beta_2, \ldots, \beta_n$ обозначаются аргументы.

Смысл встроенной функции состоит в том, что при ее появлении в каком-либо выражении вычисляется ее значение для тех значений аргументов, которые получились к данному моменту программы. Формальное различие библиотечных и встроенных функций может состоять лишь в способе их наименования. Отличаются же они тем, что

Смысл функции	Количество аргументов	Название функции	Тип функции	Тип аргументов
$\lvert\beta\rvert$	1	ABSF XABSF	Действительная Целая	Действительный Целый
Наибольшему целому, не превосходящему $\lvert\beta\rvert$, присваивается знак β	1	INTF XINTF	Действительная Целая	Действительный Действительный
$\beta_1 \pmod{\beta_2}$	2	MODF XMODF	Действительная Целая	Действительные Целые
Выбирается $\max(\beta_1, \beta_2, \ldots, \beta_n)$	Не менее 2	MAXOF MAX1F XMAXOF XMAX1F	Действительная Действительная Целая Целая	Целые Действительные Целые Действительные
Выбирается $\min(\beta_1, \beta_2, \ldots, \beta_n)$	Не менее 2	MINOF MIN1F XMINOF XMIN1F	Действительная Действительная Целая Целая	Целые Действительные Целые Действительные
Перевод целого числа в действительную форму	1	FLOATF	Действительная	Целый
Перевод действительного числа в целую форму (как в XINTF)	1	XFIXF	Целая	Действительный
$\lvert\beta_1\rvert$ берется со знаком β_2	2	SIGNF XSIGNF	Действительная Целая	Действительные Целые
$\beta_1 - \min(\beta_1, \beta_2)$	2	DIMF XDIMF	Действительная Целая	Действительные Целые

при обращении к библиотечной функции из многих мест скомпилированной программы используется соответствующая библиотечная подпрограмма лишь «в одном экземпляре»; в то же время определенный участок вставляется в скомпилированную программу всякий раз, когда производится обращение ко встроенной функции. Таким образом, программа для одной встроенной функции может появиться в основной программе несколько раз.

в. *Арифметический оператор-функция.* Такая функция представляет собой конструкцию вида $\alpha(\beta_1, \beta_2, \ldots, \beta_n) = b$, где α — название функции, $\beta_1, \ldots, \beta_n$ — ее аргументы, а b — выражение. Название составляется из букв и цифр; начинается с буквы, заканчивается буквой F и содержит от 4 до 7 символов. Если значение функции целое, то ее название должно начинаться с буквы X, а в противном случае — с любой другой буквы. b является выражением, не содержащим переменных с индексами.

Описанная конструкция является определением оператора-функции. Она должна появляться в программе перед другими конструкциями, где соответствующая функция должна вычисляться.

Аргументы в определении функции являются фиктивными переменными; они описывают лишь способ задания функции, но сами по себе не принимают никаких значений. Необходимо лишь следить, чтобы они правильно отражали тип переменной, т. е. целую или действительную. Смысл оператора-функции состоит в том, что если где-либо в программе появится требование на вычисление функции, то она будет вычислена согласно своему определению; при этом вместо фиктивных аргументов будут подставлены фактические. Фактические аргументы уже могут содержать индексы.

Примером оператора-функции может являться

$$\text{SPUNF}(X, Y) = X*A + Y*B$$

Если в дальнейшем в программе потребуется, например, вычислить выражение C + SPUNF (PSI, R (J)), то вместо X и Y будут подставлены PSI и R (J), после чего вычислится требуемое выражение. Переменные A и B, не входящие в число аргументов функции, будут рассматри-

ваться как ее параметры, т. е. примут значения, получившиеся непосредственно перед вычислением данного выражения.

г. Функция-подпрограмма. Эта конструкция будет описана ниже в разделе «Подпрограммы».

Д.8. ОПЕРАТОРЫ

Оператор является важнейшей конструкцией языка ФОРТРАН. Любая программа этого языка представляет собой последовательность операторов. Оператор представляет собой конструкцию вида

$$n\alpha\beta$$

где n — номер оператора, α — название оператора и β — содержательная часть оператора. Исключение составляет арифметический оператор, который будет описан ниже.

Операторы перфорируются на перфокартах. На перфокарте имеется 80 колонок, из которых 8 последних не используются для записи операторов. В каждой колонке можно отперфорировать один символ или ничего не отперфорировать (т. е. оставить пробел). Пробелы всегда игнорируются, кроме нескольких случаев, специально оговоренных ниже при описании соответствующих операторов. Таким образом, на одной перфокарте можно отперфорировать 72 символа, входящих в оператор.

Первая колонка карты отводится под знак примечания. Если в этой колонке отперфорировать букву С, то дальнейшие символы на этой карте игнорируются при трансляции, но выводятся на печать при отладке; это облегчает чтение программы.

Колонки со 2-й по 5-ю отводятся под номер оператора. Номером оператора может являться любое целое число, содержащее не более 4 десятичных цифр (в некоторых вариантах ФОРТРАНа это ограничение ослаблено). Колонка 6 отводится под символ продолжения. Дело в том, что один оператор может не поместиться на одной перфокарте. Тогда он продолжается на следующей (с колонки 7), а в 6-й колонке этой следующей карты перфорируется какой-либо символ (удобнее всего перфорировать там по-

рядковый номер продолжения); разумеется, если оператор умещается на одной карте, то в 6-й колонке ничего не перфорируется. Всего может быть 9 перфокарт-продолжений.

Начиная с колонки 7 перфорируется название оператора и затем — его содержательная часть (которая, быть может, продолжается на последующие карты).

Имеется два типа операторов — исполнимые и неисполнимые, или описательные. К названиям описательных операторов относятся:

DIMENSION
EQUIVALENCE
COMMON
FORMAT
FUNCTION
SUBROUTINE

К исполнимым операторам относятся:

IF (два типа)
GO TO (три типа)
ASSIGN
DO
CONTINUE
PAUSE
STOP
END
CALL
RETURN
READ
READ TAPE
READ INPUT TAPE
PUNCH
PRINT
WRITE TAPE
WRITE OUTPUT TAPE
REWIND
BACKSPACE

Д.9. ОПИСАТЕЛЬНЫЕ ОПЕРАТОРЫ

а. *Оператор* DIMENSION. Оператор DIMENSION используется для указания максимальных размеров массивов, с которыми оперирует программа. Под массивом подразумевается множество элементов, соответствующее переменной с индексами. Оператор DIMENSION записывается в виде

n DIMENSION $\alpha\,(a_1, \ldots, a_n), \beta\,(b_1, \ldots, b_m), \ldots, \gamma\,(c_1, \ldots, c_q)$

где n — номер оператора (номер может отсутствовать); α, β, . . ., γ — названия переменных с индексами; a_1, a_2, . . ., a_n — максимальные значения индексов переменной α; b_1, b_2, . . ., b_m — максимальные значения индексов переменной β; c_1, c_2, . . ., c_q — максимальные значения индексов переменной γ. Очевидно, все величины a_1, a_2, . . ., c_q должны быть целыми положительными числами. Любая переменная с индексами, появляющаяся в программе, должна быть описана в операторе DIMENSION. Сам оператор DIMENSION должен появляться в самом начале программы перед всеми исполнимыми операторами.

Оператор DIMENSION используется транслятором для отведения памяти под массивы данных. Ячейки памяти отводятся под элементы массивов таким образом, что первый индекс изменяется быстрее всего, а последний — медленнее всего. Не обязательно описывать все массивы в одном операторе DIMENSION: можно использовать несколько таких операторов.

Пример оператора DIMENSION:

DIMENSION A(10), B(7,35)

В соответствии с этим описанием под массив A отводится 10 ячеек, а под массив B — 245 ячеек памяти.

б. *Оператор* EQUIVALENCE. Этот оператор записывается в виде

n EQUIVALENCE (*a*, *b*, *c*, . . .), (*d*, *e*, *f*, . . .), . . .

где n — номер оператора (он может отсутствовать), а a, b, c, d, e, f, . . . — указания переменных. Указание переменной может иметь вид α или $\alpha\,(p)$, где α — название переменной (с индексами или без них), а p — целое без знака.

Смысл данного оператора состоит в том, что при отведении памяти под переменные с индексами или без них ячейки, где запоминаются конкретные значения, располагаются не в произвольном порядке. Именно значения, соответствующие указаниям a, b, c, . . ., запоминаются в одной и той же ячейке; значения, соответствующие указаниям d, e, f, . . ., запоминаются также в одной ячейке

и т. д. При этом под указанием α (p) понимается (p — 1)-я ячейка, следующая за той, где запоминается α. Если указание имеет вид α (без числа p), то принимается $p = 1$, т. е. рассматривается та ячейка, где содержится значение переменной α. Если α является названием переменной с индексами, то под α подразумевается первый элемент соответствующего массива. Следует помнить, что число p не является индексом; оно представляет собой порядковый номер элемента массива.

Пример оператора EQUIVALENCE:

```
EQUIVALENCE (A(1), B(1), C(5)), (D(17), E(3))
```

В соответствии с этим оператором массивы A, B и C (если, конечно, они были описаны как массивы в операторе DIMENSION) располагаются так, что первый элемент массива A, первый элемент массива B и пятый элемент массива C попадут в одну ячейку памяти. Кроме того, массивы D и E расположатся так, что 17-й элемент массива D и 3-й элемент массива E также попадут в одну ячейку.

в. *Оператор* FUNCTION. Оператор FUNCTION имеет следующий вид:

$$n \text{ FUNCTION } \alpha(\beta_1, \beta_2, \ldots, \beta_n)$$

где n — номер оператора (он может отсутствовать), α — название данной функции-подпрограммы, а $\beta_1, \beta_2, \ldots, \beta_n$ — аргументы. Название функции-подпрограммы составляется из букв или цифр, (начинается с буквы I, J, K, L, M или N, если значение функции является целым, и с любой другой буквы в противном случае) и содержит от 1 до 6 букв и (или) цифр.

Данный оператор является заголовком подпрограммы, которая и определяет способ вычисления соответствующей функции. При этом аргументы, как и в арифметическом операторе-функции, фиктивны, т. е. сами по себе не принимают никаких значений, а лишь входят в описание способа вычисления функции. В определении функции должен присутствовать хотя бы один аргумент. В программе, следующей за заголовком, каждый аргумент должен появиться хотя бы один раз в исполнимом операторе. Кроме того, название функции должно появиться хотя бы один раз

в таком исполнимом операторе, который присваивает значение этому названию (например, в левой части арифметического оператора или в списке оператора ввода и т. п.)

В аргументах не указываются индексы. Если какой-либо из аргументов является по смыслу переменной с индексами, то он должен быть описан в операторе DIMENSION, следующем за заголовком функции. Кроме того, фактическая переменная, подставляемая вместо этого фиктивного аргумента, также должна быть описана в другом операторе DIMENSION в основной программе, где вычисляется данная функция.

Пример функции-подпрограммы:

```
FUNCTION STAN(X, Y)
какие-то операторы
STAN=Z+X*C
какие-то операторы
RETURN
```

(оператор возврата в основную программу RETURN описывается ниже). Если где-либо в основной программе встретится, например, CUP*STAN (Q, P(I)), то вместо X и Y будут подставлены Q и P(I), вычислится значение функции и затем продолжатся расчеты по основной программе. Вообще, при требовании на вычисление функции-подпрограммы вместо фиктивных аргументов всегда подставляются фактические. Следует помнить, что название функции не является переменной, которую саму по себе можно использовать в вычислениях. Например, нельзя использовать просто STAN в арифметическом выражении.

Подпрограмма, определяющая данную функцию, является последовательностью операторов. Эта последовательность заканчивается либо оператором END (он указывает, что здесь закончилась соответствующая подпрограмма), либо заголовком следующей функции (он указывает, что здесь началось описание следующей функции).

Нельзя описывать фиктивные переменные (аргументы) в операторе EQUIVALENCE в подпрограмме, описывающей данную функцию.

г. *Оператор* SUBROUTINE. Данный оператор имеет вид:

$$n \text{ SUBROUTINE } \alpha(\beta_1, \beta_2, \ldots, \beta_n)$$

где n — номер оператора (он может отсутствовать), α — название подпрограммы, а $\beta_1, \beta_2, \ldots, \beta_n$ — ее аргументы. Оператор SUBROUTINE отличается от FUNCTION следующим образом: 1) если значение функции FUNCTION вычислялось всякий раз, когда в основной программе появлялось ее название с фактическими аргументами, то для вычислений по подпрограмме SUBROUTINE необходимо использовать специальный оператор CALL (описанный ниже); 2) если в функции-подпрограмме вычисляется лишь одно значение, то в подпрограмме SUBROUTINE можно вычислить множество значений различных переменных. Переменные, значения которых вычисляются, указываются в списке аргументов в качестве «аргументов». Все остальное, сказанное о FUNCTION, относится и к SUBROUTINE. Пример подпрограммы SUBROUTINE:

```
SUBROUTINE DELTA (X, Y, Z, U)
    операторы, описывающие подпрограм-
    му; каждый аргумент должен появиться
    в них хотя бы один раз
RETURN
```

Если в основной программе появится оператор CALL DELTA (A, B, S, Q), то вместо аргументов X и Y будут подставлены A и B, а переменные S и Q примут значения, вычисленные в подпрограмме.

д. *Оператор* COMMON. Оператор COMMON записывается в виде

$$n \text{ COMMON A, B, C, } \ldots$$

где n — номер оператора (он может отсутствовать), а A, B, C, . . . — названия переменных с индексами или без них (сами индексы не должны присутствовать в записи оператора COMMON). Смысл оператора COMMON состоит в следующем. Пусть где-нибудь в программе встретился оператор COMMON A, B, C, ... Под переменные и мас-

сивы, указанные в нем, отвелись некоторые ячейки памяти. Пусть дальше в программе (или в подпрограмме!) опять встретился оператор COMMON U, V, W,... . Тогда переменная (или массив) U расположится в тех же ячейках, что и A; переменная (или массив) V расположится в тех же ячейках, что и B и т. д. Если оператор COMMON встретится еще раз, то новые массивы опять будут располагаться в соответствии с описанным правилом. Таким образом, оператор COMMON служит для того, чтобы записывать данные в одни и те же ячейки памяти. Оператор COMMON отличается от оператора EQUIVALENCE. В самом деле, если написать EQUIVALENCE только один раз, то уже устанавливается, что значения нескольких переменных содержатся в одной ячейке. Если же написать один раз оператор COMMON, то перечисленные там переменные лишь подготавливаются к тому, чтобы соответствующие значения содержались в ячейках, общих с другими переменными; для соответствия одних и тех же ячеек разным переменным необходимо упомянуть оператор COMMON еще раз. Кроме того, оператор EQUIVALENCE не устанавливает соответствия между переменными программы и подпрограммы и не может содержать фиктивных переменных (аргументов). В то же время оператор COMMON устанавливает соответствие между переменными программы и подпрограмм (FUNCTION и SUBROUTINE) и не допускает описание фиктивных переменных.

Оператор COMMON удобен в тех случаях, когда не все величины, используемые в подпрограмме FUNCTION или SUBROUTINE, перечисляются в качестве аргументов. Более того, следует стремиться к тому, чтобы перечислять возможно меньше аргументов, а необходимые величины передавать в подпрограмму с помощью оператора COMMON— это улучшит распределение памяти и сократит машинное время.

Некоторые варианты ФОРТРАНа допускают более общую запись оператора COMMON:

$$n \text{ COMMON } /\alpha_1/\text{A, B, } \ldots /\alpha_2/\text{C, D, } \ldots$$

где α_1, α_2, ... — названия блоков общей памяти (от 1 до 6 букв и цифр, начинаются с буквы), а A, B, ... D, ... — названия переменных (с индексами или без

них). Обобщение состоит в том, что каждый блок α рассматривается как отдельный описанный выше оператор COMMON.

е. *Оператор* FORMAT. Оператор FORMAT является основным средством описания формы и расположения информации при ее вводе и выводе из машины. Оператор FORMAT записывается в виде

$$n \text{ FORMAT}(s_1, s_2, \ldots, s_n)$$

где n — номер оператора (он обязательно должен присутствовать), а $s_1, s_2, \ldots, s_n$ — спецификации. Для указания формы, в которой информация представлена внутри машины и вне ее, используются специальные буквы в соответствии со следующей таблицей:

Внутреннее представление	Буква	Внешнее представление
Действительная переменная	E	Десятичное число с плавающей запятой
Действительная переменная	F	Десятичное число с фиксированной запятой
Целая переменная	I	Десятичное целое число
Двоичное представление восьмеричного числа	O	Восьмеричное число

Приведенной таблице соответствуют 4 типа спецификаций: E$w.d$, F$w.d$, Iw и Ow. Здесь через w обозначается общее количество цифр числа, а через d — количество десятичных цифр после десятичной точки. Например, спецификация F18.5 означает, что на печать (при печати) будет выведено число, занимающее в строке 18 позиций, а после десятичной точки напечатаются 5 цифр.

Имеется два типа спецификаций для описания буквенно-цифровой информации: Aw и wH. Спецификация A означает, что w символов будут введены (или выведены), причем они будут являться значениями переменной или значениями элементов массива. Спецификация типа H означает, что в операторе FORMAT за спецификацией wH

следуют w буквенно-цифровых символов, причем в данном случае пробелы включаются в число символов, а не игнорируются.

Спецификация типа wX задает w пробелов (например, при печати между числами).

Спецификация типа wP (которая приписывается без запятой к спецификации типа F спереди) является масштабным множителем: десятичная точка переносится на w позиций (вправо, если w положительно, и влево, если w отрицательно). Та же спецификация, приписанная к спецификации типа E, переносит десятичную точку и одновременно изменяет порядок.

При печати, перфорации или вводе с перфокарт информация вводится построчно; одна строка печати содержит обычно 120 символов, а одна перфокарта — 72 символа. Кроме того, на магнитную ленту также можно записывать (или считывать с нее) «строки» по 120 символов. Ни одна спецификация не должна соответствовать более чем 120 (или 72 в случае перфокарт) символам. Чтобы описать несколько строк в одном операторе FORMAT, соответствующие спецификации отделяются друг от друга косой чертой «/» (которая ставится вместо запятой, причем запятая отсутствует).

Если необходимо повторить несколько одинаковых спецификаций, то можно использовать только одну, поставив перед ней целое число, равное количеству повторений. Кроме того, можно повторять целые группы спецификаций, заключая их в скобки и приписывая перед скобками целое число, равное количеству повторений.

Две косые черты, поставленные подряд, соответствуют пропуску одной строки, три черты — пропуску двух строк и т. д.

Различные варианты ФОРТРАНа имеют дополнительные средства, связанные с оператором FORMAT. Эти дополнительные средства описаны в конкретных руководствах к каждой машине.

Д.10. ИСПОЛНИМЫЕ ОПЕРАТОРЫ

а. *Оператор* GO TO. Имеются три формы этого оператора.

В первой форме он записывается в виде

$$n \text{ GO TO } m$$

где n — номер оператора GO TO, а m — целое без знака или название целой переменной. Этот оператор имеет следующий смысл: после проработки данного GO TO программа начинает работать не со следующего оператора, а с того, номер которого равен m. Если m — не целое число, а название целой переменной, то оператору GO TO должен предшествовать оператор ASSIGN (описанный ниже).

Во второй форме оператор GO TO записывается в виде

$$n \text{ GO TO } (m_1, m_2, \ldots, m_j), m$$

где n — номер этого оператора, $m_1, m_2, \ldots, m_j$ — целые без знака, а m — целая переменная без индексов. Смысл этого оператора состоит в том, что если переменная m приняла значения 1, 2, . . ., j, то следующим начинает работать оператор соответственно с номером $m_1, m_2, \ldots, m_j$. Таким образом, если значение m к моменту проработки данного GO TO равно i, то начинает работать оператор с номером m_i.

В третьей форме оператор GO TO записывается в виде

$$n \text{ GO TO } m, (m_1, m_2, \ldots, m_j)$$

где $m_1, m_2, \ldots, m_j$ — целые числа без знака, а m — целая переменная без индексов. Смысл данного оператора состоит в том, что следующим за GO TO начинает работать тот оператор, номер которого равен m, причем m должно совпадать с одним из $m_1, m_2, \ldots, m_j$. Таким образом, в отличие от предыдущей формы записи в данном случае переменная m может принимать лишь значения, перечисленные внутри скобок в операторе GO TO.

6. *Оператор* ASSIGN. Номер оператора в языке ФОРТРАН не является просто целой переменной. Поэтому нельзя написать GO TO m и присвоить целой переменной m некоторое значение. Чтобы указать, что переменная является номером оператора, используется оператор ASSIGN, который записывается в виде

$$n \text{ ASSIGN } a \text{ TO } m$$

где n — номер оператора, a — целое без знака, а m — название целой переменной.

в. *Оператор* IF. Имеются два типа оператора IF. Первый записывается в виде

$$n \ \text{IF}(a)\, n_1, n_2, n_3$$

где n — номер оператора, a — арифметическое выражение, а n_1, n_2, n_3 — целые без знака. Оператор работает так: если значение арифметического выражения внутри скобок отрицательно, то следующим за IF начинает работать оператор с номером n_1; если это выражение в точности равно нулю, то следующим работает оператор с номером n_2, а если это выражение положительно, то следующим работает оператор с номером n_3.

Оператор IF во второй форме записывается в виде

$$n \ \text{IF}(b)\, \Pi$$

где n — номер оператора, а b — булево выражение. Через П обозначен любой исполнимый оператор языка ФОРТРАН, кроме DO. Работает данный оператор следующим образом: если булево выражение внутри скобок принимает значение TRUE, то выполняется оператор П и затем следующий по порядку. Если же булево выражение приняло значение FALSE, то оператор внутри IF пропускается и сразу начинает работать следующий.

г. *Оператор* DO. С помощью данного оператора в программе организуются циклы, т. е. повторные вычисления согласно одному и тому же участку программы. Данный оператор записывается в виде

$$n \ \text{DO} \ m \ i = j_1, j_2, j_3$$

где n — номер оператора DO, m — предел действия DO (номер некоторого другого оператора), а j_1, j_2, j_3 — либо целые числа, либо целые переменные без индексов. i всегда является целой переменной без индексов. Число j_1 (или значение переменной j_1) должно быть положительным. Оператор DO работает следующим образом. Участок программы, начиная с оператора, непосредственно следующего за DO, и заканчивая оператором, номер которого

равен m (т. е. пределом действия DO), выполняется последовательно несколько раз; при этом индекс i принимает значения j_1, $j_1 + j_3$, $j_1 + 2j_3$ и т. д. Повторение заканчивается, если значение индекса i станет больше (или равно) j_2, если j_3 положительно, или меньше (или равно) j_2, если j_3 отрицательно.

Возможна сокращенная форма записи оператора DO, а именно:

$$n \text{ DO } m\, i = j_1, j_2$$

Здесь подразумевается, что $j_3 = 1$ (если $j_1 < j_2$) или $j_3 = -1$ (если $j_1 > j_2$).

Внутри одного цикла DO может быть другой, внутри этого последнего — третий и т. д. При использовании оператора DO необходимо следовать перечисленным ниже правилам.

Правило 1. Если внутри одного DO имеется другой оператор DO, то область действия внутреннего DO должна целиком находиться внутри внешнего DO.

Правило 2. Нельзя передавать управление внутрь области действия DO иначе, как через сам этот оператор. Имеются два исключения из этого правила. Во-первых, внутри цикла DO может содержаться обращение к функции или подпрограмме; тогда, конечно, управление будет фактически передаваться куда-то в другую часть программы, а потом — обратно внутрь цикла. Во-вторых, можно выйти из цикла DO до его окончания (например, с помощью оператора GO TO), а затем вернуться обратно; при этом, однако, нельзя изменять ни i, ни j_1, j_2 и j_3.

Правило 3. Нельзя изменять значений i, j_1, j_2 и j_3 внутри цикла (хотя, конечно, ими можно пользоваться в качестве целых переменных, например индексов).

Правило 4. Оператор IF не может быть последним исполняемым оператором в области действия DO.

д. *Оператор* CONTINUE. Этот оператор является «пустым», т. е. не производит никаких действий. Он записывается в виде

$$n \text{ CONTINUE}$$

где n — номер оператора. Он используется в качестве последнего в области действия DO в тех случаях, когда не-

обходимо удовлетворить четвертому правилу для оператора DO. Он не увеличивает машинного времени; поэтому обычно им всегда заканчивают область действия DO.

е. *Арифметический оператор.* Арифметический оператор записывается в виде

$$n \; a = b$$

где n — номер оператора, a — название переменной (с индексами или без них), а b — выражение. Смысл арифметического оператора состоит в том, что вычисляется выражение b и его значение присваивается переменной a. При этом переменная a и выражение b могут быть различных типов. Если выражение b принимает действительное значение, а переменная a целая, то от значения b отбрасывается дробная часть, а остаток переводится в целую форму. Обратно, если значение выражения b целое, а переменная a действительная, то полученное целое значение переводится в действительную форму.

ж. *Оператор* CALL. Этот оператор используется для обращения к подпрограммам SUBROUTINE. Он записывается в форме

$$n \text{ CALL } \alpha(b_1, b_2, \ldots, b_n)$$

где n — номер оператора, α — название подпрограммы, к которой производится обращение, а b_1, b_2, . . ., b_n — фактические параметры. Каждый аргумент может быть одной из следующих величин: 1) числом (целым или действительным); 2) переменной (целой или действительной, с индексами или без них); 3) арифметическим выражением; 4) буквенно-цифровыми символами, причем им должна предшествовать спецификация wH, где w — количество символов в аргументе, считая пробелы; 5) названием функций или других подпрограмм SUBROUTINE. Аргументы в операторе CALL должны соответствовать фиктивным аргументам вызываемой подпрограммы своим порядком следования, типом, размерами массива и т. п.

Смысл оператора CALL состоит в подстановке фактических параметров на место фиктивных в подпрограмме и в последующем вычислении по этой подпрограмме. После

выполнения оператора CALL начинает работать следующий за ним оператор.

з. *Оператор* RETURN. Этот оператор записывается в виде

n RETURN

где *n* — номер оператора. Оператором RETURN заканчивается фактическая работа операторов подпрограммы FUNCTION или SUBROUTINE. Эти подпрограммы должны обязательно содержать по крайней мере один оператор RETURN, хотя их там может быть и несколько. В результате работы оператора RETURN происходит возврат в то место основной программы, откуда вызывалась подпрограмма.

и. *Оператор* END. Этот оператор записывается в виде

END

Он является информацией для транслятора, указывая, что закончилась программа (или подпрограмма).

к. *Оператор* PAUSE. Этот оператор записывается в виде

n PAUSE *m*

или

n PAUSE

где *n* — номер этого оператора, а *m* — целое число (в некоторых системах ФОРТРАНа восьмеричное). Этот оператор заставляет машину остановиться, причем в адресном поле регистра на пульте появится число *m*. Если *m* в операторе отсутствует, то считается, что $m = 0$. После пуска машина возобновит работу, начиная со следующего оператора.

л. *Оператор* STOP. Этот оператор отличается от PAUSE тем, что машина останавливается и не возобновляет работу по прерванной программе после пуска.

Операторы PAUSE и STOP почти никогда не используются, так как вместо останова современные машины передают управление на другие программы, ожидающие своей очереди.

м. *Операторы ввода и вывода.* Имеется несколько операторов ввода и вывода. Во многих из них присутствует список переменных. Список переменных всегда записывается в виде

$$\alpha_1, \alpha_2, \ldots, \alpha_n$$

где α_1, α_2, . . ., α_n — элементы списка. Простейшим элементом списка является переменная без индексов. Элементом списка может являться также значение переменной с индексами при конкретных их значениях. Если необходимо, чтобы элемент списка перечислял все значения переменных с индексами при изменении одного из них, то такой элемент записывается в виде

$$(\beta_1(q_1), \beta_2(q_2), \ldots, \beta_m(q_m), \ j = j_1, \ j_2)$$

где β_1, β_2, . . ., β_m — названия переменных с индексами, q_1, q_2, . . ., q_m — их совокупности индексов, j — целая переменная (изменяющийся индекс), а j_1 и j_2 — пределы изменения индекса j при перечислении элементов. Индекс j должен содержаться в каждой совокупности индексов q_1, q_2, . . ., q_m. Смысл такого перечисления состоит в том, что в список заносятся все элементы массивов β_1, β_2, . . ., β_m для значений индекса j, равных j_1, $j_1 + 1$,... . . ., j_2.

Такой сложный элемент списка можно опять объединять с другими и перечислять полученную совокупность по второму индексу и т. д. При этом индекс j каждый раз является целой переменной, а индексные пределы j_1 и j_2 — числами. Заменить индексные пределы целыми переменными можно в том случае, если эти переменные уже встретились в списке раньше при их перечислении по порядку.

Список переменных может соответствовать списку спецификаций в операторе FORMAT. При этом каждой переменной соответствует определенная спецификация. Соответствие устанавливается одновременным перечислением

обоих списков по порядку. При этом, если список спецификаций закончился, а список переменных еще не исчерпан, то список спецификаций начинает разворачиваться заново, начиная с последней левой скобки.

Оператор READ записывается в виде

n READ *m*, β

где *n* — номер этого оператора, *m* — номер оператора FORMAT, где задаются соответствующие спецификации переменных, а β — список переменных. Переменные, перечисленные в списке, вводятся с перфокарт согласно спецификациям оператора FORMAT с заданным номером.

Оператор READ INPUT TAPE записывается в виде

n READ INPUT TAPE *i*, *m*, β

где *n* — номер этого оператора, *i* — целое число без знака или целая переменная, *m* — номер оператора FORMAT, а β — список переменных. Отличие этого оператора от оператора READ состоит в том, что информация считывается с *i*-го ленточного блока. При этом номер *i* является символическим: на каждой машине устанавливается соответствие между символическими и фактическими номерами ленточных блоков.

Оператор READ TAPE имеет вид

n READ TAPE *i*, β

где *n* — номер оператора, *i* — номер блока (опять символический) и β — список переменных. С помощью этого оператора данные считываются с ленты, причем если в предыдущем операторе данные присутствовали на ленте в двоично-десятичной форме, то теперь они должны быть в двоичной форме. Лента, с которой действует данный оператор, является «внутренней» (она не может использоваться, например, непосредственно для печати вне машины).

Оператор PUNCH имеет вид

n PUNCH *m*, β

где *n* — номер этого оператора, *m* — номер оператора FORMAT, где заданы спецификации, а β — список переменных. С помощью этого оператора информация выво-

дится на перфокарты в соответствии с заданной спецификацией.

Оператор PRINT имеет вид

$$n \text{ PRINT } m, \beta$$

где n — номер этого оператора, m — номер оператора FORMAT, где задаются спецификации, а β — список переменных. С помощью такого оператора печатаются значения перечисленных в списке переменных в соответствии с заданной спецификацией.

Оператор WRITE OUTPUT TAPE имеет вид

$$n \text{ WRITE OUTPUT TAPE } i, m, \beta$$

Он отличается от оператора READ OUTPUT TAPE тем, что теперь происходит не считывание, а запись.

Оператор WRITE TAPE, который записывается в виде

$$n \text{ WRITE TAPE } i, \beta$$

отличается от READ TAPE тем, что теперь происходит не считывание, а запись.

Оператор

$$n \text{ END FILE } i$$

используется для того, чтобы поставить на ленте специальную метку, означающую конец зоны с записями «строк» (см. описание оператора FORMAT).

Оператор

$$n \text{ REWIND } i$$

где i — номер ленточного блока (символический), используется для обратной перемотки ленты.

Оператор

$$n \text{ BACKSPACE } i$$

где i — номер ленточного блока, используется для установки ленты на начало записи.

В различных вариантах языка ФОРТРАН могут использоваться дополнительные средства ввода — вывода и оперирования с внешней информацией. Эти средства описываются в соответствующих руководствах.

Оглавление

Д. МАК-КРАКЕН, У. ДОРН

**ЧИСЛЕННЫЕ МЕТОДЫ
И ПРОГРАММИРОВАНИЕ НА ФОРТРАНЕ**

Редактор *А. Г. Крылов*
Художник *А. Г. Антонова*
Художественный редактор
В. М. Шаповалов
Технический редактор *Н. А. Турсукова*

Сдано в производство 26/III 1969 г.
Подписано к печати 8/IX 1969 г.
Бумага № 1 84×108$^{1}/_{32}$, 9,13 бум. л.
30,66 усл. печ. л. Уч.-изд. л. 26=75
Изд. № 1/4817. Цена 2 р. 11 к. Зак. 907

ИЗДАТЕЛЬСТВО «МИР»
Москва, 1-й Рижский пер., 2

Московская типография № 16
Главполиграфпрома
Комитета по печати при Совете
Министров СССР
Москва, Трехпрудный пер., 9